PERBANDINGAN ALGORITMA BFS DAN DFS DALAM PEMBUATAN RUTE PERJALANAN OBJEK PERMAINAN 2 DIMENSI

David Steven Wijaya
NIM : 13505044

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

ABSTRAK

***Breadth First Search* (BFS) dan *Depth First Search* (DFS) adalah dua jenis algoritma traversal di dalam graf yang sering digunakan untuk mencari simpul yang diinginkan dari graf secara cepat dan mangkus. Secara umum, kedua algoritma ini dapat menyelesaikan hamper semua permasalahan yang menyangkut pencarian simpul pada graf. Akan tetapi, kedua algoritma ini memiliki kelebihan dan kekurangan masing-masing, tergantung dari jenis permasalahan yang ingin dipecahkan. Makalah ini akan membahas penggunaan algoritma BFS dan DFS dalam masalah pembuatan rute perjalanan untuk objek permainan dua dimensi, khususnya untuk permainan yang membutuhkan komputasi yang cepat seperti strategi waktu-nyata (*real-time-strategy / RTS)* di mana objek seperti orang, tank, atau benda apapun yang diperintah oleh pemain untuk bergerak harus dapat langsung menentukan rute perjalanannya ke tempat yang dituju sehingga permainan dapat dilakukan secara waktu-nyata. Selain itu, algoritma ini juga berperan penting dalam mengatur gerakan objek-objek yang dikendalikan oleh komputer sehingga gerakan objeknya tidak kaku dan menyerupai pola pikir manusia (intelegensia buatan).**

Kata kunci: BFS, DFS, rute perjalanan, AI

1. PENDAHULUAN

Dewasa ini banyak dikembangkan permainan-permainan yang bersifat waktu-nyata, salah satunya adalah permainan strategi waktu-nyata. Permainan ini dijalankan di atas sebuah peta yang dapat berupa grid dua dimensi atau grid tiga dimensi. Dalam permainan ini, pemain dapat memegang kontrol terhadap beberapa unit (objek) dan memerintahnya untuk melakukan pergerakan ke suatu posisi tertentu dalam peta maupun melakukan penyerangan ke sebuah objek musuh yang tidak selalu statis (objek musuh dapat berupa objek yang sedang bergerak). Objek yang dikendalikan pemain harus dapat segera bergerak dan menentukan rute pergerakannya segera setelah pemain memberikan instruksi.

Gambar 1. Pengendalian objek (tentara) ke sebuah lokasi tertentu dalam permainan waktu-nyata

Spesifikasi dari permainan ini menuntut kemampuan komputasi yang besar karena objek yang dikendalikan oleh pemain biasanya sangat banyak. Pada gambar 1 diperlihatkan pemain memberikan instruksi kepada banyak objek (tentara) sekaligus. Selain itu, permainan dilakukan dengan cepat dan pemberian instruksi dapat dilakukan berulang-ulang pada objek yang sama. Permasalahan lainnya adalah lokasi yang dituju oleh objek dapat berupa objek bergerak sehingga rute perjalanan harus dikalkulasi ulang setiap kali objek tujuan bergerak.

Dalam makalah ini, penulis mencoba mengimplementasikan algoritma BFS dan DFS dalam mencari solusi permasalahan ini, yaitu dengan mencari rute perjalanan dari sebuah lokasi ke lokasi tertentu yang diinginkan di dalam peta dua dimensi.

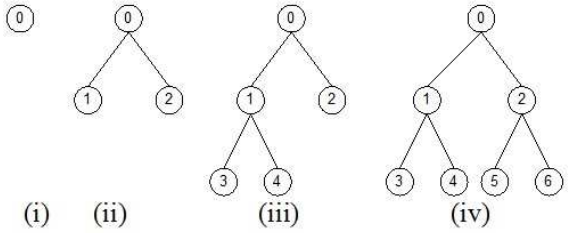
2. DASAR TEORI

Algoritma yang digunakan dalam makalah ini adalah DFS dan BFS yang merupakan algoritma yang lazim digunakan dalam graf tidak berarah. Algoritma ini bekerja secara traversal, yaitu mengunjungi simpul-simpul dalam graf dengan cara yang sistematik. Secara sederhana, algoritma ini hanya menentukan urutan pemeriksaan simpul-simpul hingga menemukan simpul solusi.

