TUGAS AKHIR – KM184801

KAJIAN ALGORITMA GENETIKA DAN ALGORITMA A* DALAM MENCARI RUTE TERPENDEK

DIMAS ARI NUGROHO
06111640000081

Dosen Pembimbing
Drs. Bandung Arry S., MI.KOMP.
M Luthfi Shahab, S.Si.,M.Si.

Departemen Matematika
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember
Surabaya
2020

TUGAS AKHIR - KM184801

KAJIAN ALGORITMA GENETIKA DAN ALGORITMA A* DALAM MENCARI RUTE TERPENDEK

DIMAS ARI NUGROHO
NRP 06111640000081

Dosen Pembimbing:
Drs. Bandung Arry S., MI.Komp
M Luthfi Shahab, S.Si, M.Si

DEPARTEMEN MATEMATIKA
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember
Surabaya 2020

FINAL PROJECT - KM4801

THE STUDY OF GENETIC ALGORITHM AND A* ALGORITHM FOR SEARCHING THE SHORTEST PATH

DIMAS ARI NUGROHO
NRP 06111640000081

Supervisors:
Drs. Bandung Arry S., MI.Komp
M Luthfi Shahab, S.Si, M.Si

DEPARTMENT OF MATHEMATICS
Faculty of Science and Data Analytics
Sepuluh Nopember Institute of Technology
Surabaya 2020

LEMBAR PENGESAHAN

KAJIAN ALGORITMA GENETIKA DAN ALGORITMA A* DALAM MENCARI RUTE TERPENDEK

THE STUDY OF GENETIC ALGORITHM AND A ALGORITHM FOR SEARCHING THE SHORTEST PATH*

Diajukan Untuk Memenuhi Salah Satu Syarat
Untuk Memperoleh Gelar Sarjana Matematika
Pada Bidang Studi Ilmu Komputer
Program Studi S-1 Departemen Matematika
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember Surabaya

Oleh:
Dimas Ari Nugroho
NRP. 06111640000081

Menyetujui,

Dosen Pembimbing II,	Dosen Pembimbing I,
M Luthfi Shahab, S.Si, M.Si	Drs. Bandung Arry S., MI.Komp
NIP. 19950331 201803 1 001	NIP. 19630605 198903 1 003

Mengetahui,
Kepala Departemen Matematika
FSAD ITS

Subchan,Ph.D.
NIP. 19700831 199403 1 003

Surabaya, 19 Agustus 2020

KAJIAN ALGORITMA GENETIKA DAN ALGORITMA A* DALAM MENCARI RUTE TERPENDEK

Nama Mahasiswa	:	Dimas Ari Nugroho
NRP	:	06111640000081
Departemen	:	Matematika FSAD-ITS
Pembimbing	:	1. Drs. Bandung Arry S., MI.Komp
		2. M Luthfi Shahab, S.Si, M.Si

Abstrak

Mencari rute terpendek merupakan masalah yang kerap kali dijumpai pada game yang terkait dengan Artificial Intelligence (AI). Tugas Akhir ini meneliti tentang bagaimana algoritma genetika dioptimalkan untuk mencari rute terpendek. Permasalahan ini akan divisualisasikan berupa prototype game yang disimulasikan menggunakan Unity 3D. Sedangkan sebagai pembanding Algoritma Genetika, Algoritma A dipilih karena merupakan algoritma eksak untuk mencari rute terpendek. Disini, graf input dipilih sebagai sampel permasalahan yang dibuat oleh peneiliti untuk membandingkan kedua algoritma. Hasil perbandingan dari segi waktu komputasi yang didapat adalah Algoritma Genetika adalah 1,029 detik sedangkan Algoritma A* adalah $1{,}565x10^{-3}$ detik. Dari segi memori Algoritma Genetika menghasilkan rata-rata memori sebesar 12,56 MB sedangkan Algoritma A* sebesar 9,12 MB. Dari segi jarak yang dihasilkan Algoritma Genetika adalah 1% mendekati jarak optimal yang dihasilkan Algoritma A* dengan masing-masing jarak adalah sebesar 36 dan 35,5.*

Kata kunci:*Algoritma genetika, Algoritma A*, Memori, Waktu, Jarak*

THE STUDY OF GENETIC ALGORITHM AND A* ALGORITHM FOR SEARCHING THE SHORTEST PATH

Name	:	Dimas Ari Nugroho
NRP	:	06111640000081
Department	:	Mathematics FSAD-ITS
Supervisors	:	1. Drs. Bandung Arry S., MI.Komp
		2. M Luthfi Shahab, S.Si, M.Si

Abstract

Find the shortest path is a problem which often encountered in games related to Artificial Intelligence(AI). This final project is studied how genetic algorithm optimized to find the shortest path. This problem which will be visualized in the form of a game prototype that will be simulated using Unity 3D. Meanwhile, as a comparison of the Genetic Algorithm, A Algorithm will be selected as an exact algorithm to find the graph searching. Here, the input of graph is selected as a sample of the problems that created by the researcher to compare both algorithms. The comparison result in terms of computation time obtained is the Genetic Algorithm is 1.029 seconds while Algorithm A* is 1,565x10^{-3} seconds. In terms of memory, the Genetic Algorithm produces an average memory of 12.56 MB while Algorithm A* is 9.12 MB. The result that generated by Genetic Algorithm is 1% near optimal to A* Algorithm, the optimal distance that produced by A* Algorithm is 35.5 and 36 by Genetic Algorithm.*

Keywords:*Genetic Algorithms, A* Algorithms, Memory, Time, Cost*

KATA PENGANTAR

Puji syukur penulis ucapkan kehadirat Allah SWT atas rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul "Kajian Algoritma Genetika dan Algoritma A* Dalam Mencari Rute Terpendek Pada Pathfinding Game" yang merupakan salah satu prasyarat akademis dalam menyelesaikan Program Sarjana Departemen Matematika FSAD Institut Teknologi Sepuluh Nopember Surabaya.

Penyelesaian penulisan Tugas Akhir ini tidak lepas dari orang-orang terdekat penulis yang telah mendukung dan memotivasi penulis. Oleh sebab itu, penulis mengucapkan terima kasih kepada:

1. Ayah, Mama, serta keluarga penulis yang selalu memberikan doa, semangat, dan dukungan dalam ibadah menuntut ilmu yang ditempuh oleh penulis selama ini.

2. Bapak Subchan, Ph.D. selaku Kepala Departemen Matematika FSAD ITS yang telah memberikan dukungan dan motivasi selama perkuliahan hingga selesainya tugas akhir ini.

3. Bapak Drs. Bandung Arry S., MI.Komp dan Bapak Muhammad Luthfi Shahab, S.Si, M.Si selaku Dosen Pembimbing yang telah memberikan bimbingan, arahan, dan motivasi kepada penulis dalam mengerjakan tugas akhir ini hingga dapat selesai dengan baik.

4. Bapak Dr. Imam Mukhlash, S.Si, MT, Bapak Dr. Budi Setiyono, S.Si, MT dan Bapak Dr. Chairul Imron, MI.Komp. selaku Dosen Penguji yang telah memberikan bimbingan, arahan, dan saran kepada penulis dalam mengerjakan tugas akhir ini.

5. Bapak Dr. Didik Khusnul Arif, S.Si, M.Si selaku Dosen Wali yang telah memberikan dukungan dan motivasi selama perkuliahan hingga selesainya tugas akhir.

6. Seluruh Bapak dan Ibu dosen Departement Matematikan ITS atas ilmu dan motivasi yang diberikan kepada penulis selama perkuliahan.

7. Seluruh Staf Departement Matematika ITS yang telah memberikan pelayanan terbaik kepada penulis selama perkuliahan hingga selesai.

8. Arek-arek joi, Pakde Joi, Bapak Nasi Goreng Joi, Bapak kumis penjaga gerbang ITS belakang, teman-temen SMA, serta seluruh teman yang telah memberi semangat, dukungan, dan memberikan doa-doa terbaik untuk penulis dan bersedia menjadi tempat curhat selama penulis menyelesaikan tugas akhir ini.

9. Setiap orang yang pernah memberikan dukungan, masukan, saran, semangat, cerita hidup serta hal-hal baik yang tidak akan pernah dilupakan penulis sehingga dapat menyelesaikan tugas akhir ini dengan baik, semoga kalian semua diberi keberkahan dan rezeki dalam segala hal.

10. Teman-teman Matematika ITS 2016 "LEMNISCATE" yang telah memberikan banyak cerita selama perkuliahan dan banyak sekali yang tidak bisa disebut satu-persatu.

    Penulis menyadari bahwa dalam penyusunan Tugas Akhir ini masih terdapat kekurangan, sehingga penulis mengharapkan kritik dan saran dari semua pihak demi kesempurnaan Tugas Akhir ini. Semoga Tugas Akhir ini dapat bermanfaat.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

Terdapat banyak algoritma untuk melakukan pencarian rute terpendek. Pemilihan algoritma yang paling optimum selalu menjadi permasalahan dalam pencarian rute terpendek, dimana masing-masing algoritma memiliki kelebihan dan kekurangannya masing-masing. Dalam masalah ini tidak dapat dikatakan secara langsung algoritma mana yang paling baik secara keseluruhan artinya setiap algoritma memiliki kelebihan dan kekurangannya msing-masing. Optimasi yang mencakup efisiensi waktu proses kerja algoritma, waktu tempuh yang diperlukan untuk mencapai tujuan akhir dan jarak tempuh yang paling pendek ini selalu tergantung dari setiap kondisi permasalahan yang ada, dan kondisi yang paling mempengaruhi adalah banyak titik[1].

Perkembangan dunia industri game saat ini membutuhkan algoritma pathfinding yang paling optimal dalam memecahkan masalah, hal ini dikarenakan sumber daya mereka telah berkembang secara kompleks. Untuk memecahkan masalah algoritma pathfinding, industri game telah membuat beberapa penelitian terkait dengan algoritma pathfinding. Akhirnya, industri game tersebut menemukan solusi dari permasalahan pathfinding berupa *Navigation Mesh* yang merupakan suatu teknik untuk menyelesaikan permasalahan pathfinding pada game dengan menggunakan poligon. Karena kesederhanaan dan efisiensi tinggi dalam menyelesaikan permasalahan pada game pathfinding, *Navigation Mesh* telah menjadi pilihan yang

paling mainstream untuk menyelesaikan permasalahan tersebut. Akan tetapi, *Navigation Mesh* ini sendiri hanya bisa memilih jalan yang masuk akal untuk setiap objek yang bergerak[4]. Sehingga dikarenakan keterbatasan tersebut, pada penelitian kali ini penulis berencana mengambil konsep *Navigation Mesh* untuk diimplementasikan pada algoritma genetika untuk mencari rute terpendek.

Algoritma A Star atau A* adalah salah satu algoritma pencarian yang menganalisis input, mengevaluasi sejumlah jalur yang mungkin dan menghasilkan solusi. Algoritma A* adalah algoritma komputer yang digunakan secara luas dalam grafik traversal dan pencarian jalur seiring dengan proses perencanaan jalur yang efisien di sekitar titik yang disebut node[10]. Algoritma Dijkstra menggunakan prinsip serakah, di mana setiap langkah dipilih dengan bobot minimum yang menghubungkan simpul yang dipilih dengan simpul yang tidak dipilih lainnya [11]. Sementara *Breadth First Search* adalah metode yang melakukan pencarian yang lebih luas yang memperluas sebuah node secara pre-order, memperluas sebuah node dan kemudian memperluas semua tetangga dari node terakhir. Setelah itu, memperpanjang *node* yang tidak diperpanjang dan berdekatan dengan node yang diperluas, dan seterusnya[12].

Pathfinding merupakan proses menentukan gerakan beberapa objek ke posisi lain tanpa terjadi tumbukan antara objek satu dengan objek yang lainnya[4]. Sebenarnya, metode ini tidak hanya digunakan dalam permainan dengan konsep *Artificial Intelligence*, tetapi juga digunakan pada bidang lain seperti sistem informasi geografis(GIS), Robotika, dan statistik. Beberapa teori telah disampaikan untuk memecahkan masalah pathfinding, seperti finite state Machine (FSM), graf, Dijkstra, dan A*. FSM adalah algoritma yang paling sederhana yang dapat diimplementasikan dalam

masalah pathfinding ini tetapi cara mengubungkan antar solusi terlalu luas dan gerakannya cukup mudah diprediksi. Sedangkan Algoritma lain yang dapat diimplementasikan dalam masalah pathfinding adalah graf. Graf sendiri membuat beberapa *waypoints* tergantung pada bobot setiap graf. Selain graf, ada juga algoritma Dijkstra dimana algoritma ini menggunakan prinsip yang serakah dimana maksud dari prinsip ini adalah, ketika setiap langkah kita memilih sisi yang memiliki bobot paling minimum dan bobot tersebut yang diambil pada setiap langkah akan dimasukkan kedalam satu set solusi pada algoritma ini[4].

Pada penelitian[5], mereka mengusulkan pendekatan yang bertujuan untuk mengurangi jumlah node yang diakses menggunakan ruang dengan algoritma genetika. Dibandingkan dengan algoritma A* pada umumnya yang digambarkan dengan berpola hambatan, sebelumnya metode mereka menunjukkan hasil yang optimal, tetapi dalam pelaksanaan tanpa pola hambatan apa pun, hasilnya jauh di bawah standar, sehingga tidak cocok untuk digunakan pada banyak kasus. Berbeda dengan RTP-GA, pendekatan baru mereka memastikan kerugian minimal dibandingkan dengan algoritma BFS di suatu *case* tanpa pola, dan juga mempertahankan keunggulan algoritma A* yang optimal pada lintasan dengan banyak hambatan.

Pada penelitian kali ini, penulis akan menggunakan meneliti Algoritma A* dan Algoritma Genetika untuk mencari rute terpendek. Output yang diharapkan adalah untuk mengetahui perbandingan Algoritma Genetika dan Algoritma A* untuk menyelesaikan permasalahan ini dari segi jarak, waktu komputasi, dan memori yang dibutuhkan sehingga akan diketahui mana algoritma yang lebih baik untuk menyelesaikan permasalahan ini.

1.2 Rumusan Masalah

Rumusan masalah dalam tugas akhir ini adalah: Mengkaji Algoritma A* dan Algoritma Genetika untuk mencari rute terpendek dan membandingkan kedua algoritma tersebut?

1.3 Batasan Masalah

Batasan masalah yang digunakan dalam Tugas Akhir ini adalah:

1. Perbandingan kedua Algoritma akan ditinjau berdasarkan waktu komputasi, memori yang dibutuhkan, dan jarak yang diambil untuk menyelesaikan permasalahan.

2. Permasalahan yang digunakan untuk menguji Algoritma Genetika dan Algoritma A* sebanyak 8 jenis input.

1.4 Tujuan

Tujuan dari pengerjaan Tugas Akhir ini adalah: Mengetahui hasil perbandingan Algoritma Genetika dan Algoritma A* untuk mencari rute terpendek.

1.5 Manfaat

Manfaat dari pengerjaan Tugas Akhir ini adalah:

1. Manfaat pada penelitian ini adalah untuk mengetahui seberapa optimal Algoritma Genetika terhadap Algoritma A* dalam mencari rute terpendek.

