# IMPLEMENTASI DAN ANALISIS ALGORITMA A*(STAR) UNTUK MENENTUKAN JALUR DENGAN MULTIPLE GOAL PADA PERGERAKAN NPC(NON-PLAYABLE CHARACTER)

## *Implementation and Analysis of A* Algorithm for Multiple Goal Pathfinding Used at NPC(Non-Playable Charactrer) Movement*

Pratama Juliantono Taufiq[1], Agung Toto Wibowo, ST., MT.[2], Gia Septiana, SSi., MSc.[3]

[1,2,3]Prodi S1 Teknik Informatika, Fakultas Informatika, Universitas Telkom
pratama.taufiq@gmail.com

**ABSTRAK**

Industri *game* sampai saat ini tidak pernah berkurang peminatnya bahkan semakin meningkat dari waktu ke waktu. Salah satu *genre game* yang banyak dimainkan adalah *turn-based strategy*, yaitu *game* (yang biasanya permainan perang terutama strategi perang) dimana pemain secara bergantian mengatur strateginya untuk mendekati pihak lawan. Selain dimainkan dengan player lain, biasanya *game* dengan *genre* ini pun bisa dimainkan dengan NPC atau biasa disebut *Non- playable character*. Oleh karena itu dibutuhkan NPC yang pintar dan bisa bersaing layaknya manusia yangmemainkannya. Pembuatan NPC yang pintar membutuhkan algoritma pencarian jalur yang bisa memudahkan NPC tersebut mencapai musuh dengan tepat dan cepat.
Untuk menciptakan NPC yang sesuai dibutuhkan algoritma yang bisa menentukan rute yang optimal yang bisa diimplementasikan untuk kasus lebih dari satu karakter. Pada umumnya algoritma A*(A *Star*) sering digunakan pada *game* untuk kasus pencarian jalur.
Penerapan algoritma A*(*Star*) dengan menggunakan nilai heuristik yang didapat dari mengkombinasikan jarak garis lurus antar masing-masing tujuan mampu menyelesaikan kasus *multiple-goal* dengan hasil yang *complete* dan optimal.

**Kata kunci** :algoritma, jalur, tercepat, optimal, *multiple-goal*, *non-playable character*, A*(*Star*), *heuristic*, *mobile game*

**ABSTRACT**

The gaming industry to date has never diminished even further increased demand from time to time . One of the many genres of games played is a turn-based strategy , the game ( which is usually a war , especially a war strategy game ) where players alternately set the strategy for approaching the opponent . In addition to playing with another player, usually the game with this

genre was could be played by the NPC or so-called non-playable character . Therefore, it needs a smart npc that can compete like men who play it . Making the smart NPC requires algorithms that can facilitate the search path to reaches the NPC enemies with precise and fast way. To create the appropriate NPC needed an algorithm that can determine the optimal route that can be implemented for the case of more than one character. In general, the algorithm A * (A Star) is often used in games to the case of search paths.
Implementation A*(Star) algorithm with heuristic point that obtained from combined every single straight line that crossed from each goals felt able to resolve the case of multiple-goal with complete and optimum result.

**Keywords** :*algorithm, path, fastest, optimal, multiple-goal, non-playable character, A*(Star), heuristic, mobile game*

## 1. PENDAHULUAN

Industri *game* sampai saat ini tidak pernah berkurang peminatnya bahkan semakin meningkat dari waktu ke waktu. Pengembangan dan implementasi aplikasi *game* pun sekarang bukan hanya terbatas pada *game console* yang umum seperti Playstation atau X-Box saja, melainkan pada perangkat kecil seperti *smartphone*.

Salah satu *genre* atau jenis *game* yang banyak dimainkan dan diimplementasikan pada *smartphone* adalah *turn-based strategy*, yaitu permainan yang biasanya berlatar belakang peperangan, dimana pemain secara bergantian mengatur strateginya untuk mendekati dan menyerang pihak lawan. Salah satu contoh permainan dengan *genre turn-based strategy* adalah permainan catur yang mana masing-masing pemain bergantian memajukan bidaknya untuk menaklukan lawan. Selain dimainkan dengan player lain, biasanya *game* dengan *genre* ini pun bisa dimainkan melawan *Non-playable character* atau biasa disebut NPC, yaitu karakter yang digerakkan secara otomatis oleh komputer. Kondisi tersebut memunculkan kebutuhan akan NPC yang pintar dan bisa bersaing layaknya manusia yang memainkannya karena *turn-based strategy game* merupakan permainan dimana karakter yang dimainkan bergerak mendekati lawan dengan berbagai strategi atau perhitungan untuk mendapatkan suatu keputusan. Semakin pintar NPC yang dilawan oleh pemain, maka pemain akan terus mendapatkan tantangan dalam bermain *game* tersebut.

