Aplikasi Shortest Path dalam Strategy Game "Mount & Blade: Warband"

Kevin Leonardo Handoyo/13509019
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
xmc_z@yahoo.com

Abstrak: ***Game "Mount & Blade: Warband" adalah sebuah game PC yang memadukan unsur Role Playing Game dan Strategy Game. Dalam permainan ini kita mengendalikan seorang karakter yang dapat memimpin pasukan untuk menguasai benteng-benteng dan kota-kota yang terdapat di dalam dunia game tersebut. Permainan ini mengharuskan kita sering berpindah-pindah tempat misalnya dari suatu kota ke kota lain agar kita bisa mempertahankan dan memperluas daerah kekuasaan kita. Selama kita berpindah dari suatu lokasi ke lokasi lain, kecepatan sangat diperlukan karena semakin lama kita di jalan maka semakin banyak kerugian yang kita alami. Karena itulah kita harus bisa mencari rute perjalanan yang efektif di dalam permainan ini. Makalah ini akan membahas tentang pemanfaatan pengetahuan ktia tentang graf berbobot yaitu masalah shortest path dalam menentukan rute perjalanan paling efektif.***

Kata kunci: ***Shortest Path, Graf berbobot, Algoritma Dijkstra, Algoritma A****

I. PENDAHULUAN

Permainan "Mount & Blade: Warband" merupakan sebuah permainan PC yang cukup unik dan menarik untuk dimainkan. Selain memiliki unsur Role Playing Game dimana kita harus mempersenjatai dan menaikan atribut dari karakter utama yang kita mainkan, permainan ini juga mempunyai unsur Strategy Game dimana kita sebagai seorang pemimpin pasukan harus menyusun komposisi pasukan, melatih pasukan, mengatur strategi dalam pertempuran, mengatur rute perjalanan dalam rangka menguasai dunia di dalam game, dan sebagainya.

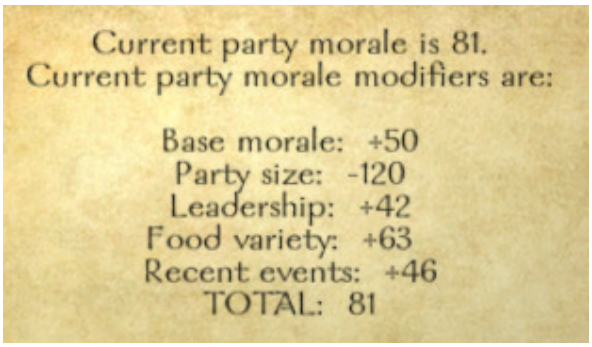
Pasukan yang kita pimpin memegang peran sangat penting dalam perjalanan kita di dalam game untuk menaklukkan pihak lawan karena sistem pertempuran memiliki skala yang besar sehingga pasukan dan strategi yang kita siapkan dapat sangat menentukan hasil pertempuran. Pasukan yang kita pimpin ini memiliki sebuah parameter yang disebut "morale". "Morale" ini sangat mempengaruhi kondisi pasukan kita, yaitu membuat pasukan kita tetap bisa efektif dalam pertempuran, serta tidak melarikan diri dari pimpinan kita.

Dalam permainan ini, untuk dapat menguasai dan menaklukkan kota-kota lawan, kita harus sering berpindah tempat, untuk misalnya melindungi kastil atau menghadiri turnamen-turnamen yang bisa menunjang karakter kita di game. Namun, ketika kita dalam perjalanan, bila terlalu lama berjalan maka "morale" pasukan kita akan terus menurun, selain itu, persediaan bahan makanan juga akan menipis sehingga pengeluaran bertambah. Karena itulah perlu dipikirkan cara agar kita bisa memiliki rute yang paling baik untuk berjalan ke beberapa kota. Pemecahan masalah ini mengacu pada cara penyelesaian shortest path dalam graf berbobot.