2. Peneliti lain dapat menggunakan Algoritma yang lebih baik untuk mencari rute terpendek berdasarkan hasil perbandingan kedua algoritma.

1.6 Sistematika Penulisan

Sistematika penulisan Tugas Akhir ini adalah sebagai berikut :

1. BAB I PENDAHULUAN

   Bab ini menjelaskan latar belakang penyusunan penulisan, rumusan masalah, Batasan masalah, tujuan, manfaat, dan sistematika penulisan.

2. BAB II TINJAUAN PUSTAKA

   Bab ini menjelaskan beberapa cara kerja, penelitian terdahulu dan algoritma yang digunakan sebagai acuan dalam pembahasan serta untuk membantu memahami permasalahan yang akan dibahas.

3. BAB III METODOLOGI

   Bab ini menjelaskan tahapan yang dilakukan dalam penyusunan penulisan.

4. BAB IV ANALISIS DAN PEMBAHASAN

   Bab ini menjelaskan secara detil mengenai hasil penelitian.

5. BAB IV KESIMPULAN DAN SARAN

   Bab ini menjelaskan kesimpulan yang diperoleh dari pembahasan masalah pada bab sebelumnya serta saran yang diberikan penulis untuk pengembangan penelitian selanjutnya.

BAB II
TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Pada tahun 2016, Moh Zikky dari *Game Technology Study Program, Multimedia Creative Department* Politeknik Elektronika Negeri Surabaya, melakukan penelitian dengan judul *Review of A* Navigation Mesh Pathfinding as the Alternative of Artificial Intelligent for Ghosts Agent on the Pacman Game*. Penelitian ini meneliti tentang masalah pencarian rute terpendek dimana masalah ini menjadi salah satu masalah yang cukup populer pada game dengan konsep *Artificial Intelligence*(AI). Penelitian ini mendiskusikan mengenai optimasi masalah yang berkaitan dengan menemukan jalur terpendek, atau yang sering disebut *Navigation Mesh*.Metode yang digunakan pada penelitian ini memiliki cakupan yang sangat luas untuk implementasinya khususnya pada dunia game.Pada penelitian ini pun, *Navigation Mesh* diimplementasikan menggunakan Algoritma A* yang diuji pada *Unity 3D Game Engine*.Algoritma A* ini sendiri merupakan algoritma yang paling efektif dalam menentukan jalur terpendek karena optimasinya menggunakan cara lacak yang efektif dengan segmentasi garis.Pac-Man game dipilih sebagai sampel penentuan jalur terpendek menggunakan *Navigation Mesh* pada *Unity 3D*.Sedangkan Algoritma A* itu sendiri diimplementasikan pada musuhnya Pac-Man(*three ghosts*), yang mana lintasan tersebut didesain dengan menggunakan konsep*Navigation Mesh*.Dengan demikian, gerakan dari *three ghosts*ketika mengejar Pac-Man merupakan review hasil dari pengujian

konsep ini.

Pada tahun 2007 Ryan Leigh dkk meneliti tentang bagaimana Algoritma Genetika diimplementasikan sebagai Algoritma pathfinding pada permainan lagoon angkatan laut. Game ini sendiri didesain 3 dimensi agar tampilannya mudah untuk dipahami dari sisi pemahaman implementasi algoritma.Game ini diimplementasikan sebagai strategi real-time angkatan laut 3D sebagai simulasi permainan dan pelatihan.Algoritma A* ini sendiri merupakan algoritma yang cukup mahal dari sisi biaya pada penggunaannya karena ketika dijalankan dengan objek yang cukup banyak, maka A* ini akan dengan mudahnya kehilangan validitas lintasan yang telah ditinjau. Meskipun banyak literatur yang menyebut bahwa algoritma ini begitu baik pada implementasinya, namun penulis memiliki pendapat lain.Dan pada paper ini pula, penulis menggunakan algoritma genetika untuk mencari ruang pada suatu jaringan seperti yang dilakukan oleh algoritma A* untuk menemukan algoritma pathfinding yang lebih optimal.

Pada tahun 2011, Alex Ferndandes dkk meneliti tentang pathfinding secara *real-time* dengan menggunakan Algoritma Genetika. Penelitian ini menyajikan metode untuk mengoptimalkan proses mencari jalur menggunakan model berdasarkan Algoritma Genetika dan A* untuk sistem real time, seperti video game, lingkungan *virtual reality*.Solusi yang diusulkan menggunakan pendeteksian pola kendala berdasarkan sistem pelatihan online yang umumnya digunakan dalam sistem *real-time* dengan persyaratan dalam lingkungan yang dinamis. Arsitektur bernama *real-time pathfinding* dengan Algoritma Genetika(RTP-GA), menggunakan algoritma ini untuk membuat agen yang disesuaikan dengan lingkungan dimana agen tersebut mampu mengoptimalkan pencarian jalur bahkan di hadapan

rintangan. Dalam kasus tertentu, arsitektur RTP-GA menghadirkan kompleksitas yang lebih baik dari Algoritma A*.

Pada tahun 2012, Alex Fernandes dkk melakukan penelitian guna menyajikan metode baru untuk mengoptimalkan proses menemukan jalur menggunakan model berdasarkan Algoritma Genetika dan *Best-First-Search* untuk sistem *real-time*, seperti video game dan realitas virtual lingkungan.Solusi yang diusulkan menggunakan hambatan deteksi pola berdasarkan pada sistem pelatihan online untuk menjamin efektifitas memori.Arsitektur bernama Pathfinding berbasis *Patterned with Genetic Algorithm*(PPGA) menggunakan teknik pembelajaran agar membuat agen yang disesuaikan dengan lingkungan yang mampu untuk mengoptimalkan pencarian jalur bahkan di ketika dihadapkan pada suatu hambatan. Penelitian ini menunjukkan bahwa arsitektur PPGA berkinerja lebih baik daripada A* pada umumnya dan *Best-First-Search* pada lingkungan yang berpola.

Pada tahun 2011, Xiao Cui dan Hao Shi meneliti tentang beberapa algoritma yang populer berdasarkan optimasi yang mengacu pada algoritma A*.Hal ini menunjukkan bahwa relasi yang bersih antara Algoritma A* dengan beberapa algoritma yang identik.Penelitian ini juga mengacu pada inti dari algoritma *pathfinding* itu sendiri merupakan *puzzle* yang belum tersusun rapi pada game berbasis *Artificial Intelligence*. Tantangan yang paling mendasar pada penelitian ini adalah bagaimana menggunakan Algoritma A* untuk memecahkan masalah yang rumit.

2.2 *Unity 3D*

Unity merupakan suatu *tools* permainan 2D dan 3D yang tersedia pada *Multiplatform.Game Engine* merupakan suatu alat untuk mendesain atau membuat suatu permainan dengan menerapkan konsep yang telah ditetapkan agar game tersebut berjalan sebagaimana mestinya.*Unity 3D* memiliki suatu agen yang telah diprogram menggunakan A* artinya ketika kita ingin melakukan pencarian graf pada suatu permasalahan labirin, agen ini dapat melakukan pencarian secara otomatis dengan menggunakan Algoritma A*[6].

2.3 *Pathfinding*

Pathfinding adalah suatu metode yang digunakan untuk pencarian jalur yang bertujuan untuk mencapai suatu tempat tujuan dari titik awal ke titik tujuan.Cara kerja *pathfinding* adalah dengan mencari sebuah grafik dengan memulai pada satu titik dan menjelajahi node yang berdekatan sampai node tujuan tercapai, umumnya dengan maksud untuk menemukan rute terpendek.Namun, tidak perlu untuk memeriksa semua jalur yang mungkin untuk menemukan satu jalur yang optimal.Dalam sistem *pathfinding* kompleksitas waktu yang lebih baik dapat dicapai dengan algoritma yang dapat pra-proses grafik untuk mencapai kinerja yang lebih baik[7].

2.4 Algoritma Genetika

Gagasan dibalik Algoritma Genetika adalah memodelkan evolusi alami dengan menggunakan pewarisan genetik[14].Ada dua hal pokok yang harus dipikirkan sebelum menggunakan Algoritma Genetika, yang pertama adalah representasi kromosom dan yang kedua adalah pemilihan fungsi *fitness*.Kromosom tersebut akan menjadi solusi untuk mencari rute optimal dan fungsi *fitness* akan menjadi alat untuk mengukur kekuatan dari suatu individu.

Algoritma genetika dimulai dengan dugaan populasi yang merepresentasikan kromosom-kromosom sebagai solusi permasalahan.Biasanya dugaan tersebut dipilih secara acak yang tersebar di seluruh search space.Algoritma genetika kemudian menggunakan tiga operator penting yaitu seleksi, *crossover*, dan mutasi untuk mengolah populasi hingga konvergen atau menemukan hasil yang terbaik[13].

Seleksi berusaha untuk memberikan tekanan pada populasi seperti pada seleksi alami yang ditemukan pada sistem biologis.Kromosom yang buruk akan terbuang dan kromosom yang baik akan memiliki kesempatan yang lebih besar untuk berkembang pada generasi berikutnya.*Crossover* memungkinkan pertukaran informasi pada kromosom untuk membentuk kromosom-kromosom baru. Mutasi digunakan agar informasi-informasi yang ada dalam kromosom tidak mudah hilang[13].

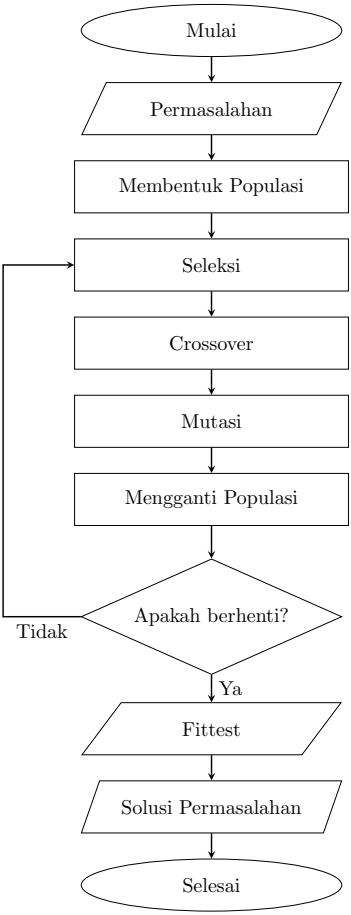
Setelah seleksi, *crossover*, dan mutasi selesai dilakukan pada populasi awal, maka populasi baru akan terbentuk dan generasi dari populasi baru tersebut telah naik satu tingkat.Proses seleksi, crossover, dan mutasi terus dilanjutkan hingga jumlah generasi tertentu telah dilampaui atau kriteria pemberhentian dipenuhi[13].Langkah-langkah Algoritma Genetika adalah seperti dalam Gambar 1.

2.4.1 Random Selection

Random selection adalah salah satu operator seleksi yang bekerja dengan memilih kromosom dalam populasi secara acak baru dilakukan *Crossover* untuk didapatkan individu yang lebih baik.

2.4.2 One-Point Crossover

One-Point Crossover adalah teknik crossover dengan mengambil 1 titik yang ada sepanjang jumlah panjang kromosom orang tua.Crossover ini menggunakan dua orang



Gambar 2.1: Langkah-langkah Algoritma Genetika

tua untuk menjalankan crossover dan menghasilkan dua keturunan. *One-Point Crossover* menghasilkan keturunan yang didapat dari hasil persilangan *gene* kedua orang tua dan menghasilkan sepasang individu *offspring*.Contoh *One-Point Crossover* dapat dilihat pada Gambar 2.2.

orang tua 01	01	10	10	11	11	00	01	10	11
orang tua 00	11	11	11	10	01	01	11	01	11
anak	01	01	10	10	10	01	01	11	01

Gambar 2.2: Contoh *One-Point Crossover*

2.4.3 Desain Awal Kromosom

Dalam penelitian ini, kromosom direpresentasikan sebagai array, dengan panjang kromosom sebagai banyaknya dimensi pada array.Sehingga, desain awal kromosom bisa digambarkan sebagai berikut:

x_1	x_2	x_3	...	x_n

Gambar 2.3: Contoh Desain awal kromosom

Dengan n sebagai jumlah dimensi yang menyatakan jumlah node yang diambil untuk menentukan rute terpendek.

2.5 Algoritma A*

Algoritma A* dikembangkan oleh Hart, Nilsson, dan Raphael, algoritma ini paling popular dan telah digunakan secara luas dalam graph traversal, penemuan jalur, serta proses perencanaannya[2].

Algoritma A* adalah yang paling banyak digunakan dalam peta permainan, yang menggunakan fungsi heuristik untuk memperkirakan jarak setiap titik ke titik target, sehingga dapat mengurangi ruang pencarian dan meningkatkan efisiensi pencarian [8].

Algoritma A* sendiri merupakan algoritma pencarian heuristik yang mengevaluasi setiap simpul berdasarkan nilai perkiraan ke titik akhir ($h(n)$) dan jarak sebenarnya ($g(n)$) pada saat *node* ke-n. Pencarian ini adalah untuk mengevaluasi

setiap simpul ke-n, dan pilih node terbaik untuk menentukan tujuan berikutnya, kemudian cari mulai dari *node* awal hingga menemukan *node* yang menjadi tujuan. Dalam pencarian heuristik, senarai *Open List* dan *Closed List* sangat penting untuk mengestimasi biaya pemindahan dari simpul saat ini ke *node* tujuan. Fungsi mendapatkan harga dari Algoritma A* pada saat *node* ke-n dinyatakan sebagai:

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

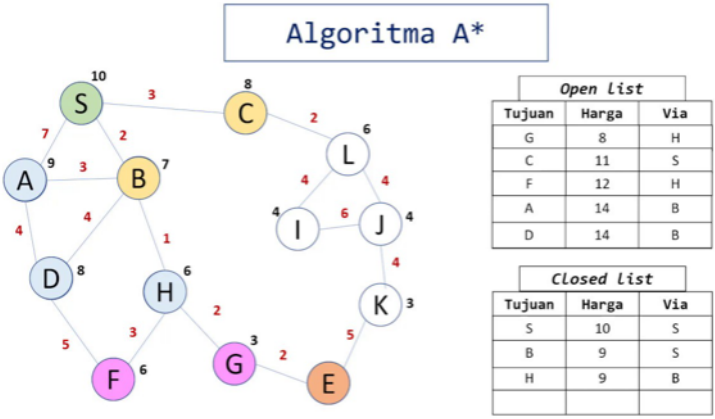
Keterangan dari persamaan diatas adalah sebagai berikut:

- f(n) = total biaya(jarak) dari titik awal ke titik tujuan.
- h(n) = perkiraan biaya(jarak) dari node n ke tujuan akhir.
- g(n) = jarak sebenarnya antara titik yang dilalui dengan titik awal.
  Berikut adalah contoh tampilan pada Algoritma A*:

Kompleksitas waktu dari A* sangat bergantung dari heuristik yang digunakannya. Pada kasus terburuk, jumlah titik yang diperiksa berjumlah eksponensial terhadap panjang solusi (jalur terpendek), tetapi A* akan memiliki kompleksitas waktu polinomial apabila fungsi memenuhi kondisi berikut:

$$|h(x) - h*(x)| \leq O(\log h*(x))$$

Dimana h* adalah heuristik optimal, yaitu ongkos sebenarnya dari node awal ke tujuan. Dengan kata lain, galat dari h tidak akan melaju lebih cepat dari logaritma dari heuristik optimal h* yang memberikan jarak sebenarnya dari x ke tujuan [18].



