Menggerakkan Karakter *Game* Menggunakan Algoritma *Breadth-First Search* (BFS) dan Algortma Algoritma A*(A Star)

Aristama Ramadhani - 13506002

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
e-mail: if16002@students.if.itb.ac.id

ABSTRAK

Makalah ini dibuat untuk membahas algoritma *Bread-First Search* (BFS) dan algoritma A* dalam metode menggerakkan karakter di suatu *game*. Kedua konsep ini sangat erat hubungannya dengan graf yang akan sangat penting dalam algoritma pencarian jalan yang akan digunakan dalam cara menggerakkan karakter nantinya.

Algoritma A* adalah keluarga dekat dari algoritma yang lebih sederhana, yaitu *Bread-First Search* (BFS). Keduanya adalah algoritma yang digunakan dalam pencarian graf, dan graf ini dapat dimisalkan sebagai suatu lokasi dalam *game*. Graf adalah kumpulan dari simpul-simpul (*nodes*) yang dihubungkan dengan sisi-sisi (*edges*). Simpul-simpul dan sisi-sisi ini dapat kita artikan sebagaimana keinginan kita. Sebagai contoh, suatu simpul A dan B pada graf adalah kota dan sisi adalah jalan yang menghubungkan kedua kota tersebut.

Pergerakan karakter di suatu *game* sangat erat kaitannya dengan *path finding* (pencarian jalan). Masalah ini dapat diturunkan dengan menajawab pertanyaan "Bagaimana saya bisa pergi ke titik B dari titik A?". Pada umunya, pencarian jalan dari satu lokasi ke lokasi lain dapat mempunyai beberapa solusi, tetapi idealnya kita ingin mendapat solusi yang memenuhi tujuan-tujuan beruikut.

1. **Cara bergerak dari titik A ke B.**
2. **Cara menghindari penghalang yang ada di jalan.**
3. **Cara mencari jalan yang terpendek.**
4. **Cara mencari jalan yang tercepat.**

Kata kunci: Bread-First Search *(BFS)*, A*, graf, simpul, sisi, pencarian jalan.

1. PENDAHULUAN

Path Finding atau pencarian jalan adalah salah satu dasar algoritma dalam konsep menggerakkan karakter dalam *game*. Tanpa algoritma pecarian jalan ini, karakter dalam *game* yang dibuat tidak akan bisa bergerak dengan benar atau seseuai keinginan kita. Walaupun bisa jalan, belum tentu karakter tersebut bergerak sesuai keinginan kita.

Pergerakan karakter dalam *game* adalah suatu hal yang paling dasar tetapi juga pakung penting dalam pembuatan game. Bila kita membuat karakter game RPG (*role playing game*) tetapi karakter kita tidak bisa bergerak, malah game yang kita buat tersebut akan menjadi tidak seru dan tidak menarik. Padahal, alur cerita, gambar, musik, dan fitur-fitur lain yang kita dari game tersebut sudah sangat bagus. Tidak akan ada artinya kalau game kita tidak seru hanya karena karakternya tidak bisa bergerak.

Walaupun pergerakan karakter dalam suatu game adalah suatu hal yang penting, hukum ini tidak berlaku pada beberapa jenis game dengan *genre* yang berbeda. Contoh game yang tidak memerlukan pergerakan karakter dengan kedua algoritma dibahas ini misalnya game bertipe visual novel. Tipe game ini hanya menggambarkan karakter-karakter yang berdialog dalam suatu layar, yang merupakan latar belakang dari game tersebut. Karakter-karakter di dalam game tipe ini biasanya tidak bergerak. Walaupun bergerak, biasanya hanya pergerakan ke kiri dan ke kanan layar untuk menunjukkan apa yang sedang dikerjakannya, atau pergerakan-pergerakan kecil seperti mengubah ekspresi wajah, gestur, dan sebagainya. Dan juga, pergerakan-pergerakan kecil ini tidak perlu algoritma pencarian jalan karena konteksnya memang lain.

2 Algoritma *Breadth-First Search* (BFS)

Banyak persoalan yang direpresentasikan dengan graf. Pencarian solusi dilakukan dengan mengunjungi (*traverse*) simpul-simpul graf. Algoritma traversal adalah mengunjungi simpul-simpul dengan cara yang sistematik. Algoritma *Breadth-First Search* diterapkan pada graf statis, yaitu graf yang sudah tersedia (dibentuk terlebih dahulu) sebelum pencarian solusi.

