Implementasi Graf (Path Finding) dengan Algoritma A* sebagai Basis Artificial Intellegence pada Musuh yang Mengutamakan Team Play

Davin Prasetya 13514003[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13514003@s.itb.ac.id

***Abstract*—Game digital sudah jauh lebih berkembang dari zaman dahulu. Saking banyaknya game yang beredar, konsumen semakin mengenal pola AI dari musuh musuh yang ada. Lama kelamaan hal ini akan berakibat pada kebosanan karena mayoritas musuh di suatu game menggunakan AI yang individualistis dalam artian mengandalkan parameter diri sendiri dan hanya mengandalkan koordinasi skill yang dapat diciptakan oleh si konsumen itu sendiri secara manual. Dalam makalah ini akan dibahas sebuah solusi dalam mengembangkan AI musuh dengan sistem mob yakni AI yang bekerja tidak sendiri melainkan bekerja sama / membuat team play dengan musuh musuh lainnya sehingga konsumen dapat merasakan suatu pengalaman bertanding melawan musuh yang dapat berkoordinasi layak nya melawan player lainnya meski dikontrol oleh AI. Pengembangan ini dapat dilakukan berdasarkan implementasi graf untuk path finding dengan algoritma A*.**

Keywords : algoritma A star, enemy teamplay, game AI, multi enemy AI

I. PENDAHULUAN

Game bermodel Role-Playing Game (RPG) merupakan suatu bentuk game yang sudah umum di masyarakat dan dimainkan juga oleh seluruh umur. Di dalam game ini, banyak faktor yang membuat game ini menarik, seperti gameplay, cerita, grafik, visual dan lain lain.

Non Player Character (NPC) juga merupakan suatu point yang penting dalam suatu game berbasis RPG mengingat jumlah karakter dari satu cerita dalam game RPG sangat banyak dan tidak mungkin semua player dapat dimainkan. Diantara NPC tersebut, ada yang merupakan sebuah karakter antagonis atau musuh yang dilawan. Musuh ini pastilah dibuat dengan suatu artificial intellegence (AI) yang menjadi salah satu poin vital dalam keberhasilan game ini dalam menarik pemainnya ke dalam dunia nya. Namun, banyak game memiliki pola serangan musuh yang dapat mudahnya terbaca oleh pemain akibat dari AI yang dibuat kurang matang. Mayoritas dari game ini menerapkan sistem musuh yang mengandalkan individualitas si musuh tersebut.

Dalam makalah ini, akan dibahas suatu konsep game yang mengutamakan kerja tim musuh-musuh dalam game, tidak hanya individualitas musuh yang biasanya hanya mengandalkan paramaternya belaka atau hanya kerja tim dari karakter protagonis yang pemain mainkan. Penyusunan teamwork sebagai basis AI untuk musuh ini dapat membuat game lebih menarik dan pola AI yang lebih sulit dibaca, namun sangat memberatkan komputansi. Dalam menyusun sebuah teamwork AI untuk si musuh ini, ada banyak faktor yang harus dipertimbangan dalam mengambil keputusan dan faktor faktor yang harus dapat dihindari oleh si AI ini, ibaratnya seperti menyusun kartu dengan berbagai macam efek dan skill dalam satu set musuh dengan tetap mempertahankan relasi skill mereka sehingga kombinasi kartu yang disusun tersebut dapat berjalan secara efektif meski dijalankan oleh AI. Akibatnya, komputansi akan menjadi sangat berat dan untuk mengatasinya diperlukan suatu algoritma yang dapat mempermudah kerja komputasi program dalam menemukan jawaban/decision.

Karena itu, melihat dari faktor AI yang digunakan mencangkup faktor benefit dan juga faktor obstacle, digunakanlah algoritma A* yang dapat mengatasi kedua hal tersebut.

II. ALGORITMA A*

*A. Penjelasan Singkat Algoritma A**

Algoritma A* adalah algoritma yang dipakai dalam pemrosesan graf secara traversal dan path finding. Keunggulan dari algoritma ini ada pada fleksibilitasnya, keefektifannya dalam mentraversal node, dan dapat digunakan dalam konteks yang luas. Algoritma ini dipelopori oleh Peter Hart, Nils Nilsson, dan Bertram Raphael pada tahun 1968 sebagai pengembangan dari algoritma Dijkstra.

