BAB I
PENDAHULUAN

1.1 Latar Belakang

Pada saat ini game tidak hanya digunakan sebagai hiburan melainkan dapat juga digunakan sebagai sarana edukasi dan simulasi[TRY-14]. Rekrutan militer Amerika Serikat menggunakan *simulator* dan *video game* yang mengasah keterampilan tempur mereka dan bahkan melindungi mereka dari tekanan mental perang[HSU-10]. Dengan menggunakan simulasi juga dapat menghemat amunisi dan menghindari kerusakan yang tidak diinginkan. Selain itu juga ramah lingkungan karena tidak menimbulkan kebisingan dan kerusakan yang tidak diperlukan. Simulasi dalam bidang militer menggunakan *video game* ataupun *virtual reality* merupakan hal yang perlu dikembangkan agar dapat meningkatkan keterampilan tempur tanpa ada kerusakan yang tidak perlu.

Maka dari itu game perlu didukung oleh *game environment* yang baik agar permainan menjadi lebih realis. Hal tersebut dapat didukung oleh grafis dan penggunaan agen cerdas dalam game. Grafis yang dibangun tidak akan menghasilkan game yang realistis jika tidak didukung oleh implementasi behaviors agen cerdas / NPC (*Non Player Character*) dalam game. Game saat ini membutuhkan pathfinding yang efisien untuk mendukung sejumlah besar agen melalui lingkungan yang ekspansif dan dinamis[WAL-10]. Namun umumnya algoritma yang digunakan tidak efisien sehingga muncul masalah dalam penggunaan memory dan sumber daya CPU.

Algoritma A* adalah algoritma yang umum digunakan untuk menyelesaikan permasalahan pathfinding. Namun Algoritma A* memiliki kelemahan yaitu jika diimplementasikan ke dalam sebuah peta permainan yang besar. Dalam skala kecil algoritma A* merupakan algoritma pencarian terbaik, namun jika peta permainan diperluas akan menambah waktu pencarian karena ruang lingkup penelusuran semakin besar[DEC-84]. Penggunaan memori menjadi besar karena node yang diingat bertambah banyak. Peningkatan pada algoritma A* sangatlah penting untuk

mendapatkan jalur terdekat dan akurat tanpa menggunakan sumber daya yang besar. Seperti penelitian sebelumnya[BOT-04], *Hierarchical Pathfinding A**(HPA*) untuk mengatasi kelemahan algoritma A*.

Algoritma pencarian yang baik dengan jalur yang optimal adalah A*[STO-96] dan IDA* adalah algoritma dengan penggunaan memori yang lebih sedikit[KOR-85]. Untuk mendapatkan kecepatan dan penggunaan memori yang sedikit maka perlu dilakukan abstraksi seperti yang dilakukan pada varian A* yang lain yaitu HPA*[BOT-04] dan HAA*[HAR-08]. Perbedaanya adalah pada HAA* dikombinasikan dengan *clearance-based pathfinding* yang digunakan untuk menyelesaikan masalah dengan ukuran unit dan *terrain* yang bermacam-macam. Sedangkan game *First Person Military Simulation Game* adalah game FPS(*First Person Shooter*) yang menggunakan satu tipe unit dan terrain. HPA* adalah algoritma yang sederhana untuk dipahami, mudah, dan efisien[BOT-04]. HPA* merupakan algoritma untuk mempersempit ruang lingkup pencarian dengan melakukan partisi pada peta grid ke dalam *clusters.* Sehingga cocok diimplementasikan pada game dengan multi-NPC karena dapat mempersingkat waktu pencarian dan memperkecil penggunaan memori. HPA* biasanya diterapkan pada peta berbasis grid.

Oleh karena itu, dalam skripsi ini dilakukan implementasi HPA* untuk pathfinding NPC dalam game *First Person Military Simulation* untuk melakukan pencarian jalur secara ringan dan cepat sehingga game *First Person Military Simulation* dapat berjalan secara optimal.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang, maka dapat dirumuskan permasalahan pada penelitian ini yaitu :

1. Bagaimana implementasi algoritma HPA* pada movement NPC di First Person Military Simulation Game?
2. Bagaimana pengaruh besar ukuran *cluster* terhadap kinerja algoritma HPA* pada movement NPC di First Person Military Simulation Game?

1.3 Batasan Masalah

Agar permasalahan yang dirumuskan lebih terfokus, maka penelitian ini dibatasi oleh hal-hal sebagai berikut :

1. Representasi ruang pencarian dari peta permainan yang digunakan berupa *regular grids* berbentuk persegi.
2. Peta permainan menggunakan *single terrain type*.
3. *Seeker* tidak bergerak mengikuti jalur yang sudah ditemukan.
4. Pencarian dilakukan di *Single-Level Graph.*

1.4 Tujuan

Tujuan yang ingin dicapai dalam implementasi skripsi ini adalah untuk menyelesaikan masalah pencarian jalur NPC pada game *First Person Military Simulation* menggunakan *Hierarchical Pathfinding A** (HPA*) dan untuk mengukur pengaruh ukuran *cluster* terhadap kinerja algoritma HPA* pada movement NPC di First Person Military Simulation Game.

1.5 Manfaat

Manfaat yang diharapkan dari implementasi skripsi ini adalah :

1. Agar *movement* NPC pada game *First Person Military Simulation* menjadi efektif dan efisien.
2. Dapat dijadikan sebagai pembanding atau literatur dalam penelitian yang menerapkan algoritma HPA* dimasa yang akan datang.

1.6 Sistematika Penulisan

Sistematika isi dan penulisan dalam skripsi ini antara lain:

BAB I Pendahuluan

Berisi tentang latar belakang, rumusan masalah, batasan masalah, tujuan dan manfaat dari penelitian serta sistematika penulisan.

BAB II Dasar Teori

Menguraikan tentang dasar teori dan teori penunjang yang berkaitan dengan *Hierarchical Pathfinding* A*.

BAB III Metodologi Penelitian

Berisi tentang gambaran umum langkah-langkah dalam implementasi pathfinding dimulai dari tahap perancangan, implementasi serta pengujian secara umum.

BAB IV Perancangan

Berisi tentang perancangan dalam implementasi dari pathfinding menggunakan *Hierarchical Pathfinding* A*.

BAB V Implementasi

Membahas tentang implementasi dari pathfinding menggunakan *Hierarchical Pathfinding* A*.

BAB VI Evaluasi

Memuat hasil evaluasi terhadap *Hierarchical Pathfinding* A* yang telah diimplementasikan.

BAB VII Penutup

Memuat kesimpulan yang diperoleh dari pembuatan dan pengujian real-time pathfinding, serta saran-saran untuk pengembangan lebih lanjut.

BAB II
TINJAUAN PUSTAKA

Bab ini membahas dasar teori yang digunakan untuk menunjang penulisan skripsi mengenai Implementasi *Pathfinding* dalam game *First Person Military Simulation* dengan menggunakan *Hierarchical Pathfinding* A*. Beberapa Dasar teori yang dimaksud adalah *Pathfinding,* Algoritma A*, *Regular Grids, Hierarchical Pathfinding* A*, dan Parameter Pengujian

2.1 Pathfinding

Pathfinding adalah proses pencarian jalur tercepat dari titik asal ke titik tujuan dengan menghindari berbagai halangan sepanjang jalur yang ditempuh. Strategi *Pathfinding* biasanya digunakan sebagai inti dari sistem gerakan AI[GRA-03]. Posisi *pathfinding* dalam AI berada diantara *decision making* dan pergerakan[MIL-09]. Karena terkadang pathfinding juga menentukan keputusan tentang kemana akan bergerak serta bagaimana menuju kesana. Secara umum pathfinding dapat dibedakan menjadi pathfinding statik dan dinamik. Ada tiga tahap utama untuk melakukan *pathfinding* dalam permainan komputer yaitu (1) pre-procesing peta permainan (2) membuat graph beserta datanya (3) menelusuri graph untuk mendapatkan solusi menggunakan algoritma pencarian

Pre-procesing pada peta permainan dilakukan dengan cara membagi peta permainan ke dalam bagian yang lebih kecil sehingga ruang lingkup pencarian dapat dikurangi. Representasi ruang pencarian dari peta permainan yang digunakan pada permainan komputer diantaranya adalah *binary space partition trees*(BSP), *corner graph, waypoint graph, circle-based waypoint graph, navigation meshes, area awarness system,* dan *regular grids.*

Pathfinding memiliki beberapa algoritma yang bisa diterapkan antara lain [FAU-12]:

a. Brute Force

Algoritma ini merupakan algoritma yang paling mudah dimengerti. Cara kerjanya adalah membandingkan posisi sekarang dengan posisi tujuan dan menentukan langkah berikutnya.

b. BFS

Breadth-First Search merupakan algortima yang menyelesaikan masalah dengan memanfaatkan struktur pohon.

c. DFS

Deep-First Search merupakan algoritma yang menyeleseaikan masalah dengan memanfaatkan struktur pohon. DFS mencari solusi ke node yang paling dalam pada pohon.

d. Branch dan A*

Branch and Bound merupakan pengembangan dari BFS. Pada Branch and Bound, setiap node memiliki harga (dengan cara penghitungan harga yang bermacam-macam). Harga node menentukan kedekatan node dengan solusi. Algoritma A* (Baca : A bintang) merupakan salah satu pengembangan dari Algoritma *Branch and Bound*. Perhitungan harga pada algoritma A* memanfaatkan unsur heuristik pada benda.

2.2 Algoritma A*

Algoritma A*[HAR-68] adalah salah satu algoritma Branch & Bound atau disebut juga sebagai sebuah algoritma untuk melakukan pencarian solusi dengan menggunakan informasi tambahan (heuristik) dalam menghasikan solusi yang optimal [TIL-11].

2.2.1 Terminologi Dasar A*

Beberapa terminologi dasar yang terdapat pada algoritma ini adalah starting point, current node, simpul, neighbor node, open set, closed set, came from, harga (cost), walkability, target point. Terminologi dasar A* yaitu [HAP-11]:

a. Starting point adalah sebuah terminologi untuk posisi awal sebuah benda.

b. Current node adalah simpul yang sedang dijalankan dalam algortima pencarian jalan terpendek.
c. Simpul adalah petak-petak kecil sebagai representasi dari area pathfinding. Bentuknya dapat berupa persegi, lingkaran, maupun segitiga.
d. Neighbor node adalah simpul-simpul yang bertetangga dengan current node.
e. Open set adalah tempat menyimpan data simpul yang mungkin diakses dari starting point maupun simpul yang sedang dijalankan.
f. Closed set adalah tempat menyimpan data simpul sebelum current node yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan.
g. Came from adalah tempat menyimpan data ketetanggaan dari suatu simpul, misalnya y came from x artinya neighbor node y dari current node x.
h. Harga (F) adalah nilai yang diperoleh dari penjumlahan nilai G, jumlah nilai tiap simpul dalam jalur terpendek dari starting point ke current node, dan H, jumlah nilai perkiraan dari sebuah simpul ke target point.
i. Target point yaitu simpul yang dituju.
j. Walkability adalah sebuah atribut yang menyatakan apakah sebuah simpul dapat atau tidak dapat dilalui oleh current node.

