Pencarian Rute untuk Perpindahan Posisi Otomatis Karakter di Game

Vincent Chuardi / 13517103
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
vchuardi@gmail.com

***Abstract*—Di zaman modern seperti sekarang, internet sudah digunakan untuk berbagai hal. Sebagai contoh, internet bisa dipakai untuk berkomunikasi dengan orang-orang yang berada di benua yang berbeda, bentuk berkomunikasi yang dilakukan juga beragam, seperti text-based message, voiced message, video call, dan beragam jenis pesan lainnya. Selain itu, internet juga digunakan untuk mengecek kondisi rumah/ tempat tinggal dan mengatur peralatan elektronik di rumah/ tempat tinggal yang terkoneksi dengan internet (Smart Home). Tetapi, efek internet yang paling besar adalah penggunaan internet oleh pembuat-pembuat game online untuk menambah pengalaman bermain game dengan orang-orang di negara lain. Genre game yang akan diambil pada makalah ini adalah RPG. Penulis akan membahas kemungkinan pemilihan algoritma-algoritma untuk kasus perpindahan posisi dalam game RPG.**

***Keywords*—Game, Graf, A*, Heuristik.**

I. Introduction

Internet (*interconnected-networking*) adalah jaringan yang saling berhubungan. Terdiri dari jaringan-jaringan komputer yang terhubung diseluruh dunia melalui satelit. Internet tersusun dari komputer-komputer yang terhubung di dunia, yang memungkinkan terjadinya pertukaran informasi (dalam bentuk text, gambar, audio, video, dan lainnya). Untuk melakukan terjadinya pertukaran informasi, digunakan protokol standar yaitu Transmision Control Protocol dan Internet Protocol (TCP/IP). Cara menghubungkan jaringan dengan prinsip seperti ini dinamakan *internetworking*.

Game online adalah permainan digital yang menggunakan koneksi internet. Ada banyak sekali platform yang mendukung untuk bermain game online, satu diantaranya adalah game mobile baik berbasis Android maupun iOS. Cakupan pemain game online bisa mencakup satu daerah, satu negara, satu benua, bahkan seluruh dunia. Game online memiliki banyak sekali jenis, seperti jenis strategi yang memerlukan penyusunan strategi tertentu agar bisa menyelesaikan objektif game tersebut. Game ini bisa berbentuk yang memerlukan kemampuan pemain untuk memimpin sebuah pasukan, yang berbentuk mengelola sumber daya hingga membangun sebuah peradaban. Game online juga ada yang berjenis RPG (*Role Playing Game*), game jenis ini memiliki ciri khas unik yang mengajak pemain untuk mengeksplorasi game itu sendiri. Segala kemampuan khusus (*skill), equipment*, dan lainnya bisa dieksplorasi secara bebas oleh *player*. Biasanya game jenis RPG ini, membuat pemain untuk mencari cara untuk menjadi pemain yang terbaik alias nomor satu di game tersebut.

Pada makalah ini, penulis akan menjelaskan mengenai algoritma-algoritma yang bisa digunakan untuk mencari jalur-jalur terpendek untuk perpindahan tempat otomatis. Penulis akan menggunakan contoh berupa game RPG yang bernama Ragnarok Mobile: Eternal Love. Perpindahan tempat yang dimaksud berupa perpindahan dari satu map ke map lainnya dan perpindahan dari satu tempat ke tempat lain dalam satu map.

II. Teori Graf, A*

A. Graf

Graf adalah himpunan benda yang berisi simpul (*vertex/node*) yang dihubungkan oleh sisi (*edge*). Graf digambarkan sebagai titik-titik yang dihubungkan oleh garis. Titik-titik disini melambangkan simpul dan garis melambangkan sisi. Garis yang memiliki arah tertentu dinamakan busur. Arah dalam busur harus diikuti, tidak boleh 'bergerak' berlawanan dengan arah yang ada. Sisi yang menghubungkan suatu simpul dengan simpul itu sendiri dinamakan gelang(kalang/*loop*).

