Kompleksitas Algoritma A* Pada Implementasi PassiveAI Untuk Game Mobile AI

Dandy Akhmad Rahadiansyah 23514098[1]
Program Magister Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]23514098@std.stei.itb.ac.id

***Abstract*— A* merupakan salah satu solusi dalam *Game Artificial Intelligence* untuk proses pencarian rute (*pathfinding*) yang dapat mensimulasikan agar karakter AI dapat berjalan seperti karakter lainnya. Ide dasarnya seperti pengimplementasian *matrix* dimana node memiliki posisi x, posisi y, dan bobot pada masing-masing node. Implementasi pada algoritma A* ini dapat ditemukan dalam Game RTS (Real Time Strategy) dan yang umum dimainkan adalah game *Candy Crush, Bejeweled* dan bahkan pada Game Mobile seperti *Samurai Vengeance*. Pada Paper ini akan dibahas implementasi algoritma A* dan komputasinya apabila dijalankan pada Game Mobile sebagai analisa yang terkait pada Teori Komputasi, dan menjadi solusi alternatif dalam perancangan game mobile AI.**

***Index Terms*—Game Mobile, PassiveAI, Artificial Intelligence, A*, matrix.**

I. INTRODUCTION

Banyak game developer nasional maupun internasional sangat memperhatikan pertanyaan seperti "Bagaimana cara game dalam menyenangkan seseorang?". Game yang menyenangkan merupakan aspek yang wajib dimiliki pada setiap aplikasi game. Selain dari multiplayer, single player masih sering digandrungi oleh kebanyakan pencandu game.

Pada game, masalah yang sering dihadapi adalah penggunaan AI sebagai salah satu metode dalam menghasilkan nilai kesenangan (*fun)* didalam dunia game itu sendiri. Ide yang digunakan sederhana, bagaimana AI dapat melakukan sesuatu yang serupa atau lebih dari player yang memainkan game (*adaptive*). Lalu ide tersebut dipecah menjadi beberapa bagian, "Bagaimana tingkah laku AI yang dapat dikontrol secara signifikan sesuai bidang (*terrain*) dasar pada game?", "Bagaimana AI dapat menghindari rintangan (*obstacle)"* dan "Bagaimana AI dapat menghindari player (*evade*) atau menyerang (*attack*) player itu sendiri?".

Algoritma yang sering kali digunakan dalam menyelesaikan masalah ini salah satunya menggunakan Algoritma A* dalam menentukan solusi yang harus diambil didalam pengambilan keputusan. Implementasi dasarnya sering menggunakan matriks untuk menentukan apakah algoritma ini layak atau tidak untuk mengatasi masalah dalam AI tersebut. Namun didalam implementasi dunia nyatanya sering kali algoritma ini yang mengganggu performasi apabila terlalu banyak objek AI yang di instansiasi pada bidang game. Terutama pada game mobile yang memiliki kapasitas memory yang jauh lebih kecil dari komputer PC. Oleh karena itu penambahan beberapa batasan sebagai salah satu solusi untuk menciptakan game AI dengan harapan berkurangnya kompleksitas AI pada game tersebut sehingga nilai *fun* tadi dapat tercapai pada game mobile.

II. TEOREMA

A. Game AI

Artificial Intelligence atau Kecerdasan Buatan merupakan ilmu dalam merepresentasikan kecerdasan manusia kedalam sistematika komputer yang memiliki aturan-aturan tertentu. Sedangkan Game AI adalah Kecerdasan Buatan yang dapat merepresentasikan pola pikir pemain dalam memainkan sebuah game untuk melakukan status tertentu.

AI pada game sering kali disebut sebagai BOT bahasa *internet slang* yang merupakan kata dari *robot*. Bot ini selalu digunakan sebagai representasi player jika kita bermain game dalam mode single player. Biasanya bot ini memiliki tingkat kesulitan easy, medium, hard. Yang mengindikasikan tingkat responsif, tingkat probabilitas, tingkat kecepatan analisa state dan lainnya.

Bot sering kali dijumpai pada game yang bersifat versus (berlawanan) seperti *chess*, *tictactoe*, dll. Dan bahkan game kompleks saat inipun sering kali digunakan dalam mode *single player* atau *campaign*. Walaupun banyak algoritma-algoritma lain yang mendukung proses pemilihan keputusan yang akurat seperti, Machine Learning.