Gambar 2.4: Contoh Algoritma A*

2.6 Fungsi Heuristik

Algoritma A* adalah algoritme pencarian yang menggunakan fungsi heuristik untuk 'menuntun' pencarian rute, khususnya dalam hal pengembangan dan pemeriksaan node-node pada peta[9]. Fungsi heuristik yang akan digunakan untuk mencari rute terpendek adalah fungsi *Manhattan* yang menjumlahkan selisih dua titik x dan y. Dalam notasi matematika dapat ditulis fungsi d sebagai berikut :

$$d(x, y) = \sum_{i=1}^{m} |x_i - y_i|$$

Sedangkan apabila fungsi *Manhattan* ditulis dalam bentuk *pseudocode* dapat ditulis sebagai berikut :

h(n)= abs(tujuan.x-n.x)+abs(tujuan.y -n.y)

Dimana h(n) merupakan perkiraan cost dari node n ke node tujuan yang dihitung dengan fungsi heuristik. Variabel n.x merupakan koordinat x dari node n, sedangkan n.y merupakan koordinat y dari node n. Variabel tujuan.x merupakan koordinat x dari node tujuan dan tujuan.y merupakan koordinat y dari node tujuan. Nilai dari h(n) akan selalu bernilai positif.

BAB III
METODE PENELITIAN

3.1 Langkah-Langkah Penelitian

Pada bab ini dilakukan pembahasan tentang metode penelitian yang digunakan dalam Tugas Akhir agar proses pengerjaan dapat terstruktur dengan baik.

1. Studi literatur
   Dilakukan studi literatur untuk mendukung pengerjaan Tugas Akhir dan pemahaman yang lebih mendalam mengenai algoritma genetika khususnya untuk optimasi fungsi. Literatur yang dipelajari dapat bersumber dari jurnal, buku, internet, maupun bimbingan dengan dosen pembimbing.

2. Membuat beberapa input yang akan diselesaikan dengan Algoritma Genetika dan Algoritma A*
   Selanjutnya adalah membuat beberapa input yang akan diselesaikan dengan kedua algoritma.

3. Mendesain Algoritma A* untuk mencari rute terpendek
   Setelah mendapatkan input untuk diselesaikan, selanjutnya Algoritma A* akan didesain untuk melakukan pencarian rute terpendek.

4. Mendesain Algoritma Genetika untuk mencari rute terpendek
   Setelah mendapatkan input untuk diselesaikan, selanjutnya Algoritma Genetika akan didesain untuk melakukan pencarian rute terpendek.

5. Implementasi Program
   Setelah melakukan kajian tentang algoritma genetika, selanjutnya akan dilakukan implementasi pada program dengan menggunakan bahasa pemrograman C# yang ada pada Unity 3D.

6. Mendapatkan hasil running program pada Algoritma A* dan Algoritma Genetika
   Hasil running program pada simulasi akan digunakan sebagai parameter perbandingan untuk kedua algoritma.

7. Membandingkan Algoritma A* dan Algoritma Genetika
   Membandingkan hasil analisa kedua Algoritma berdasarkan *Running Time*, Memori yang dibutuhkan, dan jarak yang ditempuh oleh masing-masing algoritma.

8. Penarikan Kesimpulan
   Penarikan kesimpulan dilakukan dengan mendapatkan hasil dan pembahasan yang telah diselesaikan pada tahap-tahap sebelumnya.

BAB IV
ANALISIS DAN PEMBAHASAN

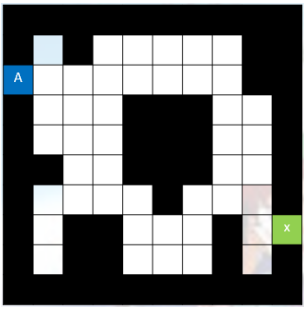
4.1 Membuat beberapa input yang akan diselesaikan dengan Algoritma Genetika dan Algoritma A*

Input yang digunakan berupa matriks dengan representasi bilangan *integer* dengan ukuran n x n. Adapun ketentuan dari *integer* pada matriks permasalahan direpresentasikan adalah sebagai berikut :

- $0 =$ *open space*
- $1 =$ *wall*
- $5 =$ *start*
- $8 =$ *finish*

Permasalahan input ini dibatasi sebanyak delapan permasalahan yang akan diselesaikan dengan kedua algoritma untuk dicari jalur terpendeknya dari beberapa alternatif yang ada. Setiap permasalahan tersebut memiliki tingkat kesulitan yang berbeda-beda bergantung pada alternatif jalur yang ada untuk menuju ke titik akhir dan banyaknya *node*. Setiap permasalahan yang sudah diketahui jalur optimalnya akan diselesaikan dengan Algoritma A*, hal ini karena Algoritma A* adalah ***complete*** dan ***optimal*** sehingga nanti akan diketahui seberapa baik atau buruk Algoritma Genetika mencari rute terpendek berdasarkan perhitungan presentase jarak yang sudah ditentukan oleh nilai *P*. Nilai *P* mengertikan seberapa baik atau buruk Algoritma Genetika terhadap Algoritma A* juga untuk mengetahui berapa persen

Algoritma Genetika mendekati keoptimalan Algoritma A*. Karena Algoritma A* merupakan Algoritma eksak untuk mencari rute terpendek sedangkan Algoritma Genetika ini merupakan algoritma pendekatan maka untuk memunculkan kesimpulan yang seharusnya pada setiap permasalahan diperlukan nilai P untuk mengartikan seberapa optimal Algoritma Genetika terhadap Algoritma A*. Berikut adalah contoh input pada permasalahan 1 pada Gambar 4.1. Dari



Gambar 4.1: Contoh input pada permasalahan 1

Gambar 4.1 didapatkan bahwa input pada permasalahan 1 terdapat 100 node dengan 2 alternatif jalur untuk menuju ke titik tujuan. Setiap input permasalahan akan semakin banyak *node* yang harus diselesaikan, artinya secara analisis semakin banyaknya node maka kedua algoritma akan menghasilkan waktu dan memori yang semakin bertambah juga. Secara umum, untuk mengetahui hasil perbandingan rute terpendek kedua algoritma akan dilihat jalur mana yang akan diambil oleh kedua algoritma tersebut untuk mencapai ke titik tujuan.

4.2 Mendesain Algoritma A* untuk mencari rute terpendek

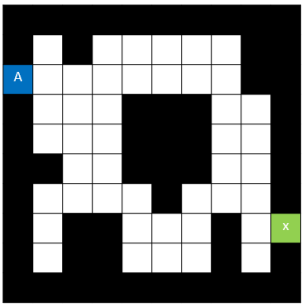
Berikut merupakan desain Algoritma A* untuk mencari rute terpendek dalam bentuk *pseudocode* pada Tabel 4.1.

```
functions A*(Shortest Path) returns solusi
Deklarasi start node, goal
  open←start node
  close←array kosong
loop ketika masih ditemukan suksesor
  if open=kosong then
    loop sampai ditemukan suksesor berikutnya
    if nilai f suksesor ¡= node saat ini then
      masukkan suksesor ke open
    end
    best node=simpul pada open dengan f terkecil
    Pindah simpul terbaik dari open ke close
    if best node=goal then
       Sukses
    end
    loop untuk setiap node terdekat
       if node berikutnya adalah wall atau sudah dicek
          teruskan pengulangan
       end
       g(suksesor)=g(best node)+act cost(best node ke suksesor)
       if f lebih besar dari g(best node) then
          ganti nilai g(suksesor) menjadi f(best node)
          ganti nilai h dari node suksesor ke goal
          ganti act node menjadi parent
             if suksesor tidak ada di open list
                tambahkan node suksesor ke open
             end
       end
    end
```

Algoritma A* merupakan algoritma yang melakukan pencarian pada graf dengan mengevaluasi setiap simpul pada saat *node* ke n berdasarkan jarak sebenarnya dari titik

awal($g(n)$) dengan jarak perkiraan terhadap titik akhir($h(n)$). Untuk nilai ($h(n)$) dihitung dengan menggunakan fungsi jarak *manhattan* dengan *node obstacle* tidak dihitung sebagai koordinat yang akan dilewati. Setelah merancang *pseudocode* Algoritma A* kemudian akan diimplementasikan dengan bahasa pemrograman. Penyelesaian pada permasalahan 1 sebagai contoh bagaimana Algoritma A* menyelesaikan input yang telah ditetapkan dan berlaku juga pada permasalahan yang lain.

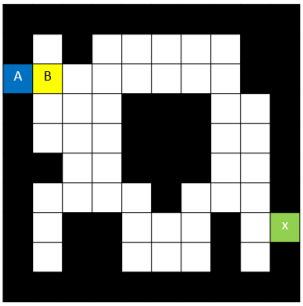
1. Diambil contoh pada permasalahan 1 seperti pada Gambar 4.2 dibawah ini.Fungsi Algoritma A* dinotasikan dengan f(n)=h(n)+g(n) dimana g(n) adalah jarak *node* saat ini dengan *node* awal sedangkan h(n) adalah perkiraan jarak *node* saat ini dengan *node* tujuan.



Gambar 4.2: Permasalahan 1

2. Menentukan *node parent* untuk melanjutkan ke *node* suksesor berikutnya dalam hal ini misal A adalah *Start Point* dan x adalah *Exit Point*. *Node* A merupakan *Closed* karena dia adalah node pertama kali yang dikembangkan oleh Algoritma A*. Untuk

proses perhitungan Algoritma A* dan *node* yang masuk dalam senarai *close list* dan *open list* akan dijelaskan pada Gambar 4.3. Fungsi Algoritma A* ini adalah

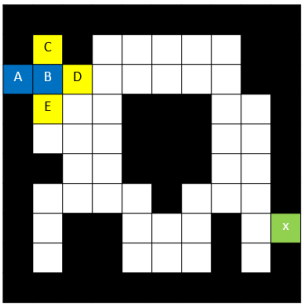


Gambar 4.3: Step 1 Penyelesaian Algoritma A*

untuk menentukan node yang menjadi *parent* untuk suksesor *node* berikutnya. Misalkan jarak antar *node* ini sama dengan 1, artinya nilai dari g(b)=1 dimana g(b) ini adalah jarak sebenarnya dari *node* A. Kemudian nilai h(b) didapatkan dengan menghitung nilai heuristik dari *node* B ke *node* tujuan, dalam hal ini koordinat titik x adalah (2,1) dan koordinat titik tujuan adalah (7,9) sehingga untuk mendapatkan nilai h akan dihitung dengan menggunakan fungsi *Manhattan Distance* $d = |7-2| + |9-1| = 13$. Setelah kita mendapatkan nilai g(b) dan h(b) maka baru selanjutnya akan ditentukan *cost* algoritma untuk bergerak dari *node* A ke *node* B yakni sama dengan f(b)=1+13 didapatkan nilai f(b)=14. Dari iterasi ini didapat *open list*=[B] dan *closed list*=[A].

3. Menentukan *node* suksesor dari B untuk melanjutkan ke *node* suksesor berikutnya dalam hal ini misal B adalah *parent* lalu *node* C, D, dan E adalah suksesor berikutnya

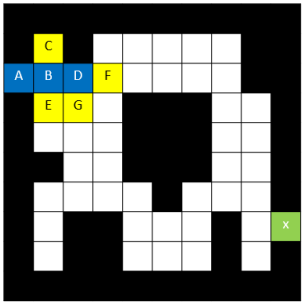
yang akan masuk dalam kategori *open list*. Untuk visualisasi langkah ini akan disajikan pada Gambar 4.4. Dengan menggunakan cara yang sama seperti pada



Gambar 4.4: Step 2 Penyelesaian Algoritma A*

nomor 2 akan didapatkan nilai f(c), f(d), dan f(e) dengan nilai g(n) adalah 2 karena nilai tersebut dihitung dari *node start*. Dari hasil perhitungan dengan menggunakan fungsi Algoritma A* didapatkan bahwa nilai dari f(c), f(d), dan f(e) masing-masing adalah 14, 12, dan 12. Oleh karena nilai f dari *node* d dan e adalah yang paling kecil maka *node* tersebut berhak untuk menjadi parent untuk melanjutkan ke *node* suksesor berikutnya. Dari hasil iterasi ini didapatkan *open list*=[D,E,C] dan *closed list*=[A,B].

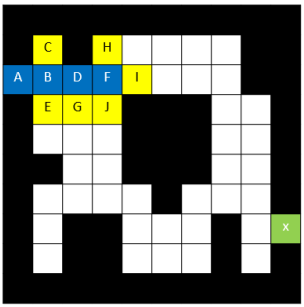
4. Menentukan *node* suksesor dari D untuk melanjutkan ke *node* suksesor berikutnya dalam hal ini misal D adalah *parent* lalu *node* F dan G adalah suksesor berikutnya yang akan masuk dalam kategori *open list*. Untuk visualisasi langkah ini akan disajikan pada Gambar 4.5. Dengan menggunakan cara yang sama seperti pada nomor 3 akan didapatkan nilai f(f) dan



Gambar 4.5: Step 3 Penyelesaian Algoritma A*

f(g) dengan nilai g(n) adalah 3 karena nilai tersebut dihitung dari *node start*. Dari hasil perhitungan dengan menggunakan fungsi Algoritma A* didapatkan bahwa nilai dari f(f) dan f(g) masing-masing adalah 14 dan 16. Oleh karena nilai f dari *node* f adalah yang paling kecil maka *node* tersebut berhak untuk menjadi parent untuk melanjutkan ke *node* suksesor berikutnya. Dari hasil iterasi ini didapatkan *open list*=[F,G] dan *closed list*=[A,B,D].