Graf didefinisikan sebagai pasangan himpunan (V, E) dengan V (*vertex/vertices*) adalah himpunan simpul dan E (*edge*) adalah himpunan sisi. Suatu graf G biasanya ditulis dalam bentuk (V,E).

Graf dapat dibedakan berdasarkan:

- Keberadaan gelang/ sisi ganda:
  a) Graf sederhana (*simple graph*)
     Graf yang tidak memiliki sisi ganda dan sisi gelang. Contoh: Graf dengan himpunan simpul V= {1, 2, 3, 4} dan himpunan sisi E= {(1,2), (2,3), (3,4), (1,3)}. Graf ini termasuk graf sederhana karena tidak ada sisi ganda dan sisi gelang.
  b) Graf tidak sederhana (*unsimple-graph*)
     Graf yang memiliki sisi ganda atau sisi gelang, sehingga graf yang memiliki salah satu dari syarat di atas sudah termasuk graf tidak sederhana. Contoh: Graf dengan himpunan simpul V= {1, 2, 3} dan himpunan sisi E= {(1,2), (2,1), (3,3)}. Graf ini adalah graf tidak sederhana karena memiliki sisi ganda dan sisi gelang. Sisi ganda pada graf ini ditandai dengan terdapat sisi (1,2) dan sisi (2,1). Sisi gelang pada graf ini ditandai dengan sisi (3,3).

- Keberadaan arah sisi:
  a) Graf berarah (*directed graph*/*digraph*)

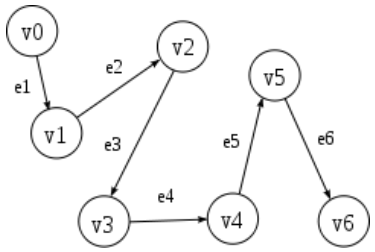
     Graf yang sisinya memiliki arah. Graf berarah biasanya diberi keterangan tambahan. Contoh: Graf berarah dengan himpunan simpul V= {1, 2, 3} dan himpunan sisi E= {(1,2), (2,3), (1,3)}. Kata penanda jenis graf ini adalah 'graf berarah'. Graf ini memiliki sisi (1,2) dengan arah dari simpul 1 menuju ke simpul 2, sisi (2,3) yang memiliki arah dari simpul 2 ke simpul 3, dan seterusnya. Graf berarah memiliki arah tertentu sehingga tidak boleh sembarangan. Simpul 1 bisa menuju ke simpul 2, tetapi simpul 2 tidak bisa ke simpul 1 karena tidak ada sisi yang menyatakan (2,1) (dari simpul 2 ke simpul 1).

  b) Graf tak-berarah (*undirected graph*)

     Graf yang sisinya tidak memiliki arah.

     Contoh: Graf dengan himpunan simpul V= {1,2,3} dan himpunan sisi E= {(1,2), (1,3)}. Ini adalah graf biasa dengan sisi yang menghubungkan simpul 1 dengan 2, dan sisi yang menghubungkan simpul 1 dengan 3

Sisi juga dapat diberikan suatu nilai. Nilai pada sisi disebut sebagai bobot. Graf yang memiliki bobot dinamakan graf berbobot (*weighted graph*). Graf yang memiliki arah dan bobot disebut sebagai jaringan. Bobot dari sisi dapat merepresentasikan banyak jenis, seperti jarak, waktu, dan sebagainya.



Gambar 2.1 Graf Berarah dan Berbobot[3]

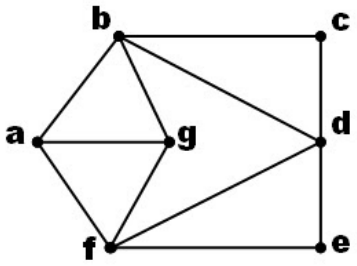
Di dalam graf terdapat banyak istilah-istilah yang sering dipakai untuk menjelaskan graf.