Ada banyak cara yang dapat diimplementasikan selain secara otomatisasi. Yaitu dengan pathfinder dengan menggunakan waypoint yang ditentukan bagaimana bot bergerak dan beraksi jika menemukan objek yang dapat digunakan. Sebagai salah satu contoh adalah game dibawah ini.

Pada tahun 2000 game fenomenal *Counter Strike*, pergerakan bot diatur secara manual, dengan menggunakan titik(*waypoint*) yang digunakan sebagai status pergerakan bot didalam map game. Sehingga bot pun tidak begitu beradaptasi dengan lingkungan sekitarnya. Namun seiring berjalannya waktu algoritma path finder ini mulai digunakan secara automatic dalam analisa pergerakan bot apabila terdapat map baru sebanyak 1 kali.

Gambar 2.1 Waypoint tegak lurus pada game Counter Strike

B. Matrix

Dalam teori matematika matriks merupakan susunan bilangan-bilangan yang diapit dalam kurung siku "[]" yang memiliki ukuran sebesar a x b dengan a merupakan baris dan b merupakan kolom. Dalam komputer matriks terbagi berdasarkan dimensi yaitu 1 dimensi, 2 dimensi dan 3 dimensi. Matriks 1D, matriks memiliki nilai kolom saja atau baris saja. Pada matriks 2D, matriks ini memiliki nilai a x b dimana a merupakan baris dan b merupakan kolom. Sedangkan Matriks 3D merupakan matriks yang memiliki kedalaman dengan nilai a x b x c. Sama seperti 2D namun nilai c merupakan nilai dari kedalaman(*volume*) dari matriks itu sendiri. matriks ini dapat dikaitkan sebagai larik(*array*) yang memiliki yang berukuran sesuai dengan dimensionalnya.

0.1	0.5	0.1	0.5
0.2	0.1	0.2	0.1
0.3	0.2	0.3	0.2
0.2	0.3	0.2	0.3

Gambar 2.2 Contoh Matriks 2D 4x4

Pada paper ini matriks yang digunakan adalah matriks 2D dimana matriks ini merupakan representasi dari objek dalam peta 2D menunjukan posisi dimana dalam game terdapat penghalang(obstacle) dan objek tertentu.

C. Algoritma Dijkstra

Algoritma Dijkstra merupakan algoritma yang digunakan sebagai pencarian rute terpendek (shortest path) diantara node didalam sebuah graf. Algoritma ini berjalan dengan melakukan kalkulasi terhadap semua kemungkinan bobot terkecil dari setiap titik. Algoritma ini pasti akan menemukan rute terpendek dalam graf. Pada paper ini algoritma dijkstra digunakan sebagai Game AI pathfinder sebagai perbandingan kompleksitas algoritma.

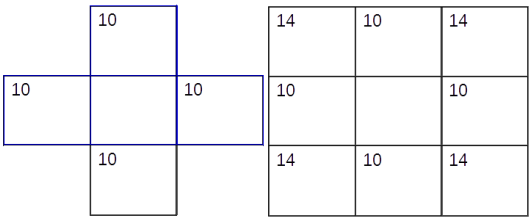
D. A (A star)*

Algoritma A* (A star) merupakan pengembangan dari algoritma Dijkstra dan Greedy Best First Search, dalam pencarian rute terpendek dan juga dapat menggunakan pencarian heuristic. Dalam definisi singkat, A* dapat melakukan pencarian secepat Best First Search. Pencarian A* dilakukan dengan membangkitkan kemungkinan yang ada dan memilih salah satu dengan bobot yang minimal pada kemungkinan tersebut. Lalu setelah kemungkinan ini dibangkitkan dan bobot diperhitungkan, kemungkinan ini dimasukan kedalam suatu daftar sampai semua node dicari.