2.2.2 Fungsi Heuristik

Algoritma A* menerapkan teknik heuristik dalam membantu penyelesaian persoalan. Heuristik adalah penilai yang memberi harga pada tiap simpul yang memandu A* mendapatkan solusi yang diinginkan. Dengan heuristik yang benar, maka A* pasti akan mendapatkan solusi (jika memang ada solusinya) yang dicari. Dengan kata lain, heuristik adalah fungsi optimasi yang menjadikan algoritma A* lebih baik dari pada algoritma lainnya. Namun heuristik masih merupakan estimasi / perkiraan biasa saja Sama sekali tidak ada rumus khususnya. Artinya, setiap kasus memiliki fungsi heuristik yang berbedabeda. Algoritma A* ini bisa dikatakan mirip dengan algoritma Dijkstra, namun pada algoritma Dijkstra, nilai fungsi heuristiknya selalu 0 (nol) sehingga tidak ada fungsi yang mempermudah pencarian solusinya.

Nilai ongkos pada setiap simpul n menyatakan taksiran ongkos termurah lintasan dari simpul n ke simpul target (target node), yaitu [TIL-11]:

$$F(n) = \textit{nilai taksiran lintasan termurah dari simpul status n ke status tujuan} \quad \text{....(1)}$$

Dengan kata lain, F (n) menyatakan batas bawah (lower bound) dari ongkos pencarian solusi dari status n. Fungsi heuristik yang terdapat pada algoritma A* untuk menghitung taksiran nilai dari suatu simpul dengan simpul yang telah dilalui yaitu:

$$F(n) = G(n) + H(n) \quad \text{..... (2)}$$

Dimana:

F(n) = ongkos untuk simpul n

G(n) = ongkos mencapai simpul n dari akar

H(n) = ongkos mencapai simpul tujuan dari simpul n

2.2.3 Langkah Algoritma A*

Algortima A* secara ringkas langkah demi langkahnya adalah sebagai berikut.

1. Tambahkan starting point ke dalam openset.
2. Ulangi langkah berikut:
   a. Carilah biaya F terendah pada setiap simpul dalam open set. Node dengan biaya F terendah kemudian disebut current node.
   b. Masukkan ke dalam closed set.
   c. Untuk setiap 8 simpul (neighbor node) yang berdekatan dengan current node:
      1. Jika tidak walkable atau jika termasuk closed set, maka abaikan.
      2. Jika tidak ada pada open set, tambahkan ke open set.
      3. Jika sudah ada pada open set, periksa apakah ini jalan dari simpul ini ke current node yang lebih baik dengan menggunakan biaya G sebagai ukurannya. Simpul dengan biaya G yang lebih rendah berarti bahwa ini adalah jalan yang lebih

      baik. Jika demikian, buatlah simpul ini (neighbor node) sebagai came from dari current node, dan menghitung ulang nilai G dan F dari simpul ini.
   d. Stop ketika:

1. Menambahkan target point ke dalam closed set, dalam hal ini jalan telah ditemukan, atau

2. Gagal untuk menemukan target point, dan open set kosong. Dalam kasus ini, tidak ada jalan.

3. Simpan jalan. *backtrack* dari target point, pergi dari masing-masing simpul ke simpul came from sampai mencapai starting point. Itu adalah jalan Anda.

2.2.4 Kompleksitas Algoritma A*

Kompleksitas waktu dari A * tergantung pada heuristiknya. Dalam kasus terburuk, jumlah simpul yang diperluas adalah eksponensial dari panjang solusi (jalur terpendek), tetapi polinomial ketika ruang pencarian adalah pohon, ada sebuah simpul tujuan tunggal, dan fungsi heuristik h memenuhi kondisi berikut:

$$|H(x) - h*(x)| = O(\log hh*(x)) \quad \text{.....(3)}$$

Dimana * adalah heuristik optimal, biaya yang tepat yang didapat dari x ke tujuan. Dengan kata lain, kesalahan dari h tidak akan tumbuh lebih cepat dari logaritma dari "heuristik sempurna" h * yang mengembalikan jarak yang sebenarnya dari x ke tujuan.

2.3 *Hierarchical Pathfinding* A*

Hierarchical Pathfinding A* pertama kali ditemukan oleh [BOT-04]. HPA* dapat memberikan peningkatan peforma yang besar di ruang pencarian yang besar bila dibandingkan dengan algoritma A*. HPA* dapat menggunakan satu atau lebih level dari abstraksi untuk mengurangi kompleksitas suatu masalah. Abstraksi ruang pencarian dapat dilakukan dengan 4 langkah[LIN-13] :

1. Peta permainan dibagi kedalam *clusters* berbentuk kotak dengan ukuran yang ditentukan
2. Lalu pada setiap *cluster* yang bertetangga ditentukan entrances yang dapat dibuat.
3. *Entrances* dari *cluster* yang bertetangga dihubungkan dengan *edge* yang bernilai 1. Tetangga yang diagonal tidak dihubungkan.

4. Semua entrance yang berada dalam satu *cluster* saling dihubungkan. Gunakan jalur A* untuk menghubungkan setiap entrances dan gunakan panjang jalur A* dalam penentuan bobot *edge*.

Untuk mencari jalur dari graf abstrak, pertama *node* awal dan *node* akhir harus ditambahkan ke dalam graf abstrak. Setelah *node* awal dan *node* akhir ditambahkan lakukan pencarian menggunakan algoritma A* dalam abstrak graf untuk mendapatkan solusi abstrak. Solusi abstrak dapat diperhalus menjadi solusi konkret dengan menghubungkan *node* yang ada pada solusi abstrak. Cara menghubungkan antar *node* yang ada pada solusi abstrak dengan melakukan pencarian dengan algoritma A* untuk menghubungkan node. Setelah dihubungkan akan dihasilkan jalur konkret untuk menyelesaikan permasalahan.

2.4 *Regular Grids*

Representasi ruang pencarian dari peta permainan yang digunakan pada permainan komputer diantaranya adalah *binary space partition trees*(BSP), *corner graph*, *waypoint graph*, *circle-based waypoint graph*, *navigation meshes*, *area awarness system*, dan *regular grids*. *Regular Grids* merupakan cara yang paling sederhana untuk merepresentasikan sebuah ruang pencarian. *Regular Grid* terbuat dari bentuk persegi, persegi panjang, *hexagons*, dan segitiga.

Ruang pencarian berdasarkan *grids* sangat populer digunakan dalam permainan komputer dua dimensi. Untuk memakai ruang pencarian berdasarkan *grids* dalam permainan tiga dimensi diperlukan modifikasi. Biasanya diperlukan *grids cells* dengan jumlah yang banyak untuk merepresentasikan sebuah peta permainan. Semakin luas peta permainan semakin banyak pula jumlah *grid cells* yang dipakai sehingga kebutuhan penyimpanan data *grid* dalam memori semakin banyak dan memperlambat *pathfinding*.

Kelebihan dari represetasi ruang pencarian berbasis *grids* adalah mendukung adanya *random-access lookup*[TOZ-04]. *Random-access lookup* dalam representasi ruang pencarian berbasis *grids* memungkinkan kita untuk menentukan *tile* atau *grid cell* mana yang sesuai dengan koordinat peta permainan dengan kompleksitas waktu

O(1). Representasi ruang pencarian lain seperti *corner graph, waypoint graph, navigation meshes* tidak mendukung *random-access lookup* sehingga diperlukan sebuah penelusuran *graph* untuk menentukan *node* terdekat sesuai dengan koordinat(X,Y) atau (X,Y,Z) yang diberikan.

2.5 Parameter Pengujian

Penelitian ini menggunakan 2 parameter dalam pengujian yaitu panjang jalur dan waktu pencarian.

2.5.1 Panjang Jalur

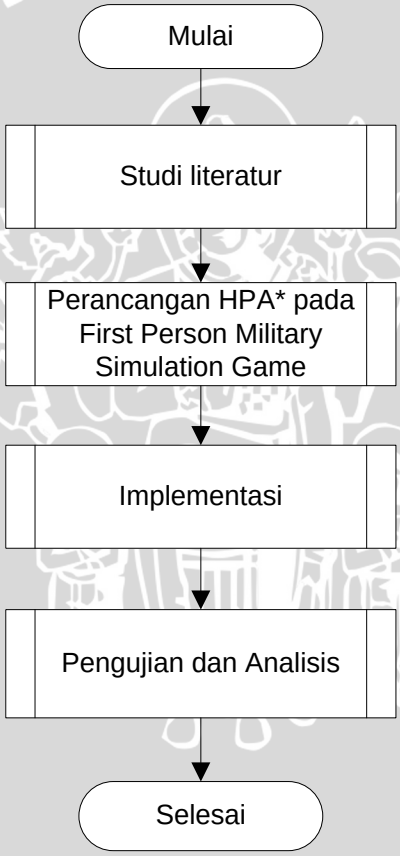
Panjang Jalur yang dimaksud adalah panjang jalur yang dibuat dalam *pathfinding*/pencarian jalur. Terdapat 5 panjang jalur yang diukur untuk pengujian, yaitu panjang jalur pencarian jalur menggunakan HPA* dengan *cluster* berukuran 10x10, 15x15, 20x20, 30x30 dan *cluster* berukuran 60x60. Lalu akan dibandingkan mana panjang jalur yang lebih pendek.

2.5.2 Waktu pencarian

Waktu pencarian adalah lama waktu algoritma pencarian dalam menyelesaikan masalah pencarian jalur. Untuk pengujian dilakukan pengukuran waktu pencarian HPA* dengan *cluster* berukuran 10x10, 15x15, 20x20, 30x30 dan *cluster* berukuran 60x60 dalam menemukan solusi.

BAB III
METODOLOGI PENELITIAN

Bab ini menjelaskan tentang langkah-langkah yang ditempuh dalam penelitian yang diusulkan. Metodologi yang dilakukan dalam penelitian ini dilakukan melalui beberapa tahapan, yakni studi literatur, perancangan *Hierarchical Pathfinding* A* pada *First Person Military Simulation Game*, implementasi, dan evaluasi. Gambar 3.1 merupakan diagram alir metode yang berisi tahapan-tahapan yang dilakukan dalam penelitian yang diusulkan.