Untuk mendapatkan NPC yang pintar dibutuhkan algoritma yang bisa memudahkan NPC tersebut mendekati dan menyerang musuh dengan tepat dan cepat. Salah satu algoritma yang di perlukan adalah algoritma untuk menentukan jalur dengan lebih dari satu tujuan yang optimal agar NPC bisa sampai ke karakter-karakter lawan, dan A*(*Star*) merupakan algoritma yang cocok dipakai dalam kasus pencarian jalur dengan *multiple goal* karena algoritma A* merupakan algoritma *Best First Search* perpaduan dari *Uniform Cost Search* yang memilih jarak paling kecil dari simpul awal ke simpul berikutnya dan *Greedy-Best First Search* yang menggunakan nilai heuristik atau nilai perkiraan untuk menentukan simpul berikutnya yang akan di lewati, sehingga hasil pencarian jalur dengan algoritma A* memiliki karakteristik yang *complete* dan optimal[6].

Di penelitian ini akan menerapkan algoritma A* pada NPC simulasi *game* agar NPC bisa menentukan jalur secara otomatis dengan lebih dari satu tujuan karena umumnya implementasi algoritma ini hanya untuk satu tujuan. Seperti pada implementasi yang dilakukan oleh Riwinto dan Alfian pada permainan *Funny English* yang menerapkan algoritma ini pada NPC-nya[1]. Agar

pencarian jalur oleh NPC tersebut bisa mendapatkan jalur yang optimal maka perlu memodifikasi penghitungan nilai heuristik. Beberapa modifikasi yang telah dilakukan pada penelitian sebelumnya adalah *Sum Heuristic*, *Progress Heuristic*, dan *Marginal Utility Heuristic*. Untuk yang pertama adalah *Sum heuristic* yaitu fungsi yang membandingkan jumlah jarak dari titik sekarang ke masing-masing titik berikutnya, fungsi heuristik ini memiliki kelemahan apabila sebuah algoritma yang bergantung hanya pada jumlah heuristik saja maka tidak akan memiliki informasi tentang node yang akan dipilih untuk di ekspansi. Kedua adalah *Progress Heuristic* yaitu fungsi yang menghitung jumlah tujuan yang searah dengan progress yang telah di buat. Dan yang ketiga adalah *Marginal-utility Heuristic*, yaitu fungsi yang menghitung keuntungan dari masing masing tujuan dan biaya perkiraannya[3]. Namun modifikasi fungsi heuristik yang penulis lakukan dalam penelitian ini berbeda dari fungsi-fungsi tersebut, penulis memodifikasi dengan menambahkan kombinasi minimum jarak garis lurus antar tujuan. Dengan modifikasi tersebut, A* yang semula hanya digunakan untuk menentukan jalur dengan satu tujuan saja juga bisa mendapatkan jalur optimal dengan lebih dari satu tujuan.

## 2. DASAR TEORI

## 2.1 Algoritma A*

### 2.1.1 Definisi

A* adalah algoritma *Best First Search* yang merupakan perpaduan *Uniform Cost Search* yang memilih jarak paling kecil dari simpul awal ke simpul berikutnya dan *Greedy-Best First Search* yang menggunakan nilai heuristik atau nilai perkiraan untuk menentukan simpul berikutnya. Oleh karena itu rumus dari algoritma ini adalah :

$$f(n) = g(n) + h(n) \quad \text{.....(1)}$$

dimana,

$f(n)$ : [illegible]

$g(n)$ : [illegible]

$h(n)$ : [illegible]

Algoritma A* ini akan menemukan rute yang *complete* (selalu menemukan solusi jika ada) dan optimal.[6]

