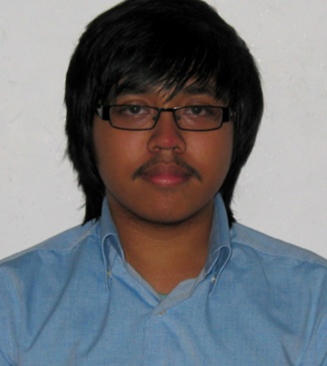
LAPORAN TUGAS BESAR II

IF3051 – Strategi Algoritma

Klotski and the Closed Sky

Aplikasi Algoritma BFS dan DFS pada permainan Klotski

Dimas Tri Ciputra 13509602

Karunia Ramadhan 13508056

Raka Mahesa 13508074

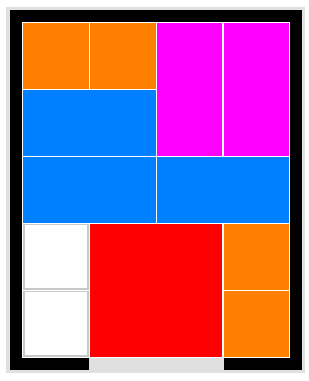
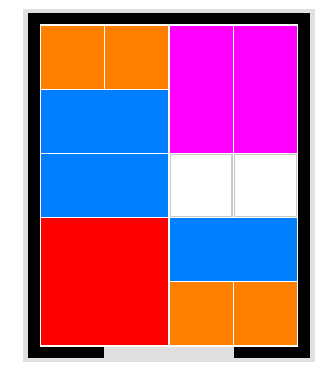
Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Sekolah Teknik Elektro dan Informatika ITB** | Nomor Dokumen | | Halaman |
| *IF3051-TB2-01* | | *16 hal* |
| Tahap | *1* | *18 Oktober 2010* |

# Deskripsi Masalah

Masalah yang diangkat dalam tugas besar II mata kuliah Strategi Algoritma ini adalah implementasi dari algoritma DFS dan BFS dalam permainan Klotski. Klotski (biasa disebut *Chinese Sliding Block Puzzle*) adalah sebuah *puzzle* yang berasal Cina, dimana obyektif permainan ini adalah mengeluarkan sebuah blok bujur-sangkar besar dari sebuah kotak. Kotak ini mempunyai ruang kosong terbatas yang akan dipakai untuk menggeser blok-blok agar blok bujursangkar besar dapat keluar. Blok yang dipindahkan itu sendiri harus cukup pada ruang kosong terdekat yang tersedia.



**Gambar 1 : Contoh soal dan solusi Klotski**

Mahasiswa diminta untuk membuat aplikasi yang mensimulasikan permainan Klotski sesuai spesifikasi tugas yang diperikan, lalu mengimplementasikan algoritma DFS dan BFSuntuk mendapatkan solusi permainan. Pengguna aplikasi bisa memilih untuk memainkan permainan Klotski yang sudah disimulasikan sendiri atau memilih pemain komputer yang dibuat berdasarkan algoritma DFS dan BFS untuk ’bermain’ dan menyelesaikan permainan.

# Dasar Teori

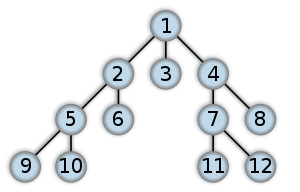
Algoritma BFS dan DFS adalah algoritma pencarian dalam graf, pohon, atau struktur pohon secara transversal. Transversal disini artinya adalah mengunjungi simpul-simpul yang ada dalam struktur dengan cara yang sistematik.

## BFS (Breadth First Search)

Dalam algoritma BFS, pencarian dimulai dari simpul akar dan mengunjungi semua tetangga dari simpul tersebut. Kemudian, pencarian akan dilanjutkan dari setiap simpul yang terdekat tersebut ke tetangga simpul yang belum dikunjungi, dan seterusnya, sampai mendapatkan solusi. Dengan kata lain :

* Traversal dimulai dari simpul v.
* Algoritma:

1. Kunjungi simpul v,
2. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu.
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.