Gambar 3.1 Diagram Alir Runtutan Metode Penelitian

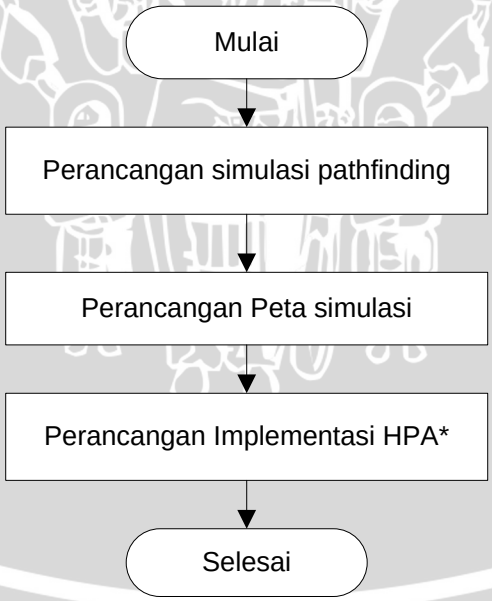
3.1 Studi Literatur

Studi literatur merupakan cara yang digunakan dalam pengumpulan informasi-informasi yang diperlukan dengan sumber yang digunakan berupa data-data yang terkait. Dalam studi literatur dikumpulkan data-data dari sumber yang terpecaya yang menjadikan sumber tersebut menjadi dasar dari teori-teori yang akan diimplementasikan lebih lanjut pada aplikasi yang dibuat. Studi literatur yang digunakan adalah sebagai berikut :

1. Algoritma A*
2. Algoritma *Hierarchical Pathfinding* A*
3. Bahasa Pemrograman C#

3.2 Perancangan

Bab perancangan terdiri dari tiga tahap, yaitu perancangan simulasi pathfinding, perancangan peta simulasi, dan perancangan implementasi Hierarchical Pathfinding A*. Tahap-tahap perancangan dijabarkan dalam diagram alir Gambar 3.2.



Gambar 3.2 Diagram Alir Perancangan Implementasi

3.2.1 Perancangan Simulasi *Hierarchical Pathfinding* A*

Pada subbab ini menjabarkan tentang perancangan simulasi pathfinding menggunakan HPA*. Simulasi dirancang untuk mengetahui valid atau tidaknya behaviour dari implementasi algoritma HPA*. Simulasi juga akan digunakan dalam pengujian.

3.2.2 Perancangan Peta Simulasi

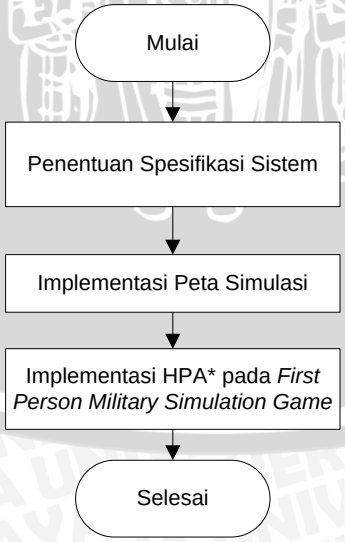
Subbab ini akan menjelaskan tentang perancangan peta permainan dari simulasi pathfinding menggunakan algoritma HPA*. Bentuk peta dan halangan yang digunakan akan dijelaskan pada subbab ini.

3.2.3 Perancangan Implementasi HPA*

Pada subbab ini berisi tentang tahapan yang dilakukan untuk merancang implementasi HPA*. Pada subbab ini dijelaskan tentang langkah-langkah dari algoritma HPA* serta perhitungan panjang langkah yang digunakan.

3.3 Implementasi

Bab implementasi terdiri dari tiga tahap, yaitu spesifikasi sistem, implementasi peta simulasi, dan implementasi HPA* pada *First Person Military Simulation Game*. Tahap-tahap implementasi dijabarkan dalam diagram alir dalam Gambar 3.3.



Gambar 3.3 Diagram Alir Implementasi

3.3.1 Penentuan Spesifikasi Sistem

Subbab ini menjabarkan spesifikasi dari lingkungan sistem tempat simulasi dijalankan. Spesifikasi sistem yang dimaksud adalah lingkungan perangkat keras dan perangkat lunak yang digunakan dalam menjalankan simulasi. Subbab ini bertujuan untuk mengetahui dalam spesifikasi tersebut, seberapa baik simulasi dijalankan untuk pengolahan data selanjutnya.

3.3.2 Implementasi Peta Simulasi

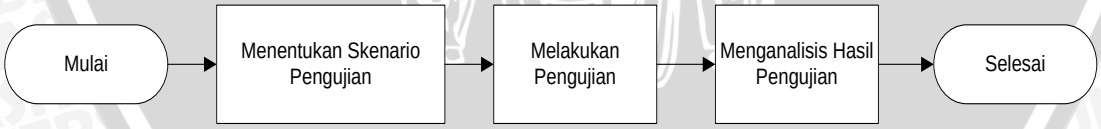
Pada subbab ini akan diimplementasikan peta yang telah dirancang sebelumnya yang akan digunakan dalam simulasi.

3.3.3 Implementasi HPA* pada *First Person Military Simulation Game*

Subbab ini berisi tentang implementasi HPA* untuk menyelesaikan permasalahan *pathfinding* pada *First Person Military Simulation Game* serta penerapan dari algoritma yang telah dirancang dan akan dipakai dalam simulasi. Metode-metode yang sudah dirancang dalam bab perancangan juga akan dijabarkan dalam subbab ini.

3.4 Pengujian dan Analisis

Akan dilakukan pengujian berdasarkan implementasi yang telah dibuat berdasarkan metriks pengujian yang telah ditentukan, yaitu panjang jalur dan waktu pencarian. Alur dari pengujian dan analisis dapat dilihat pada Gambar 3.4.



Gambar 3.4 Diagram Alir Pengujian dan Analisis

3.4.1 Menentukan Skenario Pengujian

Skenario pengujian algoritma dilakukan dengan 5 alternatif, yang pertama adalah pengujian mengukur panjang jalur dengan membandingkan antara panjang jalur saat menggunakan ukuran *cluster* 10x10 *grid, cluster* 15x15 *grid, cluster* 20x20 *grid, cluster* 30x30 *grid*, dan panjang jalur saat menggunakan ukuran *cluster* 60x60

grid, yang kedua adalah dengan pengujian menghitung waktu pencarian yang dibutuhkan dengan membandingkan waktu pencarian saat menggunakan ukuran *cluster* 10x10 *grid*, *cluster* 15x15 *grid*, *cluster* 20x20 *grid*, *cluster* 30x30 *grid*, dengan waktu pencarian saat menggunakan ukuran *cluster* 60x60 *grid*.

3.4.2 Melakukan Pengujian

Pada skenario pertama akan direkam panjang jalur yang dibuat tiap ukuran *cluster*. Pada skenario kedua akan direkam waktu pencarian dari masing-masing ukuran *cluster*.

3.4.3 Menganalisa Hasil Pengujian

Hasil pengujian skenario pertama akan dianalisis efisiensi antara saat menggunakan ukuran *cluster* 10x10 *grid*, *cluster* 15x15 *grid*, *cluster* 20x20 *grid*, *cluster* 30x30 *grid*, dan saat menggunakan ukuran *cluster* 60x60 *grid* berdasarkan panjang jalur yang dibuat. Sedangkan hasil pengujian skenario kedua akan dianalisis kecepatan antara saat menggunakan ukuran *cluster* 10x10 *grid*, *cluster* 15x15 *grid*, *cluster* 20x20 *grid*, *cluster* 30x30 *grid*, dan saat menggunakan ukuran *cluster* 60x60 *grid* berdasarkan rekaman waktu pencarian.

BAB IV
PERANCANGAN

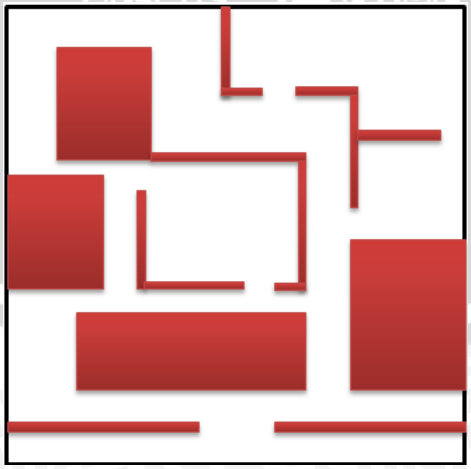
Bab ini membahas tentang perancangan implementasi dari penelitian ini. Perancangan yang dilakukan terdiri dari tiga tahapan, yaitu perancangan simulasi HPA*, perancangan peta simulasi, dan perancangan implementasi HPA*.

4.1 Perancangan Simulasi *Hierarchical Pathfinding* A*

Pada awal perancangan akan dirancang simulasi untuk pathfinding HPA* pada lingkungan 3D. Pada awal simulasi akan disediakan sebuah labirin dengan halangan statis. Aktor/*Seeker* akan ditempatkan di suatu titik pada labirin sebagai titik awal dan input berupa *mouse click* sebagai titik tujuan. Setelah mendapatkan titik tujuan akan dilakukan pencarian jalur terdekat menggunakan HPA*. Skenario ini bertujuan mengetahui valid atau tidaknya *behaviour* yang akan dirancang dan diimplementasikan dan juga untuk menguji jarak tempuh serta waktu pencarian yang dilalui jika dibandingkan antara ukuran *cluster* 10x10, 15x15, 20x20, 30x30 dan 60x60.

4.2 Perancangan Peta Simulasi

Pada simulasi Hierarchical Pathfinding A* akan dirancang sebuah peta yang digunakan pada pencarian jalur. Berikut rancangan peta simulasi dapat dilihat pada Gambar 4.1.

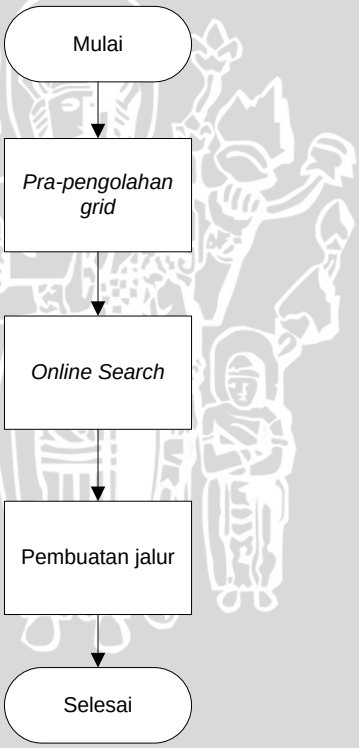


Gambar 4.1 Peta Simulasi

Dapat dilihat pada Gambar 4.1, peta terdiri dari lantai dan kotak berwarna merah. Kotak berwarna merah pada peta merupakan halangan statis. Halangan disusun sedemikian rupa sehingga peta membentuk labirin. Aktor/Seeker diletakkan di sudut kiri bawah peta sebagai titik awal.