## 2.2 Multiple Destination

*Multiple destination* terdiri dari dua kata dalam bahasa inggris yaitu, d*estination* yang dalam bahasa Indonesia adalah destinasi atau tujuan yang memiliki arti sebuah tempat yang akan di tuju atau di arahkan dan *multiple* yang berarti banyak atau lebih dari satu. Jadi, *multiple destination* diartikan lebih dari satu tempat yang akan di tuju. Adapun Psuedo code untuk kasus multiple destination adalah sebagai berikut,

```
function A*(start, goal)
  open_list = set containing start
  closed_list = empty set
  start.g = 0
  start.f = start.g + heuristic(start, goal)
  while open_list is not empty
    current = open_list element with lowest f cost
```

```
// cek setiap goal
if current = goal //jika current termasuk salah satu dari goal node
   goal_list[] = current // simpan goal node ke dalam array

   if goal_list.length == goal.length
      return cunstruct_path(goal_list) //path found
   if goal_list.length != goal.length //jika jml goal node tdk sama
      construct_path(goal)

remove current from open_list
add current to closed_list
for each neighbor in neighbors(current)
  if neighbor not in closed_list
   if neighbor is not in open_list
    add neighbor to open_list

    if neighbor == goal //jika neighbor = salah satu goal
    //tambahkan status neighbor menjadi goal

   else
    openneighbor = neighbor in open_list if
    neighbor.g < openneighbor.g
    openneighbor.g = neighbor.g
    openneighbor.parent = neighbor.parent
return false // no path exists
```

Algoritma untuk pencarian jalur dengan *multiple destination* hampir sama dengan pencarian dengan Algoritma A* biasa, yang membedakan adalah, setelah menemukan salah satu *goal* program tidak langsung berhenti melainkan menyimpan *goal* tadi ke dalam suatu *array* kemudian mencari *goal* berikutnya hingga semua *goal* didapat.

## 2.3 Fungsi Heuristik

Heuristic berasal dari kata Yunani ‘heuriskein’ berarti mencari atau menemukan. Fungsi h(n) memperkirakan biaya dari jalur termurah dari node n ke node tujuan. Heuristik ini memiliki peran yang sangat penting untuk mengontrol proses pencarian pada algoritma A*. Dengan heuristik, node yang diproses pada algoritma A* tidak akan sebanyak node jika kita menggunakan algoritma Djikstra. Beberapa macam kondisi nilai h(n) beserta pengaruhnya terhadap proses dalam algoritma A* [2]:

- Jika h(n) = 0, maka hanya g(n) yang berpengaruh, dan algoritma A* berubah menjadi algoritma djikstra yang menjamin dapat menemukan jarak terpendek.
- Jika h(n) selalu lebih rendah (atau sama dengan) dari biaya sesungguhnya (node n ke node tujuan), maka A* akan menemukan jarak terpendek. Semakin rendah nila h(n), maka

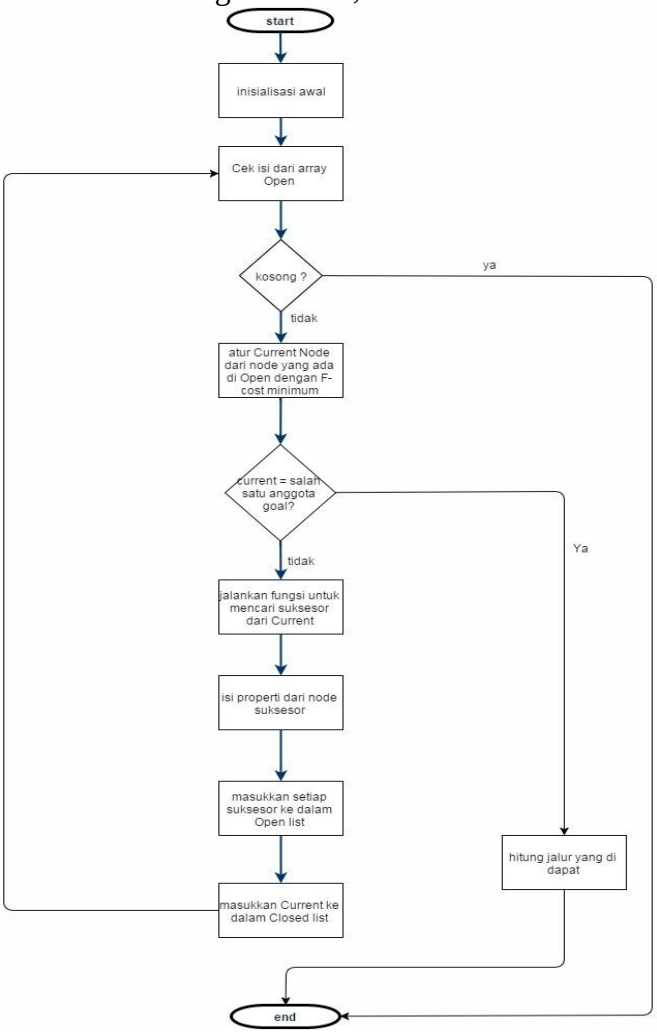
semakin banyak node yang diproses, semakin lama pula proses pencarian jalur terpendeknya.