Dapat ditentukan fungsi bobot pada algoritma ini sebagai penentuan solusi yang optimal, bobot pada sebuah node (f) dapat diformulakan sebagai berikut:

$$f = g + h$$

(g) merupakan bobot dalam setiap langkah.
Bobot diagonal pada starting point adalah g = g+14 sedangkan sisanya 4 bobot sisanya g = g+10



Gambar 2.3 4-Star, 8-Star

(h) merupakan nilai bobot estimasi gerakan dari suatu titik ke titik finish
Pencarian solusi ini diambil dengan mencari nilai (f) Terkecil (gambar 2.3) hijau sebagai starting point dan biru sebagai end point.

	G=5 H=30 F=35	G=12 H=20 F=22	G=10 H=10 F=20	G=10 H=0 F=10
G=20 H=40 F=60	G=15 H=30 F=45	G=10 H=20 F=30		G=5 H=0 F=5
G=25 H=40 F=65		G=15 H=20 F=35		
G=5 H=40 F=45	G=15 H=30 F=45	G=25 H=20 F=45		

Gambar 2.4 Solusi Pada A*

Psudocode untuk A* adalah :

```
inisialisasi open list
inisialisasi close list
Simpan starting point didalam open list

while open list tidak kosong
   Cari node dengan nilai f terkecil. Anggap itu "fmin"
```

Keluarkan “fmin” dari openlist
Cari fmin 8 tetangganya dan set induk node ke fmin

for each tetangga
if tetangga adalah solusi, keluar
Cek nilai **g**,**h**,**f**;

if node berada pada posisi yang sama sebagai solusi di dalam openlist \
Memiliki nilai f yang lebih kecil dari solusi, skip.

if node berada pada posisi yang sama sebagai solusi di dalam closelist \
Memiliki nilai f yang lebih kecil dari solusi, skip.

else, tambah node to the open list

endfor
Masukan **fmin** kedalam close list.

Endwhile

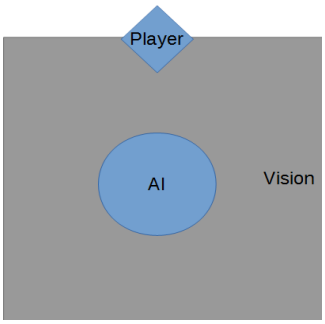
III. PENGUJIAN

A. Masalah

Masalah yang sering terjadi dalam implementasi game khususnya dalam mobile game adalah batasan memori dalam aplikasi mobile itu sendiri. Batasan memori dalam mobile Android adalah 17 mB. Gambar yang digunakan sebagai asset game sudah menguras memori yang cukup banyak dan belum termasuk pemrosesan logic game serta algoritma. Sehingga memerlukan strategi tertentu dalam menerapkan algoritma game AI salah satunya A* ini ketika diimplementasikan pada mobile game yang dimodelkan dengan matriks 2D.

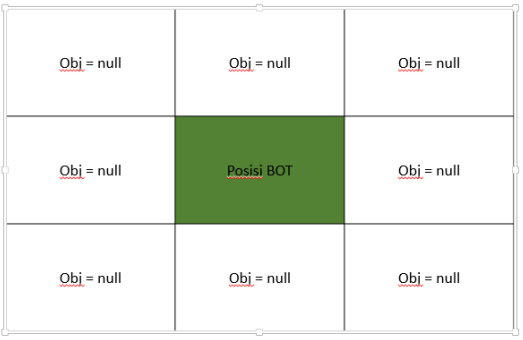
B. Pemodelan Solusi

Solusi yang dirancang untuk mengatasi masalah ini bernama *PassiveAI* dimana bot akan mulai melakukan pathfinding ketika player berada dalam nilai ambang (threshold) tertentu. AI akan mulai melakukan statusnya ketika player berada di jarak pandangan (vision) tertentu. Sehingga komputasi yang dilakukan akan lebih singkat dan ditentukan berdasarkan threshold.



Gambar 3.1 Contoh luas Matriks 2D pada Vision AI.

Pada gambar 3.1 daerah abu-abu merupakan jarak pandang AI dalam bentuk matriks (M) dengan posisi (x, y). Threshold (t) merupakan batasan jarak pandang pada AI. Dalam proses pemodelan *PassiveAI* ini model yang digunakan adalah matriks 2D sebesar 3x3.



Gambar 3.2 Matriks 2D t x t (M) menunjukan range Vision pada Bot, dengan t = 3.

