Aplikasi Graf Pada Algoritma *Pathfinding* Dalam *Video Game*

Luqman Faizlani Kusnadi and 13512054[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13512054@std.stei.itb.ac.id

Abstract—Pathfinding **adalah proses pencarian rute dari suatu titik ke titik lain.** ***Pathfinding*** **banyak digunakan dalam berbagai** ***Video Game*****, terutama pada permainan berbasis peta seperti permainan bergenre strategi atau** ***role-playing game.*** **Makalah ini akan membahas tentang bagaimana graf digunakan dan direprensentasikan dalam** ***video game*** **terutama dengan jenis** ***real-time strategy*****.**

Keywords**—Pathfinding, Video Game.**

I. PENDAHULUAN

A. Video Game

Video game adalah permainan elektronik dimana pemain berinteraksi dengan antarmuka yang disediakan. Pemain memberikan masukan menggunakan perangkat yang ada seperti *keyboard*, *joystic,* atau perangkat lainnya. Pemain juga mendapatkan *feedback* visual dari perangkat *video* seperti layar monitor. *Video game* mengalami perkembangan dari tahun ke tahun seiring dengan meningkatnya kemampuan komputer untuk melakukan komputasi dengan cepat

.

B. Real-Time Strategy

Video game dapat dikategorikan dalam berbagai jenis tergantung pada cara memainkannya seperti *role-playing games, strategy, shooter* dan lain-lain. Salah satu jenis yang banyak dikembangkan saat ini adalah *real-time strategy* (RTS). Pada permainan jenis ini, pemain bermain secara *real-time* atau tidak bergantian. Pemain biasnya memiliki satu atau lebih karakter/unit yang dapat dikendalikan. Selain itu terdapat sebuah peta dimana pemain dapat berinteraksi denngan memberi perintah pada unit-unit yang dimilikinya. Misalnya perintah untuk bergerak menuju suatu tempat dalam peta. Disinilah dibutuhkan algoritma pathfinding untuk mencari jalur terpendek dalam mencapai posisi tujuan yang diinginkan.

II. DASAR TEORI

A. Graf

Graf digunakan untuk merepresentasikan objek – objek diskrit dan hubungan antara satu objek dengan objek lainnya. Representasi dari graf dengan menyatakan objek sebagai bulatan atau titik, dan hubungan objek dengan garis.

Graf G didefinisikan sebagai pasangan himpuna (V,E) dengan

V = himpunan tak kosong dari simpul
$= \{ v_1, v_2, v_3, \ldots, v_n \}$
E = himpunan sisi yang menghubungkan sepasang simpul
$= \{ e_1, e_2, e_3, \ldots, e_n \}$

Atau dapat ditulis singkat dengan notasi G = (V,E)

Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf, maka secara umum graf dapat digolongkan menjadi dua jenis:

1. Graf sederhana

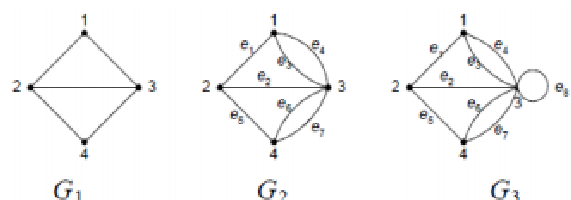
Graf yang tidak mengandung gelang maupun sisi ganda dinamakan graf sederhana.

2. Graf tak-sederhana

Graf yang mengandung sisi ganda atau gelang dinamakan graf tak-sederhana. Ada dua macam graf tak sederhana, yaitu graf ganda dan graf semu.

Graf ganda adalah graf yang mengandung sisi ganda. Sebuah graf memiliki sisi ganda jika ada 2 buah simpul yang dihubungkan lebih dari satu sisi.