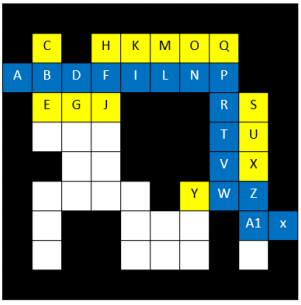
5. Menentukan *node* suksesor dari F untuk melanjutkan ke *node* suksesor berikutnya dalam hal ini misal F adalah *parent* lalu *node* h, i dan j adalah suksesor berikutnya yang akan masuk dalam kategori *open list*. Untuk visualisasi langkah ini akan disajikan pada Gambar 4.6. Dengan menggunakan cara yang sama seperti pada nomor 3 akan didapatkan nilai f(h), f(i) dan f(j) dengan nilai g(n) adalah 4 karena nilai tersebut dihitung dari *node start*. Dari hasil perhitungan dengan menggunakan fungsi Algoritma A* didapatkan bahwa nilai dari f(h), f(i) dan f(j) masing-masing adalah 16,



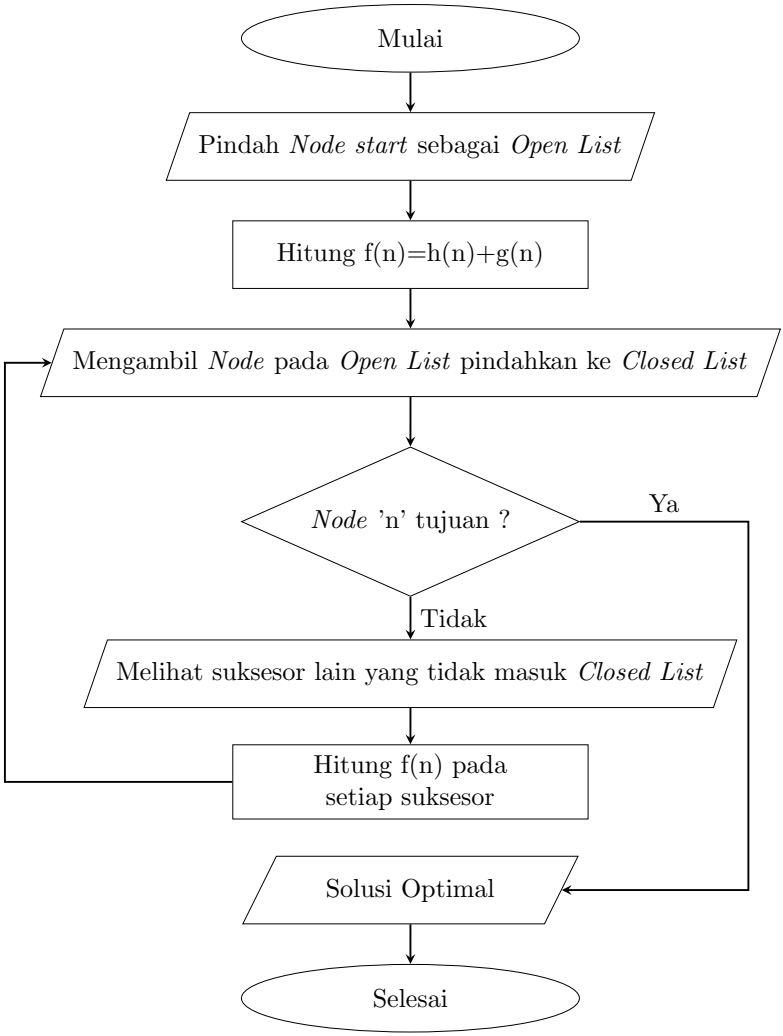
Gambar 4.6: Step 4 Penyelesaian Algoritma A*

14 dan 14. Oleh karena nilai f dari *node* i adalah yang paling kecil maka *node* tersebut berhak untuk menjadi parent untuk melanjutkan ke *node* suksesor berikutnya. Dari hasil iterasi ini didapatkan *open list*=[H,I,J] dan *closed list*=[A,B,D,F].

6. Dengan menggunakan cara yang sama seperti pada nomor 3 didapatkan *path* yang akan dilewati pada *closed list*=[A,B,D,F,I,L,N,P,R,T,V,W,Z,A1]. Untuk hasil *path* akan ditunjukkan pada Gambar 4.7.



Gambar 4.7: Step 5 Penyelesaian Algoritma A*



Gambar 4.8: Langkah-langkah Algoritma A*

4.3 Mendesain Algoritma Genetika untuk mencari rute terpendek

Algoritma Genetika yang digunakan berjenis *Population Replacement*, proses *replacement* dilakukan ketika iterasi berhenti. Karena representasi kromosom yang digunakan adalah kromosom biner, maka tidak menjamin bahwa suatu individu terbaik dalam satu generasi jika digunakan pada generasi selanjutnya akan menghasilkan individu yang berkualitas. Algoritma Genetika untuk mencari rute terpendek dalam bentuk *pseudocode* dapat dilihat pada Tabel 4.2.

Tabel 4.1: *Pseudocode* Algoritma Genetika

```
Bangkitkan penyelesaian N pencarian
Loop sampai goal ditemukan
    Loop untuk N pencarian
        Dekodekan pencarian untuk membaca arah gerak
        Evaluasi individu dengan meminimalkan jarak Manhattan
    End
        Buat satu atau dua penyelesaian terbaik
    Loop sampai didapatkan N pencarian baru
        Pilih dua pencarian sebagai parent1 dan parent2 secara acak
        [baby1, baby2]=One-Point Crossover(parent1, parent2)
        [baby1, baby1]=Mutasi tingkat bit(baby1, baby2)
    End
    Ganti N pencarian lama dengan N pencarian baru
End
```

Selanjutnya akan dijelaskan beberapa komponen pada Algoritma Genetika, yaitu: skema pengkodean, nilai *fitness*, seleksi orang tua, pindah silang(*crossover*), mutasi, penggantian populasi, dan kriteria pemberhentian.

1. Skema pengkodean yang digunakan untuk mendesain kromosom adalah *Binary Encoding*. Kromosom menyatakan 4 arah yang akan dilalui oleh Algoritma Genetika untuk menentukan arah lintasan yang akan diambil. Berikut pada Tabel 4.3 adalah desain kromosom untuk mencari rute terpendek berdasarkan arah pergerakan yang telah ditetapkan.

   Tabel 4.2: Desain Kromosom

Arah 1	Arah 2	Arah 3	.....	Arah ke-n

   Pada masalah ini, panjang kromosom menyatakan jumlah langkah yang akan diambil menuju ke titik tujuan. Untuk mencapai ke titik tujuan, maka setiap pergerakan yang diambil harus merupakan jalur yang bisa dilewati. Pada setiap arah pergerakan, direpresentasikan dengan bilangan biner 0 atau 1[16]. Berikut representasi arah pada kromosom agar mencapai ke titik tujuan:

   - ▷ 00: 0, bergerak ke koordinat (-1,0) atau Arah 1
   - ▷ 01: 1, bergerak ke koordinat (0,1) atau Arah 2
   - ▷ 10: 2, bergerak ke koordinat (1,0) atau Arah 3
   - ▷ 11: 3, bergerak ke koordinat (0,-1) atau Arah 4

2. Fungsi *fitness* digunakan untuk menentukan seberapa baik individu untuk mencapai ke titik tujuan. Fungsi *fitness* yang digunakan adalah fungsi jarak *Manhattan* untuk mencari rute terpendek pada setiap permasalahan dengan meminimalkan jarak x dan y ke titik tujuan. Nilai *fitness* dari suatu individu dikatakan optimal apabila nilai *fitness* yang diperoleh sama dengan 1[17]. Berikut perhitungan nilai *fitness* untuk meminimalkan jarak:

$$F(x,y) = \frac{1}{|jarakx|+|jaraky|+1}$$

Keterangan dari fungsi *fitness* diatas adalah sebagai berikut :

- ▷ jarak x: Selisih koordinat n dan koordinat akhir x
- ▷ jarak y: Selisih koordinat n dan koordinat akhir y

3. Tahap ini bertujuan untuk membangkitkan sebuah populasi yang berisi sejumlah kromosom yang telah ditentukan banyaknya. Misal banyaknya kromosom dalam populasi awal pada level 1 telah ditentukan sebanyak 4 individu, maka individu yang terbentuk akan diinisiasi secara acak berdasarkan tetapan yang telah ditetapkan. Untuk representasi individu ditampilkan seperti pada Tabel 4.4.

Tabel 4.3: Populasi Awal

Kromosom	Representasi Kromosom
Individu 1	01 01 01 01 01 01 01 10 01 01 00 01 01 01
Individu 2	01 10 01 01 10 10 10 10 10 10 10 01 10 01
Individu 3	01 00 00 00 11 01 01 10 10 10 10 01 10 01
Individu 4	01 00 00 00 11 00 00 00 10 10 10 01 10 01

Individu yang ada pada suatu populasi ini merupakan penyelesaian yang terdiri dari beberapa kromosom yang telah ditetapkan. Selanjutnya, Individu ini akan dilakukan perhitungan nilai *fitness* untuk diketahui seberapa baik kualitas individu tersebut untuk menemukan jalur terpendek. Pada perhitungannya didapatkan nilai fitness individu 1, 2, 3 dan 4 masing-masing adalah 0,167, 0,2, 0,067 dan 0,067. Dari hasil perhitungan nilai *fitness* ini, individu yang memiliki

nilai *fitness* terbaik akan memiliki peluang lebih besar untuk mendapatkan solusi yang diinginkan.

4. Seleksi yang digunakan dalam algoritma genetika adalah pemilihan dua kromosom dalam populasi secara acak. Hal ini dilakukan karena pemilihan dua kromosom yang baik tidak menjamin akan didapatkannya keturunan yang baik pula. Misal pada suatu populasi terdapat sekumpulan individu sabagai berikut, maka akan dilakukan pemilihan individu secara acak untuk dilakukan tahap berikutnya yaitu *Crossover*.

5. *Crossover* yang digunakan adalah *One-Point*, *crossover* tersebut dipilih karena operator *crossover* tersebut memiliki teknik yang tepat sesuai dengan representasi kromosom pada tugas akhir kali ini. Berikut Gambar 4.5 merupakan ilustrasi *One-Point Crossover*. Setelah

Tabel 4.4: Proses *One-Point Crossover*

Proses One-Point Crossover	
Parent 1	01 01 01 01 01 01 *01 10 01 01 00 01 01 01*
Parent 2	*01 10 01 01 10 10 10* 10 10 10 10 01 10 01
Baby 1	*01 01 01 01 01 01 01 10 10 10 10 01 10 01*
Baby 2	01 10 01 01 10 10 10 10 01 01 00 01 01 01

proses *crossover* nilai fitness masing-masing individu dipastikan berubah ada yang berubah jadi lebih baik dan ada juga yang berubah menjadi kurang baik. Nilai *fitness* pada masing-masing individu setelah terjadinya *crossover* adalah 0,2, 0,33, 1, dan 0,167. Untuk nilai *fitness* optimal ditemukan pada individu *Baby 1* dimana individu tersebut merupakan hasil kawin silang antara dua orang tua. Ketika nilai *fitness* sama dengan 1 artinya bahwa solusi optimal telah ditemukan.

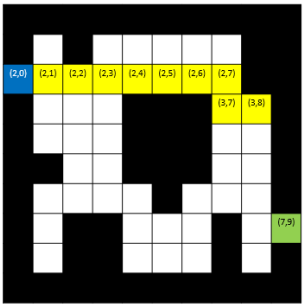
6. Mutasi dilakukan dengan cara mengubah bilangan 0 menjadi 1 untuk menambah variasi individu. Peluang terjadinya mutasi dapat ditetapkan antara 0.01 hingga 0.99.

7. Kondisi pemberhentian iterasi algoritma genetika pada tugas akhir kali ini bergantung pada batas generasi yang ditentukan atau ketika nilai *fitness* = 1. Apabila nilai jarak x dan jarak y dijumlahkan sama dengan 0, artinya bahwa nilai *fitness* dari individu ini sama dengan 1. Pada Gambar akan menejaskan contoh solusi optimal pada Algoritma Genetika.

8. Penggantian Populasi pada Algoritma Genetika berjenis *population replacement*, P individu pada suatu generasi digantikan sekaligus oleh populasi baru yang tidak membawa karakteristik dari generasi sebelumnya. Untuk mendapatkan individu terbaik pada setiap generasi, artinya harus menetapkan input yang tepat pada setiap komponen Algoritma Genetika seperti banyak populasi, peluang *crossover* dan *mutasi*.

Setelah dijelaskan mengenai komponen Algoritma Genetika beserta dengan penggantian populasi pada setiap generasi, maka selanjutnya akan diberikan contoh permasalahan 1 yang diselesaikan dengan menggunakan Algoritma Genetika. Permasalahan yang dimaksud adalah untuk mencari rute pada sebuah matriks graf dengan beberapa halangan yang ada. Artinya, pada setiap graf terdapat beberapa alternatif jalur untuk mencapai ke titik tujuan. Sehingga, ketika ditemukan individu dengan nilai *fitness* sama dengan 1 artinya solusi tersebut merupakan salah satu solusi yang optimal. Berikut akan dijelaskan secara bertahap bagaimana Algoritma Genetika ini melakukan pencarian pada suatu matriks dalam mencari rute terpendek.

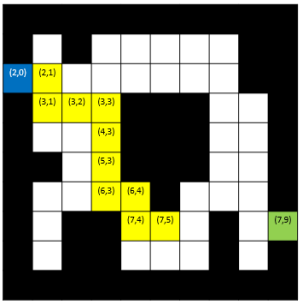
a. Individu 1 akan bergerak berdasarkan tetapan kromosom dan setelah berjalan sejauh panjang kromosom yang ada, titik akhir tersebut akan diambil untuk dihitung nilai *fitness* nya. Berikut pergerakan dari individu 1 pada Gambar 4.9. pada individu 1,



Gambar 4.9: Individu 1

pencarian berakhir pada titik (3,8) sedangkan titik tujuan ada pada titik (7,9). Berdasarkan perhitungan nilai *fitness* yang telah ditetapkan, yakni untuk meminimalkan jarak*manhattan* x dan y pada saat *node* ke-n dengan jarak akhir pada koordinat x dan y, maka didapatkan hasil perhitungan nilai fitness pada individu 1 adalah 0,167. Dengan hasil perhitungan nilai *fitness* ini didapatkan bahwa individu 1 merupakan individu terbaik kedua setelah individu 2.

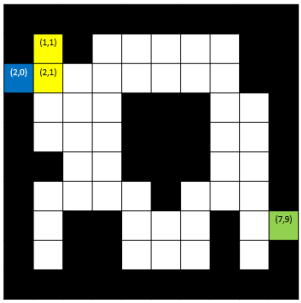
b. Individu 2 akan bergerak berdasarkan tetapan kromosom dan setelah berjalan sejauh panjang kromosom yang ada, titik akhir tersebut akan diambil untuk dihitung nilai *fitness* nya sama seperti individu 1. Berikut pergerakan dari individu 2 ditampilkan pada Gambar 4.10. pada individu 2, pencarian berakhir pada titik (7,5) sedangkan titik tujuan ada pada titik



Gambar 4.10: Individu 2

(7,9). Berdasarkan perhitungan nilai *fitness* yang telah ditetapkan, yakni untuk meminimalkan jarak*manhattan* x dan y pada saat *node* ke-n dengan jarak akhir pada koordinat x dan y, maka didapatkan hasil perhitungan nilai fitness pada individu 2 adalah 0,2. Dengan hasil perhitungan nilai *fitness* ini didapatkan bahwa individu 2 merupakan individu terbaik pada generasi ini.