4.3 Perancangan Implementasi HPA*

Algoritma HPA* memiliki tiga tahapan yaitu, pra-pengolahan *grid, online-search*, dan pembuatan jalur. Representasi ruang pada peta permainan menggunakan *Regular Grid* berbentuk persegi, karena representasi ruang pencarian berbasis *grids* sehingga mendukung adanya *random-acces lookup.* Perancangan algoritma HPA* akan dijabarkan pada diagram alir Gambar 4.2.

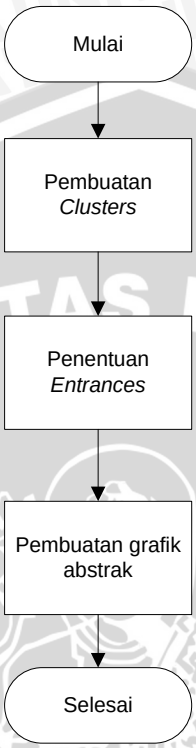


Gambar 4.2 Diagram Alir Implementasi HPA*

4.3.1 Pra-Pengolahan *Grid*

Pra-pengolahan *grid* adalah tahapan untuk membangun graf abstrak yang nantinya digunakan dalam *hierarchical search.* Tahapan dalam pra-pengolahan *grid* ada tiga yaitu, pembuatan *cluster*, penentuan entrance yang menghubungkan tiap

cluster, dan pembuatan graf abstrak. Perancangan pra-pengolahan *grid* akan dijabarkan pada diagram alir Gambar 4.3.



Gambar 4.3 Diagram Alir Pra-Pengolahan *Grid*

Adapun langkah-langkah pada diagram alir pra-pengolahan *grid* yang akan dijabarkan sebagai berikut :

1. Pembuatan *Clusters*

   Peta permainan akan direpresentasikan ke dalam *regular grid* berbentuk kotak. *Grid* akan dikelompokan ke dalam beberapa *cluster.* Misalkan isi setiap *cluster* berisi 10x10 grid.

2. Penentuan *Entrances*

   Setiap batas *cluster* akan dilakukan pemeriksaan untuk identifikasi kemungkinan pembuatan *entrance* yang menghubungkan *cluster* yang berdekatan. *Entrances* adalah segmen bebas hambatan sepanjang perbatasan *cluster* yang berdekatan. Dua *node grid* yang berdekatan dan bebas hambatan dari masing-masing *cluster* akan dihubungkan untuk membentuk *entrance.*

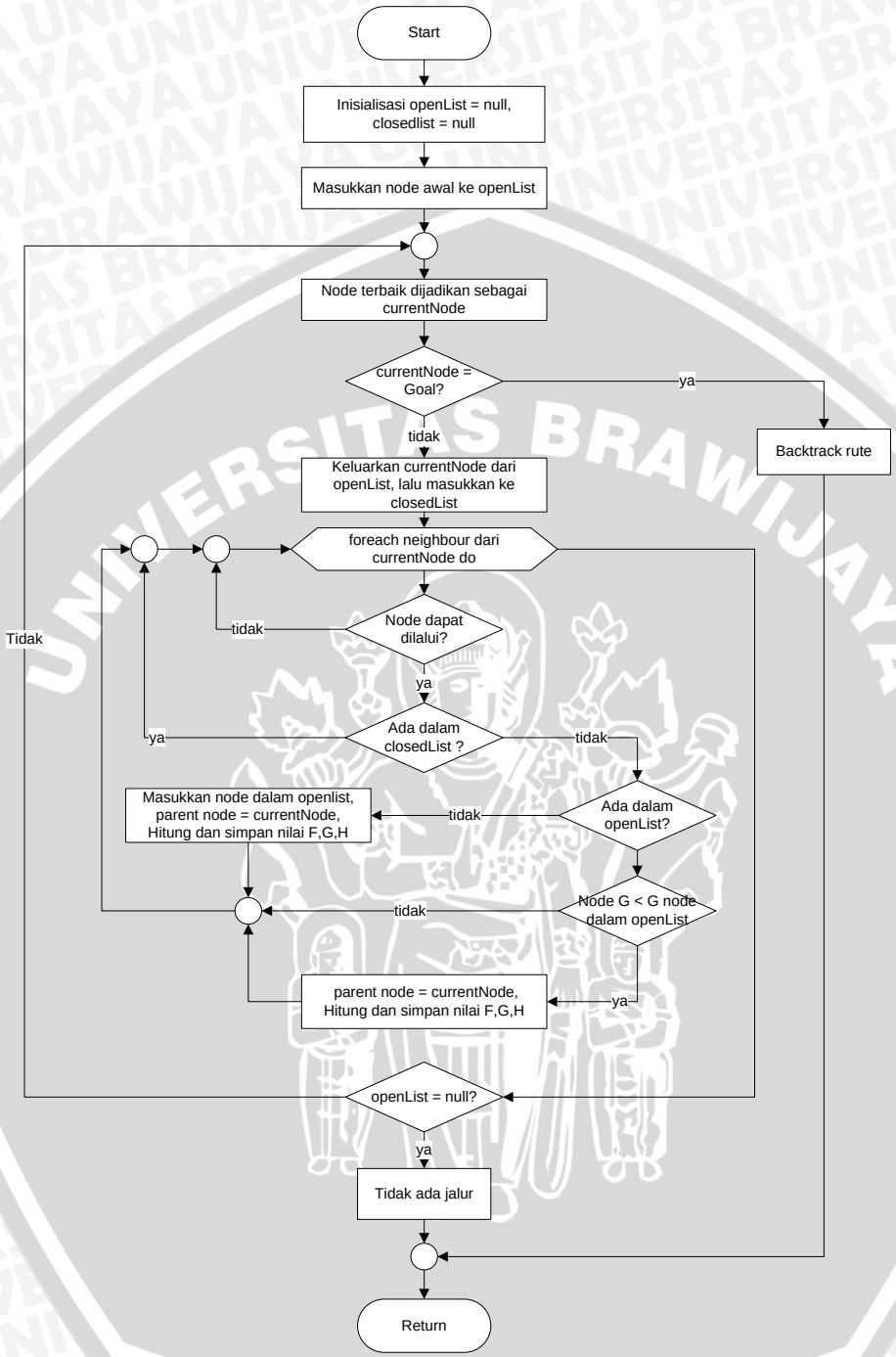
Jika *node grid* pada batas *cluster* yang memungkinkan untuk dibentuk *entrance* berjumlah kurang dari 6 maka dibuat *entrance* di tengah, sedangkan jika lebih dari 6 maka dibuat 2 *entrance* di masing-masing ujung.

3. Pembuatan graf abstrak

   Hubungkan *entrances* yang berada di *cluster* yang sama. Pencarian jalur untuk menghubungkan *entrances* menggunakan algoritma A* dengan langkah-langkah berikut :

   1) Tambahkan *node* awal ke dalam *open list*
   2) Mengulang langkah berikut:
      a) Mencari biaya F terendah pada setiap *node* dalam *open list. Node* dengan biaya F terendah akan disebut *current node*.
      b) Masukkan ke dalam *closed list*.
      c) Untuk setiap 8 simpul (*neighbour node*) yang berdekatan dengan current node:
         i. Jika tidak *walkable* atau jika termasuk *closed list,* maka abaikan.
         ii. Jika tidak ada pada *open list,* tambahkan ke *open list.*
         iii. Jika sudah ada pada open list, periksa apakah ini jalan dari *node* ini ke *current node* yang lebih baik dengan menggunakan biaya G sebagai ukurannya. Node dengan biaya G yang lebih rendah akan dibuat sebagai *parent node* dari *current node*, dan menghitung ulang nilai G dan F dari *node* ini.
      d) Pencarian jalur dihentikan ketika *current node* merupakan target point dalam hal ini jalan telah ditemukan atau gagal untuk menemukan titik tujuan, dan open list kosong.

Cara bekerja dari algoritma A* akan dijabarkan dalam diagram alir pada Gambar 4.4.

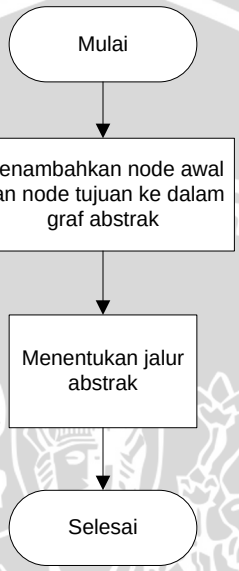


Gambar 4.4 Diagram Alir Algoritma A*

3) Jika jalur ditemukan tambahkan graf dan bobot graf yang dihitung dari panjang jalur ke graf abstrak.

4.3.2 Online Search

Pada *online search* terdapat dua tahap yaitu, menambahkan *node* awal dan *node* tujuan ke dalam graf abstrak, mencari jalur abstrak menggunakan graf abstrak. Perancangan *online search* akan dijabarkan pada diagram alir Gambar 4.5.

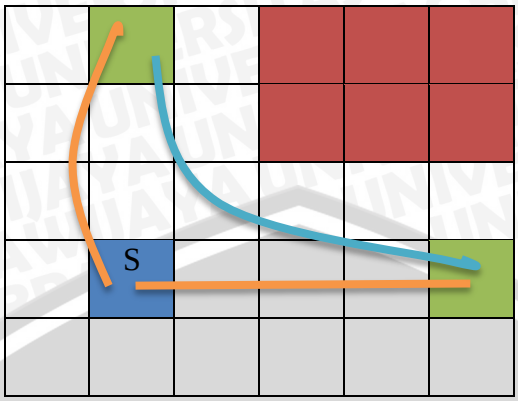


Gambar 4.5 Diagram Alir *Online Search*

Langkah-langkah pada diagram alir *online search* akan dijabarkan sebagai berikut:

1. Menambahkan *node awal* dan *node* tujuan ke graf abstrak

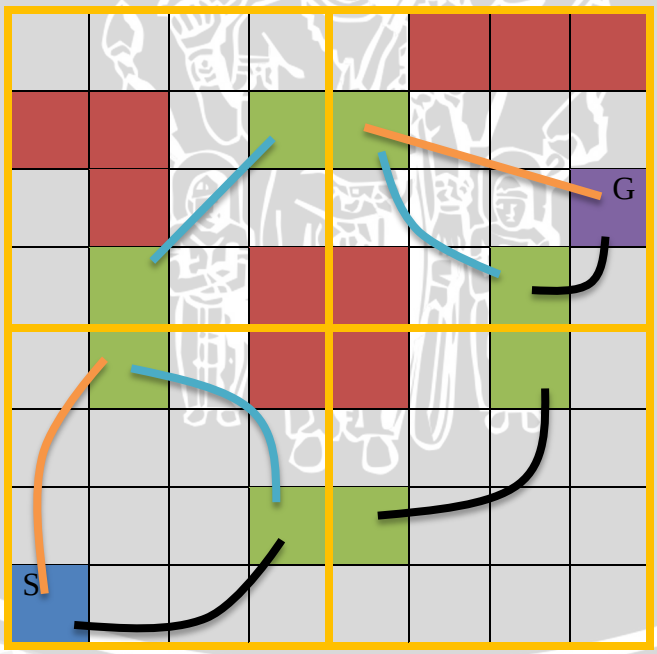
   Node awal dan *node* tujuan akan dihubungkan dengan setiap *entrance* yang ada pada *cluster* masing-masing seperti yang terlihat pada gambar 4.6. Kotak berwarna merah adalah *obstacle*, kotak hijau adalah *entrances*, kotak biru adalah node awal. Hubungkan dengan menggunakan algoritma A* dan simpan bobot dari setiap graf yang terhubung.