1. Ketetanggaan (Adjacent)

   Dua buah simpul yang terhubung langsung oleh satu sisi dinamakan simpul yang bertetangga.

   Misalnya terdapat himpunan simpul V= {1, 2, 3, 4}, dan himpunan sisi E= {(1,2), (2,3), (3,4), (1,3)}. Simpul 1 dan 2 dikatakan bertetangga karena terdapat sisi yang langsung menghubungkan simpul 1 dan simpul 2. Simpul 1 dan simpul 4 tidak bertetangga karena tidak ada sisi yang menghubungkan simpul 1 dan simpul 4 secara langsung.
2. Lintasan Euler (Path)

   Lintasan euler adalah sebuah rute yang melalui setiap sisi di dalam graf tepat satu kali.

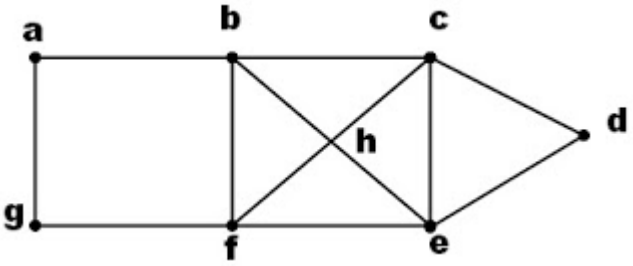


Gambar 2.2 Lintasan Euler[5]

Pada gambar 2.2, lintasan euler adalah a,b,c,d,e,f,g,b,d,f,a,g.

3. Sirkuit/ Siklus (Circuit/ Cycle)

   Sirkuit adalah sebuah rute yang melalui setiap sisi di dalam graf tepat satu kali dan berakhir di simpul awal.



Gambar 2.3 Sirkuit Euler[5]

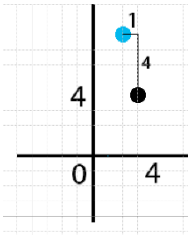
Pada gambar 2.3, sirkuit euler adalah a,b,c,d,e,c,h,b,f,h,e,f,g,a.

*B. A**

A* adalah algoritma pencarian jalur graph atau bentuk lainnya yang bagus. A* merupakan algoritma bagus karena pencarian jalur A* tidak seperti algoritma-algoritma pencarian jalur yang lain, seperti brute-force search, greedy-search, Depth First Search (DFS).

Algoritma A* mencari jalur dengan berdasarkan perhitungan-perhitungan yang spesifik. Perhitungan yang dimaksud memakai teknik heuristik yang cocok untuk melakukan penyelesaian persoalan pencarian jalur. Teknik heuristik yang bisa digunakan untuk algoritma A* banyak, beberapa diantaranya adalah teknik Manhattan Distance, teknik Diagonal Distance, dan teknik Euclidean Distance.

Teknik Manhattan Distance adalah teknik heuristik yang sederhana. Teknik Manhattan Distance mencari jarak dengan menghitung jumlah perbedaan posisi. Dalam dua dimensi, teknik Manhattan Distance menghitung jumlah dari perbedaan sumbu X titik awal dengan sumbu X titik akhir dan perbedaan sumbu Y titik awal dengan sumbu Y titik akhir. Sebagai contoh, titik awal berada pada posisi (3,4) dan titik akhir berada pada posisi (2,8). Menurut perhitungan teknik Manhattan Distance, jarak heuristik antara titik awal dengan titik akhir adalah
$|3-2| + |4-8| = 1 + 4 = 5$.
Karena teknik ini hanya melakukan penjumlahan, waktu yang dipakai untuk menjalankan algoritma Manhattan Distance tidak besar. Teknik Manhattan Distance cocok digunakan sebagai algoritma pencari heuristik yang memerlukan waktu yang singkat.



Gambar 2.4 Teknik Heuristik Manhattan Distance
Titik dengan warna hitam adalah titik awal dan titik dengan warna biru adalah titik akhir.