- Jika h(n) sama persis nilainya dengan biaya sesungguhnya dari n menuju node tujuan, maka A* akan berjalan cepat. Karena, A* hanya akan memproses node-node pada jalur terbaik/terpendek, tanpa memperdulikan node-node lain. • Terkadang, h(n) didapati lebih besar dari biaya sesungguhnya. Maka A* tidak dapat menjamin untuk menemukan jarak terpendek.
- Di lain kasus, jika h(n) teramat besar biayanya dari jarak n ke node tujuan, maka hanya h(n) yang berpengaruh. Dan A* akan berubah menjadi BFS (Best First Search).

## 3. PEMBAHASAN

### 3.1 A* Pada Player

Karena pergerakan karakter pemain tergantung pada pilihan dari pemain itu sendiri, maka digunakan algoritma A* biasa yang hanya memiliki satu tujuan, yaitu titik asal adalah tempat karakter berada dan titik akhir adalah lokasi pada *map* yang dipilih oleh pemain. Adapun flowchartdari algoritma A* ini adalah sebagai berikut,




Gambar 3.1 Flow Chart A*

3.1.1 Inisialisasi Awal

Pada inisialisasi awal ini dilakukan pemberian nilai awal pada setiap variable yang digunakan pada algoritma tersebut, diantaranya adalah

- Array Open yang diisi dengan *start node*
- Array Closed berupa array kosong

3.1.2 Isi Properti Dari Node Suksesor

Setiap node memiliki properti sebagai indentitas node tersebut. Adapun properti yang dimiliki oleh tiap node adalah sebagai berikut,

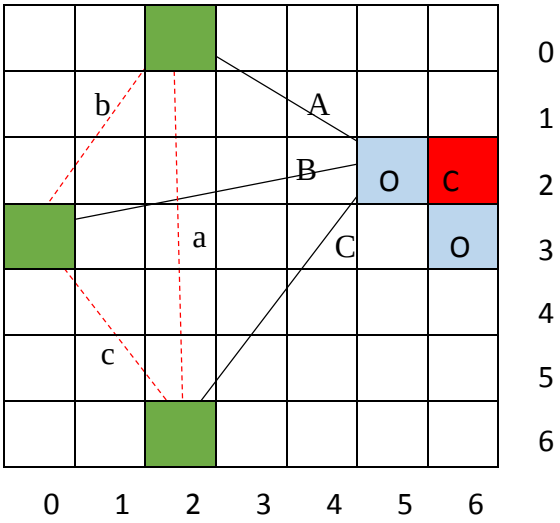
- Parent yang merupakan bestNode sebelumnya
- Value yang merupakan nilai dari node tersebut yang nilainya diambil dari penghitungan alamat koordinat node tersebut
- X, alamat node pada sumbu x
- Y, alamat node pada sumbu y
- f, nilai *f-cost* yang dihitung dari node tersebut ke node tujuan dengan menggabungkan nilai heuristik dan nilai g
- g, nilai yang di dapat dari jarak node tersebut dengan node asal melalui parent sebelumnya

**3.2 Perhitungan Fungsi Heuristik**

Penghitungan nilai heuristik untuk kasus multiple destination sedikit berbeda dengan penghitungan pada algoritma A* biasanya yang memakai Manhattan Distance, Diagonal Distance, ataupun Euclidean Distance. Penghitungan nilai heuristik untuk kasus ini dapat dilihat pada gambar 3.2 di bawah ini,