2.1. BFS

Breadth First Search (BFS) adalah algoritma pencarian simpul dalam graf (pohon) secara traversal yang dimulai dari simpul akar dan mengecek semua simpul-simpul tetangganya. Setelah itu, dari tiap simpul tetangganya, algoritma akan terus mengecek semua simpul tetangga yang belum dicek, sedemikian seterusnya hingga menemukan simpul tujuan. [wikipedia]

Urutan pengecekan simpul graf dengan menggunakan BFS ditunjukkan dengan tahapan pembentukan pohon pencarian sebagai berikut:

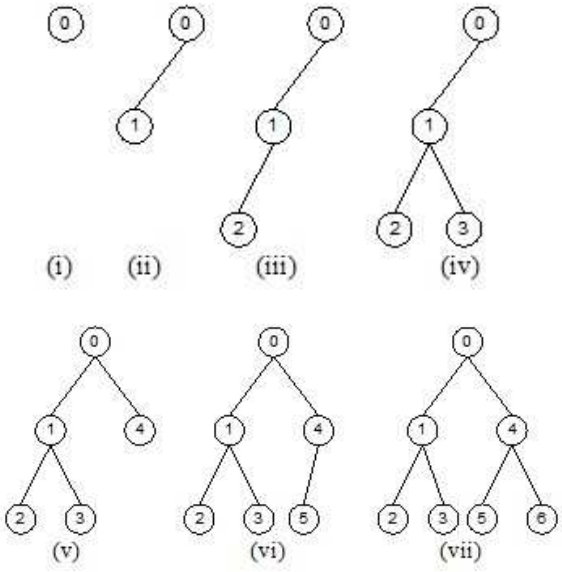


Gambar 2. Tahapan pembentukan pohon pencarian dengan BFS

2.2. DFS

Depth First Search (DFS) adalah algoritma pencarian simpul dalam graf secara traversal yang dimulai dari simpul akar dan mengecek simpul anaknya yang pertama, setelah itu, algoritma mengecek simpul anak dari simpul anak yang pertama tersebut, hingga mencapai simpul daun atau simpul tujuan. Jika solusi belum ditemukan, algoritma melakukan runut-balik (*backtracking*) ke simpul orangtuanya yang paling baru deperiksa lalu dan mengecek simpul anaknya yang belum diperiksa, sedemikian seterusnya hingga simpul solusi ditemukan.

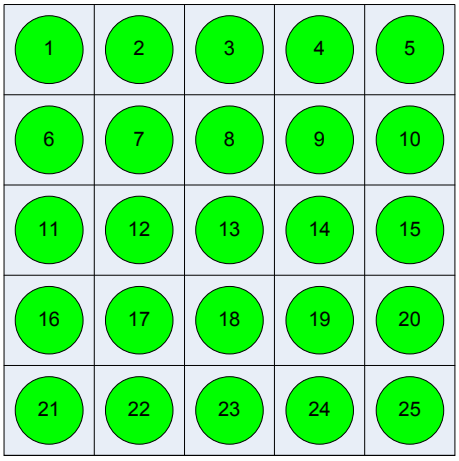
Urutan pengecekan simpul graf dengan menggunakan DFS ditunjukkan dengan tahapan pembentukan pohon pencarian sebagai berikut:



Gambar 3. Tahapan pembentukan pohon pencarian dengan DFS

3. RUANG LINGKUP PERSOALAN

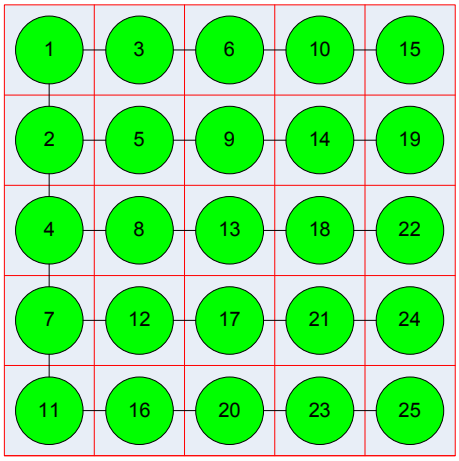
Algoritma BFS dan DFS memerlukan sebuah graf. Dalam persoalan pencarian rute perjalanan objek permainan dua dimensi ini, objek bergerak di sebuah peta dua dimensi berhingga yang direpresentasikan dengan matriks n x m yang setiap elemennya berupa simpul graf. Simpul ini mewakili posisi objek di dalam peta. Secara sederhana, peta yang digunakan dapat digambarkan sebagai berikut:



Gambar 4. Peta 5x5 yang direpresentasikan dengan kumpulan simpul-simpul (bulatan hijau)