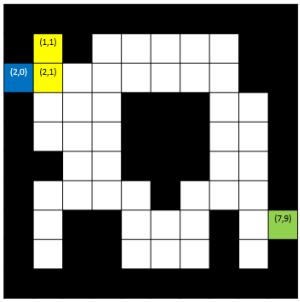
c. Individu 3 akan bergerak berdasarkan tetapan kromosom dan setelah berjalan sejauh panjang kromosom yang ada, titik akhir tersebut akan diambil untuk dihitung nilai *fitness* nya sama seperti individu 1. Berikut pergerakan dari individu 3 pada Gambar 4.11. pada individu 3, pencarian berakhir pada titik (1,1) sedangkan titik tujuan ada pada titik (7,9). Berdasarkan perhitungan nilai *fitness* yang telah ditetapkan, yakni untuk meminimalkan jarak*manhattan* x dan y pada saat *node* ke-n dengan jarak akhir pada koordinat x dan y, maka didapatkan hasil perhitungan nilai fitness pada individu 3 adalah 0,067. Dengan hasil perhitungan nilai *fitness* ini didapatkan bahwa individu 3 merupakan individu yang kurang baik pada generasi



Gambar 4.11: Individu 3

ini karena nilai *fitness* individu 3 tidak lebih baik dari individu 1 dan 2.

d. Pergerakan Individu 4 Individu 4 akan bergerak berdasarkan tetapan kromosom dan setelah berjalan sejauh panjang kromosom yang ada, titik akhir tersebut akan diambil untuk dihitung nilai *fitness* nya sama seperti individu 1. Berikut pergerakan dari individu 4 pada Gambar 4.12. pada individu 4, pancarian

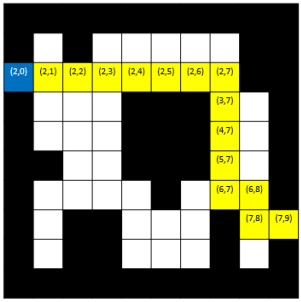


Gambar 4.12: Individu 4

berakhir pada titik (1,1) sama dengan individu 3

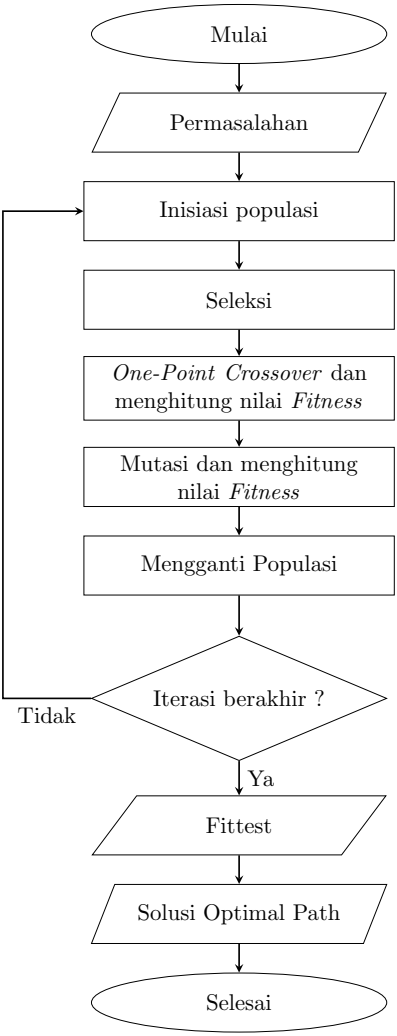
sedangkan titik tujuan ada pada titik (7,9). Berdasarkan perhitungan nilai *fitness* yang telah ditetapkan, yakni untuk meminimalkan jarak *manhattan* x dan y pada saat *node* ke-n dengan jarak akhir pada koordinat x dan y, maka didapatkan hasil perhitungan nilai fitness pada individu 4 adalah 0,067. Dengan hasil perhitungan nilai *fitness* ini didapatkan bahwa individu 4 bersama dengan individu 3 merupakan individu yang kurang baik pada generasi ini karena nilai *fitness* individu 3 tidak lebih baik dari individu 1 dan 2.

e. Individu menemukan solusi Individu hasil *crossover* akan memiliki nilai *fitness* sama dengan 1. pada individu hasil *crossover*, pancarian berakhir pada titik (7,9) sedangkan titik tujuan ada pada titik (7,9). Berdasarkan perhitungan didapatkan nilai fitness pada individu ini adalah 1. Dengan hasil perhitungan ini didapat bahwa individu hasil *crossover* ini merupakan solusi optimal pada permasalahan 1. Berikut hasil dari solusi optimal permasalahan 1 pada Gambar 4.13.



Gambar 4.13: Solusi optimal Algoritma Genetika

Berikut Langkah-langkah Algoritma Genetika untuk untuk mencari penyelesaian pada Gambar 4.14.



Gambar 4.14: Langkah-langkah Algoritma Genetika

4.4 Implementasi Program

Agar dapat diimplementasikan menggunakan Unity 3D maka masing-masing algoritma akan diimplementasikan kedalam bahasa pemrograman C#. Hal ini dilakukan untuk mengetahui bagaimana karakteristik kedua algoritma tersebut untuk mencari rute terpendek dengan beberapa input permasalahan yang telah ditetapkan.

4.4.1 Implementasi program pada Algoritma Genetika

1. Inisiasi populasi
   Method ini digunakan untuk menginisiasi populasi dalam Algoritma Genetika, pada kasus ini penulis menetapkan jumlah populasi yang optimal adalah 150-300. Dalam class ini terdapat beberapa fungsi yang dapat digunakan untuk mengolah populasi dan untuk mendapatkan kromosom terbaik yang ada dalam populasi. Keseluruhan source code dari Inisiasi populasi dapat dilihat pada Lampiran E.

2. Mencari solusi berdasarkan kromosom
   Kromosom yang dibentuk adalah bilangan biner random 0 dan 1 dibuat dari beberapa blok *gene*. *gene* ini sendiri memuat bilangan bit *random* dimana bilangan bit ini menggambarkan arah pergerakan dari algoritma genetika untuk mencari suatu solusi. Kemudian disini akan ditetapkan bahwa panjang dari gene adalah 2.
   Representasi kromosom pada tugas akhir kali ini adalah sebagai berikut:

   - ▷ Genome: Satu buah kromosom(rute).
   - ▷ Chromosomes: Memiliki panjang kromosom sesuai masukan dari penulis (Contoh : 11|10|00|01, Panjang kromosom = 4)

- ▷ Gene = Terdiri dari 2 nukleotida, artinya kemungkinan *gene* ini adalah bilangan 0 dan 1 dengan panjang gene: 2
- ▷ Nucleotide: 0 atau 1

Keseluruhan source code untuk menjalankan kromosom dapat dilihat pada Lampiran I.

3. Seleksi Orang Tua
   Seleksi untuk memilih dua orang tua untuk dihasilkan dua keturunan, pada kasus ini pemilihan dua orang tua dilakukan secara random. Keseluruhan source code untuk seleksi orang tua dapat dilihat pada Lampiran F.

4. *Crossover*
   Pada method ini akan dilakukan proses kawin silang pada dua orang tua yang terpilih untuk menghasilkan dua keturunan. Pada kasus ini, *crossover rate* untuk permasalahan satu sampai permasalahan delapan yang peluang *crossover* dapat terjadi antara 0,001 sampai 0,99. Keseluruhan source code untuk *crossover* dapat dilihat pada Lampiran G.

5. Mutasi
   Pada method ini akan dilakukan proses mutasi pada tingkat bit dengan merubah bit 0 menjadi 1, peluang mutasi dapat terjadi pada rentang 0,001 sampai 0,99. Keseluruhan source code untuk mutasi dapat dilihat pada Lampiran H.

6. Kriteria pemberhentian algoritma
   Kriteria pemberhentian algoritma adalah ketika nilai *fitness* = 1 yang artinya solusi telah ditemukan atau telah mencapai batas generasi yang telah ditetapkan. Apabila nilai fitness masih belum sama dengan 1 atau

solusi belum ditemukan dan belum mencapai batas generasi yang telah ditetapkan, maka algoritma ini akan kembali ke tahap inisiasi populasi.

4.4.2 Implementasi program pada Algoritma A*

1. Membuat fungsi Algoritma A*
   Untuk dapat menentukan *OPEN* dan *CLOSE* dari *path* yang akan ditentukan akan dicari fungsi jarak yang untuk menghitung jumlah jarak yang dilalui pada setiap iterasinya dengan memperhitungkan jarak sebenarnya ditambah dengan jarak perkiraan ke titik akhir. Dalam notasi matematika dituliskan sebagai berikut:

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

   Dengan perhitungan jarak seperti ini, Algoritma A* adalah ***complete*** dan ***optimal***. Pembuktian secara matematis dapat dilihat di [18]. Secara detail, *source code* perhitungan fungsi Algoritma A* untuk mendapatkan nilai f(n) pada node suksesor berikutnya diilustrasikan pada potongan program dibawah ini:

   ```
   int MoveCost = CurrentNode.igCost +
       GetManhattenDistance(CurrentNode,
       NeighborNode);
   ```

2. Inisiasi *Start Node* dan *Target Node*
   Algoritma A* ini akan mencari *path* dengan mengkalkulasi *node* awal dan *node* tujuan dengan memperhitungkan nilai heuristik antara *node* saat ini dan *node* tujuan serta mengkalkulasi jarak sebenarnya dengan titik awal. Untuk memulai pencarian akan diinisiasi *Start Node* dan *Target Node* yang akan dicari pada matriks:

```
Node StartNode = GridReference.
    NodeFromWorldPoint(a_StartPos);
Node TargetNode = GridReference.
    NodeFromWorldPoint(a_TargetPos);
```

3. Memulai pencarian
   Pada saat Algoritma A* ini memulai pencarian, maka dia akan menambahkan *Start Node* kedalam senarai *Open List* dengan menjalankan *Source Code* sebagai berikut:

   ```
   OpenList.Add(StartNode);
   ```

4. Menjalankan Algoritma A*
   Setelah memulai pencarian dengan menambahkan *Start Node* kedalam senarai *Open List* maka akan dimulai pencarian dengan mempertimbangkan nilai heuristik dan jarak sebenarnya. Untuk keseluruhan *source code* Algoritma A* dapat dilihat pada Lampiran G.

5. Mendapatkan *Final Path*
   Setelah Algoritma selesai dijalankan dan ditemukan tujuan akhir pada matriks, selanjutnya akan didapatkan *path* yang benar-benar valid dan dapat dilalui dengan menjalankan potongan *source code* sebagai berikut :

   ```
   void GetFinalPath(Node a_StartingNode, Node
       a_EndNode)
   {
       FinalPath = new List<Node>();//List to
           hold the path sequentially
       Node CurrentNode = a_EndNode;//Node to
           store the current node being checked

       while (CurrentNode != a_StartingNode)
       {
   ```

```
            FinalPath.Add(CurrentNode);//Add
                that node to the final path
            CurrentNode = CurrentNode.ParentNode
                ;//Move onto its parent node
        }
        FinalPath.Reverse();//Reverse the path
            to get the correct order
        GridReference.FinalPath = FinalPath;//
            Set the final path
        StartCoroutine(DrawFinalPath());
    }
```

6. Fungsi jarak Manhattan
   Fungsi Manhattan digunakan sebagai fungsi heuristik(h(n)) pada Algoritma A*. Untuk *source code* fungsi manhattan dapat dilihat pada potongan *source code* dibawah ini:

```
int GetManhattenDistance(Node a_nodeA, Node
    a_nodeB)
{
    int ix = Mathf.Abs(a_nodeA.iGridX -
        a_nodeB.iGridX);//x1-x2
    int iy = Mathf.Abs(a_nodeA.iGridY -
        a_nodeB.iGridY);//y1-y2
    return ix + iy;//Return the sum
}
```

4.5 Hasil Running Program pada Algoritma A* dan Algoritma Genetika

4.5.1 Hasil running program pada Permasalahan 1

Permasalahan 1 terdiri dari 100 *node* yang akan diselesaikan dengan kedua algoritma untuk dicari jalur terpendek menuju ke titik akhir. Terdapat dua alternatif jalur yang ada, nanti akan diketahui hasil *path* yang optimal pada masing-masing algoritma.

Pada permasalahan kali ini, Algoritma Genetika disimulasi sebanyak empat kali diambil satu yang terbaik. Algoritma Genetika pada permasalahan 1 menghasilkan jarak rata-rata 15,5, waktu rata-rata 0,29 detik, dan kebutuhan memori rata-rata 11,42 MB. Dari keempat percobaan ini akan diambil satu yang terbaik untuk dijadikan pembanding Algoritma A*, untuk hasil keempat percobaan Algoritma Genetika ini akan ditampilkan pada Tabel 4.6.

Tabel 4.5: Hasil simulasi Algoritma Genetika pada Permasalahan 1

Algoritma Genetika Permasalahan 1			
Percobaan	Jarak	Waktu	Memori
1	14	0,121 s	11,5 MB
2	18	0,473 s	11,57 MB
3	14	0,356 s	11,2 MB
4	16	0,213 s	11,4 MB

Pada permasalahan 1 Algoritma A* menghasilkan waktu komputasi 0,001 s, jarak 14, dan kebutuhan memori sebesar 8,75 MB. Berdasarkan hasil simulasi pada permasalahan 1 oleh kedua algoritma, sudah jelas bahwa waktu komputasi dan kebutuhan memori dari Algoritma A* ini lebih baik dari Algoritma Genetika meskipun dari segi jarak yang dihasilkan adalah sama.

4.5.2 Hasil running program pada Permasalahan 2

Permasalahan 2 terdiri dari 144 *node* yang akan diselesaikan dengan kedua algoritma untuk dicari jalur terpendek menuju ke titik akhir. Terdapat beberapa alternatif jalur yang ada, nanti akan diketahui hasil *path* yang optimal pada masing-masing algoritma.

Pada permasalahan 2, Algoritma Genetika disimulasi sebanyak empat kali pengujian dan diambil satu yang terbaik. Algoritma Genetika pada permasalahan 1 menghasilkan jarak rata-rata sebesar 18,75, waktu rata-rata sebesar 0,4 detik, dan kebutuhan memori rata-rata sebesar 11,7 MB. Dari keempat percobaan ini akan diambil satu yang terbaik untuk dijadikan pembanding Algoritma A*, untuk hasil keempat percobaan Algoritma Genetika ini akan ditampilkan pada Tabel 4.7.

Tabel 4.6: Hasil simulasi Algoritma Genetika pada Permasalahan 2

Algoritma Genetika Permasalahan 2			
Percobaan	Jarak	Waktu	Memori
1	18	0,49 s	11,6 MB
2	19	0,345 s	11,87 MB
3	20	0,478 s	11,75 MB
4	18	0,321 s	11,6 MB

Pada permasalahan 2 Algoritma A* menghasilkan waktu komputasi 0,001 s, jarak 18, dan kebutuhan memori sebesar 8,91 MB. Berdasarkan hasil simulasi pada permasalahan 2 oleh kedua algoritma, sudah jelas bahwa waktu komputasi dan kebutuhan memori dari Algoritma A* ini lebih baik dari Algoritma Genetika meskipun dari segi jarak Algoritma Genetika menghasilkan jarak yang lebih baik dari Algoritma A*.

4.5.3 Hasil running program pada Permasalahan 3

Permasalahan 3 terdiri dari 196 *node* yang akan diselesaikan dengan kedua algoritma untuk dicari jalur terpendek menuju ke titik akhir. Terdapat beberapa alternatif jalur yang ada, nanti akan diketahui hasil *path* yang optimal pada masing-masing algoritma.

Pada permasalahan 3, Algoritma Genetika disimulasi sebanyak empat kali pengujian dan diambil satu yang terbaik. Algoritma Genetika pada permasalahan 1 menghasilkan jarak rata-rata sebesar 23, waktu rata-rata sebesar 0,38 detik, dan kebutuhan memori rata-rata sebesar 11,88 MB. Dari keempat percobaan ini akan diambil satu yang terbaik untuk dijadikan pembanding Algoritma A*, untuk hasil keempat percobaan Algoritma Genetika ini akan ditampilkan pada Tabel 4.8.