**Gambar 2 : Contoh urutan simpul yang dikunjungi pada algoritma BFS**

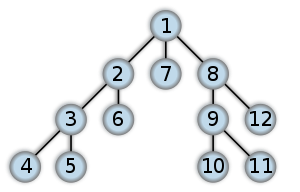
## DFS (Depth First Search)

Pada algoritma DFS, pencarian dimulai dari simpul akar, dilanjutkan dengan mengunjungi satu cabang sampao sedalam mungkin sebelum melakukan runut balik dan melanjutkan pencarian dari cabang lain.

• Traversal dimulai dari simpul v.

• Algoritma:

1. Kunjungi simpul v,
2. Kunjungi simpul w yang bertetangga dengan simpul v.
3. Ulangi DFS mulai dari simpul w.



**Gambar 3 : Contoh urutan simpul yang dikunjungi pada algoritma DFS**

# Analisis Penyelesaian Masalah

Pada algorima BFS dan DFS yang kami buat, kami tidak menggunakan struktur data pohon dalam implementasinya. Kami langsung menggunakan keadaan blok-blok yang ada untuk mendapatkan gerakan berikutnya (membentuk blok anak pada struktur pohon).

## BFS

Berikut adalah aturan-aturan yang dipakai untuk implementasi algoritma BFS pada permainan Klotski :

* Mulai dari kedalaman 0 dan kedalaman maksimal 1.
* Setiap mengunjungi node anak (menggerakkan block selain mundur), kedalaman bertambah.
* Setiap kembali ke node parent (block mundur), kedalaman berkurang.
* Untuk mencari blok yang bergerak, dilakukan iterasi terhadap semua block dan arah dari awal.
* Gerakkan blok sesuai arah pada iterasi bila sudah didapat dan bukan mundur ke node sebelumnya.
* Setiap mengunjungi node anak (block bergerak selain mundur), iterator disimpan dalam stack.
* Setiap mengunjungi node anak (block bergerak selain mundur), block dan arah gerak disimpan dalam stack.
* Setelah kembali ke node parent, iterasi dimulai dari iterator dalam stack.
* Bila iterasi tidak mendapatkan blok yang dapat bergerak lagi, maka mundur dengan menggunakan info pada stack.
* Bila iterasi tidak mendapatkan blok yang dapat bergerak lagi dan pada kedalaman 0, kedalaman maksimal bertambah.
* Setelah mencapai node anak, bila sudah kedalaman maksimal, mundur.

## DFS

Berikut adalah aturan-aturan yang dipakai untuk implementasi algoritma BFS pada permainan Klotski :

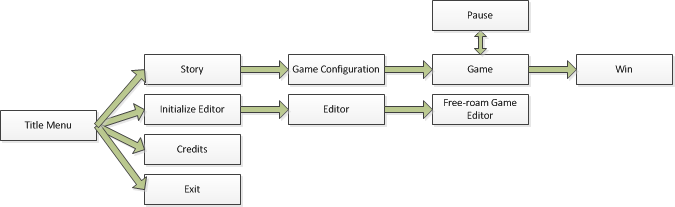
* Untuk mencari blok yang bergerak, dilakukan iterasi terhadap semua block dan arah dari awal.
* Gerakkan blok sesuai arah pada iterasi bila sudah didapat dan bukan mundur ke node sebelumnya.
* Setiap mengunjungi node anak (block bergerak selain mundur), iterator disimpan dalam stack.
* Setiap mengunjungi node anak (block bergerak selain mundur), block dan arah gerak disimpan dalam stack.
* Setelah kembali ke node parent, iterasi dimulai dari iterator dalam stack.
* Bila iterasi tidak mendapatkan blok yang dapat bergerak lagi, maka mundur dengan menggunakan info pada stack.
* Setelah mencapai node anak dan state ada di dalam daftar state, kembali ke node sebelumnya.
* Setelah mencapai node anak dan state tidak ada di dalam daftar state, state disimpan.

