**PENGEMBANGAN APLIKASI PENCARIAN SOLUSI**

**PERMAINAN “*8-PUZZLE*” MENGGUNAKAN**

**ALGORITMA PENCARIAN A\***



**Oleh :**

**Kifni Taufik Darmawan**

**15/388485/PPA/04924**

**PROGRAM STUDI MAGISTER ILMU KOMPUTER**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS NEGERI GADJAH MADA**

DAFTAR ISI

[I. Latar Belakang 2](#_Toc437935017)

[II. Dasar Teori 2](#_Toc437935018)

[A. 8-Puzzle 2](#_Toc437935019)

[B. Algoritma Pencarian A\* 2](#_Toc437935020)

[III. Metode Penelitian 3](#_Toc437935021)

[A. Studi Pustaka 3](#_Toc437935022)

[B. Perancangan & Pengembangan Aplikasi 3](#_Toc437935023)

[C. Uji Coba 3](#_Toc437935024)

[IV. Pengembangan Aplikasi 3](#_Toc437935025)

[A. Representasi Puzzle State 3](#_Toc437935026)

[B. Impelentasi Algoritma A\* 4](#_Toc437935027)

[1. Puzzle State 4](#_Toc437935028)

[2. Algoritma A\* untuk pencarian solusi 4](#_Toc437935029)

[C. Pembuatan User Interface 7](#_Toc437935030)

[V. Hasil dan Pembahasan 8](#_Toc437935031)

[VI. Kesimpulan 9](#_Toc437935032)

[VII. Daftar Pustaka 9](#_Toc437935033)

# Latar Belakang

Algoritma pencarian (*Search* *Algorithm*) adalah algoritma yang menerima data atau kunci dan dengan langkah-langkah tertentu mencari rekaman dengan data atau kunci tersebut. Algoritma pencarian merupakan cabang ilmu komputer yang terus berkembang.

Secara umum algoritma pencarian dikategorikan ke dalam 2 jenis yaitu algoritma pencarian *Uninformed* dan *Informed*. Algoritma pencarian Uninformed mencari tujuan dengan menjelajah state sampai ditemukan tujuan yang dicari, tanpa adanya dasar informasi yang digunakan untuk menentukan arah pencarian. Contoh algoritma pencarian uninformed adalah : *Random Walk*, *Depth-First search*, dan *Breadth-first search*. Sedangkan algoritma pencarian *informed* menggunakan informasi tertentu yang digunakan untuk menentukan arah dalam proses pencarian. Contoh algoritma pencarian informed adalah Dijkstra, A\* dan IDA\*.

Algoritma pencarian dapat diterapkan dalam beberapa bidang, salah satunya adalah dalam pencarian solusi untuk pada permainan puzzle. Algoritma pencarian dapat mencari solusi terbaik dalam menyelesaikan suatu permainan puzzle.

Dalam penelitian ini penulis menerapkan algoritma pencarian A\* dalam mengembangkan aplikasi untuk mecari solusi dalam permainan 8-Puzzle.

# Dasar Teori

## 8-Puzzle

8-Puzzle adalah salah satu varian dari permainan sliding block puzzle yang pertama kali diciptakan oleh Noyes Chapman. Versi pertama yang diciptakanya adalah versi 15 Puzzle. Permainan ini mulai popuper pada tahun 1880. Puzzle ini terdiri dari 15 ubin yang diberi angka 1 sampai 15 pada frame ukuran 4x4. Tujuan permainan ini adalah untuk mengurutkan ubin dari anga 1 sampai 15.

Puzzle ini dikembangkan dalam beberapa varian lain, 8-Puzzle adalah varian yang lebih sederhana dengan menggunakan 8 ubin dalam frame ukuran 3x3.

## Algoritma Pencarian A\*

*A\** ( baca : A Star), adalah salah satu jenis algoritma pencarian *informed*. Algorima ini pertama kali dijelaskan oleh Peter hard, Nils Nilsson dan Bertram Raphael dari *Stanford Research Institute* (*SRI International*) pada tahun 1968. Algoritma ini merupakan pengembangan dari algoritma pencarian *Dijkstra*. Algoritma *A\** mampu menunjukan performa yang lebih baik dengan menggunakan *heuristic* untuk memandu pencarian.

Algoritma pencarian A\* mencari node dengan nilai cost paling rendah untuk di-eskplor. Nilai cost dihitung dengan rumus :

*g(x) :* Cost yang dibutuhkan untuk berpindah dari posisi awal ke posisi node x.

h(x) : Perkiraan jarak dari node x ke node tujuan.

# Metode Penelitian

## Studi Pustaka

Pencarian referensi tentang algoritma pencarian A\* dan penerapan algoritma pencarian untuk mencari solusi dalam permainan puzzle.

## Perancangan & Pengembangan Aplikasi

Proses perancangan dan pengembangan aplikasi untuk mencari solusi pada permainan 8-puzzle dengan menerapkan algoritma pencarian A\*.

## Uji Coba

Menguji aplikasi yang dikembangkan dengan beberapa kondisi awal puzzle.