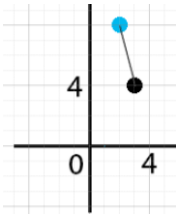
Selanjutnya adalah teknik Diagonal Distance. Teknik Diagonal Distance mencari jarak dengan mengambil nilai terbesar dari perbedaan posisi. Dalam dua dimensi, teknik Diagonal Distance membandingkan nilai dari perbedaan sumbu X titik awal dengan sumbu X titik akhir dan perbedaan sumbu Y titik awal dengan sumbu Y titik akhir. Kemudian diambil nilai yang terbesar. Sebagai contoh, titik awal berada pada posisi (3,4) dan titik akhir berada pada posisi (2,8). Menurut perhitungan teknik Diagonal Distance, jarak heuristik antara titik awal dengan titik akhir adalah |3-2| = 1 dengan |4-8| = 4, perbedaan sumbu Y yang akan diambil karena nilai perbedaan sumbu Y lebih besar dari perbedaan sumbu X. Nilai heuristik adalah 4. Karena teknik ini melakukan pengurangan dan perbandingan, waktu yang dipakai untuk menjalankan algoritma Diagonal Distance lebih besar sedikit dibandingkan waktu teknik Manhattan Distance. Teknik Diagonal Distance cocok digunakan sebagai pencarian jalur yang memungkinkan pergerakan ke delapan arah yang berbeda.

Gambar 2.5 Teknik Heuristik Diagonal Distance
Titik dengan warna hitam adalah titik awal dan titik dengan warna biru adalah titik akhir.

Teknik Euclidean Distance mencari jarak dengan menghitung panjang garis lurus yang menghubungkan titik awal dengan titik akhir. Dalam dua dimensi, teknik Euclidean Distance menghitung akar dari perbedaan sumbu X titik awal dengan sumbu X titik akhir dikuadratkan dan perbedaan sumbu Y titik awal dengan sumbu Y titik akhir dikuadratkan. Sebagai contoh, titik awal berada pada posisi (3,4) dan titik akhir berada pada posisi (2,8). Menurut perhitungan teknik Euclidean Distance, jarak heuristik antara titik awal dengan titik akhir adalah

$$\sqrt[2]{(3-2)^2+(4-8)^2} = \sqrt[2]{1+16} = \sqrt[2]{17}$$

Karena teknik Euclidean Distance melakukan operasi pengurangan, penjumlahan, kuadrat, dan akar kuadrat, waktu yang dipakai untuk menjalankan algoritma Euclidean Distance lumayan besar. Teknik Euclidean Distance cocok digunakan sebagai algoritma pencari heuristik jika tidak ada keterbatasan waktu.



Gambar 2.6 Teknik Heuristik Euclidean Distance
Titik dengan warna hitam adalah titik awal dan titik dengan warna biru adalah titik akhir

Setelah mendapatkan heuristik yang diperlukan, algoritma A* akan mencari rute yang sesuai dengan perhitungan sebagai berikut

f(n) = c(n) + h(n)

Dengan c(n) adalah *cost* yang diperlukan untuk menuju simpul/ titik n, h(n) adalah nilai heuristik dari posisi n menuju ke posisi tujuan akhir, dan f(n) adalah nilai estimasi total *cost* dari posisi n menuju ke posisi tujuan.

Jalur hasil pencarian dengan menggunakan A* selalu berupa jalur yang terpendek (jika teknik heuristik yang digunakan cocok dan tepat untuk menyelesaikan masalah yang diinginkan). Hal tersebut yang menyebabkan algoritma A* bagus. Selain selalu menghasilkan jalur yang terpendek, algoritma A* merupakan algoritma yang efisien, karena pencarian jalur yang dilakukan tidak sembarangan, sehingga waktu yang dibutuhkan untuk menjalankan algoritma A* tidak tergolong lama.