Tabel 4.7: Hasil simulasi Algoritma Genetika pada Permasalahan 3

Algoritma Genetika Permasalahan 2			
Percobaan	Jarak	Waktu	Memori
1	24	0,457 s	11,87 MB
2	22	0,389 s	11,87 MB
3	22	0,333 s	11,83 MB
4	24	0,343 s	11,95 MB

Pada permasalahan 3 Algoritma A* menghasilkan waktu komputasi 0,001 s, jarak 22, dan kebutuhan memori sebesar 8,98 MB. Berdasarkan hasil simulasi pada permasalahan 3 oleh kedua algoritma, sudah jelas bahwa waktu komputasi, kebutuhan memori dan jarak yang dihasilkan oleh Algoritma A* ini lebih baik dari Algoritma Genetika.

4.5.4 Hasil running program pada Permasalahan 4

Permasalahan 4 terdiri dari 256 *node* yang akan diselesaikan dengan kedua algoritma untuk dicari jalur terpendek menuju ke titik akhir. Terdapat beberapa alternatif jalur yang ada, nanti akan diketahui hasil *path* yang optimal pada masing-masing algoritma.

Pada permasalahan 4, Algoritma Genetika disimulasi sebanyak empat kali pengujian dan diambil satu yang terbaik. Algoritma Genetika pada permasalahan 1 menghasilkan jarak rata-rata sebesar 29, waktu rata-rata sebesar 0,75 detik, dan kebutuhan memori rata-rata sebesar 11,96 MB. Dari keempat percobaan ini akan diambil satu yang terbaik untuk dijadikan pembanding Algoritma A*, untuk hasil keempat percobaan Algoritma Genetika ini akan ditampilkan pada Tabel 4.9.

Tabel 4.8: Hasil simulasi Algoritma Genetika pada Permasalahan 4

Algoritma Genetika Permasalahan 4			
Percobaan	Jarak	Waktu	Memori
1	30	0,783 s	11,9 MB
2	28	0,509 s	11,88 MB
3	28	0,841 s	12,1 MB
4	30	0,856 s	11,95 MB

Pada permasalahan 4 Algoritma A* menghasilkan waktu komputasi 0,001 s, jarak 26, dan kebutuhan memori sebesar 9 MB. Berdasarkan hasil simulasi pada permasalahan 4 oleh kedua algoritma, sudah jelas bahwa waktu komputasi, kebutuhan memori dan jarak yang dihasilkan oleh Algoritma A* ini lebih baik dari Algoritma Genetika.

4.5.5 Hasil running program pada Permasalahan 5

Permasalahan 5 terdiri dari 324 *node* yang akan diselesaikan dengan kedua algoritma untuk dicari jalur terpendek menuju ke titik akhir. Terdapat beberapa alternatif jalur yang ada, nanti akan diketahui hasil *path* yang optimal pada masing-masing algoritma.

Pada permasalahan 5, Algoritma Genetika disimulasi sebanyak empat kali pengujian dan diambil satu yang terbaik. Algoritma Genetika pada permasalahan 5 menghasilkan jarak rata-rata sebesar 31, waktu rata-rata sebesar 0,846 detik, dan kebutuhan memori rata-rata sebesar 12,15 MB. Dari keempat percobaan ini akan diambil satu yang terbaik untuk dijadikan pembanding Algoritma A*, untuk hasil keempat percobaan Algoritma Genetika ini akan ditampilkan pada Tabel 4.10.

Tabel 4.9: Hasil simulasi Algoritma Genetika pada Permasalahan 5

Algoritma Genetika Permasalahan 5			
Percobaan	Jarak	Waktu	Memori
1	30	0,614 s	12 MB
2	32	0,784 s	12,2 MB
3	32	0,841 s	12,1 MB
4	30	1,147 s	12,3 MB

Pada permasalahan 5 Algoritma A* menghasilkan waktu komputasi 0,001 s, jarak 30, dan kebutuhan memori sebesar 9,15 MB. Berdasarkan hasil simulasi pada permasalahan 5 oleh kedua algoritma, sudah jelas bahwa waktu komputasi dan kebutuhan memori Algoritma A* ini lebih baik dari Algoritma Genetika meskipun dari segi jarak Algoritma Genetika masih lebih baik dari Algoritma A*.

4.5.6 Hasil running program pada Permasalahan 6

Permasalahan 6 terdiri dari 400 *node* yang akan diselesaikan dengan kedua algoritma untuk dicari jalur terpendek menuju ke titik akhir. Terdapat beberapa alternatif jalur yang ada, nanti akan diketahui hasil *path* yang optimal pada masing-masing algoritma.

Pada permasalahan 6, Algoritma Genetika disimulasi sebanyak empat kali pengujian dan diambil satu yang terbaik. Algoritma Genetika pada permasalahan 6 menghasilkan jarak rata-rata sebesar 59,65, waktu rata-rata sebesar 0,87 detik, dan kebutuhan memori rata-rata sebesar 12,22 MB. Dari keempat percobaan ini akan diambil satu yang terbaik untuk dijadikan pembanding Algoritma A*, untuk hasil keempat percobaan Algoritma Genetika ini akan ditampilkan pada Tabel 4.11.

Tabel 4.10: Hasil simulasi Algoritma Genetika pada Permasalahan 6

Algoritma Genetika Permasalahan 6			
Percobaan	Jarak	Waktu	Memori
1	54	0,671 s	12,27 MB
2	54	0,623 s	12,13 MB
3	65	0,897 s	12,29 MB
4	65	1,287 s	12,18 MB

Pada permasalahan 6 Algoritma A* menghasilkan waktu komputasi 0,001 s, jarak 30, dan kebutuhan memori sebesar 9,19 MB. Berdasarkan hasil simulasi pada permasalahan 6 oleh kedua algoritma, sudah jelas bahwa waktu komputasi dan kebutuhan memori Algoritma A* ini lebih baik dari Algoritma Genetika meskipun dari segi jarak Algoritma Genetika masih lebih baik dari Algoritma A*.

4.5.7 Hasil running program pada Permasalahan 7

Permasalahan 7 terdiri dari 484 *node* yang akan diselesaikan dengan kedua algoritma untuk dicari jalur terpendek menuju ke titik akhir. Terdapat beberapa alternatif jalur yang ada, nanti akan diketahui hasil *path* yang optimal pada masing-masing algoritma.

Pada permasalahan 7, Algoritma Genetika disimulasi sebanyak empat kali pengujian dan diambil satu yang terbaik. Algoritma Genetika pada permasalahan 7 menghasilkan jarak rata-rata sebesar 60, waktu rata-rata sebesar 3,25 detik, dan kebutuhan memori rata-rata sebesar 12,3 MB. Dari keempat percobaan ini akan diambil satu yang terbaik untuk dijadikan pembanding Algoritma A*, untuk hasil keempat percobaan Algoritma Genetika ini akan ditampilkan pada Tabel 4.12.

Tabel 4.11: Hasil simulasi Algoritma Genetika pada Permasalahan 7

Algoritma Genetika Permasalahan 7			
Percobaan	Jarak	Waktu	Memori
1	57	2,223 s	12,22 MB
2	68	2,623 s	12,28 MB
3	68	3,874 s	12,37 MB
4	57	4,291 s	12,36 MB

Pada permasalahan 7 Algoritma A* menghasilkan waktu komputasi 0,001 s, jarak 30, dan kebutuhan memori sebesar 9,23 MB. Berdasarkan hasil simulasi pada permasalahan 7 oleh kedua algoritma, sudah jelas bahwa waktu komputasi dan kebutuhan memori Algoritma A* ini lebih baik dari Algoritma Genetika meskipun dari segi jarak Algoritma Genetika masih lebih baik dari Algoritma A*.

4.5.8 Hasil running program pada Permasalahan 8

Permasalahan 8 terdiri dari 576 *node* yang akan diselesaikan dengan kedua algoritma untuk dicari jalur terpendek menuju ke titik akhir. Terdapat beberapa alternatif jalur yang ada, nanti akan diketahui hasil *path* yang optimal pada masing-masing algoritma.

Pada permasalahan 8, Algoritma Genetika disimulasi sebanyak empat kali pengujian dan diambil satu yang terbaik. Algoritma Genetika pada permasalahan 8 menghasilkan jarak rata-rata sebesar 78,5, waktu rata-rata sebesar 3,65 detik, dan kebutuhan memori rata-rata sebesar 12,56 MB. Dari keempat percobaan ini akan diambil satu yang terbaik untuk dijadikan pembanding Algoritma A*, untuk hasil keempat percobaan Algoritma Genetika ini akan ditampilkan pada Tabel 4.13.

Tabel 4.12: Hasil simulasi Algoritma Genetika pada Permasalahan 8

Algoritma Genetika Permasalahan 8			
Percobaan	Jarak	Waktu	Memori
1	83	8,761 s	12,65 MB
2	83	6,478 s	12,78 MB
3	83	3,874 s	12,467 MB
4	65	3,488 s	12,331 MB

Pada permasalahan 8 Algoritma A* menghasilkan waktu komputasi 0,001 s, jarak 30, dan kebutuhan memori sebesar 9,23 MB. Berdasarkan hasil simulasi pada permasalahan 8 oleh kedua algoritma, sudah jelas bahwa waktu komputasi dan kebutuhan memori Algoritma A* ini lebih baik dari Algoritma Genetika meskipun dari segi jarak Algoritma Genetika masih lebih baik dari Algoritma A*.

4.5.9 Hasil optimal Algoritma Genetika

Pada bagian ini didapatkan hasil optimal pada Algoritma Genetika dari empat kali percobaan dan diambil satu yang terbaik. Hasil optimal Algoritma Genetika ditinjau dari banyak populasi, banyak generasi yang dibutuhkan, waktu komputasi, memori, dan jarak. Hasil selengkapnya dapat dilihat pada Tabel 4.14 dibawah ini.

Tabel 4.13: Hasil Optimal Algoritma Genetika

Permasalahan	Algoritma Genetika				
	Populasi	Generasi	Waktu	Memori	Jarak
1(100 Node)	150	18	0,121 s	11,5 MB	14
2(144 Node)	150	11	0,321 s	11,6 MB	18
3(196 Node)	150	14	0,333 s	11,83 MB	22
4(256 Node)	150	27	0,509 s	11,88 MB	28
5(324 Node)	150	52	0,614 s	12 MB	30
6(400 Node)	150	36	0,623 s	12,13 MB	54
7(484 Node)	200	49	2,223 s	12,22 MB	57
8(576 Node)	200	37	3,488 s	12,3 MB	65

Pada Tabel 4.4 dapat dilihat bahwa pada permasalahan 1 sampai permasalahan 6 Algoritma Genetika mampu mendapatkan solusi dengan rata-rata waktu dibawah 1 detik, sedangkan untuk permasalahan 7 dan 8 waktu yang dibutuhkan lebih dari 2 detik. Hal ini disebabkan karena semakin kompleks permasalahan akan mempengaruhi kerja dari Algoritma Genetika itu sendiri sehingga akan menghasilkan waktu komputasi yang lebih lama. Jumlah populasi yang diinisiasi adalah 150 sampai 200, jumlah ini ditetapkan agar tetap memperhitungkan waktu komputasi yang tidak terlalu lama dan meminimalkan kebutuhan memori serta dapat menghasilkan solusi yang optimal.

4.5.10 Hasil optimal Algoritma A*

Pada bagian ini didapatkan hasil optimal pada Algoritma A* dari empat kali percobaan dan hasil yang didapatkan adalah sama. Tampilan hasil optimal pada Algoritma A* akan ditinjau dari segi waktu komputasi, memori, dan jarak. Hasil selengkapnya dapat dilihat pada Tabel 4.15 dibawah ini.

Tabel 4.14: Hasil Simulasi Algoritma A*

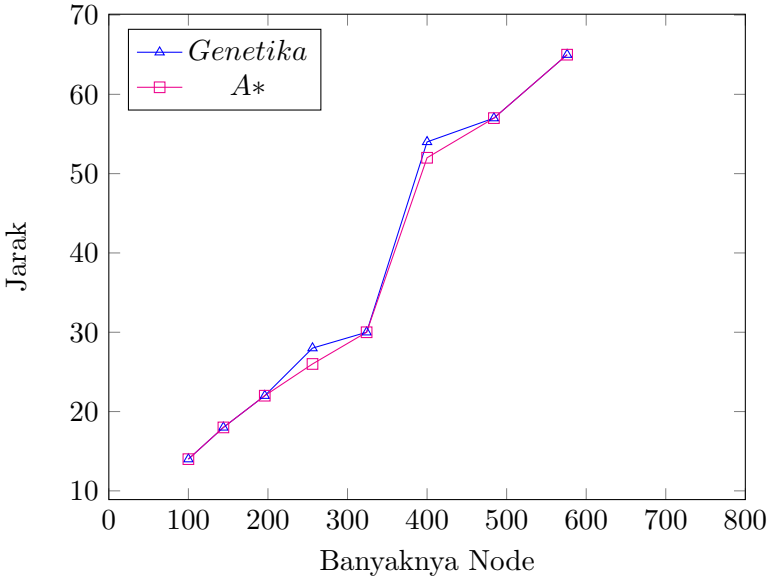
Permasalahan	Algoritma A*		
	Waktu	Memori	Jarak
1(100 Node)	$1x10^{-3}$ s	11,5 MB	14
2(144 Node)	$1{,}23x10^{-3}$ s	11,6 MB	18
3(196 Node)	$1{,}37x10^{-3}$ s	11,83 MB	22
4(256 Node)	$1{,}46x10^{-3}$ s	11,88 MB	28
5(324 Node)	$1{,}78x10^{-3}$ s	12 MB	30
6(400 Node)	$1{,}83x10^{-3}$ s	12,13 MB	54
7(484 Node)	$1{,}88x10^{-3}$ s	12,22 MB	57
8(576 Node)	$1{,}97x10^{-3}$ s	12,3 MB	65

Pada Tabel 4.4 dapat dilihat bahwa Algoritma A* mampu menghasilkan waktu yang sangat cepat dengan kebutuhan memori yang tidak terlalu banyak, sehingga algoritma ini dapat berjalan optimal pada setiap permasalahan. Pada tugas akhir kali ini, Algoritma A* akan dipilih sebagai pembanding Algoritma Genetika. Algoritma ini dipilih karena algoritma ini merupakan algoritma eksak untuk mencari rute terpendek. Dikatakan eksak karena dapat mengevaluasi *node* yang akan dilalui berdasarkan nilai heuristik terhadap titik akhir dan jarak dari *node* n ke titik awal.

4.6 Perbandingan Algoritma Genetika dan Algoritma A*

4.6.1 Perbandingan Jarak Algoritma

Pada bagian kali ini, akan didapatkan jumlah perbandingan jarak yang ditempuh oleh kedua Algoritma untuk mencari rute terpendek. Grafik perbandingan jarak akan ditampilkan pada Gambar 4.15.