Gambar 3.2 Flow Chart Pencarian Nilai Heuristik

Dimulai dengan menghitung jarak antar goal dengan metode Euclidean Distance, kemudian masing-masing jarak tersebut ditambahkan dengan metode permutasi dari goal-goal yang ada sehingga menghasilkan susunan arah yang memiliki total jarak minimum. Setelah mendapatkan masing-masing susunannya sesuai dengna masing-masing goal, ditambahkan dengan jarak garis lurus dari node suksesor ke masing-masing goal. Setelah itu dipilih jarak yang paling kecil untuk di jadikan nilai heuristik. Untuk menentukan mana jalur yang selanjutkan akan dipilih adalah jalur yang berada di array Open dan memiliki f-cost minimum. Penghitungan f-cost dilakukan denganmengkombinasikan jarak garis lurus seperti yang terlihat pada gambar 3.3. Kombinasi minimumlah yang menjadi f-cost dari node tersebut, seperti contoh pada gambar, garis A yang memiliki panjang 3,606 di tambahkan dengan kombinasi minimum dari jarak masing-masing node tujuan yang dimulai dari tujuan yang dituju oleh garis A yaitu garis b ditambah c dengan total 7,212. Sehingga f-cost untuk node pada koordinat (5,2) adalah garis A + garis b + garis c dengan total 10,818.




Gambar 3.3 Penghitungan F-Cost

## 4. PENGUJIAN DAN ANALISIS

4.1 Menentukan Tujuan Dari Garis Lurus Terdekat

Dengan menentukan tujuan awal menggunakan garis lurus yang terdekat dengan tujuan, maka didapkan hasil seperti berikut

Gambar 4.1 Penentuan Tujaun Awal

Pada gambar 4.1 terlihat bahwa tujuan terdekat berjarak dua langkah sehingga NPC akan bergerak ke goal tersebut dengan menggunakan algoritma A* yang fungsi heuristiknya didapat dari perhitungan *Euclidean Distance* antara titik awal dan tujuan, sehingga menghasilkan jalur seperti gambar di bawah ini


Gambar 4.2 Jalur yang di tempuh ke tujuan pertama

Pada gambar 4.2 jalur yang di tempuh memiliki biaya sebesar 14 langkah untuk sampai di lokasi karakter pemain tersebut. Setelah selesai dari tujuan pertama tersebut, NPC memilih lagi tujuan kedua yaitu seperti terlihat pada gambar 4.3 di bawah ini

Gambar 4.3 Tujuan Kedua

Jarak garis lurus pada gambar adalah 5,09902 yang mana lebih kecil dibandingkan jarak ke tujuan yang lain yaitu sebesar 7,07107. Oleh karena itu jalur berikutnya yang akan dilalui NPC adalah sebagai berikut


Gambar 4.4 Jalur Tujuan Kedua

Jalur yang di tempuh kali ini memiliki bobot sebesar 16 langkah untuk sampai di lokasi tujuan kedua. Begitupun akhirnya ke tujuan ke tiga seperti yang terlihat pada gambar 4.4 di bawah ini

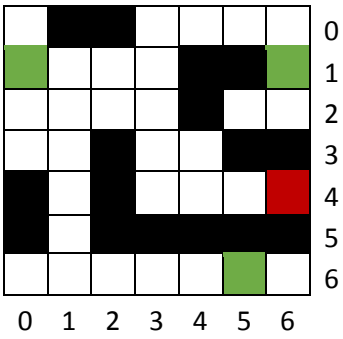

Gambar 4.5 Jalur Tujuan Ketiga

Bobot yang ditempuh pada gambar di atas adalah sebesar delapan langkah untuk sampai di lokasi terakhir.

Dari scenario di atas menghasilkan bobot total sebesar 38 langkah untuk mencapai keseluruhan goal.

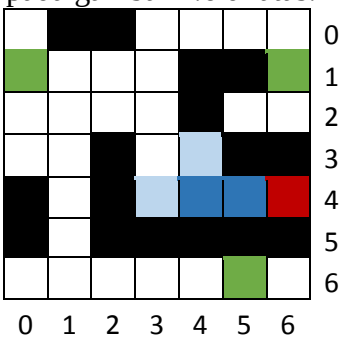
4.2 Menentukan Jalur Dengan Modifikasi Fungsi Heuristik

Disini digunakan fungsi heuristik yang menghasilkan nilai dari perhitungan kombinasi antar tujuan. Berikut pengujian algoritmanya pada scenario ini



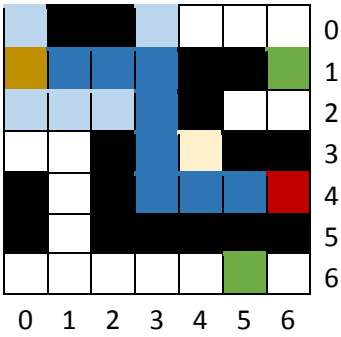

Gambar 4.6 *Layout Map*

Inisialisasi dilakukan dengan menjadikan titik awal yang di tandai kotak berwarna merah pada gambar 4.6 di atas.