Algortima pencarian menggunakan BFS dapat ditulis sebagai berikut.

1. Traversal akan dilakukan dalan suatu graf, misalnya dimulai dari simpul *v*.
2. Kunjungi simpul *v*, bila simpul yang dicari ditemukan, maka pencarian selesai dan kembalikan hasil.
3. Bila tidak ditemukan, kunjungi semua simpul yang bertetangga dengan *v*, bila tidak ditemukan, cari lagi di simpul yang belum dikunjungi yang bertetangga dari simpul yang dikunjungi tadi. Begitu seterusnya sampai pencarian selesai (pencarian berhasil atau tidak ditemukan).

2.1 Kompleksitas Algoritma *Breadth-First Search* (BFS)

Meskipun metode BFS kelihatannya bagus untuk mencari solusi, namun kompleksitas algoritmanya eksponensial. Untuk menjelaskan hal ini, misalkan setiap simpul dapat membangkitkan *b* simpul buah baru. Simpul akar akan membangikitkan simpul pada aras ke-1, masing-masing simpul membangkitkan *b* buah simpul, total seluruhnya b^2 pada aras ke-2. Toap-toap simpul ini akan membangkitkan *b* buah simpul baru pada aras ke-3, total ada b^3 buah simpul. Misalkan solusi ditemukan pada aras ke-*d*, maka jumlah maksimum seluruh simpul yang dibangkitkan adalah

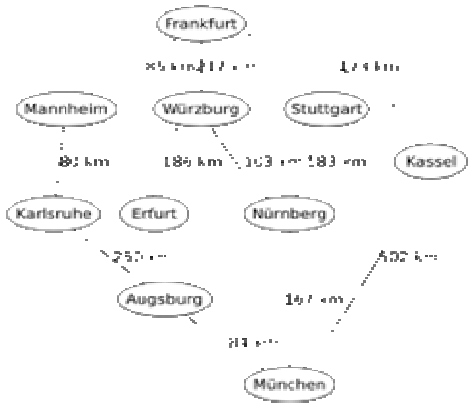
$$1 + b + b^2 + b^3 + ... + b^d = (b^{d+1} - 1)/(b-1)$$

Jadi kompleksitas waktu algoritmanya adalah $O(b^d)$. Kompleksitas ruang algoritma BFS sama dengan kompleksitas waktunya, karena semua sipul daun dari pohon harus disimpan di dalam memori selama proses pencarian.

2.2 Penerapan BFS dalam Pencarian Jalan untuk Menggerakkan Karakter Game

Misalnya, kita ingin mencari jalan terpendek dari Frankfurt ke Munchen (lihat gambar 1). Dari gambar 1, solusinya gampang ditentukan, tapi bagaimana membuat kodenya? Salah satu caranyaadalah dengan menggunakan Algoritma *Breadth-First Search.*

Algoritma BFS adalah mengunjungi sebuah simpul (kota dalam contoh ini) dalam satu waktu. BFS mengunjungi simpul berdasarkan jarak dari simpul awal, dimana jaraknya dihitung dari jumlah sisi-sis yang dilewati. Jadi, dengan BFS, semua simpul dengan jarak satu sisi dari simpul awal dikunjungi, lalu du simpul, dan seterusnya dampai semua simpul dikunjungi. Dengan cara ini, kita akan menemukan jalan dari *start* ke *goal* dengan jumlah sisi minimum yang dilewati. Dengan kata lain, bisa disebut seperti ini: kunjungi simpul-simpul tetangga, lalu tetangga dari simpul tetangga, dan seterusnya sampai simpul tujuan ditemukan.



Gambar 1. Peta Jerman

Secara sederhana, pseudo-kode algoritmanya bisa ditulis sebagai berikut.

```
procedure BFS(input v:integer)
{Traversal graf dengan algoritma pencarian BFS.
Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpul yang dikunjungi dicetak ke layar}

Deklarasi
   w : integer
   q : antrian;

   procedure BuatAntrian
  (input/output q : antrian)
   { membuat antrian kosong, kepala(q)
  diisi 0 }

   procedure MasukAntrian
  (input/output     q:antrian,     input
   v:integer)
   { memasukkan v ke dalam antrian q
   pada posisi belakang }

   procedure HapusAntrian
  (input/output         q:antrian,output
   v:integer)
   { menghapus v dari kepala antrian q
   }

   function AntrianKosong
  (input q:antrian) → boolean
```

```
   { true jika antrian q kosong, false
  jika sebaliknya }

Algoritma:
   BuatAntrian(q){ buat antrian kosong
}

   write(v) { cetak simpul awal yang
dikunjungi }

   dikunjungi[v]←true  { simpul v
telah dikunjungi, tandai dengan true}

   MasukAntrian(q,v)   { masukkan
simpul awal kunjungan ke dalam
antrian}

  { kunjungi semua simpul graf selama
antrian belum kosong }
   while not AntrianKosong(q) do
       HapusAntrian(q,v)        {
simpul v telah dikunjungi, hapus dari
antrian }
       for w←1 to n do
          if A[v,w] = 1 then         {
v dan w bertetangga }
             if not dikunjungi[w] then
                write(w)
{cetak simpul yang dikunjungi}
                MasukAntrian(q,w)
                dikunjungi[w]←true
             endif
          endif
      endfor
   endwhile
  { AntrianKosong(q) }
```

