Metode Path Finding pada Game 3D Menggunakan Algoritma A* dengan Navigation Mesh

Freddi Yonathan - 13509012
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509012@std.stei.itb.ac.id

***Abstrak* — Path finding merupakan metode yang sangat dibutuhkan pada berbagai game, terutama game 3d. Path finding digunakan untuk menentukan arah pergerakan suatu objek berdasarkan keadaan lokasi dan object di sekitarnya dari satu titik ke titik lain. Navigation Mesh merupakan suatu bentuk struktur data pengganti graf yang dapat lebih menyederhakan perhitungan path finding. Dalam makalah ini akan dijelaskan lebih lanjut tentang metode path finding serta penggunaan navigation mesh dengan algoritma A* pada metode path finding.**

***Kata Kunci*—A*, A star, Greedy, Navigation Mesh, Path Finding.**

I. PENDAHULUAN

Permasalah path finding merupakan masalah yang sangat penting dalam berbagai game, terutama game 3d. Path finding digunakan untuk menentukan arah pergerakan suatu objek dari satu tempat ke tempat lain berdasarkan keadaan peta dan object lainnya. Dalam pemecahan path finding akan dibutuhkan algoritma yang dapat dengan cepat memproses dan menghasilkan arah yang terpendek untuk mencapai suatu lokasi. Bayangkan apabila algoritma yang digunakan cukup lama untuk menemukan path, seorang pemain yang menggerakan karakter permainannya harus menunggu beberapa detik supaya karakternya bergerak setelah perintah diberikan akan sangat mengganggu permainan, terutama dalam permaian real-time.

Algoritma yang digunakan untuk path finding sudah banyak yang ditemukan. Misalnya Bellman–Ford, Dijkstra, Floyd–Warshall, dan A*. Semua algortima tersebut sebenarnya mencari rute terpendek melalui graf. Namun pada game, dalam pencarian rute, lokasi akan direpresentasikan sebagai graf juga. Dalam makalah ini algoritma yang akan dibahas hanyalah A* karena dirasa sebagai algoritma yang cukup mangkus dan dipakai dalam berbagai game.

II. ALGORITMA GREEDY

Algoritma Greedy dibahas karena algoritma A* merupakan perkembangan algortima Dijkstra yang merupakan turunan dari algoritma Greedy.

Algoritma Greedy merupakan algoritma heuristic yang digunakan untuk mencari solusi optimum secara umum. Algoritma ini tidak dapat memastikan semua solusi yang dihasilkan merupakan solusi optimum, namun sebagaian besar permasalahan yang diselesaikan dengan Greedy diharapkan mencapai solusi optimum.

Algoritma Greedy memiliki 5elemen utama, yaitu :

1. Himpunan Kandidat
2. Himpunan Solusi
3. Fungsi Seleksi
4. Fungsi Kelayakan
5. Fungsi Objektif

Himpunan Kandidat adalah himpunan pembentuk solusi, yaitu himpunan dari sekumpulan pilihan yang dapat diambil salah satunya sebagai solusi dari setiap tahap.

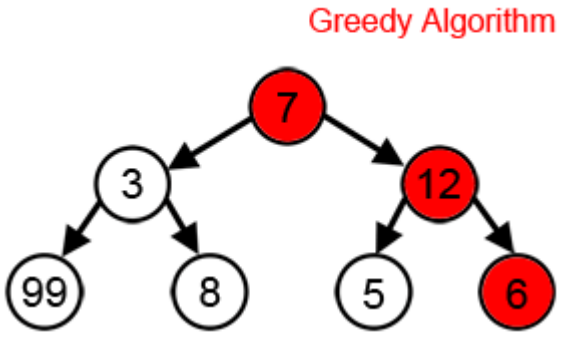
Himpunan Solusi adalah himpunan bagian dari himpunan kandidat yang terpilih sebagai solusi pada suatu tahap tertentu.

Fungsi Seleksi adalah fungsi yang memilih bagaimana himpunan solusi terpilih sebagai solusi dari himpunan kandidiat.

Fungsi Kelayakan adalah fungsi yang memeriksa apakah suatu kandidat layak untuk menjadi solusi. Kandidat yang tidak layak tidak akan diperiksa oleh fungsi seleksi serta tidak akan menjadi solusi.