# Pengembangan Aplikasi

Dalam penelitian ini, Penulis menggunakan bahasa pemrograman Javascript untuk implementasi Algoritma pencarian A\* untuk mencari solusi pada permainan 8-Puzzle.

## Representasi Puzzle State

Setiap kondisi (state) pada permainan 8-Puzzle dapat direpresentasikan dalam bentuk array yang berisi angka 0 sampai 8 dengan urutan sesuai dengan kondisi puzzle. Berikut contoh inisialisasi state dalam bentuk array.

var goal\_state = [0, 1, 2, 3, 4, 5, 6, 7, 8];

var initial\_state = [1, 2, 3, 0, 4, 5, 6, 8, 7];

## Impelentasi Algoritma A\*

### Puzzle State

Setiap state yan dilalui dalam proses pencarian menggunakan, akan disimpan dalam suatu object Puzzle yang dengan property *state* (kondisi puzzle) dan *path* (jalur yang dilalui) :

function Puzzle(state, path){

this.state = state;

this.path = path;

}

Property state menjunjukan state puzzle dalam bentuk array. Sedangkan property path berisi jalur yang dilalui untuk mencapai state tersebut. Penulis menggunakan dengan tipe data String dengan separator, untuk menghemat penggunaan memory ketika pencarian, selain itu proses untuk mengubah string tersebut ke dalam array dalam javascript relatif mudah dan cepat. Dalam object puzzle tersebut juga tidak terdapat property *children* dan *parent* yang biasa digunakan untuk merepresentatikan tree, karena selain untuk menghemat penggunaan memory, property path sudah cukup untuk mendapatkan *state* – *state* yang dilalui dari *initial* *state* sampai dengan *goal* *state*.

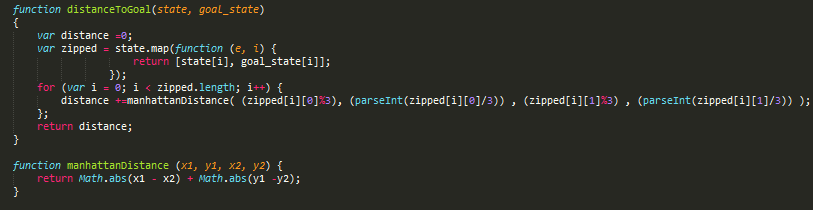
### Algoritma A\* untuk pencarian solusi

Dalam implementasi algoritma pencarian A\* dalam pencarian solusi permainan *8-Puzzle*, setiap node yang dieksplorasi adalah kondisi (state) puzzle. Dalam penelitian ini, setiap state puzzle disimpan dalam array. Iterasi pertama dimulai dengan eksplorasi initial state, kemudian mencari possible state dengan aksi berbeda (*left*, *right*, *up*, *down*). Puzzle state yang dari aksi tersebut kemudian dihitung *cost*-nya, dan kemudian indeks puzzle state tersebut di masukkan ke dalam *PriorityQueue*. Eksplorasi puzzle state selanjutnya dilakukan pada puzzle *state* dengan *cost* paling rendah yang diambil dari *PriorityQueue.* Perhitungan *cost* pada tiap state merupakan jumlah langkah yang dibutuhkan untuk mencapai state tersebut ditambah dengan perkiraan langkah yang dibutuhkan untuk mencapai *goal state* dari *state* tersebut.

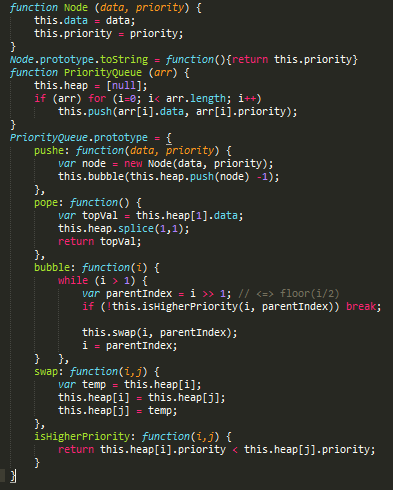
fungsi tersebut membutuhkan fungsi untuk menghitung perkiraan langkah yang dibutuhkan untuk mencapai goal state. Berikut adalah fungsi yang digunakan :







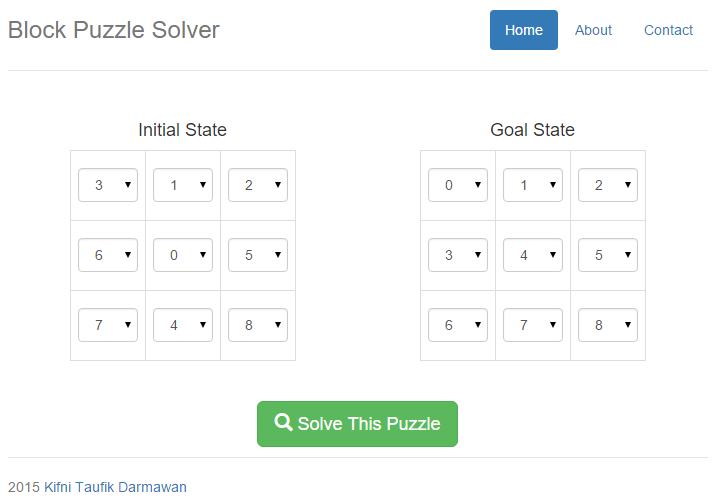
Algoritma A\* akan mengeksplor state dengan nilai cost yang paling rendah, untuk itu dibutuhkan priority queue yang digunakan untuk menyimpan sementara indeks *puzzle* *state* dan *cost-nya menggunakan fungsi push,* mengengeluarkan *puzzle state* dengan *cost* paling rendah mengunakan fungsi *pop* untuk dieksekusi pada tiap iterasi. Penulis mengunakan fungsi PriorityQueue yang dibuat oleh GRIFFnDOOR (<https://jsfiddle.net/GRIFFnDOOR/r7tvg/> ), yang penulis modifikasi untuk memperbaiki error dan mempersingkat kode.