Itulah yang bisa dijelaskan dari algoritma BFS. Tetapi bila dilihat sekali lagi, mudah ditemukan bahwa ada masalah dalam pencarian ini: kita menemukan jalan dengan jumlah sisi paling sedikit, tetapi setiap sisi bisa mempunyai "*cost*" yang berbeda-beda. Sebagai contoh di atas, cost untuk melewati satu sisi bisa 80 km dan cost untuk mengunjungi sisi lain bisa 502 km. Sudah jelas, melewati jalan 80 km akan lebih cepat daripada melewati jalan 502 km. Dan dalam persolan ini, pencarian jalan terdekat dengan menggunakan algoritma BFS akan menghasilkan jawaban yang salah. Bila algoritma BFS digunakan, simpul simpul yang akan dilewati untuk mengunjungi Munchen dari Frankfurt adalah sebagai berikut:

Frankfurt(start awal) → Kassel → Munchen

Ini akan menghasilkan cost (jarak) = 173 + 502 = 675 km, yang merupakan cost (jarak) terbesar dalam kenyataan, walaupun dalam gambar melewati sisi paling sedikit.

Dalam dunia nyata, ada rute lain yang lebih pendek misalnya:

1. Frankfurt → Mannheim → Karlsruhe → Augsburg → Munchen
   Dengan rute ini, cost (jarak) = 85 + 80 + 250 + 84 = 499 km.
2. Frankfurt → Wurzburg → Nurnberg → Munchen
   Dengan rute ini, cost (jarak) = 217 + 103 + 167 = 487 km.

Bila kita ingin melewati jalur terpendek, sudah jelas kita akan memilih rute Frankfurt → Wurzburg → Nurnberg → Munchen dengan jarak terpendek, yaitu 487 km. Hanya saja, algoritma BFS tidak mampu mendapatkan solusi ini karena algoritma ini mengasumsikan semua sisi mempunyai cost yang sama sehingga ini tidak cukup bagus dalam pencarian jalan terpendek. Nah, disinilah algoritma A* akan berguna untuk pencarian jalan ini, dan itu hanya untuk menggerakkan karakter game kita saja melalui jalan terpendek. Walah...

3 Algoritma Pencarian A* (A Star)

Pencarian menggunakan algoritma A* mempunyai prinsip yang sama dengan algoritma BFS, hanya saja dengan 2 faktor tambahan.

1. Setiap sisi mempunyai "cost" yang berbeda-beda, seberapa besar cost untuk pergi dari satu simpul ke simpul yang lain.
2. Cost dari setiap simpul ke simpul tujuan bisa diperkirakan. Ini membantu pencarian, sehingga lebih kecil kemungkinan kita mencari ke arah yang salah.

Cost untuk setiap sompul tidak harus berupa jarak. Cost bisa saja berupa waktu bila kita ingin mencari jalan dengan waktu tercepat untuk dilalui. Sebagai contoh, bila kita berkendaraan melewati jalan biasa bisa saja merupakan jarak terdekat, tetapi melewati jalan tol biasanya memakan waktu lebih sedikit.

Algoritma A* bekerja dengan prinsip yang hampir sama dengan BFS, kecuali dengan 2 perbedaan.

1. Simpul-simpul di list "terbuka" diurutkan oleh cost keseluruhan dari simpul awal ke simpul tujuan, dari cost terkecil sampai cost terbesar. Dengan kata lain, menggunakan *priority queue* (antrian prioritas). Cost keseluruhan dihitung dari cost dari simpul awal ke simpul sekarang (*current node*) ditambah cost perkiraan menuju simpul tujuan.
2. Simpul di list "tertutup" bisa dimasukkan ke list "terbuka" bi;a jalan terpendek (cost lebih kecil) menuju simpul tersebut ditemukan.