II. DASAR TEORI

A. Sistem Morale dan Persediaan Makanan

Morale pasukan di dalam permainan ini mempengaruhi banyak faktor yaitu keefektifan bertarung di medan pertempuran, kecepatan bergerak, serta bila morale terlalu rendah, pasukan kita akan kabur dari kita. Morale akan menurun nilainya apabila kita mengalami beberapa hal yaitu kalah bertempur, kehabisan bahan makanan, serta bila terlalu lama dalam perjalanan tanpa menang pertempuran. Sebaliknya, morale dapat ditambah dengan cara menang turnamen, menang pertempuran, punya banyak variasi persediaan makanan, *leadership skill*, dan beberapa faktor lain. Jumlah pasukan yang kita miliki juga mempengaruhi morale. Semakin banyak pasukan maka akan semakin besar nilai pengurangan moralenya.



Gambar 1: Tampilan penghitungan morale

Persediaan makanan juga mempunyai peran penting dan juga dapat mempengaruhi morale pasukan kita. Persediaan makanan harus bervariasi dan mencukupi agar morale pasukan kita tidak turun. Semakin banyak waktu dihabiskan maka persediaan makanan akan semakin menipis.

B. Graf

Graf adalah sebuah struktur diskrit yang digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Sebuah graf memiliki notasi sbb:

Graf G = (V,E)

Dimana V = himpunan tidak kosong dari simpul-simpul (disebut juga verticles), dan E = himpunan sisi (edges) yang menghubungkan sepasang simpul.

Bila ditinjau dari ada atau tidaknya gelang atau sisi ganda pada suatu graf, maka jenis graf dapat dibedakan menjadi:

1. Graf sederhana : Graf yang tidak mengandung gelang ataupun sisi ganda
2. Graf tak sederhana : Graf yang mengandung sisi ganda atau gelang

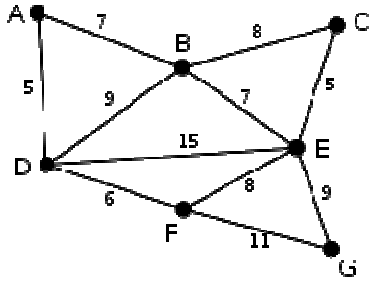
Sedangkan bila ditinjau dari ada atau tidaknya arah pada sisi-sisi graf, maka graf dapat dibedakan menjadi 2 jenis yaitu:

1. Graf tak berarah : Graf yang sisinya tidak memiliki orientasi arah
2. Graf berarah : Graf yang setiap sisinya memiliki orientasi arah

Dalam sebuah graf ada yang disebut sirkuit dan lintasan. Lintasan yang memiliki panjang n dari simpul awal V0 ke simpul tujuan Vn dalam graf G adalah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk V0,E1,V1,E2,…,Vn-1,En,Vn sedemikian sehingga E1=(V0,V1),…, En=(Vn-1,Vn) adalah sisi-sisi dari graf G. Sirkuit dalam sebuah graf adalah lintasan yang berawal dan berakhir pada simpul yang sama.

C. Graf Berbobot

Graf berbobot adalah graf yang memiliki sebuah nilai yang disebut juga bobot pada setiap sisi yang terdapat dalam graf tersebut. Bobot tersebut merupakan angka. Graf berbobot ini dapat diaplikasikan dalam berbagai persoalan, misalnya tentang graf yang setiap simpulnya mewakili kota, dan setiap sisinya merupakan jalan antar kota sedangkan bobotnya merupakan panjang jalan tersebut.



Gambar 2: Graf berbobot

D. Shortest Path Problem

Shortest Path Problem dalam teori graf adalah permasalahan untuk mencari lintasan dari suatu simpul tertentu, sebut saja simpul asal, menuju simpul tujuan, agar lintasan yang dipilih merupakan lintasan dengan bobot yang paling sedikit. Persoalan ini sering digunakan untuk merepresentasikan masalah dimana kita harus mencari jalan yang paling singkat dari suatu tempat untuk mencapai tempat lain.