*B. Cara Kerja Algoritma A**

Dalam Algoritma A*, ia menggabungkan algoritma Djikstra dan algoritma Greedy Best-First-Search dengan menambahkan suatu Heuristik. Secara lebih jelasnya, ia pertama mencari sebuah node yang berkemungkinan

besar mencapai goal. Bersamaan dengan itu, ia mencatat juga jarak yang sudah ia tempuh ke suatu fungsi untuk menentukan nilai dan menyimpan pula sebuah priority queue dari open set. Open set adalah node yang perlu untuk di proses secara traversal. Seiringan dengan berjalannya dari node sebelumnya ke salah satu node, queue dengan nilai terkecil akan dihapus, lalu ditambahkan di queue tersebut dengan nilai elemen tetangganya dan mengupdate nilai masing-masing queue sesuai dengan perubahan jarak yang ditempuh tersebut. Proses traversal akan berhenti ketika goal node/ node tujuan yang diinginkan memiliki nilai yang lebih kecil dari semua node di priority queue atau saat queue sudah kosong/empty.

C. Heuristik

Fungsi penentu nilai di algoritma A* dapat bermacam-macam sesuai dengan keperluan program. Hal ini disebut Heuristik. Heuristik adalah fungsi perkiraan/estimasi yang menentukan nilai minimum dari satu node untuk sampai ke goal. Dalam konteks paling sederhana, fungsi yang dipakai sebagai heuristik adalah

$f(n) = g(n) + h(n)$

dengan g(n) adalah fungsi jarak yang ditempuh dari initial node, h(n) adalah estimasi nilai yang diperlukan untuk sampai ke goal, dan f(n) merupakan heuristik nilai.

Heuristik dapat digunakan dalam mengontrol pergerakan A*, berikut sifat dari heuristik sebagai fungsi yang mengontrol jalannya program

1.h(n) = 0, fungsi heuristik f(n) akan dikontrol oleh hanya 1 parameter yaitu g(n) yang mengakibatkan algoritma A* sama seperti Dijikstra.

2.h(n) selalu lebih rendah atau sama dengan nilai yang diperlukan untuk pindah dari node n ke goal, maka A* dipastikan sampai ke goal dengan nilai terpendek. Semakin rendah nilai h(n) maka perjalanan node A* makin tersebar.

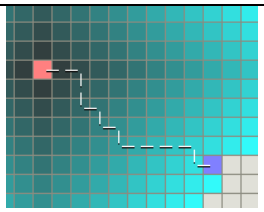
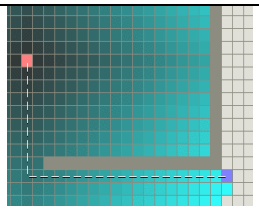
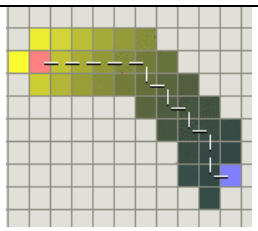
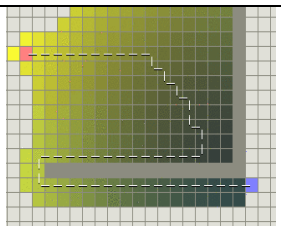
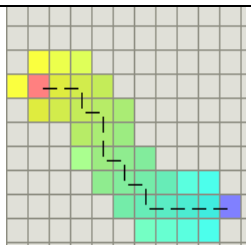
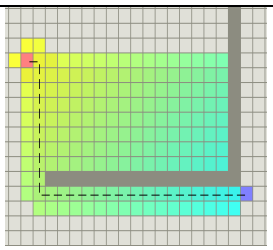
3. h(n) sama dengan nilai yang diperlukan untuk pindah dari node n ke goal, A* tidak akan tersebar dan sampai ke goal dengan nilai terpendek secara sangat cepat. Hal ini yang paling diharapkan dari algoritma A*

4. h(n) lebih besar dari nilai yang diperlukan untuk pindah dari node n ke goal, maka algoritma A* belum tentu menemukan shortest path, tetapi bisa lebih cepat.

5. h(n) jauh lebih besar dari g(n), algoritma A* akan menjadi sama seperti algoritma Greedy Best-First-Search diakibatkan oleh nilai g(n) yang diabaikan.