C. Implementasi A*

Untuk mencari solusi optimal dari sebuah matriks untuk menghasilkan rute terpendek dalam *PassiveAI*, pada paper ini akan menggunakan algoritma A*. Dengan asumsi bahwa AI akan dipicu oleh aksi player didalam game. Algoritma A* bekerja jika player memicu aksi tertentu dalam game.

Langkah awal pada pengerjaan algoritma A* pada *PassiveAI* ini dengan menghitung keseluruhan nilai (g) dari lalu mengisi matriks (M) dengan nilai (g) dan nilai (h), lalu lakukan formula mencari nilai (f) dan isikan kembali kedalam matriks (M).

Inisialisasi list untuk openlist dan closelist sebagai referensi matriks yang telah diperiksa nilai (f) nya. Untuk setiap perulangan kita melakukan pencarian nilai (f) paling kecil, kemudian setelah didapat, kita hapus dari openlist dan dimasukan kedalam closelist. Hingga ditemukannya titik akhir pada matriks dimana pada kasus ini disebut sebagai objek *player*. Apabila *PassiveAI* tidak menemukan adanya player. *PassiveAI* berada didalam state idle. Dengan melakukan patroli dengan melakukan looping diseluruh area threshold. (Gambar 3.3).

D. Komputasi A*

Dalam solusi *PassiveAI* ini terdapat 2 step yang memerlukan analisa komputasi diantara lain:

1. Komputasi nilai (g) berdasarkan matriks (M) dengan nilai threshold (t) dimana waktu kompleksitas dalam menyelesaikan masalah ini adalah **$O(n^2)$**. Dimana pencarian nilai (g) dilakukan diseluruh matriks (M) sehingga waktu kompleksitas adalah (t x t).

2. Komputasi pencarian solusi nilai (f) dalam pencarian rute terpendek akan membutuhkan waktu kompleksitas exponensial terhadap solusi yang ditemukan **$O(x^n)$** dimana x merupakan nilai child yang dimiliki posisi saat ini dan n merupakan kedalaman dari pencarian.

E. Simulasi Algoritma

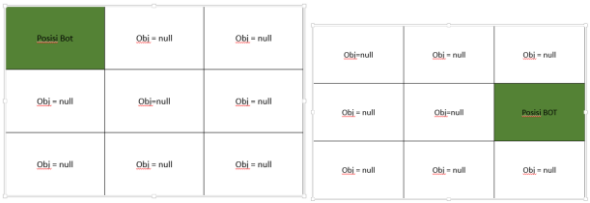
Dalam implementasi A* dalam kasus ini terdapat beberapa batasan rule yang digunakan adalah:

1. AI akan bergerak ketika Player memicu status tertentu. Contoh: membuka pintu, masuk ke daerah baru.
2. Area pada map game ini tidak ada objek penghalang hanya pembatas map game saja. Contoh : *Samurai Vengeance*.

Dengan batasan tambahan ini dilakukan implementasi dengan test case sebagai berikut:

Test 1: Asumsikan bahwa matriks M tidak memiliki rintangan.

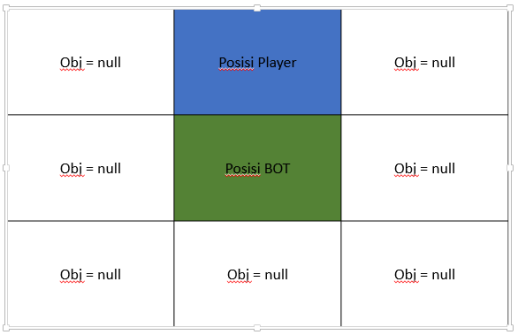
Solusi 1: AI akan diam tidak melakukan komputasi. Namun akan bergerak dari posisi awal hingga ke posisi awal.



Gambar 3.3 a) solusi 1 n=1; b) solusi 1 n=4;

Test 2: Asumsikan bahwa matriks M menemukan player didepan.

Solusi 2: AI akan melakukan Algoritma A* untuk mendekati player didepannya.



Gambar 3.4 Solusi 2 bot berada lurus didepan player.

Test 3: Asumsikan bahwa matriks M menemukan player di daerah diagonal AI.

Solusi 3: AI akan menggeser posisinya sehingga tegak lurus dengan player. Lalu melakukan komputasi pencarian rute terpendek.

