**LAPORAN**

**PEMBUATAN GAME**

**“NUMERIC PUZZLE”**

*Diajukan untuk memenuhi salah satu tugas mata kuliah*

*Analisis Algoritma*

****

**Disusun Oleh :**

**KELOMPOK 01**

Disca Amelia (10114345)

Fitriani Nur Solihat (10114373)

Putri Sarnianty Achmad (10114219)

**Kelas** : AA-05

**PROGRAM STUDI TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK DAN ILMU KOMPUTER**

**UNIVERSITAS KOMPUTER INDONESIA**

**2016**

# **PENJELASAN GAME**

## **Pengertian**

Apa itu Numeric Puzzle? Numeric Puzzle adalah permainan puzzle ciptaan Sam Loyd yang terdiri dari sebuah papan kosong n x n yang berisi kotak-kotak angka dalam keadaan teracak dari angka 1 sampai (n2 – 1) angka dan satu kotak kosong. Satu kotak kosong (disisi paling kanan) digunakan untuk memindahkan papan-papan disekelilingnya. Pemindahan bisa dilakukan dengan memindahkan papan kosong tersebut ke atas, ke bawah, ke kanan, atau ke kiri. Papan yang kosong harus terus dipindah-pindahkan agar solusi akhir didapat atau kondisi puzzle telah terurut.

## **Sejarah**

Sejak dulu tidak jelas kapan puzzle slide pertama diciptakan atau dibuat. Tetapi fakta yang diketahui bahwa pada tahun 1878 Sam Loyd, ahli puzzle terbaik Amerika, "*drove the whole world crazy*" (dalam kata-katanya sendiri) dengan yang baru "ditemukan" 14-15 puzzle-nya. Ini adalah variasi "Puzzle 15" yang dibuat dan dijual oleh embossing Perusahaan dari New York sekitar 10 tahun sebelumnya. Puzzle ini terdiri dari 15 nomor potongan persegi yang bisa di geser di dalam kotak persegi yang cukup besar untuk menampung 16 buah. Potongan kotak angka yang harus ditempatkan secara acak di dalam kotak dan Anda harus mengurutkan kotak angka itu tanpa mengeluarkan kotak angka dari bingkai (jadi satu-satunya yang diperbolehkan adalah menggeser kotak kotak tersebu). Tidak setiap pola acak dari potongan kotak itu dapat diurutkan dengan hanya menggeser potongan kotak itu dan Sam Loyd cerdik dalam memanfaatkan fakta ini.

Tidak mengherankan bahwa Sam melaju seluruh dunia gila dengan variasi nya dari puzzle 15. Masalahnya rumusnya tidak mungkin untuk memecahkan. Ketika Anda membeli Sam 14-15 puzzle kotak kosong diposisikan kanan bawah. Kotak-kotak yang diberi nomor dalam urutan dari kiri ke kanan dan dari atas ke bawah; hanya potongan nomor 1 sampai 14 atau 15 diacak. Anda harus menyusun potongan kotak tersebut sehingga semua potongan kotak dalam posisi yang benar dan tempat kosong harus diposisikan kanan bawah lagi. Sebuah sliding puzzle dengan potongan-potongan kotak persegi hanya dapat diselesaikan ketika jumlah pertukaran yang diperlukan untuk memecahkan puzzle terjadi. 14-15 puzzle menarik perhatian dunia luas yang hanya bisa dibandingkan dengan Cube Rubik yang menaklukkan dunia 100 tahun kemudian. Erno Rubik sebenarnya terinspirasi oleh sliding puzzle ketika ia merancang kubus terkenal yang dapat dilihat sebagai Versi 3 dimensi dari sliding puzzle.