Karena list "terbuka" diurutkan berdasarkan perkiraan cost keseluruhan, algoritma mengecek simpul-simpul yang mempunyai perkiraan cost yang paling kecil terlebih

dahulu, jadi algoritmanya mencari simpul-simpul yang kemungkinan mengarah ke simpul tujuan. Karena itu, lebih baik perkiraan cost-nya, lebih cepat pencariannya.

Cost dan perkiraannya ditentukan oleh kita sendiri. Bila cost-nya adalah jarak, akan menjadi mudah. Cost antara simpul adalah jaraknya, dan perkiraan cost dari suatu simpul ke simpul tujuan adalah penjumlahan jarak dari simpul tersebut ke simpul tujuan. Atau agar lebih mudahnya bisa ditunjukkan seperti berikut ini.

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

Perhatikan bahwa algoritma ini hanya bekerja bila cost perkiraan tidak lebih besar dari cost yang sebenarnya. Bila, cost perkiraan lebih besar, bisa jadi jalan yang ditemukan bukanlah yang terpendek.

3.1 Kompleksitas Algoritma A* (A Star)

Kompleksitas waktu dari algoritma A* tergantung dari heuristiknya. Dalam kasus terburuk (*worst case*), jumlah simpul yang diekspansi bisa eskponensial dalam solusinya (jalan tependek). Akan tetapi, kompleksitasnya bisa berupa polinomial bila fungsi heuristik *h* bertemu kondisi berikut:

$$|h(x) - h^*(x)| \leq O(\log h^*(x))$$

dimana *h** adalah heuristik optimal, atau cost pasti untuk menuju tujuan dari *x*. Dengan kata lain, kesalahan (*error*) dari *h* tidak boleh tumbuh lebih cepat dari logaritma "*perfect heuristic*" *h** yang mengembalikan jarak sebenarnya dari *x* menuju tujuan (Russel dan Norvig 2003, hal. 101).

3.2 Penerapan A* dalam Pencarian Jalan untuk Menggerakkan Karakter Game

Menggunakan algoritma pencarian A* untuk menggerakkan karakter dalam game mengharuskan menginterpretasikan lingkungan game sebagai graf. Dengan kata lain, menentukan apa yang direpresentasikan oleh simpul dan sisinya.

Sebagai contoh, dalam game 2D (dua dimensi), berbentuk ubin-ubin atau kotak-kotak seperti papan catur, ini mudah. Simpul-simpul merepresentasikan ubin-ubin yang ada. Kita bisa menggerakkan karakter dari ubin satu ke ubin-ubin lain yang merupakan tetangga dari ubin tadi (kecuali dalam ubin tetanga terdapat penghalang).

Algoritma A* juga bisa digunakan untu mengimplementasikan pergerakan karakter dalam game 3D, tetapi ini tidak akan dibahas di makalah ini karena terlalu rumi. Sejauh ini, dalam ilmu komputer (*computer science*), algoritma A* merupakan algoritma terbaik di antara algoritma-algoritma pencarian graf untuk mencari jalan dengan cost terkecil dari simpul awal menuju simpul akhir.

4. KESIMPULAN

Algoritma *Breadth-First Search* (BFS) dan A* (A Star) sama-sama merupakan algoritma pencarian graf. Algoritma A* merupakan algoritma yang lebih baik baik daripada algoritma *Breadth-First Search* (BFS) karena bisa menemukan jalan terpendek (*shortest path*) dalam masalah ini. Selain itu, algoritma A* juga merupakan algoritma terbaik dalam ilmu komputer (*computer science*) dalam pencarian graf untuk mencari jalan dengan cost terkecil, paling tidak sampai saat ini.

REFERENSI

1. Munir, Rinaldi. 2006. Diktat Kuliah IF 2251 Strategi Algoritmik. Bandung: Laboratorium Ilmu dan Rekayasa Komputasi. Hal. 108, 120. Program Studi Teknik Informatika ITB.
2. http://en.wikipedia.org/wiki/A%2A_search_algorithm Diakses tanggal 18 Mei 2008 ~ 07.05
3. http://en.wikipedia.org/wiki/Breadth-first_search Diakses tanggal 18 Mei 2008 ~ 07.13
4. http://www.peachpit.com/articles/aspx?p=101142 Diakses tanggal 18 Mei 2008 ~ 07.02