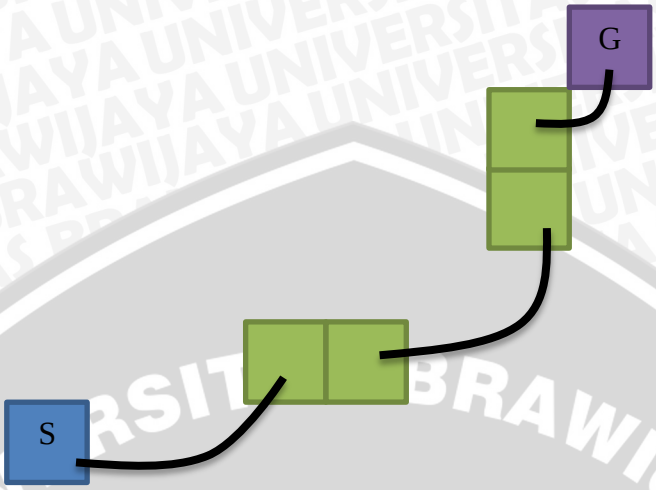
Gambar 4.6 Penambahan Node Pada Graf Abstrak

2. Menentukan jalur abstrak

   Setelah menambahkan *node* awal dan *node* tujuan pada graf abstrak, lalu lakukan pencarian pada graf abstrak menggunakan algoritma A* untuk menemukan jalur abstrak antara node awal dan node tujuan. Langkah-langkah menentukan jalur abstrak akan dijabarkan pada Gambar 4.7 dan Gambar 4.8.



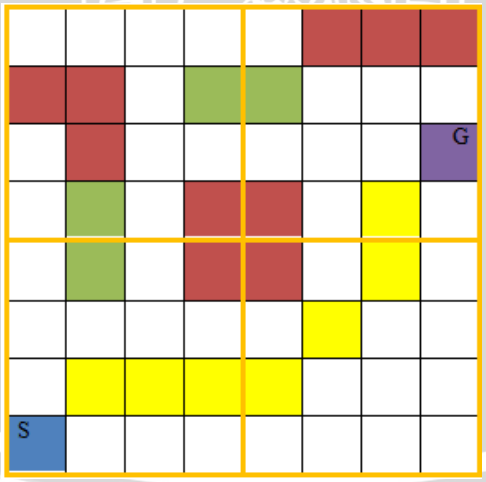
Gambar 4.7 Pencarian Jalur Abstrak



Gambar 4.8 Jalur Abstrak

4.3.3 Pembuatan Jalur

Setelah jalur abstrak ditemukan perlu dilakukan *refine path* untuk mendapatkan jalur konkret. *Refine path* dengan melakukan pencarian terhadap jalur abstrak menggunakan algoritma A* sehingga akan didapatkan jalur sebenarnya seperti pada Gambar 4.9.



Gambar 4.9 Kotak Berwarna Kuning Sebagai Jalur Konkret

4.3.4 Perhitungan Panjang Langkah

Berikut akan dilakukan perhitungan untuk menentukan panjang langkah menuju node yang tegak lurus dan yang diagonal. Ukuran peta adalah 60x60 *node grid*, diameter tiap node adalah 1. Karena units dapat bergerak ke segala arah (8 arah pergerakan) maka perhitungan panjang langkah menggunakan *Manhattan distance* dengan persamaan seperti pada persamaan(4).

$$h = \sqrt{(dx - sx)^2 + (dy - sy)^2} \quad \text{.....(4)}$$

Dimana h adalah panjang langkah, (dx,dy) adalah *node* tujuan dan (sx,sy) adalah *node* awal. Sehingga panjang langkah yang tegak lurus adalah 1 dan panjang langkah menuju node diagonal adalah$\sqrt{1^2 + 1^2} = \sqrt{2} = 1.4142$ dan dibulatkan menjadi 1,41.

BAB V
IMPLEMENTASI

Bab ini membahas mengenai implementasi algoritma berdasarkan hasil yang telah didapatkan dari analisis kebutuhan dan proses perancangan. Pembahasan terdiri dari penjelasan tentang spesifikasi sistem, batasan-batasan dalam implementasi, dan implementasi rancangan algoritma.

5.1 Penentuan Spesifikasi

Hasil analisis kebutuhan dan perancangan perangkat lunak yang telah dijelaskan pada bab 3 menjadi acuan untuk melakukan implementasi *pathfinding* pada *First Person Military Simulation Game* menggunakan *Hierarchical Pathfinding A** dapat berfungsi sesuai dengan kebutuhan. Spesifikasi sistem untuk implementasi algoritma dijelaskan pada spesifikasi perangkat keras dan perangkat lunak.

5.1.1 Spesifikasi Perangkat Keras

Spesifikasi perangkat keras yang dipakai dalam proses pengembangan dijelaskan dalam Tabel 5.1.

Tabel 5.1 Spesifikasi Perangkat Keras

Nama Komponen	Spesifikasi
Processor	Intel® Core™ i7-2630QM CPU @2.00GHz (8 CPUs), ~2.00GHz
Memory	4096MB RAM
Hardisk	500GB
Graphic Card	NVIDIA GeForce GT 540M, 1GB

5.1.2 Spesifikasi Perangkat Lunak

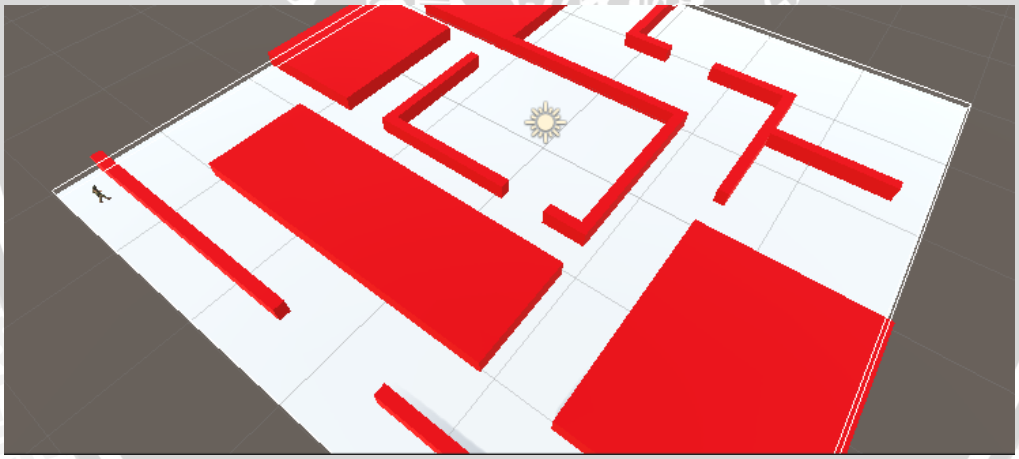
Spesifikasi perangkat lunak yang dipakai dalam proses pengembangan dijelaskan dalam Tabel 5.2.

Tabel 5.2 Spesifikasi Perangkat Lunak

Perangkat Lunak	Spesifikasi
Operating System	Windows 7 Home Premium 64-bit (6.1, Build 7600)
DirectX Version	DirectX 11
Programing Language	C#
Software Development Kit	Unity Version 5.0.0f4

5.2 Implementasi Peta Simulasi

Implementasi peta permainan untuk simulasi *pathfinding* akan langsung dirancang dalam Unity3D pada *editor scene view*. Objek-objek yang akan digunakan antara lain adalah *plate* dan *cube. Plate* akan digunakan sebagai lantai. *Cube* akan digunakan sebagai halangan statis. Hasil implementasi dari masing-masing peta ditunjukkan pada Gambar 5.1.



Gambar 5.1 Implementasi Peta Simulasi

5.3 Implementasi HPA* pada First Person Military Simulation Game

Implementasi algoritma AI diperlukan untuk menjadikan NPC dapat menemukan jalur terdekat ke titik tujuan. Oleh karena itu algoritma AI yang sudah dirancang akan diterapkan ke *First Person Military Simulation Game.*

5.3.1 Implementasi Pra-Pengolahan *Grid*

Implementasi pra-pengolahan *grid* akan ditunjukkan dalam Tabel 5.3.

Tabel 5.3 Pseudocode untuk pra-pengolahan *grid*

Pseudocode untuk Pra-Pengolahan Grid	
Deklarasi nodeDiameter, clusterDiameter, gridSizeX, gridSizeY, clusterSizeX, clusterSizeY	
1	Void Awake() {
2	CreateGrid();
3	CreateCluster();
4	InsertClusterNode();
5	VerticalEntrances();
6	HorizontalEntrance();
7	AbstractGraph ();
8	}

Penjelasan :

1. Baris 1 merupakan pemanggilan method Awake yang hanya dijalankan sekali pada awal *load scene.*
2. Baris 2 pemanggilan fungsi CreateGrid() untuk membuat grid pada peta.
3. Baris 3-4 pemanggilan fungsi CreateCluster untuk mengelompokkan grid ke dalam *cluster*.
4. Baris 5-6 pembuatan *entrances* dengan melakukan pemeriksaan secara vertikal dan horizontal.
5. Baris 7 untuk pembuatan graf abstrak.

Implementasi fungsi CreateGrid(), CreateCluster(), InsertClusterNode(),dan AbstractGraph() ditunjukkan dalam Tabel 5.4, 5.5, 5.6, dan 5.7.