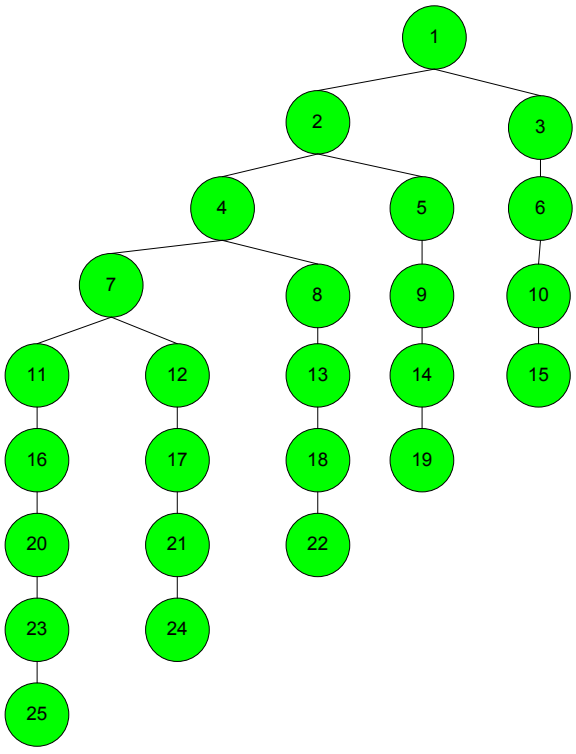
Pembentukan pohon dari simpul-simpul tersebut dilakukan jika diketahui simpul awal (akar),

yaitu posisi awal objek yang akan dicari rute perjalanannya. Apabila simpul 1 adalah simpul akar dan diasumsikan bahwa objek hanya dapat bergerak ke empat arah (utara, selatan, barat, dan timur) , maka simpul tetangga dari simpul 1 adalah simpul 6 dan simpul 2. Jika dilakukan pembentukan pohon dengan algoritma BFS, maka didapat pohon berikut:



Gambar 4 a. Pohon yang dibentuk dari peta

Jika posisi peta dihilangkan, maka pohon dapat dilihat sebagai berikut:



Gambar 4 b. Pohon yang dibentuk dari peta

Dari pohon ini, dapat dilakukan pencarian rute dengan algoritma BFS dan DFS.

4. IMPLEMENTASI ALGORITMA

Algoritma BFS dan DFS diimplementasikan penulis dengan menggunakan C++. Berikut adalah implementasi BFS dan DFS yang dituangkan ke dalam kode:

4.1. BFS

Pada dasarnya, implementasi algoritma BFS dilakukan dengan struktur data antrian (*queue*). Antrian adalah struktur data berupa larik dengan urutan pengaksesan *First in First out.* Berikut adalah pseudo-code dari algoritma BFS:

1. Inisialisasi antrian kosong
2. Tambahkan simpul akar ke dalam antrian
3. Jika antrian kosong, hentikan pencarian (tidak ada solusi)
4. Ambil simpul s dari antrian (*head*)
   a. Jika s adalah simpul tujuan, hentikan pencarian (solusi ditemukan)
   b. Lakukan ekspansi dari simpul s, tambahkan simpul-simpul yang dihasilkan ke dalam antrian (*tail*)
5. Kembali ke langkah 3

Ekspansi dari simpul s akan menghasilkan semua simpul yang bertetanggaan dengan s (simpul anak dari s) yang belum diakses sebelumnya. Urutan ekspansi adalah dari simpul anak terkiri.

Pada akhir pencarian, antrian akan berisi urutan simpul-simpul yang diakses, yaitu rute perjalanan objek dari simpul awal ke simpul tujuan.

Berikut adalah potongan kode yang mengimplementasikan algoritma BFS dalam C++:

```
//fungsi BFS
//fungsi mengembalikan larik dari simpul-simpul
solusi
```

```
Simpul* BreadthFirst::search() {
  Simpul* node; // larik dari simpul-simpul
                   solusi
  // selama antrian tidak kosong (kondisi tidak
     ada solusi)
  while (!_queue.empty()) {
  // ambil elemen head antrian
  node = _queue.front();
  _queue.pop();
  // jika solusi ditemukan
  if (problem.isGoal(node)){
    return node;
  }
  // ekspansi node, masukkan elemen yang
```

```
      dihasilkan ke antrian
    problem.successors(node, _queue);
  }
  return 0; // jika tidak ada solusi
}
```