## Pembuatan User Interface

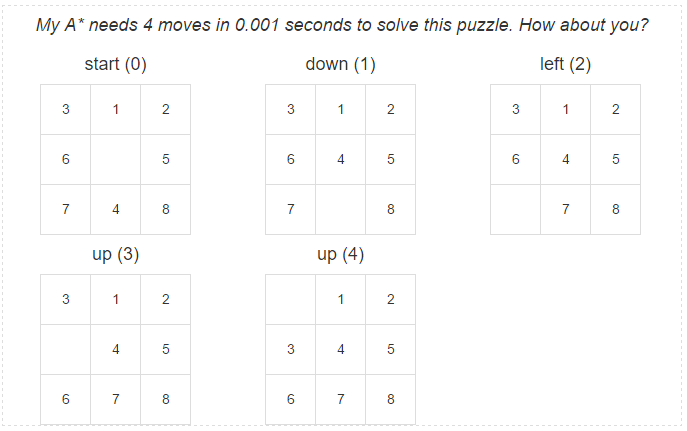
Aplikasi yang dikembangkan dalam penelitian ini berbasis web. Penulis menggunakan library CSS *Twitter Bootstrap* dan library Javascript *JQuery* untuk membantu pembuatan antar muka. Aplikasi yang dikembangkan dalam penelitian ini memungkinan pengguna untuk melakukan pencarian solusi dengan goal state dan initial state yang dapat ditentukan.

Berikut adalah tampilan antar muka aplikasi yang dikembangkan :



# Hasil dan Pembahasan

Aplikasi yang dikembangkan dalam penelitian ini mampu melakukan pencarian solusi dari 8-puzzle dengan initial state dan goal state yang dapat diinputkan melalui antar muka aplikasi web. Berikut adalah contoh tampilan penyelesaikan puzzle :



Berikut beberapa hasil percobaan yang dilakukan :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Initial State | Goal State | Jumlah Langkah | Waktu (detik) |
| 1 | [1,3,4  ,8,6,2,  7,0,5] | [1,2,3  ,8,0,4,  7,6,5] | 5 | 0.002 |
| 2 | [2,8,1  ,0,4,3  ,7,6,5] | [1,2,3  ,8,0,4,  7,6,5] | 9 | 0.006 |
| 3 | [2,8,1  ,4,6,3,  0,7,5] | [1,2,3  ,8,0,4,  7,6,5] | 12 | 0.024 |
| 4 | [2, 1, 0,  6, 8, 3,  4, 7, 5] | [1,2,3  ,8,0,4,  7,6,5] | 16 | 5.198 |
| 5 | [5, 6, 7,  4, 0, 8,  3, 2, 1] | [1,2,3  ,8,0,4,  7,6,5] | 32 | 1165.9 |

# Kesimpulan

Dari hasil pembuatan aplikasi pencarian solusi pada permainan 8-Puzzle dapat disimpulkan bahwa :

1. Aplikasi yang dibuat mampu menyelesaikan puzzle berdasarkan input initial dan goal state yang dapat ditentukan melalui antar muka.
2. Metode pencarian A\* dapat digunakan untuk mencari solusi permainan 8-puzzle.
3. Waktu yang digunakaan untuk menyelesaikan puzzle dengan aplikasi yang dikembangkan cukup cepat untuk menyelesaikan puzzle dengan jumlah langkah solusi <20, tetapi untuk puzzle yang sangat rumit (solusi > 20) membutuhkan waktu relatif lama.

# Daftar Pustaka

Danger, J. (2013, January 2). *Graph Traversal: solving the 8-puzzle with basic A.I.* Retrieved December 11, 2015, from Jack Danger Canty: https://jdanger.com/solving-8-puzzle-with-artificial-intelligence.html

GRIFFnDOOR. (n.d.). *Priority Queue*. Retrieved December 10, 2015, from JSFiddle: https://jsfiddle.net/r7tvg/53/

Spaans, R. G. (Improving sliding-block puzzle solving using meta-level reasoning). *2010.* Trondheim: Norwegian University of Science and Technology.

Source Code : <https://bitbucket.org/kifni41/block-puzzle-solver>

Live App : <http://jagallab.kifni.com/block-puzzle-solver>