Gambar 3.5 a) Menemukan posisi diagonal Player. b) Bot berada lurus didepan player.

Berdasarkan test case diatas dapat diperoleh hasil simulasi analisa algoritma A* terhadap nilai threshold yang mempengaruhi kecepatan *PassiveAI* dalam melakukan state selanjutnya setelah Player memasuki batas jarak pandang.

Hasil Simulasi A* pada game mobile android:

Threshold	Kecepatan Menghasilkan Solusi	Perubahan State *Passive AI*	Memory
3	0.008771	0.010233	6 MB
5	0.014812	0.015339	8.87 MB
7	0.033227	0.039122	13.22 MB

IV. ANALISIS

Pada kasus ini terdapat beberapa nilai threshold yang berbeda akan mempengaruhi performa pada game mobile yang cukup signifikan. Algoritma A* dapat menemukan solusi yang cepat dalam menemukan rute terpendek namun tetap berada pada ambang batas memory mobile game yaitu 17 MB.

Pada simulasi algoritma A* ini hanya dijalankan sampai threshold bernilai 7. Mungkin saja hasilnya berbeda apabila jumlah *PassiveAI* yang digunakan cukup banyak, dan tentunya akan menambah beban pengerjaan pada perangkat mobile itu sendiri. Dikarenakan instansiasi objek *PassiveAI* yang memiliki deskripsi yang dan memiliki tujuan yang sama.

Algoritma A* bekerja secara efisien jika sebelumnya kita membuat list untuk openlist dan closelist sebagai referensi matriks yang telah diperiksa nilai (f). Untuk setiap perulangan kita melakukan pencarian nilai (f) paling kecil, kemudian setelah didapat, kita hapus dari openlist dan dimasukan kedalam closelist. Hingga ditemukannya titik akhir pada matriks dimana pada kasus ini disebut sebagai objek *player*. Cara ini secara otomatis akan menghindari apabila terdapat objek diantara *PassiveAI*, sehingga setelah perulangan selesai didapat *PassiveAI* akan melakukan state tertentu seperti menyerang atau mendekati *player*.

V. KESIMPULAN

Tidak seperti game PC pada umumnya dimana keterbatasan memori yang cukup kecil membuat *PassiveAI* merupakan salah satu solusi dalam representasi pergerakan bot dalam sebuah game khususnya game mobile. Dikarenakan bot dapat bertingkah laku layaknya *player*. Seperti mendekatkan diri pada objek lawan, atau menyerang lawan. Tetapi *PassiveAI* ini akan lebih menarik jika terdapat unsur *unpredictable* dalam melakukan perubahan state dari mendekat ke menyerang dengan menggunakan algoritma probabilistik seperti psudo-random, dll.

REFERENCES

[1] A. Patel, "Amit A* Page," 12 11 2015. [Online]. Available: http://theory.stanford.edu/~amitp/GameProgramming/. [Accessed 18 12 2015].

[2] R. Eranki, "Pathfinding using A* (A-Star)," 2002. [Online]. Available: http://web.mit.edu/eranki/www/tutorials/search/. [Accessed 16 12 2015].

[3] A. S. Azhar, "DuniaDigit," 8 2013. [Online]. Available: http://duniadigit.blogspot.co.id/2013/08/belajar-algoritma-untuk-pencarian-jalur.html. [Accessed 17 12 2015].

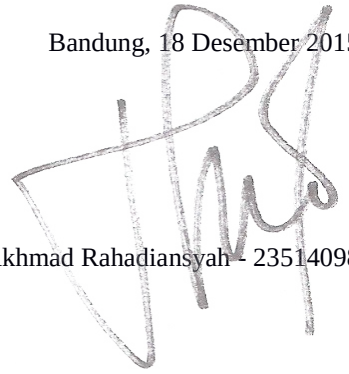
[4] R. Graham, H. McCabe and S. Sheridan, Pathfinding in Computer Games, Blanchardstown: Institute of Technology Blanchardstown , 2000.

[5] A. Stentz, Optimal and Efficient Path Planning for Partially-Known Environments, Pittsburgh: Carnegie Mellon University, 1994.

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, 18 Desember 2015



Dandy Akhmad Rahadiansyah - 23514098