Tabel 5.4 Pseudocode untuk CreateGrid

Pseudocode untuk CreateGrid()	
1	for (int x = 0; x < gridSizeX; x ++)
2	for (int y = 0; y < gridSizeY; y ++)
3	Vector3 worldPoint = worldBottomLeft + Vector3.right * (x * nodeDiameter + nodeRadius) + Vector3.forward * (y * nodeDiameter + nodeRadius) Bool walkable=
4	!(Physics.CheckSphere(worldPoint,nodeRadius, unwalkableMask))
5	grid[x,y] = new Node(walkable,worldPoint, x,y)
6	endfor
7	endfor

Penjelasan :

1. Baris 1-2 merupakan awal perulangan berdasarkan axis X dan axis Y *grid.*
2. Baris 3 merupakan inisialisasi posisi dan radius pada peta permainan dari *node grid.*
3. Baris 4 untuk pengecekan apakah *node grid* dapat dilalui atau tidak.
4. Baris 5 untuk pembuatan *node grid.*
5. Baris 6, akhir perulangan pada axis Y.
6. Baris 7, akhir perulangan pada axis X.

Tabel 5.5 Pseudocode untuk CreateCluster

Pseudocode untuk CreateCluster()	
1	`for (int x = 0; x < clusterSizeX; x ++)`
2	`for (int y = 0; y < clusterSizeY; y ++)`
3	`Vector3 cworldPoint = worldBottomLeft+Vector3.right* (x*clusterDiameter+clusterRadius)+Vector3.forward*(y *clusterDiameter +clusterRadius)`
4	`cluster[x,y] = new Cluster(cworldPoint, x,y)`
5	`endfor`
6	`endfor`

Penjelasan :

1. Baris 1-2 merupakan awal perulangan berdasarkan axis X dan axis Y *cluster.*
2. Baris 3, inisialisasi posisi dan radius *cluster*.
3. Baris 4, pembuatan *cluster.*
4. Baris 5, akhir perulangan axis Y.
5. Baris 6, akhir perulangan axis X.

Tabel 5.6 Pseudocode untuk InsertClusterNode

Pseudocode untuk InsertClusterNode()	
1	`for (int x = 0; x < clusterSizeX; x ++)`
2	`for (int y = 0; y < clusterSizeY; y ++)`
3	`for(inti=x*(int)clusterDiameter;i<(x*(int)clusterDiameter)+(int)clusterDiameter; i++)`
4	`for(intj=y*(int)clusterDiameter;j<(y*(int)clusterDiameter)+(int)clusterDiameter;j++)`
5	`if(i < gridSizeX && j < gridSizeY)`
6	`grid[i,j].clustX = x`
7	`grid[i,j].clustY = y`

```
8                                   end if
9                               endfor
10                          endfor
11                      endfor
12  endfor
```

Penjelasan :

1. Baris 1-2 untuk awal perulangan sebesar ukuran *cluster*.
2. Baris 3-4 untuk awal perulangan sebesar isi *cluster.*
3. Baris 5 merupakan pengecekan apakah koordinat x dan y melebihi ukuran *grid.*
4. Baris 6-7 untuk mengelompokkan grid ke dalam *cluster*.

Tabel 5.7 Pseudocode untuk AbstractGraph

```
Pseudocode untuk AbstractGraph()
1   for (int x = 0; x < clusterSizeX; x ++)
2      for (int y = 0; y < clusterSizeY; y ++)
3         foreach(Node pivot in cluster[x,y].entrances)
4               foreach(Node n in cluster[x,y].entrances)
5                    List<Node> edge = new List<Node>();
6                     if(n!=pivot)
7                        edge.Clear();
8                        edge = Connect(pivot,n);
9                     end if
10                    if(edge.Count > 0)
11                       pivot.graph.Add(new Graph(n,n.fCost))
12                    end if
13              end foreach
14         end foreach
15     end for
16  end for
```

Penjelasan :

1. Baris 1-2 merupakan awal perulangan sebesar ukuran *cluster.*
2. Baris 3 merupakan awal perulangan sebagai pivot pada setiap *entrance* di *cluster* yang sama.
3. Baris 4 merupakan awal perulangan pada setiap entrance di *cluster* yang sama.
4. Baris 5 untuk inisialisasi list edge untuk menyimpan jalur penghubung *entrances.*
5. Baris 6 melakukan pengecekan agar tidak menghubungkan *node* yang sama.
6. Baris 7-8 untuk menghubungkan entrances dengan method FindPath()

7. Baris 10-12, jika *path* ditemukan maka akan menambahkan graf dengan bobot sebesar panjang jalur yang ditemukan.

5.3.2 Implementasi *Online Search*

Implementasi *Online Search* dilakukan dengan method HPAfindPath yang ditunjukkan dalam Tabel 5.8

Tabel 5.8 Pseudocode untuk HPAfindPath

```
Pseudocode untuk HPAfindPath()
Deklarasi
openVertex, closedVertex, Path
Deskripsi
Input : startPos, targetPos
```

```
1  Node startNode = grid.NodeFromWorldPoint(startPos)
2  Node targetNode = grid.NodeFromWorldPoint(targetPos)
3  List<Node> Path = new List<Node> ()
4  InsertNode (startNode)
5  InsertNode (targetNode)
6  if (startNode.walkable && targetNode.walkable)
7     List<Node> openVertex = new List<Node> ()
8     List<Node> closedVertex = new List<Node> ()
9     openVertex.Add (startNode)
10     while (openVertex.Count > 0)
11       Node currentNode = openVertex [0]
12       for (int i = 1; i < openVertex.Count; i ++)
13           if(openVertex[i].fgCost<currentNode.fgCost||openVertex
   [i].fgCost==currentNode.fgCost&&openVertex[i].hCost<currentNode.
   hCost)
14               currentNode = openVertex [i]
15           end if
16       end for
17       openVertex.Remove (currentNode)
18       closedVertex.Add (currentNode)
19       if (currentNode == targetNode)
20          List<Node> AbsPath = new List<Node> ()
21          List<Node> temp = new List<Node> ()
22          AbsPath = RetracePath (startNode, targetNode)
23       for (int x = 1; x < AbsPath.Count; x++)
24          temp = FindPath (AbsPath [x - 1], AbsPath [x])
25             for (int y = 0; y < temp.Count; y++)
26                Path.Add (temp [y])
27             end for
28       end for
29       return Path
30       end if
31       foreach (Graph neighbour in currentNode.graph)
32          if (closedVertex.Contains (neighbour.point))
33             continue
34          end if
35          float newMovementCostToNeighbour = currentNode.ggCost
   + GetDistance(currentNode, neighbour.point)+neighbour.weight
```

```
            if (newMovementCostToNeighbour<neighbour.point.ggCost
||!openVertex.Contains (neighbour.point))
                neighbour.point.ggCost=newMovementCostToNeighbour
                neighbour.point.hCost=GetDistance(neighbour.point,
targetNode)
                neighbour.point.parent=currentNode
                if (!openVertex.Contains (neighbour.point))
                    openVertex.Add (neighbour.point);
                end if
            end if
        end foreach
    end while
end if
return Path
```

Penjelasan :

1. Baris 1 untuk mendapatkan node awal berdasarkan posisi titik awal dengan bantuan fungsi NodeFromWorldPoint.
2. Baris 2 untuk mendapatkan node tujuan berdasarkan posisi titik tujuan dengan bantuan fungsi NodeFromWorldPoint.
3. Baris 3 merupakan inisialisasi list Path.
4. Baris 4-5 untuk menambahkan node awal dan node tujuan ke dalam graf abstrak.
5. Baris 6 merupakan pengecekan sebelum pencarian agar node awal dan node tujuan dapat dilalui, jika tidak dapat dilalui maka tidak akan melakukan pencarian.
6. Baris 7-8 merupakan inisialisasi list openVertex dan list closedVertex.
7. Baris 9, menambahkan node awal ke dalam list openVertex.
8. Baris 10, permulaan while selama list openVertex tidak kosong
9. Baris 11-16, list index ke 0 dari openVertex dijadikan *current node* lalu lakukan pengecekan nilai fcost dari setiap node yang ada di openVertex. Jika fcost lebih rendah maka node akan dijadikan *current node*, jika nilai fcost sama maka akan dibandingkan nilai hcost mana yang lebih rendah untuk dijadikan *current node*.
10. Baris 17-18 merupakan remove current node dari openVertex dan tambahkan di closedVertex

11. Baris 19-30, cek apakah current node adalah tujuan, jika iya telusuri dengan method RetracePath(), lalu telusuri setiap node dari solusi abstrak dengan method FindPath() untuk mendapatkan solusi abstrak
12. Baris 31-44, cek setiap graph yang terhubung dengan current node. Jika graph yang terhubung terdapat pada closedVertex maka continue. Jika gcost dari current node kurang dari graph yang terhubung dan tidak terdapat di openVertex maka set gcost dan hcost dari graph dan atur parent dari graph menjadi current node.
13. Baris 47 mengembalikan list path yang sudah terbentuk.

BAB VI
PENGUJIAN DAN ANALISIS

Pada bab ini dilakukan proses pengujian dan analisis terhadap implementasi *pathfinding* pada *First Person Military Simulation Game* menggunakan *Hierarchical Pathfinding* A*. Pengujian dilakukan dengan menguji tiap skenario dan pencacatan hasil berdasarkan parameter pengujian. Parameter pengujian berupa panjang jalur yang dibuat NPC dan juga waktu pencarian yang dibutuhkan saat melakukan pencarian jalur.

6.1 Spesifikasi Pengujian

Spesifikasi perangkat keras yang digunakan dalam uji coba adalah procesor Intel Core i7, harddisk 500GB, memori 4GB RAM, kartu grafis NVIDIA GeForce GT 540M 2GB. Sedangkan spesifikasi perangkat lunak yang digunakan adalah Windows 7 Home Premium 64-bit, *game engine* Unity v5.0.