# Implementasi dan Pengujian

## Spesifikasi Teknis Program

Berikut adalah spesifikasi dari program yang kami buat :

* Program dibuat dengan bahasa C#, menggunakan *framework* XNA 3.1 dan platform .NET 3.5 dengan bantuan Flat Red Ball (FRB) Game Engine, Tomshane Neoforce GUI Engine, dan FMOD Sound Engine.
* Program dijalankan pada sistem operasi *Windows* yang sudah mempunyai .NET (minimal) 3.5 dan DirectX minimal 9.0c.
  + Untuk menjalankan program, diperlukan *file* dll. tambahan (Flat Red Ball dan Neoforce) pada *folder* program.
* Program dibagi menjadi empat macam *State* besar dan beberapa *State* tambahan yang diatur oleh *State Manager*. *State-state* kemudian yang akan menyusun program.



**Gambar 4 : Flow diagram sederhana Klotski**

Fitur program :

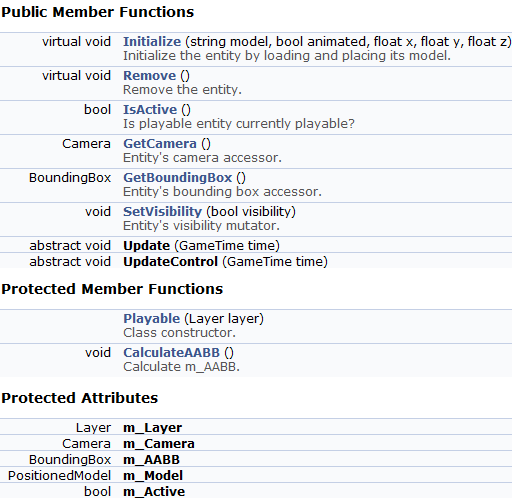
* Program memiliki empat pilihan pada menu utama, yaitu menuju permainan, pilihan untuk membuat *puzzle* (editor), melihat halaman *Credits,* dan keluar dari permainan.
* Permainan dibagi kedalam tiga tahap, yaitu cerita, konfigurasi permainan, dan permainannya sendiri.
  + Pada halaman *Story,* pengguna mampu memberhentikan cerita dan langsung melanjutkan bila tidak ingin melihat seluruh cerita.
  + Pada halaman konfigurasi, pengguna bisa memilih karakter yang akan digunakan untuk memainkan permainan, Klotski (dimainkan oleh pemain), BFS, maupun DFS (dikendalikan komputer) dan memilih *board* (yang diambil dari *file* eksternal) permainan yang akan dimainkan.
* Permainan ditampilkan dalam bentuk 3D, dengan animasi karakter dan *chase camera* (bila memainkan Klotski) dan tampilan kamera *bird view* untuk melihat keadaan beserta tampilan waktu yang berjalan dan langkah yang telah dilakukan.
  + Klotski memiliki tiga nyawa yang akan berkurang bila Klotski jatuh dari kapal. Bila nyawa Klotski habis, permainan akan berakhir.
  + Klotski harus melompati kapal-kapal dan mengendalikannya melewati ruang-ruang kosong pada *puzzle* untuk akhirnya mengeluarkan *The Royal Zeppelin* dan memenangi permainan.
  + BFS dan DFS akan menggerakkan kapal secara *remote* sesuai dengan algoritma mereka untuk memenangi permainan.
  + Bila pemain adalah BFS atau DFS, pengguna mampu mengatur kecepatan pergerakan kapal yang dijalankan pemain.
  + Permainan dapat diberhentikan sewaktu-waktu untuk memunculkan tutorial sekaligus menu untuk keluar permainan, kembali kedalam permainan, maupun mengulang permainan.
  + Pemain akan mendapat notifikasi yang berisi informasi permainan baik ketika pemain kalah maupun menang dalam permainan.
* Pengguna mampu membuat *board* permainan, baik dari segi luas maupun lokasi kapal yang ada pada editor.
  + *Board* yang didesain pada editor adalah *board* permainan dalam kondisi akhir atau kondisi solusi dari *puzzle.*
  + Editor memiliki jendela tutorial yang bisa dibuka dan ditutup sewaktu-waktu.
* Setelah mendesain *board*, pengguna akan masuk kedalam mode *free-roam* editor. Disini pengguna program mampu mengacak *board* kondisi solusi yang dibuat sebelumnya.
  + *Free-roam* memiliki fitur *pause* yang hampir sama dengan yang ada pada permainan.
  + Pengguna mampu menyimpan *board* yang telah dia acak kedalam *file* eksternal yang nantinya bisa digunakan dalam permainan.
* Halaman *Credits* berisi informasi tentang pembuat permainan, orang lain serta hal-hal yang terlibat dalam pembuatan permainan ini.
  + Pada halaman *Credits,* pengguna juga mampu memberhentikan *credits* dan kembali bila tidak ingin melihat seluruhnya.