Pada tahun 2006 Jerry Slocum dan Dic Sonneveld menerbitkan buku mereka: The 15 Puzzle. Banyak hal mengejutkan terungkap dalam buku ini, termasuk:

"*Sam Loyd tidak menciptakan puzzle 15 dan tidak ada hubungannya dengan mempromosikan atau mempopulerkan itu. Puzzle menggila yang diciptakan dari 15 Puzzle dimulai pada Januari 1880 di AS dan pada bulan April di Eropa. Puzzle menggila berakhir pada bulan Juli 1880 dan Artikel pertama Sam Loyd tentang puzzle tidak dipublikasikan sampai enam belas tahun kemudian, Januari 1896. Loyd pertama mengklaim pada tahun 1891 bahwa ia menemukan puzzle, dan terus kampanye 20 tahun sampai kematiannya. Penemu yang sebenarnya adalah Noyes Chapman, yang Postmaster dari Canastota, New York, dan ia diterapkan untuk paten Maret 1880.* "

## **Cara Main**

Cara bermain game numeric puzzle :

1. Setelah muncul Angka-angka secara acak, disitu terdapat kolom kosong yang merupakan tempat yang digunakan untuk membantu memindahkan/menggerakan kolom yang memiliki bilangan di dalamnya(kolom Bilangan).
2. Kolom bilangan tadi dipindahkan dengan cara menggeserkan ke kolom kosong, sehingga kolom kosong dapat berubah tempat.
3. Pilih kolom bilangan yang terdekat dengan tempat kolom kosong.
4. Geser kolom bilangan ke kolom kosong.
5. Dan terus lakukan step by step nya sehingga game ini selesai , menjadi urutan angka yang sesuai.

## **Aturan-Aturan Permainan :**

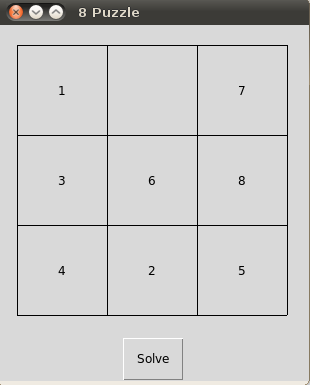
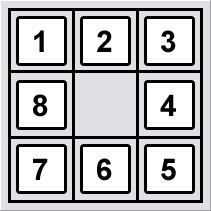
|  |  |  |
| --- | --- | --- |
| **Aturan ke-** | **Kondisi** | **Pergerakan Papan Kosong** |
| 1 | Jika Papan Kosong berada di tengah. | Atas, Bawah, Kiri,Kanan |
| 2 | Jika Papan Kosong Berada di paling kanan | Atas, Bawah, Kiri |
| 3 | Jika Papan Kosong Berada di paling kiri | Atas, Bawah, Kanan |
| 4 | Jika Papan Kosong Berada di paling atas | Kiri, Kanan, Bawah |
| 5 | Jika Papan Kosong Berada di paling bawah | Kiri, Kanan, Atas |
| 6 | Jika Papan Kosong Berada di sudut Kanan Atas | Kiri, Bawah |
| 7 | Jika Papan Kosong Berada di sudut Kiri Atas | Kanan, Bawah |
| 8 | Jika Papan Kosong Berada di sudut Kanan Bawah | Kiri, Atas |
| 9 | Jika Papan Kosong Berada di sudut Kiri Bawah | Kanan, Atas |

## **Kondisi** **Menang**

Kondisi menang adalah saat angka acak dalam puzzle telah terurut dari 1-8 dan kotak kosong kembali berada disisi kanan bawah.

## **Gambar Puzzle**

### C:\Users\8.1\Pictures\soal-puzzle.jpg



# **Penjelasan Strategi Algoritma**

## **II.1.** **Pengertian Best-first search**

Algoritma best first search ini merupakan kombinasi dari algoritma depth first search  dengan algoritma breadth first search  dengan mengambil kelebihan dari kedua  algoritma tersebut. Apabila pada pencarian dengan algoritma  hill climbing tidak  diperbolehkan untuk kembali ke node pada level yang lebih rendah meskipun node di  level yang lebih rendah tersebut memiliki nilai heuristik yang lebih baik, lain halnya pada algoritma best first search, pencarian diperbolehkan mengunjungi node yang ada di level yang lebih rendah, jika ternyata node di level yang lebih tinggi memiliki nilai heuristik yang lebih buruk (Kusumadewi, 2003).