4.2. DFS

Implementasi algoritma DFS tidak jauh berbeda, kecuali dalam struktur data dan cara ekspansi. Struktur data yang digunakan adalah stack, di mana berlaku urutan pengaksesan *Last in First out*. Berikut adalah pseudo-code dari algoritma DFS:

1. Inisialisasi stack kosong
2. Tambahkan simpul akar ke dalam stack
3. Jika stack kosong, hentikan pencarian (tidak ada solusi)
4. Ambil simpul s dari stack (*pop*)
   a. Jika s adalah simpul tujuan, hentikan pencarian (solusi ditemukan)
   b. Ekspansi simpul s dan masukkan simpul-simpul yang dihasilkan ke dalam stack (*push*)
5. Kembali ke langkah 3

Ekspansi dari simpul s akan menghasilkan sebuah simpul anak yang belum diakses (s1) dan sebuah simpul anak yang belum diakses dari s1 tersebut (s2) dan seterusnya hingga mencapai simpul daun (simpul yang tidak mempunyai anak / semua anaknya telah diakses). Urutan ekspansi adalah dari simpul anak terkiri.

Pada akhir pencarian, stack akan berisi urutan simpul-simpul yang diakses, yaitu rute perjalanan objek dari simpul awal ke simpul tujuan.

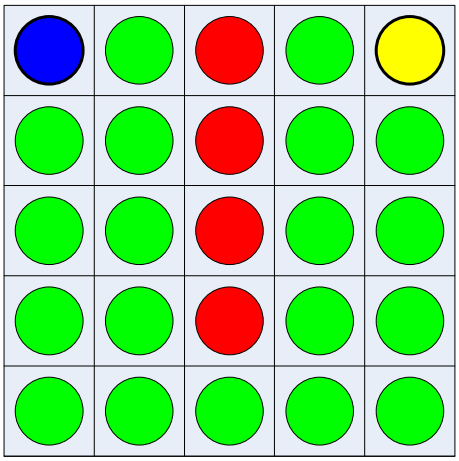
Berikut adalah potongan kode yang mengimplementasikan algoritma BFS dalam C++:

```
//fungsi DFS
//fungsi mengembalikan larik dari simpul-simpul
solusi
```

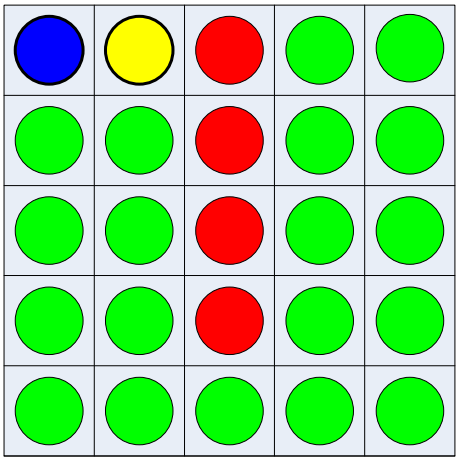
```
Simpul* DepthFirst::search() {
  Simpul* node; // larik dari simpul-simpul
                   solusi
  // selama antrian tidak kosong (kondisi tidak
     ada solusi)
  while (!_stack.empty()) {
  // ambil elemen head antrian
  node = _stack.top();
  _stack.pop();
  // jika solusi ditemukan
  if (problem.isGoal(node)){
    return node;
  }
  // ekspansi node, masukkan elemen yang
      dihasilkan ke antrian
    problem.successors(node, _stack);
  }
  return 0; // jika tidak ada solusi
}
```

5. HASIL EKSPERIMEN

Untuk menguji performa dari algoritma BFS dan DFS, penulis merancang dua buah peta sederhana dan mengujikan masing-masing algoritma ke dalam kota tersebut. Berikut adalah kota-kota yang digunakan dalam pengujian:



Gambar 5 a. Kota A



Gambar 5 b. Kota B

Keterangan dari gambar kota :

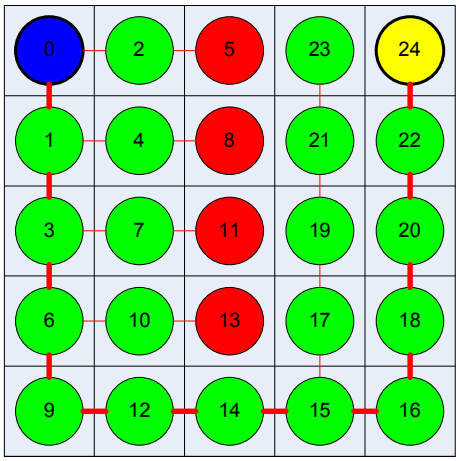
 : simpul (posisi) awal (akar pohon)