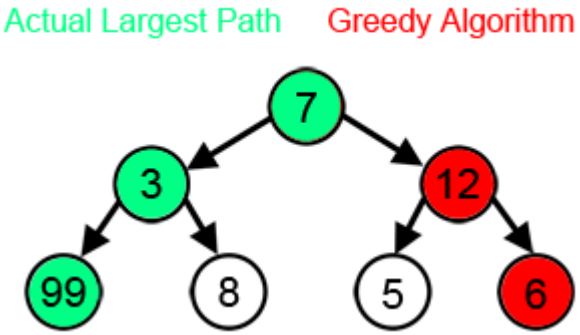
Fungsi Objektif adalah fungsi yang memaksimumkan atau meminimumkan solusi.

Algoritma Greedy akan selalu mangambil solusi optimum pada saat itu dan tidak memperhitungkan langkah lainnya untuk pengambilan keputusan selanjutnya. Namun, solusi optimum yang dipilih saat waktu tertentu belum tentu merupakan solusi optimum pada akhirnya. Pada kenyataannya seringkali algoritma Greedy tidak menghasilkan solusi optimum. Pada gambar di bawah ini menunjukkan bagaimana cara kerja Greedy dalam mengambil path terbesar.



Gambar 1. Ilustrasi Persoalan dengan Greedy

Pada gambar di atas terlihat bahwa Greedy akan selalu mengambil keputusan optimal pada saat itu saja. Ketika berada di node akar (node berbobot 7) dengan algoritma Greedy akan mengambil solusi untuk bergerak ke node berbobot 12, karena tentu saja 12 lebih besar daripada 3. Kemudian pada node berbobot 12 akan diambil solusi untuk bergerak ke node berbobot 6. Namun pada kenyataannya solusi optimum untuk permasalahan ini adalah dengan bergerak ke node berbobot 3 dan bergerak ke node berbobot 99, seperti yang terlihat pada gambar di bawah ini.



Gambar 2. Ilustrasi Solusi Greedy Dibandingkan Solusi Optimum

III. ALGORITMA A*

A* merupakan algoritma yang pertama kali dikembangkan oleh Nils Nilsson pada 1964 berdasarkan algoritma Dijkstra. Saat itu, algoritma ini dinamakan algoritma A1. Pada 1967 Bertram Raphael mengembangkan lebih jauh algoritma ini dan menyebutnya A2, namun tidak dapat membuktikan keunggulannya dibandingkan algoritma sebelumnya. Kemudian pada 1968 Peter E. menunjukkan bukti keoptimalan algoritma A2 dibandingkan dengan A1. Kemudian algoritma A2 dinyaatakan sebagai algoritma paling optimal untuk kasus tersebut, dan diganti namanya menjadi A*. Berdasarkan waktu, algoritma ini lebih baik daripada algoritma Dijkstra dengan pencarian heuristik.

A* memiliki 2 fungsi utama dalam menentukan solusi terbaik. Fungsi pertama disebut sebagai g(x) merupakan fungsi yang digunakan untuk menghitung total cost yang dibutuhkan dari node awal menuju node tertentu. Fungsi kedua yang biasa disebut sebagai h(x) merupakan fungsi perkiraan total cost yang diperkirakan dari suatu node ke node akhir.

Pada A*, setiap node dari node awal ditelusuri kemudian dihitung cost dari tiap-tiap node dan dimasukkan ke tabel prioritas. Node dengan cost paling rendah akan diberikan tingkat prioritas paling tinggi. Kemudian pencarian dilanjutkan pada node dengan nilai prioritas tertinggi pada tabel.

Berikut adalah pseudo code untuk algoritma A*.

```
function A*(start,goal)
   closedset := the empty set
   openset := {start}
   came_from := the empty map

   g[start] := 0
   h[start] := heuristic_cost(start, goal)
   f[start] := g[start] + h[start]

   while openset is not empty
     x := the node in openset having the lowest
          f[] 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[x] +
                            dist_between(x,y)

       if y not in openset
           add y to openset
           tentative_is_better := true
       else if 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[y] := tentative_g_score
           h[y] := heuristic_cost(y, goal)
           f[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
```

`closedset` merupakan tabel node yang telah ditelusuri

`openset` merupakan tabel prioritas node

`came_from` path terbaik untuk mencapai suatu node

`g[]` merupakan nilai dari fungsi g(x).

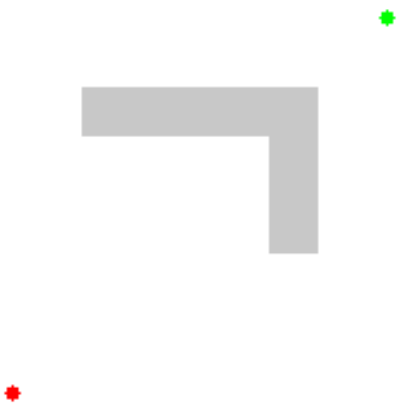
`h[]` merupakan nilai dari fungsi h(x).