Algoritma best first search merupakan salah satu bagian dari tipe informed search. Algoritma ini menggunakan nilai-nilai heuristik tiap simpul yang dibuka. Simpul dengan nilai heuristik terbaik akan dibuka lebih dahulu. Bila goal state belum ditemukan, akan dilakukan pemeriksaan pada simpul berikutnya dengan nilai heuristik terbaik pada kedalaman yang sama. Simpul tersebut kemudian dibuka dan diperiksa apakah terdapat goal state pada cabang-cabangnya. Bila goal state belum ditemukan,  akan dilakukan proses yang sama pada simpul berikutnya.

## **II.2.** **Cara Kerja Algoritma BFS**

Secara umum berikut adalah cara kerja algoritma Best-First Search :

OPEN berisi initial state dan CLOSED masih kosong.

Ulangi sampai goal ditemukan atau sampai tidak ada nodes di dalam OPEN

a. Ambil simpul terbaik yang ada di OPEN

b. Jika simpul tersebut sama dengan goal, maka sukses

c. Jika tidak, masukkan simpul tersebut ke dalam CLOSED

d. Bangkitkan semua suksesor dari simpul tersebut

e. Untuk setiap suksesor kerjakan :

i. Jika suksesor tersebut belum pernah dibangkitkan, evaluasi sukseso tersebut, tambahkan ke OPEN, dan catat parent atau orang tuanya.

ii. Jika suksesor tersebut sudah pernah dibangkitkan, ubah parentnya jika jalur melalui parent ini lebih baik daripada jalur melalui parent yang sebelumnya. Selanjutnya, perbarui biaya untuk suksesor tersebut dan nodes lain yang berada di level bawahnya.

Pada algoritma tersebut diatas, OPEN adalah senarai (list) yang digunakan untuk menyimpan simpulsimpul yang pernah dibangkitkan dan nilai heuristiknya telah dihitung tapi belum terpilih sebagai simpul terbaik (best node). Dengan kata lain, OPEN berisi simpul-simpul yang masih memiliki peluang (peluangnya masih terbuka) untuk terpilih sebagai simpul terbaik. Sedangkan CLOSED adalah senarai untuk menyimpan simpul-simpul yang sudah pernah dibangkitkan dan sudah pernah terpilih sebagai simpul terbaik. Artinya, CLOSED berisi simpul-simpul tidak mungkin terpilih sebagi simpul terbaik (peluang untuk terpilih sudah tertutup).

|  |  |  |
| --- | --- | --- |
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | 0 | 5 |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 8 | 0 | 4 |
| 7 | 6 | 5 |