Beberapa algoritma yang dapat digunakan untuk menyelesaikan masalah shortest path ini yaitu:

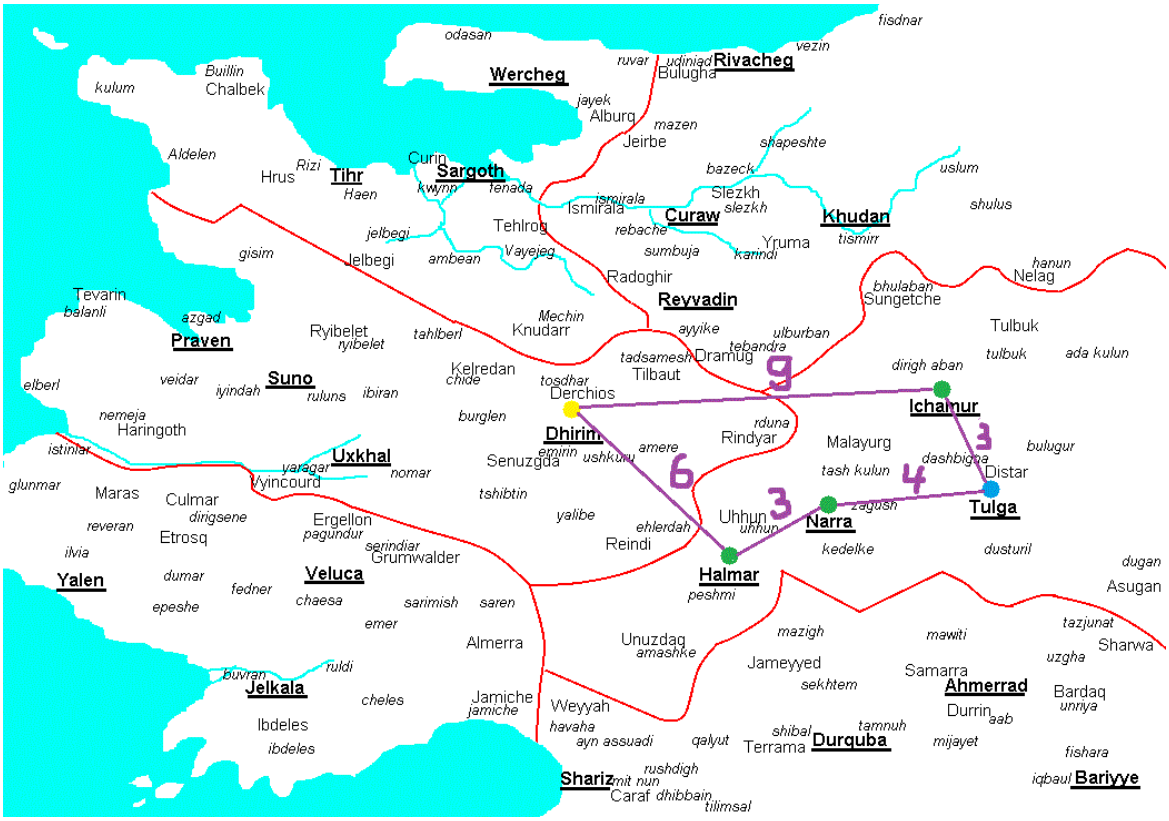
1. Algoritma Dijkstra untuk menyelesaikan masalah lintasan satu pasang, satu asal, satu tujuan
2. Algoritma Bellman-Ford untuk menyelesaikan masalah lintasan satu asal bila bobot boleh negatif
3. Algoritma A* untuk menyelesaikan masalah lintasan satu pasang dengan cara heuristic untuk mempercepat pencarian
4. Algoritma Floyd-Warshall untuk menyelesaikan masalah lintasan semua pasangan
5. Algoritma Johnson untuk menyelesaikan masalah lintasan semua pasangan pada graf sparse

III. METODE

Penyelesaian dari permasalahan Shortest Path dapat diterapkan dalam permainan ini karena inti dari permasalahannya yaitu sama-sama mencari jalan tercepat dari suatu tempat ke tempat lain. Oleh sebab itu, maka penyelesaian dari masalah shortest path juga dapat diterapkan dalam kasus game ini.

Dari beberapa algoritma penyelesaian yang tersedia, yang paling sesuai dengan permasalahan dalam permainan ini yaitu dengan menggunakan penyelesaian dengan algoritma Dijkstra dan algoritma A* karena memiliki masalah lintasan satu pasang, satu tujuan, dan satu asal, serta tidak memiliki bobot negatif.