`f[]` merupakan jumlah dari g(x) dan h(x).

`dist_between` merupakan fungsi yang menghasilkan jarak antara 2 node yang bersebelahan.

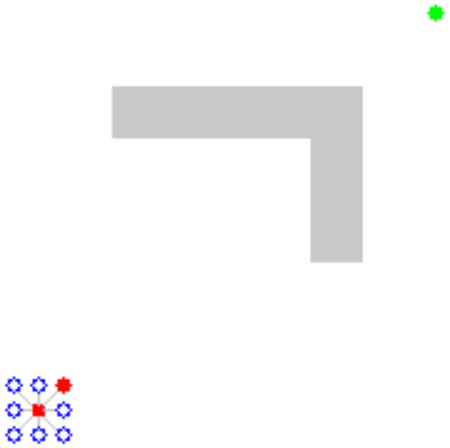
`heuristic_cost` merupakan fungsi yang menghasilkan suatu nilai cost perkiraan berdasarkan kriteria tertentu.

Untuk dapat lebih mengerti mari kita lihat contoh berikut :



Gambar 3. Ilustrasi Contoh Soal untuk Algoritma A*

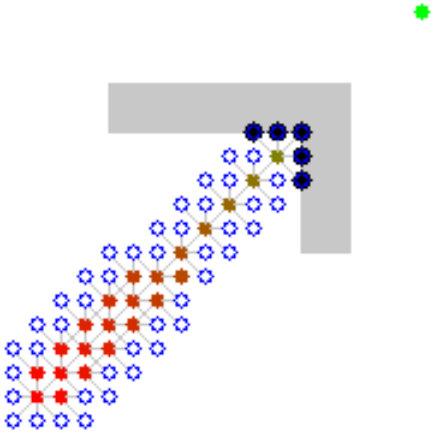
Gambar diatas memperlihatkan soal awal. Titik merah merupakan titik awal, titik hijau merupakan titik akhir yang ingin dituju. Bangun berwarna abu-abu menggambarkan penghalang yang tidak dapat dilewati. Algoritma A* dimulai dengan mengecek tiap node yang bersebelahan dengan titik yang sedang ditelusuri, dalam hal ini titik awal. Kemudian priority tiap titik tersebut dimasukkan ke dalam sebuah tabel dan node dengan prioritas tertinggi pada tabel dihapus dari tabel dan dijadikan node yang ditelusuri.



Gambar 4. Ilustrasi Langkah Pertama dengan Algoritma A*

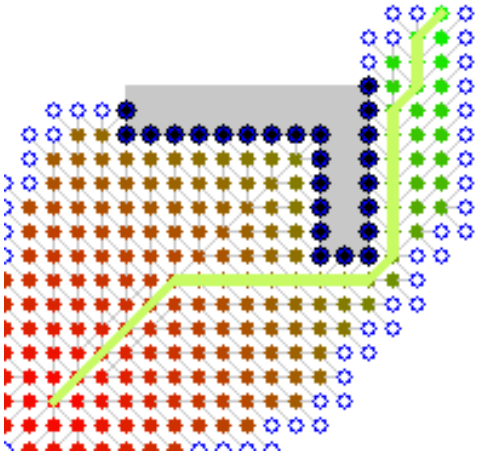
Gambar di atas memperlihatkan tahap pertama. Titik merah kedua merupakan node dengan prioritas tertinggi, titik putih merupakan node yang ada dalam tabel prioritas.

Setelah proses diulang beberapa kali akan terlihat seperti gambar dibawah ini.



Gambar 5. Ilustrasi setelah beberapa kali proses dengan Algoritma A*

Titik biru merupakan node yang bertabrakan dengan penghalang dan tidak akan dimasukkan ke tabel.