### *Struktur Data*

Dalam laporan ini, kami hanya akan membahas struktur data dari kelas yang paling penting dalam keberlangsungan permainan Klotstki ini yaitu kelas abstrak *Playable*, kelas *Ship*, *Actor*, *Game Data* dan *State Game*. Hal ini disebabkan karena banyaknya kelas dalam aplikasi ini yang berfungsi sebagai pengatur (*State Manager, Sound Manager*) dan komponen dari permainan itu sendiri (*Sky, etc.*).

Keterangan : *Header* dari tiap kelas tidak menampilkan atribut maupun fungsi *private* dari kelas tersebut.

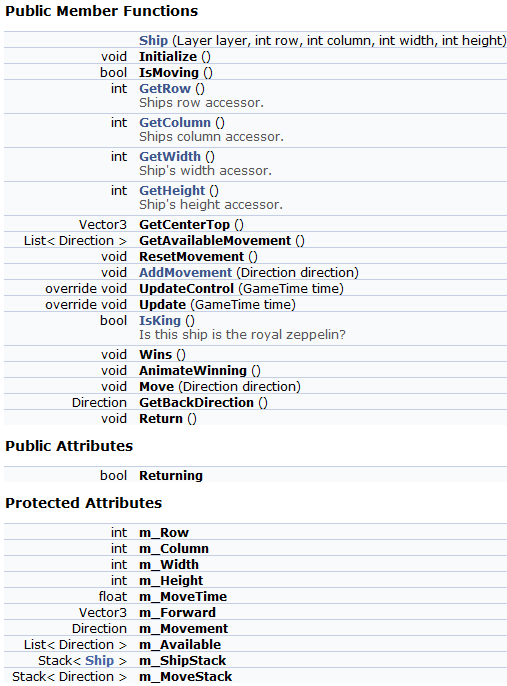
#### Playable



**Gambar 5 : *Header* fungsi dan atribut kelas Playable**

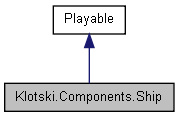
Kelas ini merupakan kelas abstrak yang implementasinya akan merepresentasikan objek yang bisa dikontrol oleh pemain. Atribut yang penting dari kelas ini adalah model yang mewakili kelas tersebut dalam permainan, *bounding box* dari model tersebut, dan kamera yang ada pada model.

#### Ship



**Gambar 6 : *Header* fungsi dan atribut kelas Ship**

Kelas ini merupakan kelas kapal dalam permainan, yang merupakan sebuah blok dalam *puzzle* Klotski. Kelas ini menyimpan atribut tinggi, lebar, dan lokasi kapal dalam logika, serta mengatur pergerakan kapal. Sebagai blok dalam permainan, kelas ini juga mempunyai atribut penyimpan *state* kapal tersebut selama permainan dalam sebuah *stack* yang akan digunakan dalam implementasi algoritma BFS dan DFS.



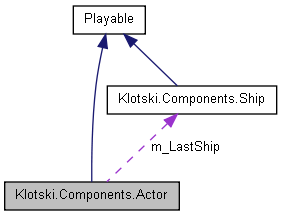
**Gambar 7 : *Collaboration Diagram* dari kelas Ship**

#### Actor

****

**Gambar 8 : *Header* fungsi dan atribut kelas Actor**

Kelas ini adalah kelas yang mengatur pemain utama dari permainan ini, Klotski (bernama sama dengan *puzzle* yang menginspirasi permainan ini). Klotski mempunyai atribut *Life* yang menandakan nyawa hidupnya dalam permainan dan berbagai informasi lain yang menandakan statusnya selama permainan berlangsung.