Dari ke lima sifat tersebut, pemakai algoritma dapat menyusun sifat mana yang ingin digunakan sebagai path finder, nilai h(n) yang besar yang mengakibatkan perjalananan traversal node dari initial node ke goal node tidak mengikuti alur shortest path tetapi dapat meningkatkan kecepatan traversal, atau nilai h(n) yang kecil sehingga node berada pada akurasi yang baik dalam mengikuti shortest path tetapi membutuhkan waktu yang lama dalam melakukan traversalnya. Hal ini harus dipertimbangkan karena meski nilai heuristik yang diharapkan adalah nilai h(n) yang sama persis dengan nilai yang diperlukan untuk pindah dari node n ke goal untuk mencapai shortest path tanpa menurunkan kecepatan traversal, nilai ini susah untuk tercapai.

D. Perbandingan Algoritma A dengan Algoritma Pathfinding lainnya*

	Free Obstacle	Obstacle
Djikstra	 Fig 1.Djikstra - Free	 Fig 2. Djikstra - Obstacle
Greedy	 Fig 3. Greedy Best-First-Search Free	 Fig 4. Greedy Best-First-Search Obstacle
A*	 Fig 5. A* Free	 Fig 6. A* Obstacle

Tabel I. Rute Traversal Ketiga Algoritma
Source :
http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

III. AI MUSUH BERBASIS TEAM PLAY

A. Penjelasan Singkat

Tujuan yang ingin dicapai dalam AI musuh ini adalah musuh yang dapat bekerja sama dengan musuh lainnya sehingga pemain dapat merasakan pengalaman seperti melawan pemain lawannya / Player Versus Player (PVP) padahal yang ia lawan adalah sebuah AI / Player Versus Enemy (PVE). Faktor pengambilan decision disini sangat penting untuk variasi kombinasi dari musuh sehingga pemain tidak dapat menemukan suatu pola secara keseluruhan. Faktor yang diambil berupa faktor dari karakter pemain/player, faktor diri sendiri dan faktor musuh yang lain dalam suatu arena.

B. Faktor Penggerak

Faktor penggerak AI seperti yang telah disebutkan di sub bab sebelumnya terbagi menjadi 3, faktor dari karakter pemain/player, faktor diri sendiri dan faktor musuh yang lain dalam suatu arena. Berikut rinciannya

1. Faktor pemain/player
   a) Parameter statik pemain (serangan/pertahanan)
   b) Parameter dinamis pemain (hp/mp)
   c) Kelemahan
   d) Skill set
   e) Status peningkat/buff dan penurun/debuff
   f) Jumlah
   g) Aksi sebelumnya
2. Faktor diri sendiri
   a) Parameter dinamis diri sendiri (hp/mp)
   b) Skill set
   c) Status peningkat/buff dan penurun/debuff
   d) Perbandingan parameter dengan pemain
   e) Perbandingan parameter dengan enemy lain
   f) Tingkat kesulitan game
   g) Progress game
3. Faktor musuh lain
   a) Kesamaan jenis dengan musuh utama
   b) Kombinasi skill set yang dapat diperoleh
   c) Status peningkat/buff dan penurun/debuff
   d) Jumlah
   e) Posisi dari pemain
   f) Posisi dari musuh utama

Dari jumlah faktor secara kasar diatas, sudah didapat kombinasi gerakan yang sangat banyak (7*7*6 = 294), dan lebih buruknya lagi, jumlah dari setiap faktor tidak hanya 1,skill set bisa dapat sangat banyak, dari faktor parameter dinamis juga bermacam,seperti contoh saat healt point (HP) >=60%, 30%<= HP < 60%, dan HP <30% dan faktor faktor lainnya yang dapat bercabang menjadi banyak. Karena hal demikian, sebenarnya yang diproses menjadi sesuatu faktor matriks bukanlah hanya 2 dimensi melainkan 3 dimensi faktor set. Karena itu diperlukanlah suatu algoritma yang dapat mencari path namun juga dapat membatasi perintah perintah yang dilarang untuk dilakukan, seperti saat musuh lain tidak terkena efek penurun/debuff ia tidak disembuhkan dari penyakit, atau saat hp musuh lainnya masih full ia tidak di heal.