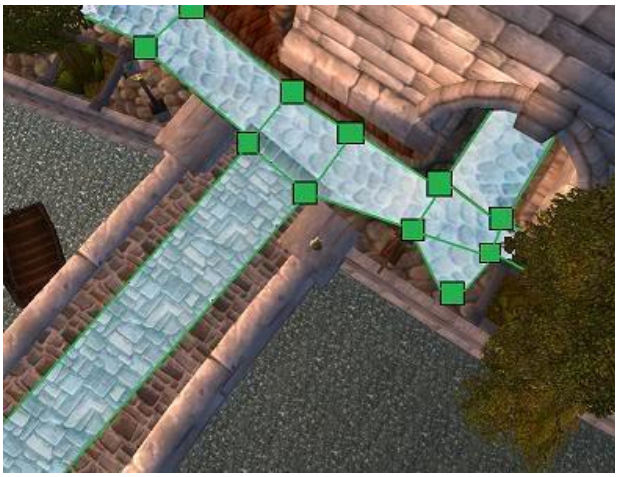
Gambar 6. Ilustrasi Kondisi Solusi Algoritma A*

Setelah proses mencapai titik akhir, maka akan terlihat seperti gambar di atas. Garis hijau merupakan solusi yang didapat, Sedangkan tiap titik yang berwarna merupakan node yang sudah ditelusuri.

IV. NAVIGATION MESH

Navigation Mesh merupakan suatu bentuk struktur data yang terutama sangat dibutuhkan dalam path finding dalam game 3d. Mesh merupakan suatu bentuk yang dapat dianggap sebagai graf dengan node yang berbentuk poligon. Dalam path finding, setiap bentuk poligon dalam navigation mesh akan dijadikan sebagai node dari graf, dengan nilai koordinat biasanya diambil dari titik tengah poligon tersebut. Ilustrasi berikut akan memperlihatkan bagaimana navigation mesh direpresentasikan.

Gambar 7. Contoh Gambar Suatu Lokasi pada Game



Gambar 8. Contoh Gambar Suatu Lokasi dengan Navigation Mesh

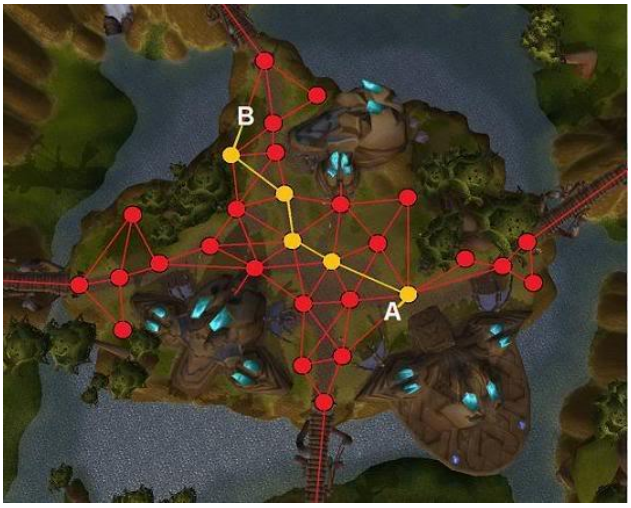
Terlihat pada contoh gambar di atas bagaimana Navigation Mesh dibentuk. Bagian yang ditandai dengan biru, yaitu bagian yang bisa dilewati pemain dan hanya bagian tersebut yang dihitung dalam penentuan path finding.

Ada beberapa alasan mengapa game 3D menggunakan Navigation Mesh dan bukan graf.

1. Perhitungan dengan Navigation Mesh lebih sedikit dibandingkan dengan graf, sehingga lebih cepat.
2. Pergerakan objek akan lebih halus.
3. Lebih mudah menghitung ulang arah apabila objek penghalang ditambahkan.

Untuk menjelaskan 3 keuntungan navigation Mesh dibandingkan dengan graf, lihat gambar-gambar berikut.

Gambar 9. Contoh Lokasi yang Belum Ditambahkan Graf atau Navigation Mesh



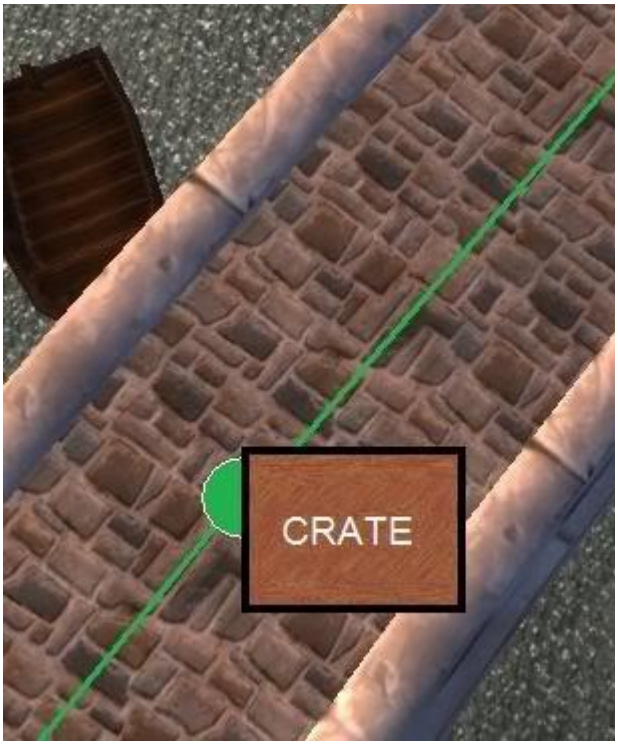
Gambar 10. Contoh Lokasi Setelah Ditambahkan Graf