Inisial State Goal State

## **II.3.** **Penerapan Strategi dalam Menyelesaikan Game**

**#include <stdio.h>**

**#include <stdlib.h>**

**int difference(int A[][3], int B[][3]);**

**int alter(int A[][3], int B[][3]);**

**int diffup(int A[][3], int B[][3]);**

**int diffdown(int A[][3], int B[][3]);**

**int diffleft(int A[][3], int B[][3]);**

**int diffright(int A[][3], int B[][3]);**

**int minimum(int a, int b, int c, int d);**

**void display(int A[][3]);**

**int main(int argc, char \*argv[])**

**{**

**int A[3][3] = {{2,8,3}, {1,6,4}, {7,0,5}}; // Initial Array (Design Masih simple Belum di Edit)**

**int B[3][3] = {{1,2,3}, {8,0,4}, {7,6,5}};**

**int i, j;**

**int d; // perbedaan antara array yang diinginkan dan array yang akan uji**

**int steps =0 ;**

**printf("\n\n\n\t\t\t===================================\n");**

**printf("\t\t\t\tSELAMAT Datang di GAME\n");**

**printf("\t\t\t Numeric Puzzle\n");**

**printf("\t\t\t===================================\n");**

**getch();**

**system("cls");**

**printf("Klik Enter di Setiap Step Agar Mencapai Goal State : \n");**

**printf("Goal State: \n");**

**display(B);**

**printf("Inial State: \n");**

**display(A);**

**while(1)**

**{**

**d = difference(A, B);**

**if(d==0) { printf("files matched in steps: %d", steps); system("PAUSE"); return 0; }**

**steps++;**

**printf("\nStep: %d \n", steps);**

**alter(A, B); // mengubah aray**

**display(A);**

**system("PAUSE");**

**}**

**system("PAUSE");**

**return 0;**

**}**

**void display(int A[][3])**

**{**

**int i,j;**

**for(i=0;i<3;i++)**

**{**

**for(j=0;j<3;j++)**

**printf("%d ", A[i][j]);**

**printf("\n");**

**}**

**}**

**int difference(int A[][3], int B[][3])**

**{**

**int counter =0 ,i,j;**

**for(i=0;i<3; i++)**

**for( j=0;j<3;j++)**

**if(A[i][j] != B[i][j]) counter++;**

**return counter;**

**}**

**int alter(int A[][3], int B[][3])**

**{**

**int dup, ddown, dleft, dright;**

**int temp, i , j, flag=0, serial=0;**

**char ran[4];**

**dup = diffup(A, B);**

**ddown = diffdown(A, B);**

**dleft = diffleft(A, B);**

**dright = diffright(A, B);**

**printf("%d %d %d %d\n", dright, dleft, dup, ddown);**

**int min = minimum(dup, ddown, dleft, dright);**

**printf("%d\n", min);**

**if (min == dright)**

**ran[serial++] = 'r';**

**if (min == dleft)**

**ran[serial++] = 'l';**

**if (min == dup)**

**ran[serial++] = 'u';**

**if (min == ddown)**

**ran[serial++] = 'd';**

**int sel = rand()%serial;**

**char change = ran[sel];**

**if(change == 'r')**

**{for(i=0;i<3;i++)**

**for (j=0;j<2;j++)**

**if(A[i][j]==0) { A[i][j] = A[i][j+1]; A[i][j+1] = 0; printf("right\n"); return 0; }**

**}**

**else if(change == 'l')**

**{for(i=0;i<3;i++)**

**for (j=1;j<3;j++)**

**if(A[i][j]==0) { A[i][j] = A[i][j-1]; A[i][j-1] = 0; printf("left\n");return 0; }**

**}**

**else if(change == 'u')**