III. Metode Penyelesaian Pencarian Rute dalam Game

A. *Perpindahan Tempat dalam Satu Map*

Sebelum memulai pembahasan mengenai penyelesaian pencarian rute dalam game, akan dilakukan penyeragaman penggunaan istilah-istilah yang akan digunakan dalam pembahasan makalah ini. Pertama, perpindahan tempat akan digunakan untuk perpindahan posisi/ lokasi karakter di sebuah map. Kedua, perpindahan map akan digunakan untuk perpindahan karakter menuju ke map lainnya.

(a) Posisi Awal (b) Proses Berjalan (c) Posisi Akhir
Gambar 3.1 Perpindahan Tempat Karakter dari (a) ke (c)

Pada gambar 3.1, posisi karakter ditandai dengan bentuk kepala panah berwarna jingga. Tujuan akhir posisi karakter (posisi yang dipilih pemain) ditunjukan dengan kepala panah berwarna hijau muda yang menunjuk ke bawah. Awalnya, karakter berada pada posisi awal (a), yang berada pada bagian paling kiri dari map. Kemudian, player memilih suatu lokasi tertentu pada map tersebut (lokasi tertentu tersebut ada pada (b) yang ditandai dengan ujung panah menunjuk ke bawah yang berwarna hijau muda). Pada saat player menekan lokasi tersebut, karakter secara otomatis jalan menuju posisi yang dituju.

(a) Posisi Awal (b) Posisi Tujuan

Gambar 3.2 Perpindahan Map Karakter dari (a) ke (b)

Pada gambar 3.2, posisi karakter ditandai dengan gambar mirip orang berwarna biru. Pilihan map yang diinginkan ditandai dengan daerah empat sudut kotak yang berwarna kuning (di kiri posisi karakter sekarang).

Setelah pengenalan istilah-istilah dilakukan, dibawah ini dimulai pembahasan mengenai perpindahan tempat dalam satu map.

Perpindahan tempat dalam satu map bisa diimplementasikan dalam beberapa cara. Di makalah ini, penulis akan menjelaskan dengan dua cara, pertama dengan menggunakan representasi graph dan kedua dengan menggunakan peta yang diasumsikan berbentuk dua dimensi.

Pembahasan pertama yang menggunakan representasi graph. Semua titik sebelum lokasi yang tidak bisa dilewati dibuat sebagai simpul graph. Sehingga simpul-simpul graph yang pertama terdiri dari batas-batas tempat yang bisa dilewati karakter. Selanjutnya, masukan posisi sekarang karakter dan posisi tujuan karakter ke dalam graph.

Apabila simpul posisi awal karakter dan simpul posisi tujuan karakter tidak dibatasi oleh simpul-simpul yang lainnya, berarti karakter bisa melakukan pergerakan secara lurus langsung menuju posisi tujuan mulai dari posisi awal.

Apabila simpul posisi awal karakter dan simpul posisi tujuan karakter dibatasi oleh simpul lainnya, berarti karakter tidak bisa menuju posisi akhir dengan melakukan pergerakan lurus. Solusinya adalah dengan menggunakan algoritma A* untuk mencari rute paling pendek yang menghubungkan simpul posisi awal dengan simpul tujuan. Teknik heuristik yang bisa dipakai adalah teknik Manhattan Distance dan teknik Euclidean Distance. Baik teknik Manhattan Distance maupun teknik Euclidean Distance akan memberikan rute yang paling optimal. Jika diperlukan waktu yang sesingkat mungkin, maka teknik Manhattan Distance akan menjadi kandidat yang bagus. Jika memperhatikan aspek elegan, maka teknik Euclidean Distance merupakan kandidat yang lebih cocok dibanding teknik Manhattan Distance.

Setelah heuristik yang diperlukan telah dibuat, dengan menggunakan algoritma Manhattan Distance maupun Euclidean Distance, langkah selanjutnya adalah mengimplementasikan algoritma A* untuk mencari rute paling bagus.