Gambar 11. Contoh Lokasi Setelah Ditambahkan Navigation Mesh

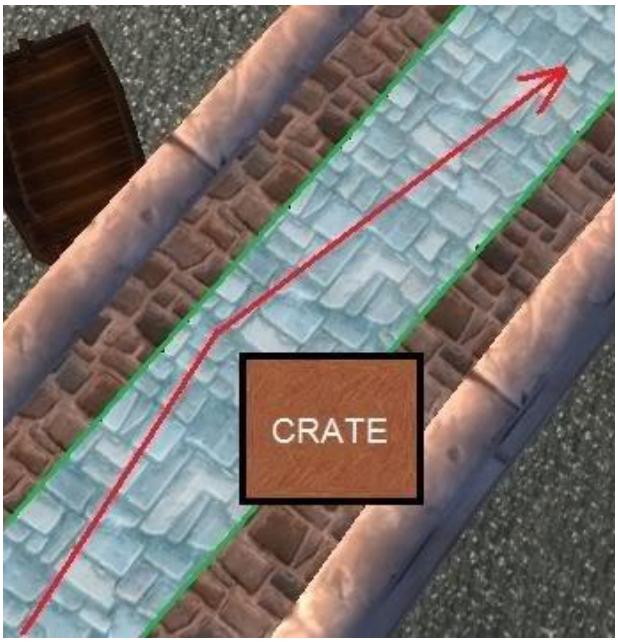
Dari gambar-gambar tersebut terlihat bahwa jumlah node pada graf jauh lebih banyak daripada navigation mesh, yang menyebabkan perhitungan lebih cepat. Selain itu, dengan menggunakan graf, arah pergerakan objek

akan zig-zag seperti yang terlihat pada gambar 10.

Dengan suatu algoritma lain, maka arah gerakan pada Navigation Mesh dapat menjadi lurus, dan dengan algoritma yang sama, apabila suatu objek penghalang ditambahkan, maka rute baru (gambar 13) akan segera dibuat pada Navigation Mesh yang sama. Sedangkan pada graf, apabila suatu objek penghalang ditambahkan menutupi banyak node, maka graf butuh digambar ulang.



Gambar 12. Contoh Penambahan Penghalang pada Jalur Gerak



Gambar 13. Contoh Pengubahan Jalur Gerak Akibat Penghalang

V. KESIMPULAN

Metode Path Finding merupakan metode yang sangat penting dalam game 3D untuk menunjang cara bergerak objek, terutama objek yang digerakkan oleh AI. Metode Path Finding perlu dilakukan sangat cepat, sebab setiap pergerakan yang ada pada game tidak hanya soal bergerak ke suatu lokasi tertentu, tetapi juga melakukan hal lainnya bersamaan dengan bergerak. Selain itu, sering kali perubahan tujuan gerak kurang dalam 1 detik, misalkan dalam permainan Defense of the Ancients, pemain biasanya akan terus menerus memerintahkan hero yang dimainkan untuk terus berpindah tempat secara cepat. Maka dari itu, algoritma A* dan Navigation Mesh diperlukan supaya dapat melakukan arah pergerakan ke tempat tujuan dengan sangat cepat.

REFERENCES

[1]. Munir, Rinaldi, *Diktat Kuliah IF3051 Strategi Algoritma*. Bandung: STEI ITB.

[2]. http://en.wikipedia.org/wiki/A*_search_algorithm diakses pada 12.00 tanggal 8Desember 2011

[3]. http://en.wikipedia.org/wiki/Greedy_algorithm diakses pada 12.00 tanggal 8Desember 2011

[4]. http://www.ai-blog.net/archives/000152.html diakses pada 10.00 tanggal 8Desember 2011

[5]. http://udn.epicgames.com/Three/NavigationMeshReference.html diakses pada 16.00 tanggal 8Desember 2011

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, 8 Desember 2011

Freddi Yonathan / 13509012