6.2 Menentukan Skenario Pengujian

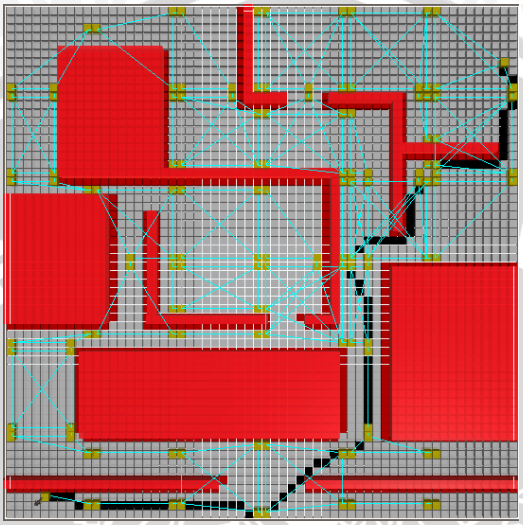
Pengujian dilakukan untuk mengetahui valid atau tidaknya *behaviour* yang sudah dirancang dengan implementasinya. Selain itu, pengujian juga dilakukan untuk mengetahui efisiensi dari sistem saat menjalankan real-time pathfinding. Efisiensi dari sistem dapat ditinjau dari panjang jalur yang dibuat dan waktu pencarian jalur. Skenario pengujian dilakukan dengan melakukan pencarian HPA* dengan ukuran *cluster* yang berbeda, yaitu 10x10, 15x15, 20x20, 30x30, dan 60x60 *grid.*

6.3 Melakukan Pengujian

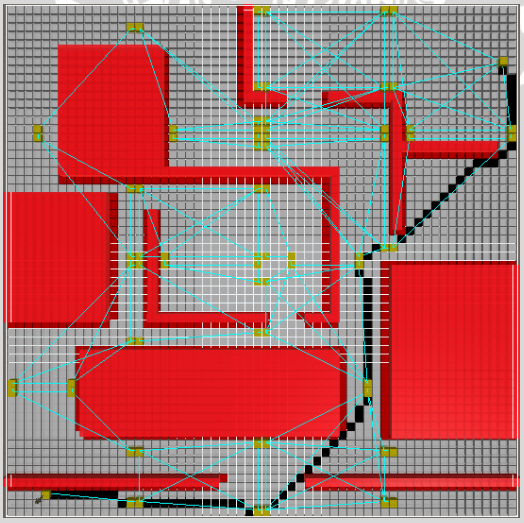
Pada subbab ini akan dijabarkan uji coba dan hasil pengujian dari penggunaan *cluster* dengan ukuran 10x10, 15x15, 20x20, 30x30, dam 60x60 *grid*.

Uji coba dilakukan dengan cara mengatur ukuran *cluster* yang digunakan pada tahap pra-pengolahan *grid.* Akan dilakukan pencarian jalur dengan menggunakan HPA* pada ukuran *cluster* 10x10, 15x15, 20x20, 30x30 dan 60x60 *grid*. Tujuan dari skenario ini untuk mengetahui valid atau tidaknya algoritma yang sudah dirancang dengan implementasinya dan untuk menguji efisiensi penggunaan ukuran *cluster*

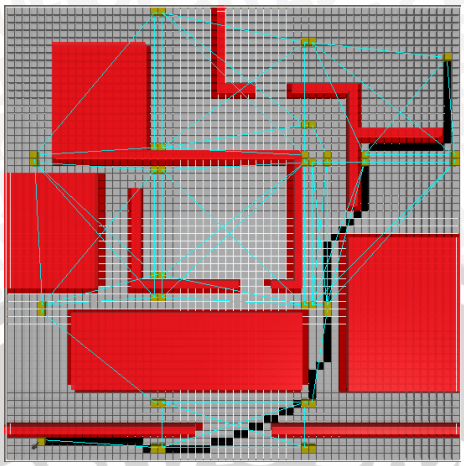
pada setiap uji coba. Akan dilakukan 10 kali percobaan dari masing-masing ukuran *cluster* yang akan merekam koordinat awal, koordinat akhir, apakah sampai tujuan, panjang jalur, dan waktu pencarian. Contoh pengujian yang dilakukan dapat dilihat pada Gambar 6.1,6.2,6.3,6.4 dan Gambar 6.5.



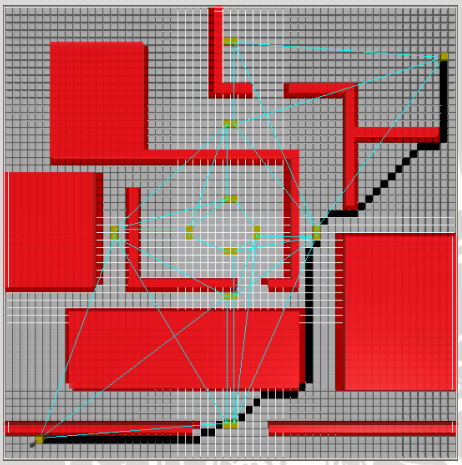
Gambar 6.1 Contoh Pengujian dengan *Cluster* 10x10 *Grid*



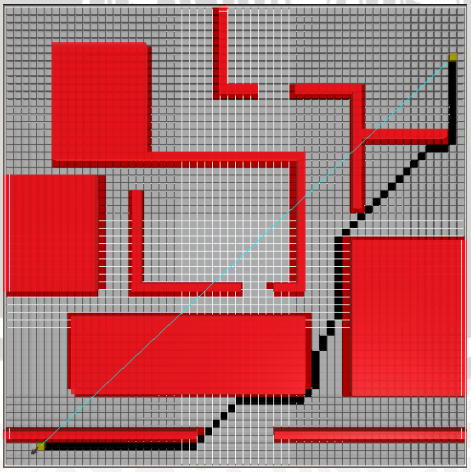
Gambar 6.2 Contoh Pengujian dengan *Cluster* 15x15 *Grid*



Gambar 6.3 Contoh Pengujian dengan *Cluster* 20x20 *Grid*



Gambar 6.4 Contoh Pengujian dengan *Cluster* 30x30 *Grid*



Gambar 6.5 Contoh Pengujian dengan *Cluster* 60x60 *Grid*

Hasil dari pengujian pada masing-masing ukuran *cluster* dapat dilihat pada Tabel 6.1 dan 6.2.

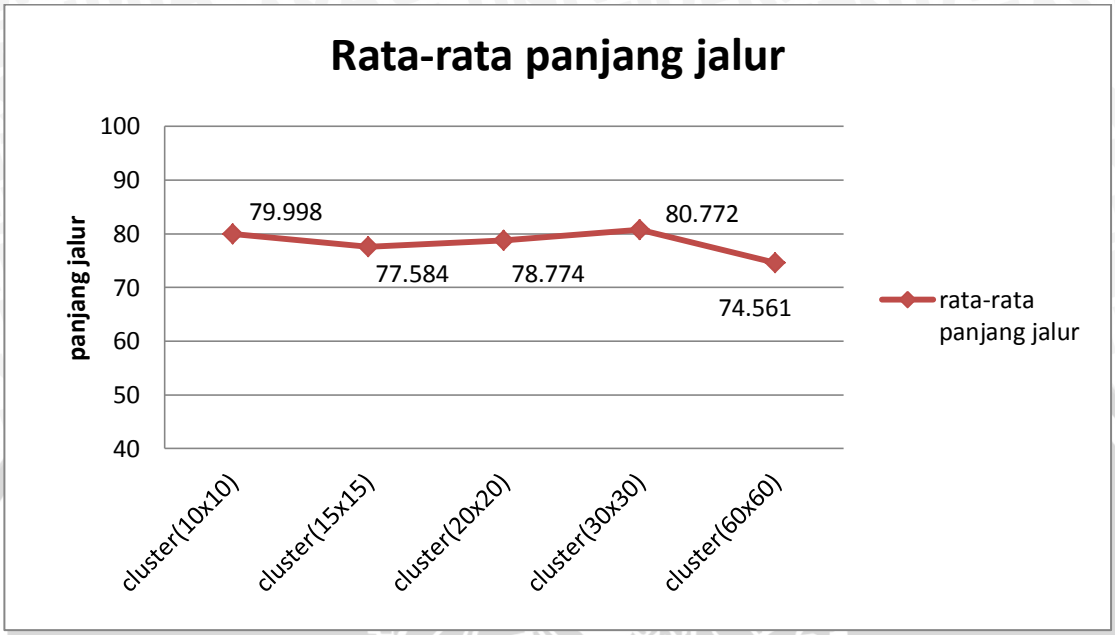
Tabel 6.1 Hasil Pengujuan Terhadap Panjang Jalur

No. Tes	Koordinat Awal	Koordinat Akhir	Sampai tujuan	Panjang jalur				
				Cluster (10x10)	Cluster (15x15)	Cluster (20x20)	Cluster (30x30)	Cluster (60x60)
1	4,2	25,34	Ya	78.11	75.18	77.28	85.04	71.28
2	4,2	22,56	Ya	103.11	100.18	99.94	117.46	95.7
3	4,2	53,52	Ya	107.18	97.08	98.7	93.77	92.01
4	4,2	24,57	Ya	103.28	100.36	100.11	119.87	95.87
5	4,2	14,37	Ya	77.38	76.21	81.04	75.38	75.38
6	4,2	33,5	Ya	33.07	33.07	31.07	30.24	30.24
7	4,2	4,16	Ya	52.97	53.56	52.38	51.56	51.56
8	4,2	56,8	Ya	56.14	56.14	55.31	54.49	54.49
9	4,2	36,38	Ya	80.04	76.53	79.21	73.21	73.21
10	4,2	0,59	Ya	108.7	107.53	112.7	106.7	105.87
			Rata-rata	79.998	77.584	78.774	80.772	74.561

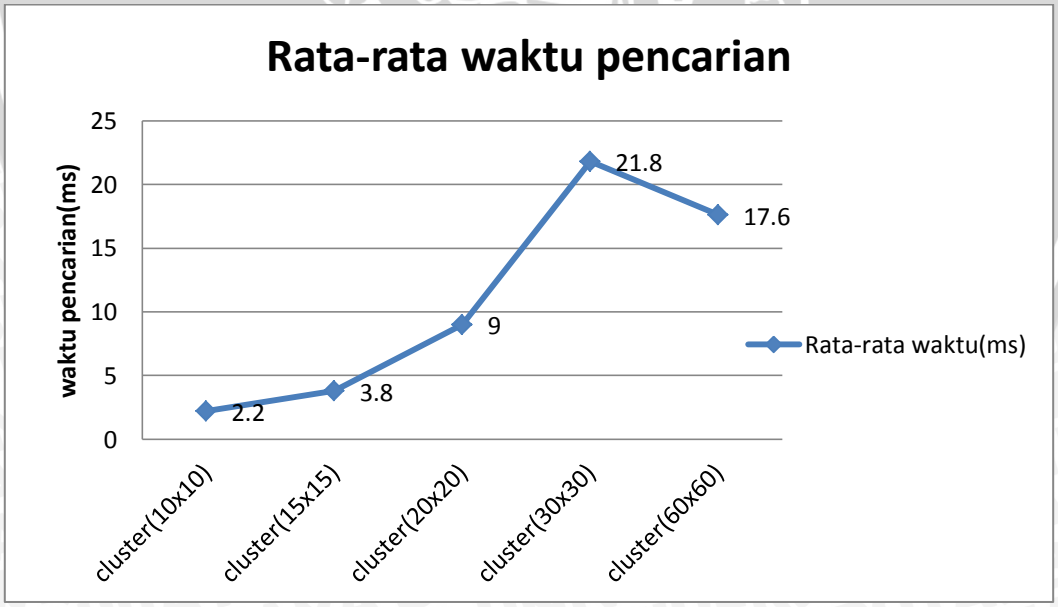
Tabel 6.2 Hasil Pengujuan Terhadap Waktu Pencarian

No. Tes	Koordinat Awal	Koordinat akhir	Sampai tujuan	Waktu pencarian(ms)				
				Cluster (10x10)	Cluster (15x15)	Cluster (20x20)	Cluster (30x30)	Cluster (60x60)
1	4,2	25,34	Ya	2	6	9	19	15
2	4,2	22,56	Ya	4	6	13	26	29
3	4,2	53,52	Ya	2	3	19	39	30
4	4,2	24,57	Ya	3	6	6	26	29
5	4,2	14,37	Ya	2	2	4	18	13
6	4,2	33,5	Ya	1	2	2	15	5
7	4,2	4,16	Ya	1	3	6	21	6
8	4,2	56,8	Ya	1	2	3	17	7
9	4,2	36,38	Ya	3	5	16	13	22
10	4,2	0,59	Ya	3	3	12	24	20
			Rata-Rata	2.2	3.8	9	21.8	17.6

Agar lebih mudah untuk melihat data maka tampilan grafik dari hasil pengujian skenario 1 ditampilkan pada Gambar 6.6, dan 6.7.