*C. Implementasi Algoritma A**

Dari faktor faktor yang telah disebutkan sebelumnya, bukan tidak mungkin untuk melakukan mapping decision node dengan cara pemberian value setiap tindakan. Value dari node dapat berupa kombinasi aritmetika dari berbagai faktor. Penyusunan value ini merupakan sesuatu yang vital untuk mengimplementasikan algoritma A* dan sudah mencakup semua faktor diatas dengan pengelompokan yang rapi. Bentuk berupa 3d faktor set pun bukanlah suatu masalah bagi algoritma A* karena yang ia lihat bukanlah matriks melainkan hubungan/path dari node node yang ada, sehingga hal demikian hanya memberatkan proses tetapi masih dapat terselesaikan. Goal node yang diinginkan adalah suatu value dengan nilai yang diset tertentu secara random dengan range value yang disesuaikan dengan difficulty. Sebagai contoh goal yang diinginkan adalah value 4 yang menandakan gerakan pada pencarian path kali ini akan menghasilkan suatu aksi yang vital dan berbahaya. Goal diset secara random agar terdapat variasi dari gerakan dan ada suatu goal yang dituju. Dari path yang program jalankan hingga suatu keputusan goal itulah yang menjadi landasan gerakan si musuh dengan cara mengambil nilai yang ia lewati sebagai suatu masukan menjadi suatu index dari list of tindakan yang mungkin dilakukan oleh si musuh utama.

D. Pertimbangan Penting Menentukan Heuristik

Dalam menentukan heuristik, selain faktor penggerak, perlu beberapa hal yang harus diperhatikan seperti skala nilai, tujuan dari algoritma A*, bentuk grid yang dimungkinkan,

1. Skala
   Penentuan f(n),h(n), dan g(n) tidak boleh sampai berbeda satuan, bila kesamaan skala ini dilanggar, dapat berakibat pada pengambilan path yang buruk dan memperlambat kerja komputasi

2. Tujuan
   Seperti yang sudah dijelaskan di bab sebelumnya, heuristik dapat menentukan banyak karakteristik, untuk program bermodel seperti yang diinginkan di makalah ini, maka tindakan yang diperlukan adalah kecepatan yang pasti dengan mengurangkan akurasi, yaitu dengan mengambil pilihan h(n) lebih besar dari nilai yang diperlukan untuk pindah dari node n ke node goal.

3. Grid yang mungkin untuk mempercepat algoritma
   Ada beberapa grid yang dapat mempercepat pengambilan keputusan Heuristik, yaitu
   i) grid persegi tanpa obstacle (square grid) dengan 4 kemungkinan jalan, menggunakan Manhattan Distance (L_1)

```
function heuristic(node) =
  dx = abs(node.x - goal.x)
  dy = abs(node.y - goal.y)
  return D * (dx + dy)
```

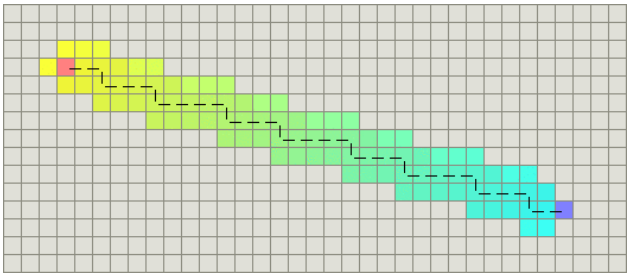


Fig 7. Square Grid 4 Kemungkinan Jalan
Source : http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

ii) square grid dengan 8 kemungkinan jalan, menggunakan Diagonal Distance (L_∞)

```
function heuristic(node) =
  dx = abs(node.x - goal.x)
  dy = abs(node.y - goal.y)
  return D * (dx + dy) + (D2 - 2 * D) * min(dx,
dy)
```

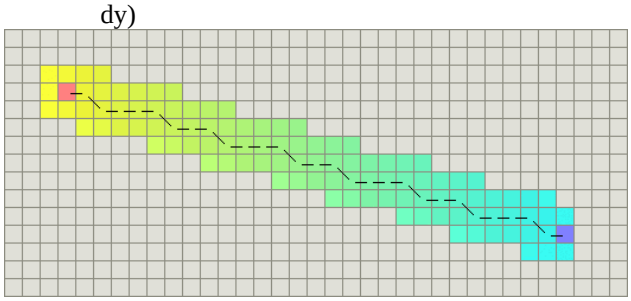


Fig 8. Square Grid 8 Kemungkinan Jalan
Source : http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

iii) square grid dengan semua kemungkinan jalan, menggunakan Euclidean Distance

```
function heuristic(node) =
  dx = abs(node.x - goal.x)
  dy = abs(node.y - goal.y)
  return D * sqrt(dx * dx + dy * dy)
```