**{for(i=1;i<3;i++)**

**for (j=0;j<3;j++)**

**if(A[i][j]==0) { A[i][j] = A[i-1][j]; A[i-1][j] = 0; printf("up\n"); return 0;}**

**}**

**else if(change == 'd')**

**{for(i=0;i<2;i++)**

**for (j=0;j<3;j++)**

**if(A[i][j]==0) { A[i][j] = A[i+1][j]; A[i+1][j] = 0; printf("down\n"); return 0; }**

**}**

**return 0;**

**}**

**int diffup(int A[][3], int B[][3])**

**{**

**int temp[3][3], i, j;**

**for(i=0;i<3;i++) //menyalin array A dalam array untuk pengujian**

**for (j=0;j<3;j++)**

**temp[i][j] = A[i][j];**

**for(i=1;i<3;i++) // swapping the space in upward direction**

**for (j=0;j<3;j++)**

**if(A[i][j]==0)**

**{**

**temp[i-1][j] = 0;**

**temp[i][j] = A[i-1][j];**

**}**

**return difference(temp, B);**

**}**

**int diffdown(int A[][3], int B[][3])**

**{**

**int temp[3][3], i, j;**

**for(i=0;i<3;i++)**

**for (j=0;j<3;j++)**

**temp[i][j] = A[i][j];**

**for(i=0;i<2;i++) // swapping the space in downward direction**

**for (j=0;j<3;j++)**

**if(A[i][j]==0)**

**{**

**temp[i+1][j] = 0;**

**temp[i][j] = A[i+1][j];**

**}**

**return difference(temp, B);**

**}**

**int diffleft(int A[][3], int B[][3])**

**{**

**int temp[3][3], i, j;**

**for(i=0;i<3;i++)**

**for (j=0;j<3;j++)**

**temp[i][j] = A[i][j];**

**for(i=0;i<3;i++) // swapping the space in the left direction**

**for (j=1;j<3;j++)**

**if(A[i][j]==0)**

**{**

**temp[i][j-1] = 0;**

**temp[i][j] = A[i][j-1];**

**}**

**return difference(temp, B);**

**}**

**int diffright(int A[][3], int B[][3])**

**{**

**int temp[3][3], i, j;**

**for(i=0;i<3;i++)**

**for (j=0;j<3;j++)**

**temp[i][j] = A[i][j];**

**for(i=0;i<3;i++) // swapping the space in the right direction**

**for (j=0;j<2;j++)**

**if(A[i][j]==0)**

**{**

**temp[i][j+1] = 0;**

**temp[i][j] = A[i][j+1];S**

**}**

**return difference(temp, B);**

**}**

**int minimum (int a, int b, int c , int d)**

**{**

**int min = a;**

**if(b<min)**

**min= b;**

**if(c<min)**

**min = c;**

**if(d<min)**

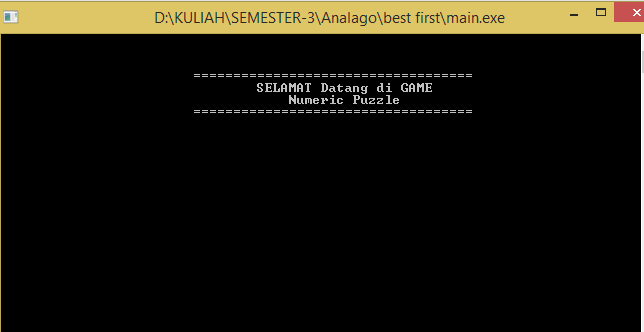
**min = d;**

**return min;**

# **Implementasi**

## **III.1.** **Screenshoot Tampilan Langkah Perlangkah**

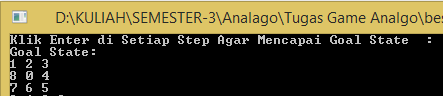
1. **Tampilan Awal Game**



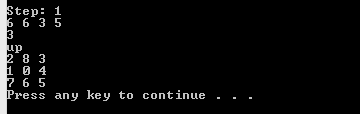
1. **Tampilan Inisial State**



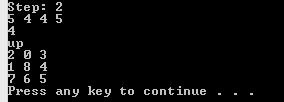
1. **Tampilan Goal State/ Tujuan Menang Puzzle**



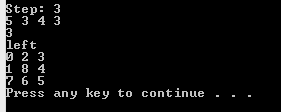
1. **Step 1**



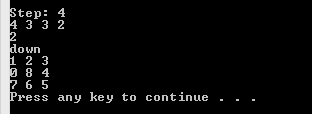
1. **Step 2**



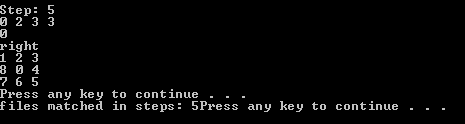
1. **Step 3**



1. **Step 4**



1. **Step ke-5 (Mencapai Goal State)**



## **III.3.** **Algortima dan** **Perhitungan Kompleksitas Strategi Algoritma Penyelesaian Game**

**Procedure** diffup(int A[][3], int B[][3])

**Kamus** :

int temp[3][3], i, j;

**Algoritma**

**For** i=0 **to** i<3 **do**

**for** j=0 **to** j<3 do

temp[i][j] 🡨 A[i][j]