Pseudocode

```
h = calculating heuristic from all node to destination node
// By using Manhattan Distance or Euclidean Distance

initialize open list
initialize closed list
put starting node to open list

while open list not empty
    find node 'q' with the least f on open list
    pop q from open list
    generate q's successors and set q's successors parents to q
    for each successor
        if successor.h equal to 0, stop and return
        successor.c = q.c + distance from node successor to q
        successor.h = h(successor)
        successor.f = successor.c + successor.h

        if node with same position as successor in open list \
           that has a lower f than successor, skip successor
        if node with same position as successor in closed list \
           that has a lower f than successor, skip successor
        otherwise, add node to open list
    end
    push q on closed list
end
```

Pembahasan kedua adalah menggunakan peta dalam bentuk dua dimensi. Dalam membuat game, sebenarnya pembahasan kedua ini lebih gampang untuk dibuat dan pengecekan A*-nya juga lebih mudah untuk diimplementasi.

Langkah awal dalam mencari rute menggunakan peta matriks dua dimensi sama seperti pencarian rute menggunakan graph, yaitu menentukan heuristik yang dipakai. Untuk penentuan rute menggunakan matriks ini, lebih cocok digunakan teknik heuristik Manhattan Distance, karena pengambilan data lebih gampang dan pengolahan datanya juga lebih mudah. Jika menggunakan Euclidean Distance, kelemahannya adalah lebih lambat. Lebih tepatnya, penggunaan heuristik Manhattan Distance dan Euclidean Distance lebih cocok digunakan untuk pencarian rute yang menggunakan matriks daripada pencarian rute yang menggunakan graph.

Setelah heuristik yang diperlukan selesai dikalkulasi, gunakanlah algoritma A* untuk melanjutkan pencarian rute yang diinginkan.

Mengapa penulis hanya menggunakan algoritma A*? Karena algoritma-algoritma searching lainnya tidak selalu menghasilkan rute yang paling optimal, misalkan greedy-search yang mengandalkan pengambilan rute yang paling pendek dari rute-rute lain yang bisa dilewati. Selain itu, tidak diambil algoritma-algoritma uninformed-search meskipun selalu menghasilkan rute paling optimal, karena waktu yang dibutuhkan algoritma-algoritma uninformed-search sangatlah lama.

Pseudocode

```
h = calculating heuristic from all point to destination point
// By using Manhattan Distance or Euclidean Distance

initialize open list
initialize closed list
put starting point to open list

while open list not empty
    find point 'q' with the least f on open list
    pop q from open list
    generate q's successors and set q's successors parents to q
    for each successor
        if successor.h equal to 0, stop and return
        successor.c = q.c + one unit
        successor.h = h(successor)
        successor.f = successor.c + successor.h
```

```
        if point with same position as successor in open list \
           that has a lower f than successor, skip successor
        if point with same position as successor in closed list \
           that has a lower f than successor, skip successor
        otherwise, add point to open list
    end
    push q on closed list
end
```

Perbedaan pseudocode A* representasi graph dan representasi matriks, adalah tipe data yang digunakan. Selain tipe data berbeda, perhitungan penambahan c(n) di representasi matriks juga lebih mudah karena berpindah ke posisi manapun, berpindahnya itu sesuai satu satuan tertentu, tidak dimungkinkan untuk berpindah secara tidak beraturan.

Penyelesaian pencarian rute paling optimum bisa diselesaikan semuanya dengan menggunakan algoritma A*, tetapi sebuah game bisa menghemat waktu yang diperlukan untuk pencarian rute optimum. Sebagai contoh, ketika karakter berpindah ke map lain, karakter akan selalu dimunculin di posisi tertentu (misalkan dekat pintu). Ketika tujuan karakter adalah pintu yang lain, rute bisa di hard-code agar waktu tetap efisien. Alasan hard-code bisa diterima, karena posisi-posisi tertentu selalu sama. Sebagai contoh, posisi pintu selalu sama, tidak pernah berubah, sehingga hard-code bisa diterima untuk mengefektifkan waktu.