Graf semu adalah graf yang memiliki sisi gelang (loop). Sisi gelang adalah sisi yang menghubungkan sebuah simpul dengan simpul itu sendiri. Berikut merupakan contoh ketiga jenis graf

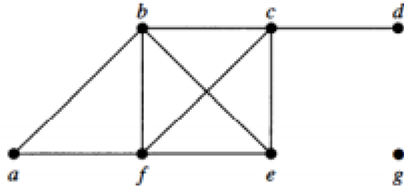


Gambar 2.1 – (a) graf sederhana, (b) graf ganda, dan (c) graf semu[3]

Sisi graf dapat memiliki orientasi arah. Berdasarkan arah dari sisi, graf dibedakan menjadi 2 jenis :

1. Graf tak-berarah

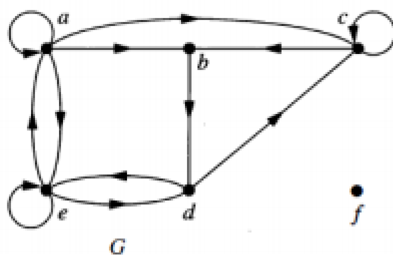
   Graf yang sisinya tidak memiliki orientasi arah disebut graf tak-berarah. Pada graf tak-berarah, urutan pasangan simpul pada sisi tidak diperhatikan. Sebuah sisi e = (u, v) sama dengan e = (v, u)



Gambar 2.2 – Graf tak berarah[3]

2. Graf berarah

   Graf yang setiap sisinya memiliki orientasi arah disebuh graf berarah. Pada graf berarah, sebuah sisi dikenal juga sebagai busur (*arc*). Pada graf berarah, (u, v) dan (v, u) menyatakan dua buah sisi yang berbeda. Pada sebuah sisi (u, v), simpul u menyatakan simpul asal (*initial vertex)* dan simpul v menyatakan simpul terminal (*terminal vertex).*



Gambar 2.3 – Graf berarah[3]

Sisi pada graf dapat memiliki bobot atau tidak. Berdasarkan bobot pada sisinya, graf dapat digolongkan menjadi dua :

1. Graf berbobot (*weighted graph)*

   Graf berbobot adalah graf yang setiap sisinya memiliki bobot. Bobot pada sisi graf dapat merepresentasikan kapasitas, biaya, atau keuntungan.

2. Graf tak-berbobot (*unweighted graph)*

   Graf tak-berbobot adalah graf yang setiap sisinya tidak memiliki bobot.

A. *Pathfinding*

Pathfinding adalah proses mencari rute dari dua buah titik berbeda. Inti dari *pathfinding* adalah pencarian pada graf, dimulai dari sebuah simpul dan dilanjutkan ke simpul-simpul lain sampai simpul tujuan tercapai, yang biasanya bertujuan untuk mencari rute terdekat.

Beberapa algoritma telah dikembangkan untuk melakukan *pathfinding* ini Algoritma yang umum digunakan adalah:

1. Algoritma Djikstra

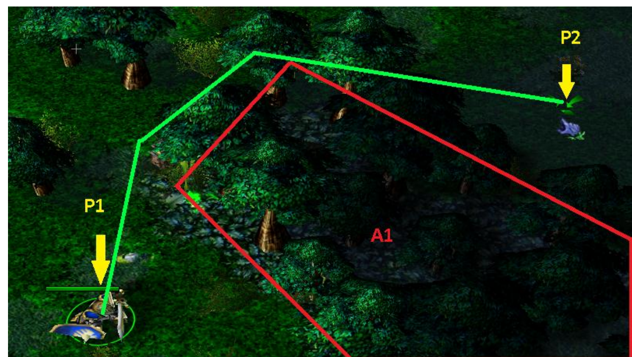
   Algoritma ini dimulai dengan sebuah simpul awal dan himpunan simpul-simpul yang bisa dicapai langsung dari simpul tersebut. Dari himpunan tersebut dipilih simpul dengan jarak terdekat dan diberi tanda bahwa simpul tersebut telah dikunjungi. Masukkan semua simpul yang bisa dicapai langsung dari simpul tersebut dalam himpunan awal. Hal ini diulangi sampai simpul tujuan tercapai. Pseudo code algoritma Djikstra dapat dilihat pada Appendix A

2. Algoritma A*

   A* merupakan variasi dari algoritma Djikstra. Algoritma A* menggunakan *heuristic* untuk meningkatkan performa algoritma Djikstra. Jika nilai *heuristic* adalah 0, algoritma ini akan sama dengan algoritma Djikstra. Pseudo code algoritma A* dapat dilihat pada Appendix B.