**endfor**

**endfor**

**for** i🡨1 **to** i<3 **do**

**for** j🡨0 **to** j<3 **do**

if(A[i][j] = 0) then

{

temp[i-1][j] 🡨 0;

temp[i][j] 🡨 A[i-1][j];

}

**endif**

**endfor**

**endfor**

difference(temp, B);

**Endprocedure**

**Kompleksitas :**

Omax(n2,n2) = **O(n2)**

**Procedure** diffdown(int A[][3], int B[][3])

**Kamus**

int temp[3][3], i, j;

**Algoritma**

**For** i🡨0 **to** i<3 **do**

**For** j🡨0 **to** j<3 **do**

temp[i][j] = A[i][j];

**EndFor**

**EndFor**

**For** i🡨0 **to** i<2 **do**

**for** j🡨0 **to** j<3 **do**

**if** (A[i][j] = 0 ) **then**

{

temp[i+1][j] 🡨 0;

temp[i][j] 🡨 A[i+1][j];

}

**EndIF**

**EndFor**

**EndFor**

difference(temp, B);

**EndProcedure**.

**Kompleksitas :**

Omax(n2,n2) = **O(n2)**

**Procedure** diffleft(int A[][3], int B[][3])

**Kamus**

int temp[3][3], i, j;

**Algoritma**

**for** i🡨0 **to** i<3 **do**

**for** j🡨0 **to** j<3 **do**

temp[i][j] 🡨 A[i][j];

**EndFor**

**EndFor**

**for** i🡨0 **to** i<3 **do**

**for** j🡨1 **to** j<3 **do**

**if** (A[i][j] = 0) **then**

{

temp[i][j-1] 🡨 0;

temp[i][j] 🡨 A[i][j-1];

}

**EndIF**

**EndFor**

**EndFor**

difference(temp, B);

**EndProcedure**

**Kompleksitas :**

Omax(n2,n2) = **O(n2)**

**Procedure** diffright(int A[][3], int B[][3])

**Kamus**

int temp[3][3], i, j;

**Algoritma**

**For** i🡨0 **to** i<3 **do**

**For** j🡨0 **to** j<3 **do**

temp[i][j] 🡨 A[i][j];

**EndFor**

**EndFor**

**for** i🡨0 **to** i<3 **do**

**for** j🡨0 **to** j<2 **do**

**if**(A[i][j] = 0) **then**

{

temp[i][j+1] 🡨 0;

temp[i][j] 🡨 A[i][j+1];

}

**EndIF**

**EndFor**

**EndFor**

difference(temp, B);

**EndProcedure**

**Kompleksitas :**

Omax(n2,n2) = **O(n2)**

# **KESIMPULAN DAN SARAN**

## **IV.1.** **Kesimpulan**

Pada permainan Numeric Puzzle algoritma yang lebih mangkus dalam pencarian kata adalah algoritma Best First Search. Hal ini dikarenakan pada algoritma BFS dapat menggabungkan keuntungan atau kelebihan dari pencarian Breadth-First Search dan Depth-First Search dan Best First Search selalu mencari apabila *goal state* belum ditemukan, akan dilakukan pemeriksaan pada simpul berikutnya dengan nilai heuristik terbaik pada kedalaman yang sama. Simpul tersebut kemudian dibuka dan diperiksa apakah terdapat *goal state* pada cabang-cabangnya. Bila *goal state* belum ditemukan, akan dilakukan proses yang sama pada simpul berikutnya sampai ditemukannya goal state.

## **IV.2.** **Saran**

*Game Numeric Puzzle* ini masih dapat dikembangkan lebih lanjut sesuai dengan spesifikasi kebutuhan pengguna sistem yang harus dipenuhi dalam mencapai tahap yang lebih tinggi dan kinerja sistem yang lebih baik. Berikut adalah beberapa saran untuk pengembangan lebih lanjut:

1. Dapat mencoba menggunakan algoritma yang lebih baik untuk mencapai hasil yang lebih baik.

2. Diharapkan pada pengembangan selanjutnya tampilan atarmuka/ tampilan game dapat dibuat lebih menarik.