**Gambar 9 : *Collaboration Diagram* dari kelas Actor**

#### Game Data



**Gambar 10 : *Header* fungsi dan atribut kelas Game Data**

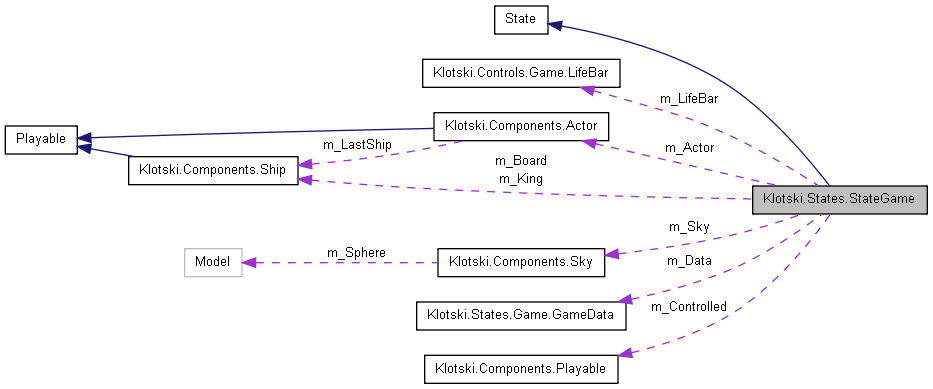
Kelas Game Data merupakan kelas yang menyimpan data permainan seperti daftar dari *Ship* yang ada beserta lokasi *goal* dari *puzzle.* Kelas ini juga yang berhubungan dengan *file* eksternal, dimana dia akan menyimpan data kelas keluar dan menginput data kelas kedalam.

#### State Game



**Gambar 11 : *Header* fungsi dan atribut kelas State Game**

Kelas ini merupakan kelas utama, *game* *state* yang mengatur permainan dari Klotski. Kebanyakan dari fungsi yang ada dalam kelas ini sendiri adalah fungsi *private* yang mengatur logika permainan . Implementasi algoritma **BFS** dan **DFS** berada langsung di dalam kelas ini, dan akan dipakai bergantung kepada pilihan pemain dalam konfigurasi yang ada.



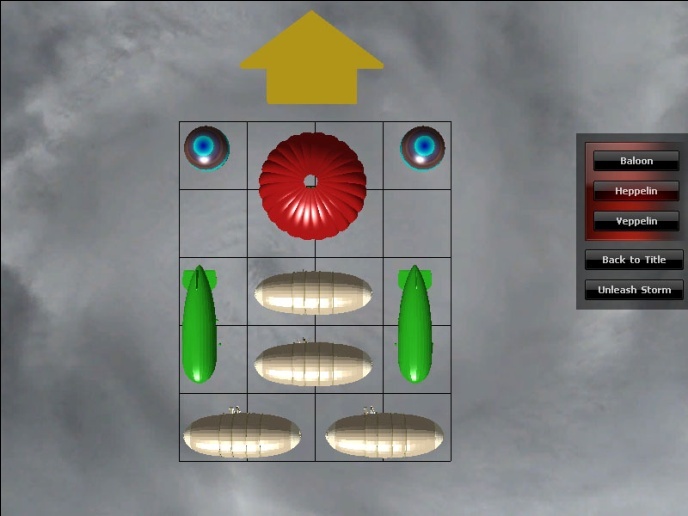
**Gambar 12 : *Collaboration Diagram* dari kelas State Game**