 : simpul (posisi) tujuan

 : simpul yang tidak dapat dilewati

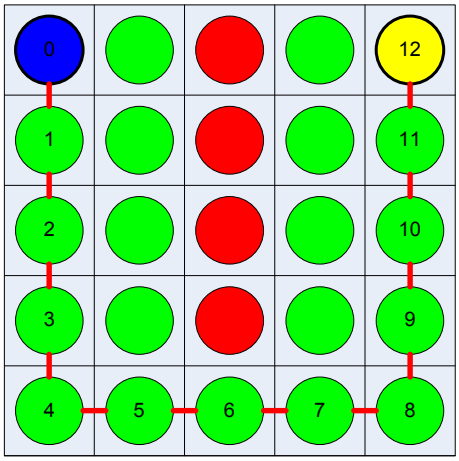
 : simpul yang dapat dilewati

Dari pengimplementasian algoritma BFS pada peta A, didapat pohon solusi sebagai berikut:



Gambar 6 a. Solusi kota A dengan BFS

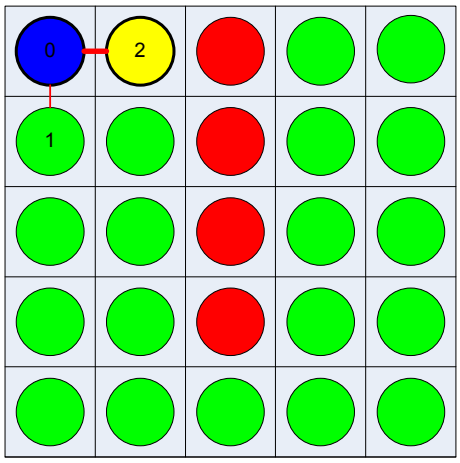
Sedangkan dengan algoritma DFS didapat pohon solusi sebagai berikut:



Gambar 6 b. Solusi kota A dengan DFS

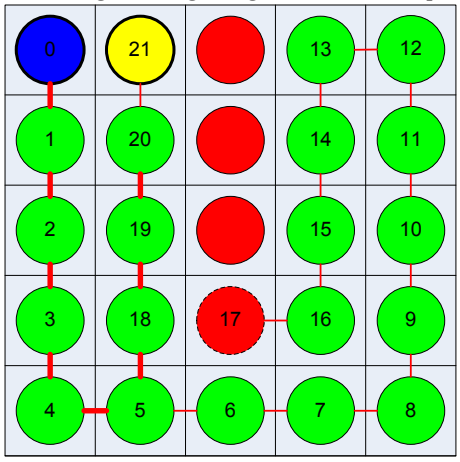
Dari hasil yang didapat dapat dilihat bahwa baik BFS maupun DFS menghasilkan rute perjalanan yang sama (lihat garis merah tebal). Akan tetapi, untuk mendapatkan solusi tersebut, algoritma BFS membutuhkan jauh lebih banyak pemeriksaan simpul dibandingkan dengan DFS, yaitu 24 : 12 (Urutan perbandingan / pembentukan pohon dapat dilihat dari angka yang tertera dalam simpul).

Untuk peta B, algoritma BFS menghasilkan solusi sebagai berikut:



Gambar 7 a. Solusi kota B dengan DFS

Sedangkan dengan algoritma DFS didapat:



Gambar 7 b. Solusi kota B dengan BFS

Ternyata untuk kasus yang sederhana seperti ini, di mana simpul tujuan merupakan simpul tetangga dari simpul akar, algoritma BFS jauh lebih cepat menemukan solusi dan rute yang benar, sedangkan algoritma DFS terjebak dengan upapohon memanjang dari simpul anak pertama yang dihasilkan, hinga menemukan jalan buntu di simpul ke 17 dan melakukan backtracking hingga ditemukan simpul solusi. Akan tetapi, rute perjalanan yang dihasilkan ternyata tidak optimal karena harus memutar terlebih dahulu.

6. ANALISIS

Dari hasil eksperimen yang didapat, kedua algoritma baik BFS maupun DFS dapat menemukan solusi. Akan tetapi dalam hal performa, jelas untuk peta A algoritma DFS lebih baik daripada BFS, tetapi untuk peta B algoritma BFS jauh mengungguli algotritma DFS.