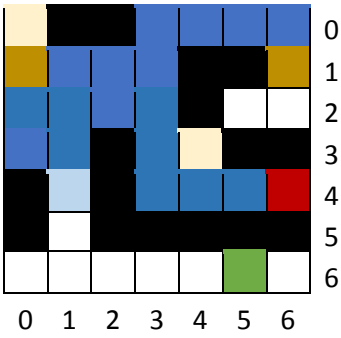

Seperti sebelumnya maka bukalah jalur hingga menemukan goal
OPEN = {(4,3),(3,4)}
CLOSED = {(4,4),(5,4),(6,4)}
f(4,3) = 17,26 ; f(3,4) = 16,93

Gambar 4.7 Pencarian Jalur



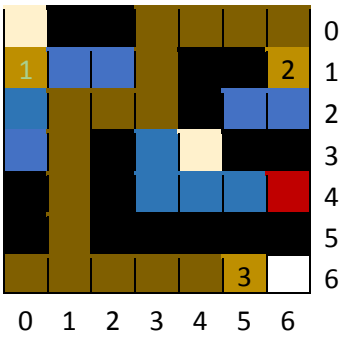

Menemukan goal pertama, namun proses belum berhenti karena semua goal belum terlewati

Gambar 4.8 Menemukan Salah Satu Goal




Goal kedua telah ditemukan, namun karena jumlah total goal adalah 3 maka proses masih di teruskan
Jika jalur yang dilalui harus kembali, maka dibandingkan isi dari goal list pada node suksesor Dengan parentnya

Gambar 4.9 Menemukan Goal Kedua




Ketika menemukan Goal terakhir balikkan seluruh path berdasar pada parent nodenya

Gambar 4.10 Semua Goal Telah Ditemukan

Dari pencarian menggunakan cara di atas dengan jalur sesuai urutan goal yang tertera pada gambar 4.10 total bobotnya sebesar 33 langkah.

## 5. KESIMPULAN

Dengan menggunakan kombinasi jarak antar goal dari masing-masing tujuan, maka algoritma A*(*Star*) dapat menghasilkan jalur yang *complete* dan optimal pada kasus *Multiple Destination*. Sehingga NPC dapat menentukank jalur mana yangakan di lalui untuk mencapai seluruh tujuannya. Penunjang dari hasil tersebut adalah nilai heuristik yang didapat akibat pengaruh dari tujuan lain, bukan seperti *Euclidean* atau *Manhattan Distance* yang hanya mengandalkan acuan satu tujuan.

## DAFTAR PUSTAKA

[1] Alfian[1], Riwinoto[2,] , Implementasi*Pathfinding*denganAlgoritma A* pada*Game Funny English*Menggunakan Unity 3D Berbasis Graf. Batam :Politeknik Negeri Batam

[2] Ayuningtyas, Vina.2006.Aplikasi Penelusuran Rute Angkutan Kota di Kota Bandung Dengan Menggunakan Implementasi GIS dan Algoritma A*.Bandung: Sekolah Tinggi Teknologi Telkom

[3] Davidov, Dmitry dan Markovitch, Shaul.2006. *Multiple-Goal Heuristic Search.* Computer Science Department. Technion, Haifa 32000, Israel.

[4] Mosaud Nosrati, Ronak Karini, dan Hojat Allah Hasanvad.2012. *Investigation of the *(Star) Search Algorithms: Characteristics, Methods and Approach*. Islamic Azad University, Shaneh, Iran.

[5] Sukiman, Jeffrey.2012. *Simulasi Pencarian Shortest Distance Path*. Teknik Informatika, STMIK IBBI, Medan, Indonesia.

[6] Suyanto. 2007. *Artificial Intelligence Searching, Reasoning, Planning dan Learning*. Bandung, Indonesia. Juni 2007