Contoh ilustrasi permasalahannya adalah sebagai berikut:



Gambar 3: Ilustrasi graf jarak dalam peta game

Dari gambar ilustrasi di atas, contoh permasalahannya adalah pada saat kita berada di kota Dhirim, lalu kita ingin pergi ke kota Tulga, tampak ada 2 alternatif lintasan, yaitu dengan melalui kota Ichamur lalu langsung menuju Tulga, atau dengan lintasan lain yaitu melalui kota Halmar, kota Narra, kemudian baru ke kota Tulga. Dari kota Dhirim menuju kota Ichamur, dapat diperkirakan jarak sekitar 9 satuan, dari kota Ichamur ke kota Tulga jarak dianggap 3 satuan, jarak kota Dhirim dan kota Halmar dianggap 6 satuan, jarak kota Halmar dan Narra dianggap 3 satuan, dan jarak kota Narra dengan kota Tulga dianggap 4 satuan.

```
function Dijkstra(Graph, source):
    for each vertex v in Graph:           // Initializations
        dist[v] := infinity ;              // Unknown distance function from source to v
        previous[v] := undefined ;         // Previous node in optimal path from source
    end for ;
    dist[source] := 0 ;                    // Distance from source to source
    Q := the set of all nodes in Graph ;
    // All nodes in the graph are unoptimized - thus are in Q
    while Q is not empty:                  // The main loop
        u := vertex in Q with smallest dist[] ;
        if dist[u] = infinity:
            break ;                        // all remaining vertices are inaccessible from source
        fi ;
        remove u from Q ;
        for each neighbor v of u:          // where v has not yet been removed from Q.
            alt := dist[u] + dist_between(u, v) ;
            if alt < dist[v]:              // Relax (u,v,a)
                dist[v] := alt ;
                previous[v] := u ;
            fi  ;
        end for ;
    end while ;
    return dist[] ;
end Dijkstra.
```

Gambar 4: Pseudocode untuk algoritma Dijkstra

```
function A*(start,goal)
    closedset := the empty set                 // The set of nodes already evaluated.
    openset := set containing the initial node // The set of tentative nodes to be evaluated.
    came_from := the empty map                 // The map of navigated nodes.
    g_score[start] := 0                        // Distance from start along optimal path.
    h_score[start] := heuristic_estimate_of_distance(start, goal)
    f_score[start] := h_score[start]           // Estimated total distance from start to goal through y.
    while openset is not empty
        x := the node in openset having the lowest f_score[] value
        if x = goal
            return reconstruct_path(came_from, came_from[goal])
        remove x from openset
        add x to closedset
        foreach y in neighbor_nodes(x)
            if y in closedset
                continue
            tentative_g_score := g_score[x] + dist_between(x,y)

            if y not in openset
                add y to openset
                tentative_is_better := true
            elseif tentative_g_score < g_score[y]
                tentative_is_better := true
            else
                tentative_is_better := false
            if tentative_is_better = true
                came_from[y] := x

                g_score[y] := tentative_g_score
                h_score[y] := heuristic_estimate_of_distance(y, goal)
                f_score[y] := g_score[y] + h_score[y]
    return failure

function reconstruct_path(came_from, current_node)
    if came_from[current_node] is set
        p = reconstruct_path(came_from, came_from[current_node])
        return (p + current_node)
    else
        return current_node
```

*Gambar 5: Pseudocode untuk algoritma A**

A. *Penyelesaian dengan Algoritma Dijkstra*

Langkah penyelesaian masalah dengan algoritma Dijkstra dimulai dari simpul awal, diperiksa semua sisinya, kemudian dipilih sisi yang mempunyai bobot paling kecil. Setelah sampai ke simpul berikutnya, proses tersebut diulang lagi, dicari lagi sisi dengan bobot terkecil dan menuju simpul selanjutnya, dan begitu seterusnya.