### *Antarmuka Program*

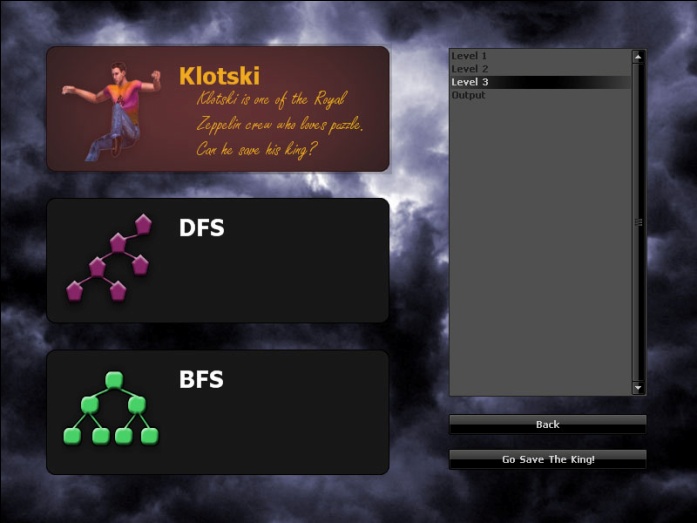
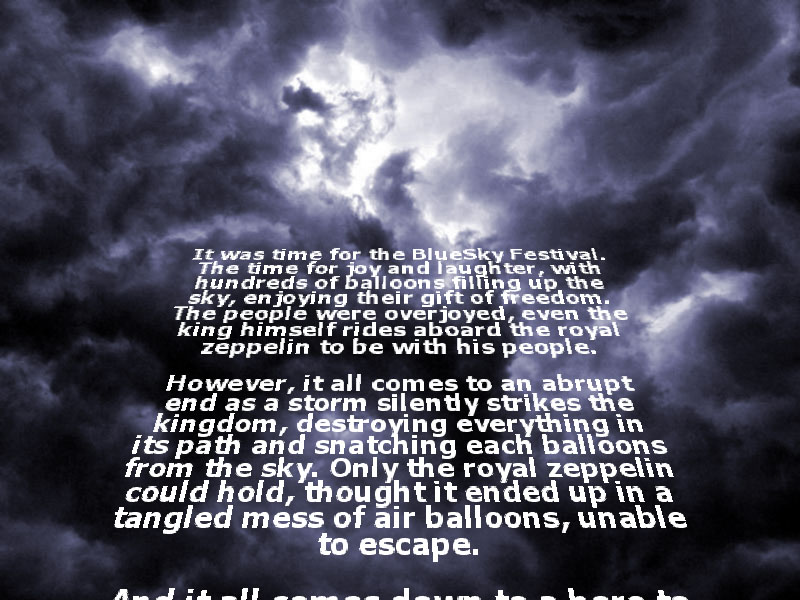
Berikut ini adalah beberapa tampilan antarmuka dari permainan yang kami buat :

****

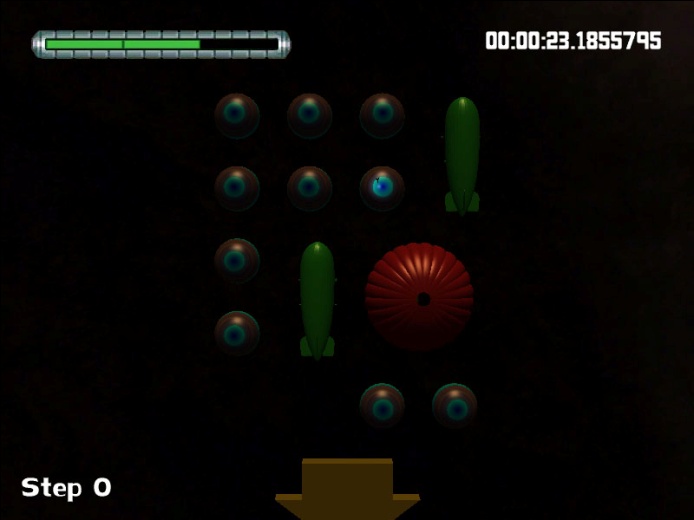
**Gambar 13 : Game Menu Gambar 14 : Initialize Editor**

****

**Gambar 15 : Editor Gambar 16 : Free-Roam Editor**

****

**Gambar 15 : Story Gambar 16 : Configuration**

****

**Gambar 15 :Antarmuka Gameplay Gambar 16 : Bird-View Camera**

****

**Gambar 15 : Contoh DFS Gambar 16 : Win Interface**

****

**Gambar 15 : Game Over Gambar 16 : Credits**

## Eksperimen/Pengujian

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Map** | **BFS** | | | **DFS** | | |
|  | **Langkah** | **Waktu** | **Simpul** | **Langkah** | **Waktu** | **Simpul** |
| Level 1 | 90 | 20 | 19 | 16 | 4 | 8 |
| Level 2 | 1141313 | 35028 | 160 | 63 | 14 | 55 |
| Level 3 |  |  |  | 1164 | 253 | 450 |
| Level 4 | 47 | 11 | 8 | 7 | 2 | 7 |
| Level 5 | 91357 | 18036 | 233 | 35 | 8 | 34 |

Note : (merah berarti belum mencapai solusi sampai waktu tersebut)

* Waktu dalam satuan detik.
* Simpul = jumlah simpul yang dikunjungi.
* Pengujian dilakukan dengan kecepatan animasi yang tercepat.