Gambar 4.15: Perbandingan jarak

Jarak didapatkan dengan menghitung banyaknya titik yang dilalui untuk mencapai tujuan. Untuk mengetahui presentase optimal pada perbandingan jarak akan digunakan nilai P pada persamaan berikut:

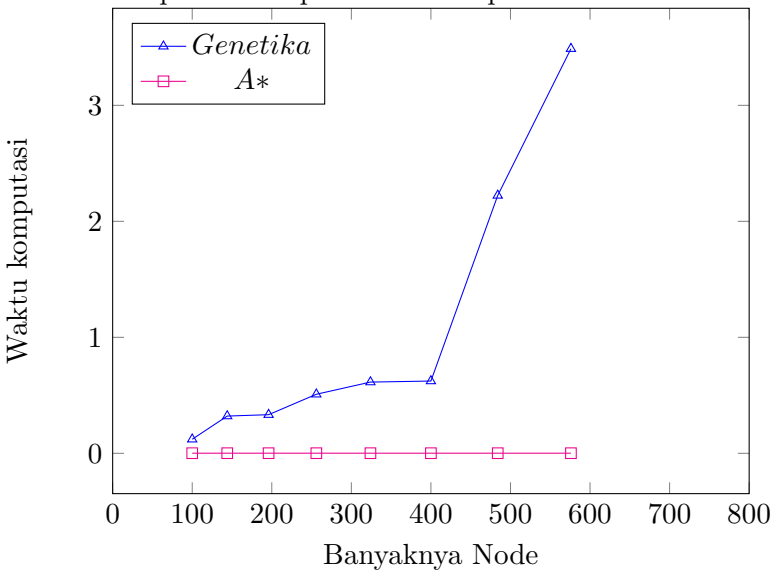
$$P = \frac{jarak1 - jarak2}{jarak1} * 100\%$$

jarak 1 dan jarak 2 masing-masing adalah jarak yang diperoleh Algoritma Genetika dan Algoritma A*. Nilai P mempunyai arti berapa persen Algoritma Genetika lebih baik atau lebih buruk dari Algoritma A*.

Pada Gambar 4.15 dapat dilihat bahwa Algoritma Genetika dapat mengambil jarak sama dengan Algoritma A* pada permasalahan 1, 2, 3, 5, 7, dan 8. Pada permasalahan 4 jarak yang ditempuh oleh Algoritma Genetika sebesar 28 sedangkan Algoritma A* sebesar 26, sehingga dapat dihitung persentase nilai P untuk mengetahui berapa persen Algoritma Genetika lebih buruk dari Algoritma A* pada permasalahan 4 $\left|P = \frac{28-26}{28} * 100\% = 7\%\right|$. Dari hasil perhitungan nilai P pada permasalahan 4, dapat diketahui bahwa Algoritma Genetika menghasilkan presentase jarak sebesar 7% lebih buruk daripada Algoritma A*. Hal ini tentu bukan merupakan perbedaan yang besar akan tetapi presentase itu sudah cukup menggambarkan bahwa Algoritma A* pada permasalahan 4 masih bisa bekerja lebih baik dalam mencari rute terpendek. Pada permasalahan 6 jarak yang ditempuh oleh Algoritma Genetika sebesar 54 sedangkan Algoritma A* sebesar 52, sehingga dapat dihitung persentase nilai P untuk mengetahui berapa persen Algoritma Genetika lebih buruk dari Algoritma A* pada permasalahan 4 $\left|P = \frac{54-52}{54} * 100\% = 4\%\right|$.Dari hasil perhitungan nilai P pada permasalahan 6, dapat diketahui bahwa Algoritma Genetika menghasilkan presentase jarak sebesar 4% lebih buruk daripada Algoritma A*. Hal ini tentu bukan merupakan perbedaan yang besar akan tetapi presentase itu sudah cukup menggambarkan bahwa Algoritma A* pada permasalahan 4 masih bisa bekerja lebih baik dalam mencari rute terpendek.

4.6.2 Perbandingan Waktu komputasi Algoritma

Perbandingan waktu komputasi kedua Algoritma dihitung melalui waktu proses kedua algoritma untuk mendapatkan solusi. Untuk Algoritma Genetika lama proses ini selain dihitung hingga algoritma ini mendapatkan solusi dapat juga dihitung hingga iterasi yang ditetapkan sudah berakhir. Artinya, Algoritma Genetika ini bisa saja tidak menemukan solusi setelah melalui proses yang telah dilakukan dan tetap dapat dihitung lama waktu proses tersebut bekerja hingga iterasi yang dilakukan Algoritma Genetika berakhir. Untuk Algoritma A* waktu proses untuk mendapatkan solusi relatif lebih cepat karena algoritma ini merupakan algoritma yang optimal jika digunakan untuk mencari rute terpendek. Untuk hasil perbandingan waktu komputasi dapat dilihat pada Gambar 4.16.



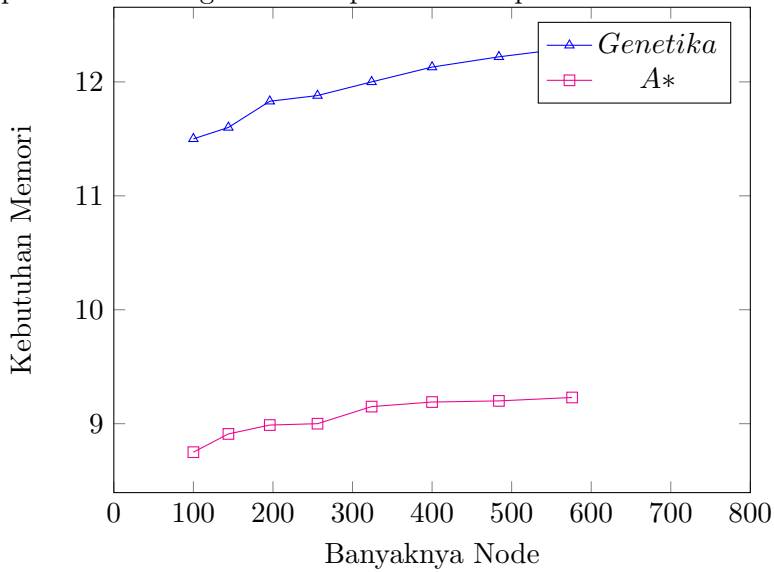
Gambar 4.16: Perbandingan Waktu komputasi

Berdasarkan hasil perhitungan, didapat bahwa nilai rata-rata waktu komputasi Algoritma Genetika adalah sebesar 1,029 detik sedangkan untuk Algoritma A* adalah $1{,}565 \times 10^{-3}$ detik, hasil ini menggambarkan bahwa untuk mencari rute terpendek Algoritma Genetika membutuhkan waktu yang lebih banyak daripada Algoritma A*. Jika dilihat dari grafik yang ada, dapat disimpulkan bahwa pertumbuhan waktu yang dibutuhkan bersifat logaritmik. Hal ini menggambarkan bahwa apabila banyaknya titik yang harus dioptimalkan oleh Algoritma Genetika ini semakin banyak maka waktu yang dibutuhkan pun akan semakin banyak seiring bertambahnya panjang kromosom yang dibutuhkan dan jumlah populasi yang semakin meningkat. Dari beberapa aspek inilah yang menyebabkan waktu komputasi Algoritma Genetika kurang optimal jika dibandingkan dengan Algoritma A*.

Dari segi hasil jika kedua algoritma tersebut dibandingkan melalui proses waktu komputasi, didapat bahwa Algoritma Genetika mendapatkan hasil yang kurang optimal dibandingkan Algoritma A*. Hal ini disebabkan karena semakin banyaknya *node* yang ada pada setiap permasalahan maka semakin panjang pula kromosom yang dibutuhkan oleh Algoritma Genetika dan jumlah populasi yang berkisar antara 150 sampai 200 menyebabkan semakin lama waktu komputasi untuk mendapatkan solusi yang diinginkan. Hal ini berbeda dengan Algoritma A* karena algoritma ini mengevaluasi setiap *node* berdasarkan jarak *node* ke n pada titik awal serta nilai heuristik ke titik akhir sehingga didapatkan perhitungan yang lebih baik. Algoritma A* mampu mengevaluasi setiap titik dengan baik, dan menghasilkan solusi yang lebih optimal maka algoritma ini lebih baik dari segi waktu untuk menghasilkan solusi.

4.6.3 Perbandingan penggunaan memori kedua Algoritma

Pada bagian kali ini, akan didapatkan jumlah perbandingan memori yang digunakan oleh kedua Algoritma. Penggunaan memori disini menggambarkan seberapa banyak *space* yang diperlukan oleh kedua algoritma untuk melakukan pencarian. Hasil yang didapat untuk perbandingan memori pada kedua algoritma dapat dilihat pada Gambar 4.17.



Gambar 4.17: Perbandingan penggunaan memori

Berdasarkan hasil perhitungan, didapat bahwa rata-rata kebutuhan memori Algoritma Genetika adalah 12,56 MB sedangkan untuk Algoritma A* adalah 9,12 MB sehingga Algoritma Genetika membutuhkan memori lebih banyak untuk menyelesaikan permasalahan.

4.6.4 Perbandingan Algoritma secara keseluruhan

Pada bagian kali ini akan ditampilkan hasil perbandingan dari segala aspek pada kedua Algoritma dari segi jarak, waktu komputasi, dan kebutuhan memori. Alasan membandingkan kedua algoritma dari segi jarak adalah karena pada Algoritma A* solusi eksak hanya memiliki satu permasalahan sedangkan Algoritma Genetika memiliki banyak variasi penyelesaian. Dari beberapa variasi penyelesaian yang dihasilkan oleh Algoritma Genetika pada implementasinya hanya mampu mendekati optimal dan tidak bisa seoptimal Algoritma A* dari segi jarak dan bahkan masih kalah dari segi waktu komputasi dan penggunaan memori. Berdasarkan hasil simulasi kedua algoritma, didapat rincian perbandingan pada Tabel 4.16.

Tabel 4.15: Perbandingan Algoritma secara keseluruhan

Node	Algoritma Genetika			Algoritma A*		
	Jarak	Waktu	Memori	Jarak	Waktu	Memori
100	14	0,121 s	11,5 MB	14	0,001 s	8,75 MB
144	18	0,321 s	11,6 MB	18	0,0012 s	8,91 MB
196	22	0,333 s	11,83 MB	22	0,0013 s	8,988 MB
256	28	0,509 s	11,88 MB	26	0,0014 s	9 MB
324	30	0,614 s	12 MB	30	0,0017 s	9,15 MB
400	54	0,623 s	12,13 MB	52	0,0018 s	9,19 MB
484	57	2,223 s	12,22 MB	57	0,0019 s	9,2 MB
576	65	3,488 s	12,5 MB	65	0,00197 s	9,23 MB

Dari segi kebutuhan memori dan waktu komputasi, Algoritma Genetika menghasilkan rata-rata memori sebesar 12,56 MB dan waktu komputasi sebesar 1,029 detik sedangkan Algoritma A* membutuhkan rata-rata memori sebesar 9,12 MB dan rata-rata waktu komputasi konstan sebesar 1,565x10^{-3} detik, sehingga jika ditinjau berdasarkan

algoritma yang lebih optimal dapat diketahui bahwa Algoritma A* lebih mampu meminimalkan kebutuhan memori dan waktu komputasi untuk mencari rute terpendek.

Dari segi jarak, didapat bahwa rata-rata jarak yang didapat Algoritma Genetika sebesar 36 sedangkan Algoritma A* sebesar 35,5. Dari hasil ini menghasilkan presentase secara keseluruhan Algoritma Genetika 1% lebih buruk dari Algoritma A*. Adapun nilai P dihitung seperti bersamaan berikut $\left|P = \frac{36-35,5}{36} * 100\% = 1\%\right|$ didapat nilai 1% berdasarkan rumus P seperti yang sudah dijelaskan sebelumnya. Berdasarkan data hasil simulasi pada Tabel 4.15 dapat diketahui bahwa seiring makin kompleks permasalahan atau jumlah *node* semakin banyak mengakibatkan kerja Algoritma Genetika kurang optimal dan menghasilkan waktu komputasi yang lebih lama begitupun dari segi penggunaan memori yang semakin banyak. Sedangkan Algoritma A* mampu bekerja lebih optimal dari segi penggunaan memori dan meminimalkan waktu komputasi sehingga algoritma ini merupakan algoritma yang lebih optimal untuk digunakan pada masalah pencarian jalur terpendek.

BAB V
PENUTUP

5.1 Kesimpulan

Berdasarkan pada hasil dan pembahasan yang telah dipaparkan, dapat diambil kesimpulan:

1. Dari segi jarak, Algoritma A* menghasilkan rata-rata jarak sebesar 35,5 sedangkan Algoritma Genetika sebesar 36. nilai P=1% yang artinya Algoritma A* 1% lebih baik dari Algoritma Genetika dari segi pengambilan jarak untuk mencapai ke titik tujuan. Rata-rata waktu komputasi Algoritma Genetika secara keseluruhan sebesar 1,029 sedangkan Algoritma A* dengan menghasilkan rata-rata sebesar 0,001 detik, artinya Algoritma A* mampu meminimalkan waktu komputasi untuk mencari rute terpendek. Rata-rata kebutuhan memori Algoritma Genetika secara keseluruhan sebesar 12,56 MB dan Algoritma A* menghasilkan rata-rata sebesar 9,12 MB, artinya Algoritma A* mampu meminimalkan kebutuhan memori untuk mencari rute terpendek.

2. Jumlah populasi terbaik untuk Algoritma Genetika untuk menyelesaikan permasalahan ini berkisar antara 150-200. Apabila populasi semakin banyak maka solusi yang dihasilkan akan semakin bagus tetapi akan menghasilkan waktu komputasi dan kebutuhan memori yang kurang optimal.

3. Dari seluruh aspek yang dijadikan parameter

perbandingan, Algoritma A* merupakan Algoritma yang lebih baik dari Algoritma Genetika untuk mencari rute terpendek karena algoritma tersebut dapat meminimalkan kebutuhan waktu, memori dan menghasilkan jarak tempuh yang lebih pendek dari Algoritma Genetika.

5.2 Saran

Saran yang dapat diberikan oleh penulis adalah sebagai berikut:

1. Diharapkan ada penelitian yang lebih spesifik terkait perbandingan kedua Algoritma dengan permasalahan yang lebih kompleks agar dapat diketahui kelebihan dan kekurangan masing-masing algoritma untuk mencari rute terpendek.

2. Memodifikasi Algoritma Genetika agar dapat mencari rute terpendek dengan lebih meminimalkan kebutuhan waktu komputasi dan memori.

DAFTAR PUSTAKA

[1] Adipranata, Rudy, Felicia Soedjianto, Wahyudi Tjondro, 2009 "Perbandingan Algoritma Exhaustive, Algoritma Genetika Dan Algoritma Jaringan Syaraf Tiruan Hopfield Untuk Pencarian Rute Terpendek" Tugas Akhir Teknik Informatika, Fakultas Teknologi Industri, Universitas Kristen Petra.

[2] J. Yao, B. Zhang and Q. Zhou, "The Optimization of A* Algorithm in the Practical Path Finding Application," 2009 WRI World Congress on Software Engineering, Xiamen, 2009, pp. 514-518, doi: 10.1109/WCSE.2009.412.