B. Perpindahan Antar Map

Perpindahan dari satu map ke map lainnya memiliki banyak skenario. Skenario-skenario yang dimaksud adalah perpindahan karakter dari pintu masuk suatu map ke map lain, perpindahan karakter dari posisi random suatu map ke map lain, perpindahan karakter dari pintu masuk suatu map ke suatu posisi tertentu di map lain, dan perpindahan karakter dari posisi random suatu map ke suatu posisi tertentu di map lain.

Perpindahan karakter dari pintu masuk suatu map ke map lain bisa menggunakan algoritma A* yang sebelumnya dengan tambahan modifikasi-modifikasi tertentu. Modifikasi yang dimaksud adalah pencarian map-map yang perlu dilewati untuk sampai map tujuan. Pencarian map bisa menggunakan algoritma A*, dengan heuristik mirip dengan matriks dua dimensi. Jika data yang diinginkan berupa graph, map-map yang ada bisa ditransformasikan ke dalam bentuk graph. Transformasi dari data awal menjadi graph lebih mudah diimplementasikan untuk map-map ini dibandingkan dengan transformasi tempat dari satu map yang dibahas sebelumnya. Bentuk graph lebih mudah dibayangkan untuk map besar dari game Ragnarok Mobile. Setelah rute map diketahui, selanjutnya adalah penggerakan karakter.

Karakter diarahin dengan algoritma A* yang sebelumnya dengan tujuan posisi pertama adalah pintu mendekati map yang diinginkan. Karena skenario yang sedang dibahas merupakan skenario pertama, pemindahan pertama bisa menggunakan cara hard-code agar waktu bisa tergunakan dengan efisien. Apabila map yang dimasuki sudah sesuai dengan yang diinginkan pemain, maka karakter akan berhenti pas setelah memasuki map tersebut atau sesuai dengan keinginan developer. Apabila map yang dimasuki belum sesuai dengan yang diinginkan pemain, maka karakter akan diarahi menuju pintu map yang sudah ditentukan diawal. Proses ini akan terus diulangi jika map karakter belum sesuai dengan yang diinginkan pemain. Proses ini akan berhenti ketika karakter sudah sampai di map diinginkan.

Skenario kedua adalah perpindahan karakter dari posisi random di suatu map ke map lain. Langkah awal tetap sama seperti sebelumnya, yaitu mencari heuristik yang cocok. Setelah heuristik berhasil didapatkan, dilakukan pencarian dengan menggunakan algoritma A*. Pada skenario ini, hard-code tidak cocok untuk dipakai sebagai pencarian rute yang pertama karena posisi karakter random. Jika dibuatkan dalam bentuk hard-code, kode yang dihasilkan sangat tidak bagus karena kemungkinan-kemungkinan yang ada sangat banyak. Apabila karakter belum sampai di map yang diinginkan, pemakaian rute selanjutnya bisa menggunakan yang sudah di hard-code karena posisi karakter berada di posisi awal yang sudah didefinisikan. Apabila karakter sudah sampai di tujuan yang diinginkan, karakter akan berhenti di tempat awal yang sudah didefinisikan.

Skenario ketiga adalah perpindahan karakter dari pintu masuk suatu map ke suatu posisi tertentu di map lain. Langkah awalnya adalah menentukan rute map besar yang akan dilewati. Titik awalnya berupa lokasi map yang sedang ditempati, titik akhirnya berupa tujuan map yang player inginkan. Teknik heuristik yang bisa digunakan berupa Manhattan Distance, Diagonal Distance, dan Euclidean Distance. Diagonal Distance bisa digunakan karena pencarian rute untuk map besar bisa berpindah secara unik sebanyak satu satuan.