## Analisis Hasil Pengujian

Dari hasil pengujian yang kami lakukan terhadap permainan Klotski, kesimpulan secara umum adalah bahwa **algoritma DFS lebih efektif dalam menyelesaikan *puzzle* Klotski dibandingkan algorima BFS**.

Pada DFS, algoritma akan mencoba menggerakkan blok-blok sampai suatu keadaan tidak bisa digerakkan lagi atau keadaan yang sudah pernah dikunjunginya (*State* yang berulang), mengunjungi simpul-simpul sampai kedalaman yang terdalam, baru runut-balik dan mencoba simpul lain. Dari hasil percobaan sendiri, implementasi DFS lebih cocok dan cepat dalam mendapatkan solusi *puzzle* yang diuji. Implementasi DFS mampu mendapatkan solusi dalam hitungan detik untuk soal yang dimana implementasi BFS sudah melakukan aksi berjam-jam tapi masih belum mendapatkan solusi. Salah satu penyebab hal ini adalah implementasi BFS mencoba semua keadaan terlebih dahulu baru mengunjungi simpul yang lebih dalam. Hal ini menyebabkan pengecekan yang meluas dan boros, sedangkan pengecekan yang dilakukan DFS cukup efisien dalam menemukan solusi.

Animasi yang dilakukan BFS juga banyak mempengaruhi waktu yang digunakan untuk mencapai solusi *puzzle.* Pada BFS, implementasi akan menganimasikan semua gerakan blok ke masing-masing keadaan sehingga waktu yang diperlukan untuk mencapai solusi akan lama. Hal ini menyebabkan perbandingan waktu BFS dan DFS akan selalu berbeda jauh.

# Kesimpulan dan Saran

## Kesimpulan

* Algoritma BFS dan DFS dapat digunakan untuk menyelesaikan *puzzle* Klotski.
* Banyak cara yang bisa dilakukan untuk mengimplementasikan algoritma BFS dan DFS pada permainan ini, termasuk cara yang kami gunakan yaitu membuat pohon secara implisit dengan *Stack.*
* Secara umum, algoritma DFS lebih efektif untuk mendapatkan solusi dari *puzzle* Klotski.

## Kritik dan Saran

* **Dalam implementasinya, algoritma DFS dan BFS yang dibandingkan harus setara.**

Perbandingan waktu BFS dan DFS di sini akan selalu berbeda jauh, sebab pada BFS untuk mencapai node-nya diperlukan waktu yang lama. Akan lebih baik bila ada suatu mekanisme untuk mencapai suatu node secara instan agar waktu prosesnya dapat dibandingkan langsung dengan DFS. Meskipun begitu, mekanisme tersebut dapat menghancurkan visualisasi gerakan pada BFS sehingga menyulitkan pemeriksaan apakah pergerakannya benar atau salah.

* **Diperlukan perancangan aplikasi yang matang.**

Pada aplikasi ini, tidak ada struktur data untuk algoritma BFS-DFS, yang ada hanya struktur untuk game (model, kolisi, gerakan, dsb) sehingga menyulitkan implementasi algoritma. Akan lebih baik bila dari awal sudah dirancang sebuah struktur yang mendukung aplikasi baik dari segi game maupun segi logika algoritma.

# Referensi

[**http://www.flatredball.com/frb/docs/index.php?title=Category:Tutorials**](http://www.flatredball.com/frb/docs/index.php?title=Category:Tutorials)

[**http://www.fmod.org/**](http://www.fmod.org/)

[**http://kur2003.if.itb.ac.id/rencana.php?kode=IF2251**](http://kur2003.if.itb.ac.id/rencana.php?kode=IF2251)

[**http://www.tomshane.cz/neoforce/default.aspx**](http://www.tomshane.cz/neoforce/default.aspx)

[**http://webmail.informatika.org/~rinaldi/Stmik/2009-2010/stmik09-10.htm#SlideKuliah**](http://webmail.informatika.org/~rinaldi/Stmik/2009-2010/stmik09-10.htm#SlideKuliah)