Gambar 6.6 Grafik Rata-rata Panjang Jalur



Gambar 6.7 Grafik Rata-rata Waktu Pencarian

6.4 Analisis Hasil Pengujian

Analisis bertujuan untuk mendapatkan kesimpulan dari hasil pengujian implementasi *pathfinding* menggunakan *Hierarchical Pathfinding* A*. Proses analisis terhadap hasil pengujian yaitu dilihat dari pembuatan jalur yang sampai tujuan, panjang jalur yang dihasilkan, waktu dalam pencarian jalur.

Pada uji coba panjang jalur, dari hasil pengujian panjang jalur dapat dilihat pada tabel 6.1 dan grafik pada gambar 6.6 bahwa 10 dari 10 uji coba yang dijalankan, semua dapat membuat jalur sampai tujuan dan panjang jalur yang dihasilkan pada *cluster* berukuran 30x30 mempunyai rata-rata panjang jalur paling tinggi daripada *cluster* berukuran 10x10,15x15,20x20, dan 60x60 dengan rata-rata panjang jalur yang dibuat adalah 80.772. Tetapi pada percobaan ke-5,6,7,8, dan 9 *cluster* berukuran 30x30 mendapatkan jalur yang optimal. Panjang jalur yang dibuat oleh *cluster* berukuran 60x60 dari seluruh percobaan menghasilkan panjang jalur yang optimal dengan rata-rata panjang jalur adalah 74.561. *Cluster* berukuran 10x10 pada percobaan ke-3,6,8, dan 9 menghasilkan jalur yang paling panjang diantara *cluster* yang lain. Rata-rata panjang jalur yang dibuat oleh *cluster* berukuran 10x10 adalah 79.998. Pada percobaan ke-6,7, dan 8 panjang jalur yang paling panjang dibuat oleh *cluster* berukuran 15x15. Sedangkan pada cluster berukuran 20x20 menghasilkan jalur yang tidak optimal pada percobaan ke-5 dan 10 dengan panjang jalur yang lebih panjang dibandingkan *cluster* yang lain. Rata-rata pajang jalur yang dibuat oleh *cluster* berukuran 15x15 adalah 77.584, dan rata-rata pajang jalur yang dibuat oleh *cluster* berukuran 20x20 adalah 78.774.

Dari data-data pada hasil pengujian panjang jalur dapat diketahui bahwa dari semua hasil pengujian (10 uji coba), semua berhasil membuat jalur menuju titik tujuan sehingga dapat disimpulkan bahwa implementasi *Hierarchical Pathfinding* A* pada *First Person Simulation Game* berhasil dilakukan. Selain itu, dari hasil semua uji coba dapat dilihat bahwa panjang jalur yang dihasilkan menggunakan *cluster* berukuran 30x30 rata-rata lebih panjang dibandingkan *cluster* berukuran 10x10, 15x15, 20x20 dan 60x60. Tetapi *cluster* 30x30 tidak selalu menghasilkan jalur yang

tidak optimal, *cluster* 30x30 tidak optimal pada percobaan ke-1,2 dan 4. *Cluster* 10x10 pada percobaan ke-3,6,8 dan 9. *Cluster* 15x15 pada percobaan ke-6,7,dan 8. *Cluster* 20x20 pada percobaan ke-5,dan 10. Rata-rata panjang jalur terpendek dihasilkan oleh *cluster* berukuran 60x60 dibandingkan *cluster* berukuran 10x10, 15x15, 20x20 dan 30x30.

Pada uji coba waktu pencarian, dari hasil pengujian waktu pencarian jalur dapat dilihat pada tabel 5.2 dan grafik pada gambar 5.4 dan 5.5 bahwa 10 dari 10 uji coba yang dilakukan dengan mencatat waktu yang dibutuhkan dalam pencarian jalur, rata-rata waktu yang dibutuhkan dalam pencarian paling lama adalah saat menggunakan *cluster* berukuran 30x30 dengan rata-rata waktu 21.8ms. Rata-rata waktu pencarian yang paling cepat adalah saat menggunakan *cluster* berukuran 10x10 dengan rata-rata waktu 2.2ms. Sedangkan rata-rata waktu pencarian yang dibutuhkan saat menggunakan *cluster* berukuran 15x15 adalah 3.8ms, saat menggunakan *cluster* berukuran 20x20 adalah 9ms,dan saat menggunakan *cluster* berukuran 60x60 adalah 17.6ms.

Dari data-data pada hasil pengujian waktu pencarian jalur dapat diketahui bahwa dari semua hasil pengujian (10 uji coba), dapat diketahui bahwa pencarian jalur yang dilakukan lebih cepat menggunakan *cluster* berukuran 10x10 dibandingkan dengan *cluster* berukuran 15x15, 20x20, 30x30, dan 60x60. Pencarian jalur yang dilakukan lebih lama menggunakan *cluster* berukuran 30x30 dibandingkan dengan *cluster* berukuran 10x10, 15x15, 20x20, dan 60x60. Tetapi pada percobaan ke-2,4 dan 9 waktu pencarian pada *cluster* berukuran 30x30 lebih kecil daripada *cluster* berukuran 60x60. Pada perhitungan waktu pencarian menggunakan *cluster* berukuran 60x60 lama waktu pencarian ditentukan berdasarkan jauh/dekatnya titik tujuan. Sehingga dapat disimpulkan bahwa besar ukuran *cluster* yang digunakan pada pencarian jalur dengan *Hierarchical Pathfinding* A* mempengaruhi waktu pencarian & panjang jalur yang dibuat.

BAB VI
PENUTUP

7.1 Kesimpulan

Berdasarkan hasil pengujian dan analisis yang dilakukan terhadap implementasi *pathfinding* pada *First Person Military Simulation* menggunakan *Hierarchical Pathfinding* A* dapat diambil kesimpulan sebagai berikut :

1. Implementasi *pathfinding* pada *First Person Military Game* berhasil dilakukan dengan menggunakan algoritma *Hierarchical Pathfinding* A*.
2. Implementasi HPA* dapat mempercepat waktu pencarian jalur.
3. Ukuran *cluster* pada HPA* mempengaruhi kecepatan pencarian dan panjang jalur yang ditemukan.
4. *Cluster* berukuran 10x10 memiliki kecepatan pencarian yang paling cepat jika dibandingkan dengan penggunaan *cluster* yang lain.
5. *Cluster* berukuran 60x60 memiliki panjang jalur yang paling optimal dibandingkan dengan penggunaan *cluster* yang lain.

7.2 Saran

Untuk meningkatkan hasil yang telah dicapai dari penelitian ini dapat dilakukan beberapa perbaikan sebagai berikut:

1. Diharapkan pada penelitian selanjutnya, penggunaan *pathfinding* dapat dikembangkan menjadi *tactical pathfinding* agar dapat mendeteksi faktor ancaman.
2. Modifikasi algoritma HPA* untuk menciptakan suatu algoritma baru yang lebih efisien dari segi panjang jalur yang dihasilkan.

DAFTAR PUSTAKA

[BOT-04] Botea, A., Muller, M., & Schaeffer, J. 2004, "Near Optimal Hierarchical Path-Finding", Department of Computing Science, University of Alberta.

[DEC-84] Dechter, Rina., & Pearl, Judea. 1984, "Generalized Best-First Search Strategies and The Optimality of A*", University of California.

[FAU-12] Fauzan, Ahmad. 2012, "Penggunaan Algoritma Pathfinding pada Game", Institut Teknologi Bandung.

[GRA-03] Graham, R., McCabe, H., Sheridan, S. 2003, "Pathfinding in Computer Games", Institute of Technology Blanchardstown.

[HAR-68] Hart, P., Nilsson, N., & Raphael, B. 1968, "A Formal Basis for Heuristic Determination of Minimum Cost Paths", IEEE Trans on Systems Science and Cybern, 4:100-107.

[HAR-08] Harabor, D., & Botea, A. 2008, "Hierarchical Path Planning for Multi-size Agent in heterogeneous environments", IEEE Symposium on Computational Intelligence and Games, Perth Australia.

[HSU-10] Hsu, Jeremy. 2010, "For the U.S. Military, Video Games Get Serious". http://www.livescience.com/10022-military-video-games.html. [21 Juni 2015].

[KOR-85] Korf, R. 1985, "Depth-first Iterative Deepening: An Optimal Admissible Tree Search", Department of Computer Science, Columbia University.

[LIN-13] Linus, van Elswijk. 2013, "Hierarchical Pathfinding Theta*", Radboud University Nijmegen.

[MIL-09] Millington, Ian And John Funge. 2009, Artificial Intelligence For Games, Second Edition, Morgan Kaufmann Publishers Inc., San Francisco, California.

[STO-96] Stout, B. 1996, Smart Moves: Intelligent Pathfinding, Game Developer Magazine.

[TIL-11] Tilawah, Hapsari. 2011, "Penerapan Algoritma A-star(A*) Untuk Menyelesaikan Masalah Maze", Institut Teknologi Bandung.

[TOZ-04] Tozour, Paul. 2004, Search Space Representation, AI Game Programming Wisdom 2, Charles River Media.

[TRY-14] Trybus, Jessica. 2014, "Game Based Learning: What it is, Why it Works, and Where it's Going". http://www.newmedia.org/game-based-learning--what-it-is-why-it-works-and-where-its-going.html. [23 Juni 2015].

[WAL-10] Walsh, M. 2010, Dynamic Navmesh – AI in the Dynamic Environtment of Splinter Cell: Conviction. Game Developer Conference.