III. Penggunaan Pathfinding pada *Video Game Real-Time Strategy*

Pada *video game* berjenis *real-time* strategy (RTS) biasanya terdapat sebuah peta dimana karakter atau unit yang dimainkan pemain dapat berinteraksi dengan peta atau dunia yang ada. Salah satunya adalah berjalan dari posisi saat ini menuju posisi lain yang diinginkan. Cara yang umum digunakan untuk memindahkan posisi karakter adalah dengan menentukan titik tujuan yang diiningkan dengan klik menggunakan tetikus maupun dengan cara lain. Sehingga pemain tidak menentukan rute yang ingin dilewati, namun komputer yang menentukan rute yang terdekat. Berikut contoh memindah posisi karakter pada sebuah permainan RTS yaitu Warcraft III.



Gambar 3.1 – Pathfinding pada Warcraft III

Pada gambar tersbut terlihat sebuah karakter sedang diberi perintah untuk bergerak dari posisi awal (P1) menuju posisi akhir (P2), namun diantara kedua posisi tersebut terdapat area pepohonan (A1) yang tidak dapat dilewati. Sehingga komputer mencari rute terdekat untuk mencapai tujuan, yang dalam hal ini digambarkan dengan garis berwarna hijau.

Seperti yang telah dijelaskan sebelumnya, *pathfinding* merupakan pencarian pada graf. Untuk itu diperlukan suatu cara untuk merepresentasikan peta dalam sebuah *game* dalam bentuk graf. Beberapa cara yang sering

digunakan antara lain menggunakan:

1. Kisi (*Grid*)

Berdasarkan peta yang ada dibuat kisi-kisi dengan ukuran seragam yang merepresentasikan area mana yang dapat dilewati dan tidak. Berikut contohnya.

Gambar 3.2 – Representasigraf dengan kisi[1]

Pada gambar diatas area yang dapat dilewati direpresentasikan dengan kisi berwarna hijau. Graf yang digunakan adalah graf lengkap dari 4 simpul sudut tiap kisi yang digabung dengan dengan kisi-kisi yang lain. Dengan cara ini,representasi graf menjadi mudah, namun cara ini akan memakan banyak memori dan komputasi karena menghasilkan banyak simpul. Selain itu, untuk peta berbentuk rumit seperti pada gambar diatas akan membuat representasi menjadi tidak akurat.

2. *Waypoints*

Cara ini dilakukan dengan membuat titik-titik pada area yang dapat dilewati dan juga keterhubungan satu sama lain. Berikut contoh representasi dengan *waypoints*

Gambar3.3 – Representasi dengan waypoint[1]

Cara lebih fleksibel untuk berbagai bentuk peta karena kita hanya perlu menyesuaikan tempat simpul-simpul tersebut diletakkan.

Misalkan sebuah karakter akan bergerak dari titik A menuju titik B.Hal pertama yang dilakukan adalah mencari simpul terdekat dengan titik A dan titik B. Setelah itu algoritma *pathfinding* dijalankan untuk mencari simpul terdekat dari titik B dari simpul terdekat dari titik A. Kelemahan dengan menggunakan cara ini adalah karakter akan selalu bergerak mengikuti sisi graf sehingga pergerakan karakter menjadi kurang natural, seperti pada contoh karakter akan bergerak dari titik A menuju titik B secara *zig-zag*.

3. *Navigation mesh (Navmesh)*

Cara yang ketiga adalah dengan mneggunakan *navmesh*. Dengan cara ini dibuntuk *polygon-polygon* untuk merepresentasikan area yang bias dilewati pada peta seperti pada gambar di bawah ini:

Gambar 3.4 – Representasi dengan navmesh[1]

Polygon-polygon yang dibentuk haruslah *polygon* konveks, karena kita dapat yakin bahwa dua buah titik dalam area *polygon* konveks dapat dihubungkan dengan sebuah garis lurus. Tiap *polygon* adalah simpul dalam graf, jika sebuah *polygon* bersentuhan dengan *polygon* lain, maka kedua simpul terhubung. Dengan cara ini jumlah simpul menjadi lebih sedikit, dan juga cara ini dapat digunakan pada peta berbentuk rumit sekalipun.