Jadi bila contoh di atas diselesaikan dengan algoritma Dijkstra akan mengikuti langkah sebagai berikut:

1. Dari Dhirim dipilih kota berikutnya adalah Halmar karena jaraknya adalah 6 (lebih dekat daripada ke Ichamur dengan jarak 9)
2. Dari Halmar hanya bisa satu jalan yaitu melanjutkan ke Narra dengan jarak 3
3. Dari Narra menuju tujuan yaitu Tulga dengan jarak 4

Total jarak yang diambil dengan algoritma Dijkstra adalah 6+3+4=13 satuan. Padahal bila kita perhatikan, jarak bila menempuh lintasan menuju Ichamur dan langsung ke Tulga hanya akan memakan jarak 12 satuan. Hal ini disebabkan karena algoritma ini hanya memilih sisi terpendek untuk setiap node sehingga hanya jarak local yang diperhitungkan dan tidak bisa melihat jarak total secara keseluruhan. Namun, algoritma ini memiliki kecepatan yang tinggi dalam menyelesaikan permasalahan shortest path karena tidak melakukan banyak proses untuk melihat jarak secara keseluruhan.

B. *Penyelesaian dengan Algoritma A**

Langkah penyelesaian dengan algoritma A* berbeda cukup jauh dengan algoritma Dijkstra, bila algoritma Dijkstra hanya memilih jalan yang tersingkat tanpa mempedulikan jalan lainnya, sedangkan algoritma A* menyimpan setiap kemungkinan lintasan yang ada. Prosesnya ia menyimpan setiap jarak dari simpul, setelah itu ia memilih jarak terkecil dari semua lintasan yang tersimpan dan maju ke simpul berikutnya, tetapi lintasan yang tersimpan tidak dibuang dan bila lintasan lain yang tersimpan suatu saat memiliki jarak terkecil, maka proses akan "mundur" lagi ke simpul terakhir lintasan itu dan seterusnya sampai semua kemungkinan lintasan mencapai simpul akhir.

Jika contoh di atas diselesaikan dengan algoritma A* maka langkahnya:

1. lintasan 1 : Dhirim-Ichamur = 9
   lintasan 2 : Dhirim-Halmar = 6
2. lintasan 2 lebih pendek sehingga diproses terlebih dulu
   lintasan 1 : Dhirim-Ichamur = 9
   lintasan 2: Dhirim-Halmar-Narra = 9

3. lintasan sama panjang, lanjutkan lintasan 2
   lintasan 1: Dhirim-Ichamur = 9
   lintasan 2: Dhirim-Halmar-Narra-Tulga = 13
4. lintasan 2 sudah sampai tujuan dan lintasan 1 lebih kecil jaraknya dan belum selesai. Lanjutkan lintasan 1
   lintasan 1: Dhirim-Ichamur-Tulga = 12
   lintasan 2: Dhirim-Halmar-Narra-Tulga = 13
5. Semua lintasan telah mencapai tujuan, dibandingkan mana nilai terkecilnya. Karena nilai lintasan 1 lebih kecil maka dipilih penyelesaian dengan lintasan 1.

Penyelesaian dengan algoritma A* terbukti lebih akurat dibandingkan dengan algoritma Dijkstra karena algoritma A* memperhitungkan semua kemungkinan lintasan yang ada dalam graf. Namun, proses yang ditempuh akan jauh lebih panjang dan memakan waktu lama.

IV. KESIMPULAN

Di dalam game ini, yang dibutuhkan terutama adalah keakuratan untuk mencari jalan yang memang terpendek. Selain itu, rute perjalanan yang ada di dalam game tidak begitu rumit sehingga tidak akan memakan waktu terlalu banyak apabila menggunakan algoritma A*. Oleh sebab itu, maka algoritma A* dapat diterapkan untuk menyelesaikan permasalahan dalam game untuk dapat mengoptimalkan rute perjalanan agar morale pasukan tidak berkurang sia-sia.

REFERENSI

[1] Munir, Rinaldi. 2005. Matematika Diskrit. Bandung: Penerbit Informatika
[2] http://mountandblade.wikia.com/wiki/Party#Morale_Modifiers
[3] http://en.wikipedia.org/wiki/Dijkstra's_algorithm
[4] http://en.wikipedia.org/wiki/A*_search_algorithm
[5] http://en.wikipedia.org/wiki/Shortest_path_problem

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, 17 Desember 2010

ttd

Kevin Leonardo Handoyo/13509019