Hal ini diakibatkan oleh karena BFS selalu membandingkan simpul-simpul yang bertetanggaan, sehingga untuk rute perjalanan yang memutar seperti kasus pada peta A ini BFS terpaksa memeriksa semua kemungkinan yang ada. Di lain persoalan, algoritma BFS memiliki keunggulan jika lokasi simpul solusi berada di dekat simpul akar seperti pada peta B karena algoritma dapat menemukan solusi sebelum membentuk pohon yang lebar.

BFS dalam kenyataannya memerlukan memori yang lebih besar, karena struktur data antrian yang digunakan harus menyimpan banyak informasi simpul yang bertumbuh secara eksponensial, sehingga untuk peta yang besar, jika posisi tujuan yang ingin dicapai cukup jauh, algoritma ini akan sangat menghabiskan memori. Akan tetapi BFS tidak akan terjebak dalam rute yang salah dan akan selalu mendapatkan solusi terbaik (*shortest path*). Jadi, algoritmaBFS cocokditerapkan jika posisi yang dituju berada dalam jarak yang tidak terlalu jauh dari posisi awal.

Di pihak lain, DFS yang langsung memeriksa semua simpul anak pertama (dalam hal ini urutan pengecekan anak adalah selatan-timur-utara-barat) hingga ditemukan simpul solusi mendapat keuntungan dalam kasus peta A karena simpul solusi selalu berada di dalam upapohon anak pertama sehingga tidak terjadi *backtracking.* Akan tetapi pada kasus petaB, DFS mengalami jalan buntu setelah melakukan pengecekan hingga simpul ke-17, bahkan menghasilkan solusi yang tidak optimal.

Dari segi pemakaian memori, DFS cenderung memerlukan kapasitas memori yang lebih kecil dibandingkan dengan BFS karena struktur data stack hanya akan menyimpan informasi simpul-simpul yang berada dalam rute yang sedang dicek. Akan tetapi algoritma ini sangat sulit dalam menemukan solusi optimal dan dapat terjebak dalam rute yang salah. Jadi, algoritma DFS cocok digunakan untuk permasalahan yang memiliki banyak solusi (simpul tujuan) tanpa harus memperhatikan solusi optimal, sehingga untuk kasus pencarian rute perjalanan objek dua dimensi ini, DFS tidak cocok digunakan (simpul tujuan selalu hanya ada satu)

6. KESIMPULAN

Dari hasil analisis, ternyata baik algoritma BFS dan DFS masing-masing memiliki keunggulan dan kerugian dalam memecahkan masalah pencarian rute perjalanan objek permainan dua dimensi, sehinggadalam kenyataannya kedua jenis algoritma ini jarang digunakan. Hal ini dikarenakan kedua algoritma ini hanya memberikan sebuah solusi pertama yang ditemukannya dan seringkali solusi yang diberikan bukanlah solusi optimal, padahal aplikasi dalam bidang permainan membutuhkan solusi optimal (misal : shortest path) untuk menentukan rute perjalanan. Algoritma BFS dan DFS memiliki keterbatasan karena mereka tidak tahu keadaan peta secara menyeluruh sehingga kedua algoritma hanya dapat bekerja secara *exhaustive search* tanpa petunjuk-petunjuk yang dapat menghasilkan solusi optimal.

Dalam kenyataannya, pencarian solusi optimal dalam menentukan rute perjalanan objek permainan dua dimensi pada umumnya menggunakan algoritma yang mengetahui keadaan peta sehinggapencarian rute dilakukan berdasarkan kriteria-kriteria tertentu, seperti algoritma A*. Akan tetapi, untuk pencarian solusi optimal yang tidak diketahui keadaan graf sama sekali, algoritma BFS masih layak untuk digunakan.

REFERENSI

Alison (1994). *Depth First vs Breadth First Search.* *http://www.macs.hw.ac.uk/%7Ealison/ai3notes* *Tanggal akses : 21 Mei 18.00*

Brainy Creatures (2005). *Artificial Intelligence.* *http://www.brainycreatures.org/ai* *Tanggal akses : 21 Mei 2007, 17.40*

Munir, Rinaldi (2007). Diktat Kuliah IF2251 – Strategi Algoritmik, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung

Wikipedia (2007). *Breadth-First Search.* *http://en.wikipedia.org/wiki/Breadth-first_search* *Tanggal akses : 21 Mei 2007, 17.30*

Wikipedia (2007). *Depth-First Search.* *http://en.wikipedia.org/wiki/Depth-first_search* *Tanggal akses : 21 Mei 2007, 17.30*