Misalkan karakter akan bergerak dari titik A menuju titik B, hal pertama yang dilakukan adalah menentukan titik A dan titik B berada dalam area *polygon* / simpul mana. Selanjutanya dijalankan algoritma *pathfinding* untuk mencari rute terpendedk.

V. KESIMPULAN

Algoritma *pathfinding* dapat diterapkan pada *video game* berjenis *real-time strategy*, namun tidak secara langsung. Untuk itu dibutuhkan cara untuk merepresentasikan peta yang ada dalam bentuk graf. Dari beberapa cara untuk merepresentasikan, dapat diambil

kesimpulan bahwa representasi dengan menggunakan *navigation mesh (navmesh)* adalah cara yang paling effisien, karena membutuhkan simpul yang lebih sedikit dan juga dapat dugunakan untuk peta dengan bentuk yang kompleks.

VII. APPENDIX

A. *Pseudo code* algoritma Djikstra [3]

```
procedure Dijkstra (input m: matrix, a: first node)
{ To find the shortest distance from first vertex a to all other vertices.
         Input: adjacency matrix (m) from a weighted graph G and first vertex a
         Output: shortest route from a to all other vertices
}
DICTIONARY
        S1, S2, ..., Sn : integer {Array of Integer}
         D1, D2, ..., Dn : integer {Array of Integer}
ALGORITHM
        {Initialization}
         For i←1 to n do
                 Si ← 0
                Di ← Mai

        {Step 1}
        Sa ← 1 {for the first step, the initial vertex must be the shortest}
        Da ← ∞ {there is no shortest route from vertex a to a}

         {Step 2, 3, ..., n-1}
        For i←2 to n-1 do
                Find j so that Sj = 0 and Dj = min(D1, D2, ..., Dn)
                Sj ← 1 {j chosen as the shortest route}
                 Renew Di, for i=1, 2, 3, ..., n with Di(new)=min { di(old), dj + mij }
```

B. *Pseudo code* algoritma A*

```
function A*(start,goal)
    closedset := the empty set    // The set of nodes already evaluated.
    openset := {start}    // The set of tentative nodes to be evaluated, initially
containing the start node
    came_from := the empty map    // The map of navigated nodes.

    g_score[start] := 0    // Cost from start along best known path.
    // Estimated total cost from start to goal through y.
    f_score[start] := g_score[start] + heuristic_cost_estimate(start, goal)

    while openset is not empty
        current := the node in openset having the lowest f_score[] value
        if current = goal
            return reconstruct_path(came_from, goal)

        remove current from openset
        add current to closedset
        for each neighbor in neighbor_nodes(current)
            if neighbor in closedset
                continue
            tentative_g_score := g_score[current] + dist_between(current,neighbor)

            if neighbor not in openset or tentative_g_score < g_score[neighbor]
                came_from[neighbor] := current
                g_score[neighbor] := tentative_g_score
                f_score[neighbor] := g_score[neighbor] + heuristic_cost_estimate(neighbor,
goal)
                if neighbor not in openset
                    add neighbor to openset

    return failure

function reconstruct_path(came_from,current)
    total_path := [current]
```

VII. ACKNOWLEDGMENT

Ucapan terima kasih sebesar-besarnya saya tujukan kepada dosen kuliah Matematika Diskrit saya Bapak Rinaldi Munir.atas bimbingan di dalam maupun luar kelas. Berbagai referensi saya dapatkan dari kuliah ataupun kegiatan di luar kuliah sehingga saya dapat menyelesaikan makalah ini. Saya juga berterima kasih kepada teman – teman dan pihak lain yang telah membantu saya dalam proses pembuatan makalah ini.

REFERENCES

[1] http://www.ai-blog.net/archives/000152.html, diaksespada 9 Desember 2014
[2] http://mgrenier.me/2011/06/pathfinding-concept-the-basics/, diakses pada 9 Desember 2014
[3] Munir, Rinaldi. 2009. Matematika Diskrit. Bandung : Informatika.

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, 10 Desember 2014

Luqman Faizlani Kusnadi, 13512054