Setelah heuristik yang diperlukan berhasil didapatkan, akan diterapkan pencarian rute untuk map besar terlebih dahulu. Kemudian, berdasarkan rute yang didapatkan, karakter akan digerakkan untuk melewati rute tersebut. Skenario ini memiliki karakter yang bermula dari pintu masuk suatu map, sehingga bisa diambil hasil rute perpindahan tempat yang sudah di hard-code. Karakter akan terus bergerak sampai karakter tersebut berada di tujuan yang tepat.

Skenario keempat adalah perpindahan karakter dari posisi random suatu map ke posisi tertentu di map lain. Langkah awalnya adalah menentukan rute map besar yang akan dilewati. Titik awalnya berupa lokasi map yang sedang ditempati, titik akhirnya berupa tujuan map yang player inginkan. Teknik heuristik yang bisa digunakan berupa Manhattan Distance, Diagonal Distance, dan Euclidean Distance.

Setelah heuristik yang diperlukan berhasil didapatkan, akan diterapkan pencarian rute untuk map besar terlebih dahulu. Kemudian, berdasarkan rute yang didapatkan, karakter akan digerakkan untuk melewati rute tersebut. Skenario ini memiliki karakter yang bermula dari pintu masuk suatu map, sehingga bisa diambil hasil rute perpindahan tempat yang sudah di hard-code. Karakter akan terus bergerak sampai karakter tersebut berada di tujuan yang tepat.

Pseudocode

```
hbigmap = calculating heuristic from all point of map to \
                destination point of map
          // By using Manhattan Distance or Diagonal Distance
          // or Euclidean Distance

initialize open list
initialize closed list
put starting point of map to open list

while open list not empty
    find point ‘q’ with the least f on open list
```

```
    pop q from open list
    generate q's successors and set q's successors parents to q
    for each successor
        if successor.h equal to 0, stop and return
        successor.c = q.c + one unit
        successor.h = h(successor)
        successor.f = successor.c + successor.h

        if point with same position as successor in open list \
           that has a lower f than successor, skip successor
        if point with same position as successor in closed list \
           that has a lower f than successor, skip successor
        otherwise, add point to open list
    end
    push q on closed list
end
// Path for big map found
for each BigMapPath
    // A* Chapter 3.A path finding
```

IV. ACKNOWLEDGMENT

Menyusun sebuah makalah bukanlah hal yang mudah. Saya berterima kasih kepada Dr. Nur Ulfa Maulidevi ST,M.Sc. yang telah memberikan saya pengetahuan dan kesempatan untuk menyusun makalah mengenai Strategi Algoritma. Tanpa pengetahuan yang diberikan kepada saya, saya tidak akan berhasil menyelesaikan makalah yang berjudul 'Pencarian Rute untuk Perpindahan Posisi Otomatis Karakter di Game'. Saya juga berterima kasih kepada pemilik-pemilik sumber teori dasar yang saya pakai untuk kelengkapan makalah ini.

REFERENCES

[1] https://budinugroho24.wordpress.com/about/pengertian-internet-atau-definisi-internet-2/, tanggal akses terakhir: 8 Desember 2018.
[2] https://www.pricebook.co.id/article/review/2016/01/26/3593/ada-4-jenis-dan-11-genre-game-yang-mana-favorit-kamu#close-modal, tanggal akses terakhir: 8 Desember 2018.
[3] http://danysatriokintoko.blogspot.com/, tanggal akses terakhir: 8 Desember 2018.
[4] Munir, Rinaldi, Matematika Diskrit, Ed.3, Bandung: Informatika Bandung, 2007
[5] http://kuliahmsi.blogspot.com/2010/07/lintasan-dan-sirkuit-euler.html, tanggal akses terakhir: 8 Desember 2018
[6] Makalah Matematika Diskrit Vincent Chuardi (NIM 13517103).
[7] https://www.geeksforgeeks.org/a-search-algorithm/, tanggal akses terakhir: 24 April 2019
[8] https://i.stack.imgur.com/8nAs8.png, tanggal akses terakhir: 24 April 2019

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, 26 April 2019

Vincent Chuardi
13517103