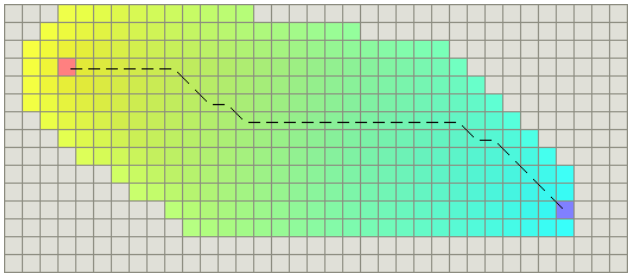


Fig 9. Square Grid Semua Kemungkinan Jalan
Source : http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

Untuk kasus ini , ada kemungkinan bermasalah dalam algoritma A* karena fungsi nilai g(x) akan tidak sesuai dengan fungsi heuristik h(x). Tetap akan didapat shortest path untuk jalan tetapi A* akan membutuhkan waktu yang lebih banyak.

iv) hexagon grid yang memungkinkan 6 kemungkinan jalan, menggunakan Manhattan Distance untuk hexagonal grid

```
function cube_distance(a, b):
  return max(abs(a.x - b.x), abs(a.y - b.y),
abs(a.z - b.z))
```

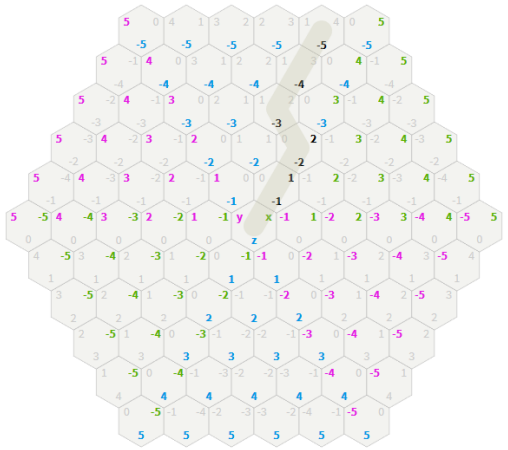


Fig 10. Hexagon Grid 6 Kemungkinan Jalan
Source : http://www.redblobgames.com/grids/hexagons/#distances

E. Permasalahan

Permasalahan utama adalah waktu komputansi, mengingat jumlah dari decision matrix yang sangat besar, tetapi diperlukan waktu yang sangat cepat untuk mengambil keputusan merupakan hal yang sangat berat karena apabila proses pengambilan keputusan/loading tersebut lama, pemain akan lebih malas untuk memainkannya. Lebih lagi untuk pengimplementasian algoritma ini dalam sebuah game real time / action dimana perubahan state dengan cepat terjadi.

Permasalahan utama yang kedua adalah bagaimana menyusun heuristik dari program ini sehingga menjadi sesuatu yang solid dan seimbang. Keseimbangan / Balance dalam game adalah sesuatu yang sangat penting, bila game tidak seimbang, maka pemain dapat dengan cepat bosan untuk memainkan game tersebut atau kasus lainnya adalah ketidak seimbangan itu di eksploitasi sehingga tujuan dari pembuatan game yang dirangkai dari awal tidak tersalurkan dengan baik. Akibat dari hal ini juga adalah munculnya variasi pemain/player yang sedikit sehingga rancangan awal game yang sudah disusun sedemikian rupa dengan sulit tidak terpakai oleh pemain dengan efektif.

IV. KESIMPULAN

1. Algoritma A* merupakan solusi yang baik dalam menyelesaikan kompleksitas path dalam representasi node value AI musuh karena mencakup semua kasus.
2. Nilai Heuristik yang digunakan dalam fungsi harus jelas dan baik disusun karena keoptimalan path dan pemilihan decision yang baik sangat bergantung pada fungsi Heuristik.
3. Kasus AI musuh berbasis team play merupakan sesuatu yang berat untuk diimplementasikan secara nyata tetapi bukan berarti mustahil untuk direalisasikan

REFERENSI

[1] http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html diakses tanggal 10 Desember 2015

[2] http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html diakses tanggal 10 Desember 2015

[3] http://www.redblobgames.com/grids/hexagons/#distances diakses tanggal 11 Desember 2015

[4] http://gamedevelopment.tutsplus.com/articles/balancing-turn-based-rpgs-the-big-picture--gamedev-8286 diakses tanggal 10 Desember 2015

[5] http://www.edenwaith.com/products/pige/tutorials/a-star.php diakses tanggal 11 Desember 2015

[6] http://www.cs.ubc.ca/~poole/aibook/slides/ch03/lect3.pdf diakses tanggal 10 Desember 2015

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 11 Desember 2015

Davin Prasetya 13514003