[3] Suyanto. Artificial Intelligence Searching, Reasoning, Planning, Learning Revisi Kedua. 2014. Yogyakarta: Informatika.

[4] Xiao Cui and Hao Shi, 2012, An Overview of Pathfinding in Navigation Mesh, IJCSNS International Journal of Computer Science and Network Security, 48 VOL.12 No.12.

[5] MACHADO, A. F. V. ; CLUA, E. W. ; GONÇALVES, R.; VALE, H. ;SANTOS, U. O. ; NEVES, T. ; OCHI, L. S.,2011, "Real Time Pathfinding with Genetic Algorithm". In: SBGames, 2011, Salvador, BA. Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGames).

[6] Goldstone, Will 2009, Unity Game Development Essentials, Packt Publishing, Birmingham.

[7] LEIGH, R., LOUIS, S. J. AND MILES, C., 2007. "Using a Genetic Algorithm to Explore A*-like Pathfinding Algorithms". In: IEEE Congress on Computational Intelligence and Games.

[8] Faathir, M. W. 2018. Perbandingan Algoritma Hosrpool dan Not So Naïve dalam Pembuatan Kamus Bahasa Indonesia – Bahasa Aceh Berbasis Android. Skripsi. Universitas Sumatera Utara.

[9] Adipranata R, Handojo A, Setiawan H. 2007. Aplikasi Pencari Rute Optimum Pada Peta Guna Meningkatkan Efisiensi Waktu Tempuh Pengguna Jalan Dengan Metode A* dan Best First Search. [laporan penelitian]. Surabaya : Universitas Kristen Petra.

[10] H. Reddy, "PATH FINDING - Dijkstra ' s and A * Algorithm ' s," pp. 1–15, 2013.

[11] P. Singal Dcsa and R. R. S. Chhillar, "Dijkstra Shortest Path Algorithm using Global Positioning System," Int. J. Comput. Appl., vol. 101, no. 6, pp. 975–8887, 2014.

[12] A. A. Jamal and W. J. Teahan, "Alpha multipliers breadth-first search technique for resource discovery in unstructured peer-to-peer networks," Int. J. Adv. Sci. Eng. Inf. Technol., vol. 7, no. 4, 2017.

[13] Coley, D.A. 1999. **An Indtroduction to Genetic Algorithms for Scientists and Engineers**. New Jersey : World Scientific.

[14] Nazif, H. dan Lee, L.S. 2012. *"Optimised Crossover Genetic Algorithm for Capacitated Vehicle Routing*

Problem". **Applied Mathematical Modelling** 36, 2110-2117.

[15] Cui, Xiao and Shi, Hao. 2011. *A*-based Pathfinding in Modern Computer Games*. School of Engineering and Science, Victoria University, Melbourne, Australia. IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.1, January 2011.

[16] F. Saptono dan T. Hidayat(2007). *Perancangan Algoritma Genetika untuk Menentukan Jalur Terpendek*. Seminar Nasional Aplikasi Teknologi Informasi. Yogyakarta. 16 Juni 2007.

[17] D.E. Goldberg(1989). *Genetic Algorithms in search, optimization and machine learning*.

[18] Russel, Stuart, and Norvig, Peter, 1995, "Artificial Intelligence:A Modern Approach".Ptentice Hall International,Inc.

LAMPIRAN

LAMPIRAN A

```
1  /*
2  Method ini ada pada class GeneticAlgorithm.cs
       difungsikan untuk menjalankan pathfinding
       dengan algoritma genetika
3  */
4  private IEnumerator StartAlgorithm()
5  {
6      yield return new WaitForSeconds(0.3f);
7      time = new Stopwatch();
8      time.Start();
9      CreateStartPopulation();
10     int epoch = 0;
11     while (epoch < totalEpochs && !solutionFound
           )
12     {
13         generation = epoch;
14         UIManager.Instance.SetGenerationText(
               generation);
15
16         //Update Fitness Scores
17         StartCoroutine(UpdateFitnessScores());
18         yield return new WaitUntil(() =>
               fitnessScoresUpdated);
19
20         if (!solutionFound)
21         {
22             //Reproduce genomes to create new
                   Genertion
23             ReproduceGeneration();
24             epoch++;
25             yield return null;
26         }
```

```
27          else
28          {
29              print("Solution Found");
30          }
31      }
32      ShowPath(bestRouteCoordintes, false);
33      print("Running Time = " + time.
            ElapsedMilliseconds + " ms");
34      print("Total Fitness Score : " +
            totalFitnessScore);
35      print("Process Completed");
36  }
```

LAMPIRAN B

```
1  /*
2  \ Creates a gene
3  */
4  public class Gene
5  {
6      public List<int> Nucleotides { get; set; }
7      public int Lenght { get { return Nucleotides
           .Count; } }
8
9      public Gene() { }
10
11     public Gene(int totNucleotides)
12     {
13         Nucleotides = new List<int>();
14         for (int index = 0; index <
               totNucleotides; index++)
15         {
16             Nucleotides.Add(new int());
17         }
18     }
19
20     public static Gene GetRandom(int geneSize)
21     {
22         Gene gene = new Gene
23         {
24             Nucleotides = new List<int>()
25         };
26         for (int index = 0; index < geneSize;
               index++)
27         {
28             int nucleotide = UnityEngine.Random.
                   Range(0, 2);
```

```
29              gene.Nucleotides.Add(nucleotide);
30          }
31          return gene;
32      }
33
34      public int Decode()
35      {
36          string binary = this.ToString();
37          int code = Convert.ToInt32(binary, 2);
38          return code;
39      }
40
41      public override string ToString()
42      {
43          string gen = string.Empty;
44          for (int index = 0; index < Nucleotides.
                Count; index++)
45          {
46              gen += Nucleotides[index].ToString()
                    ;
47          }
48          return gen;
49      }
50
51      public override bool Equals(object obj)
52      {
53          Gene other = (Gene)obj;
54          if (other == null)
55              return false;
56
57          if (Lenght != other.Lenght)
58              return false;
59
60          for (int index = 0; index < Lenght;
                index++)
61          {
62              if (Nucleotides[index] != other.
                    Nucleotides[index])
```

```
63                      return false;
64              }
65
66              return true;
67          }
68
69          public override int GetHashCode()
70          {
71              return base.GetHashCode();
72          }
73      }
```

LAMPIRAN C

```
1  /*
2  \ Creates a chromosome
3  */
4  public class Chromosome
5  {
6      public List<Gene> Genes { get; set; }
7      public int Lenght { get { return Genes.Count
           ; } }
8
9      public Chromosome() { }
10
11     public Chromosome(int totGenes)
12     {
13         Genes = new List<Gene>();
14         for (int index = 0; index < totGenes;
               index++)
15         {
16             Genes.Add(new Gene(2));
17         }
18     }
19
20     public static Chromosome GetRandom(int
           chromoLenght, int genesLenght)
21     {
22         Chromosome chromosome = new Chromosome
23         {
24             Genes = new List<Gene>()
25         };
26         for (int index = 0; index < chromoLenght
               ; index++)
27         {
```

```
28              chromosome.Genes.Add(Gene.GetRandom(
                    genesLenght));
29          }
30          return chromosome;
31      }
32
33      public override string ToString()
34      {
35          string genes = string.Empty;
36          for (int index = 0; index < Genes.Count;
                index++)
37          {
38              genes += Genes[index].ToString();
39          }
40          return genes;
41      }
42
43      public override bool Equals(object obj)
44      {
45          Chromosome other = (Chromosome)obj;
46          if (other == null)
47              return false;
48
49          if (Lenght != other.Lenght)
50              return false;
51
52          for (int index = 0; index < Lenght;
                index++)
53          {
54              if (!Genes[index].Equals(other.Genes
                    [index]))
55                  return false;
56          }
57
58          return true;
59      }
60
61      public override int GetHashCode()
```

```
62        {
63            return base.GetHashCode();
64        }
65    }
```

LAMPIRAN D

```
1  /*
2  \ Creates a genome
3  */
4  public class Genome
5  {
6      public List<Chromosome> Chromosomes { get;
           set; }
7      public float Fitness { get; set; }
8      public int Lenght { get { return Chromosomes
           .Count; } }
9      /*public static Genome Null = new Genome
10     {
11         Chromosomes = new List<Chromosome>(),
12         Fitness = -1
13     };*/
14     public Genome() { }
15
16     public Genome(int totChromos, int totGenes)
17     {
18         Chromosomes = new List<Chromosome>();
19         for (int index = 0; index < totChromos;
               index++)
20         {
21             Chromosomes.Add(new Chromosome(
                   totGenes));
22         }
23         Fitness = 0;
24     }
25
26     public static Genome GetRandom(int
           genomeLeght = 1, int chromosLeght = 35,
           int genesLenght = 2)
```

```
27      {
28          Genome genome = new Genome
29          {
30              Fitness = 0,
31              Chromosomes = new List<Chromosome>()
32          };
33          for (int index = 0; index < genomeLeght;
                index++)
34          {
35              genome.Chromosomes.Add(Chromosome.
                    GetRandom(chromosLeght,
                    genesLenght));
36          }
37          return genome;
38      }
39
40      public Gene GetNextGene(ref Genome.
            Extraction extrationInfo)
41      {
42          if (extrationInfo.CanStract)
43          {
44              Gene g = Chromosomes[extrationInfo.
                    ChromosomeIndex].Genes[
                    extrationInfo.GeneIndex];
45              extrationInfo.GeneIndex++;
46              if (extrationInfo.GeneIndex ==
                    Chromosomes[extrationInfo.
                    ChromosomeIndex].Lenght)
47              {
48                  extrationInfo.ChromosomeIndex++;
49                  if (extrationInfo.
                        ChromosomeIndex == Lenght)
50                  {
51                      extrationInfo.
                            ChromosomeIndex = -1;
52                      extrationInfo.GeneIndex =
                            -1;
53                      extrationInfo.CanStract =
```

```
                        false;
54                  }
55                  else
56                  {
57                      //Continue with extraction
58                      extrationInfo.GeneIndex = 0;
59                  }
60              }
61              return g;
62          }
63          else
64          {
65              return null;
66          }
67      }
68
69      public override string ToString()
70      {
71          string genome = string.Empty;
72          for (int index = 0; index < Chromosomes.
                Count; index++)
73          {
74              genome = Chromosomes[index].ToString
                    ();
75          }
76          return genome;
77      }
78
79      public override bool Equals(object obj)
80      {
81          Genome other = (Genome)obj;
82          if (other == null)
83              return false;
84
85          if (Lenght != other.Lenght)
86              return false;
87
88          for (int index = 0; index < Lenght;
```

```
                index++)
89          {
90              if (!Chromosomes[index].Equals(other
                    .Chromosomes[index]))
91                  return false;
92          }
93
94          return true;
95      }
96
97      public override int GetHashCode()
98      {
99          return base.GetHashCode();
100     }
101
102     public class Extraction
103     {
104         public int ChromosomeIndex { get; set; }
105         public int GeneIndex { get; set; }
106         public bool CanStract { get; set; }
107
108         public Extraction()
109         {
110             ChromosomeIndex = 0;
111             GeneIndex = 0;
112             CanStract = true;
113         }
114     }
115 }
```

LAMPIRAN E

```
1  /*
2  \ Creates a initial random population with size
       = populationSize
3  */
4  private void CreateStartPopulation()
5  {
6      population = new List<Genome>();
7      for (int index = 0; index < populationSize;
           index++)
8      {
9          population.Add(Genome.GetRandom(1,
               chromosomesLenght, genesLenght));
10     }
11 }
```

LAMPIRAN F

```
1  /*
2  \ Get the parents to reproduces based on the
       ReproductionMethod selected
3  */
4      private void GetParents(ref Genome mom, ref
           Genome dad)
5      {
6          Genome referent;
7          int survivorsLenght;
8          List<Genome> sortedPopulation = new List
               <Genome>();
9          switch (parentSelectionMethod)
10         {
11             case ParentSelectionMethod.Random:
12                 mom = population[Random.Range(0,
                       population.Count)];
13                 dad = population[Random.Range(0,
                       population.Count)];
14                 break;
15             default:
16                 break;
17         }
18     }
```

LAMPIRAN G

```
1  /*
2  \ Cross two genomes for a new genome, in this
       case, two new babies
3  */
4  private void Crossover(Genome mom, Genome dad,
       out Genome baby1, out Genome baby2)
5  {
6      if (Random.Range(0, 1f) > crossoverRate ||
           mom.Equals(dad))
7      {
8          //There is no crossover
9          baby1 = mom;
10         baby2 = dad;
11         return;
12     }
13     int splitPoint = Random.Range(0,
           chromosomesLenght);
14     baby1 = new Genome(1, chromosomesLenght);
15     baby2 = new Genome(1, chromosomesLenght);
16
17     for (int index = 0; index < splitPoint;
           index++)
18     {
19         baby1.Chromosomes[0].Genes[index] = mom.
               Chromosomes[0].Genes[index];
20         baby2.Chromosomes[0].Genes[index] = dad.
               Chromosomes[0].Genes[index];
21     }
22
23     for (int index = splitPoint; index <
           chromosomesLenght; index++)
24     {
```

```
25          baby1.Chromosomes[0].Genes[index] = dad.
                Chromosomes[0].Genes[index];
26          baby2.Chromosomes[0].Genes[index] = mom.
                Chromosomes[0].Genes[index];
27      }
28  }
```

LAMPIRAN H

```
1  /*
2  \ Cross two genomes for a new genome, in this
       case, two new babies
3  */
4  private void Mutate(ref Genome baby)
5  {
6      for (int gene = 0; gene < baby.Chromosomes
           [0].Lenght; gene++)
7      {
8          for (int bit = 0; bit < baby.Chromosomes
               [0].Genes[gene].Lenght; bit++)
9          {
10             if (Random.Range(0, 1f) <
                   mutationRate)
11             {
12                 baby.Chromosomes[0].Genes[gene].
                       Nucleotides[bit] = baby.
                       Chromosomes[0].Genes[gene].
                       Nucleotides[bit] == 0 ? 1 :
                       0;
13             }
14         }
15     }
16 }
```

LAMPIRAN I

```
1  /*
2  \ Try To Move is the way for Genetic Algorithm
       to find the path
3  */
4  private void TryToMove(Vector2 actualPos,
       Vector2 movement, out Vector2 newPos, bool
       allowMoveOverPath)
5  {
6      Vector2 initialPos = actualPos;
7      newPos = actualPos + movement;
8      //Verify that new pos is inside matrix
9      if ((newPos.x > -1 && newPos.x < MazeMatrix.
           GetLength(0)) && (newPos.y > 0 && newPos.
           y < MazeMatrix.GetLength(1)))
10     {
11         //verify that newPos is a wall
12         if (MazeMatrix[(int)newPos.x, (int)
               newPos.y] == (int)MazeElement.Wall)
13         {
14             newPos = initialPos;
15         }
16         if (!allowMoveOverPath)
17         {
18             if (pathMemory[(int)newPos.x, (int)
                   newPos.y] == (int)MazeElement.
                   Path)
19             {
20                 newPos = initialPos;
21             }
22         }
23     }
24     else
```

```
25      {
26          newPos = initialPos;
27      }
28  }
```

LAMPIRAN J

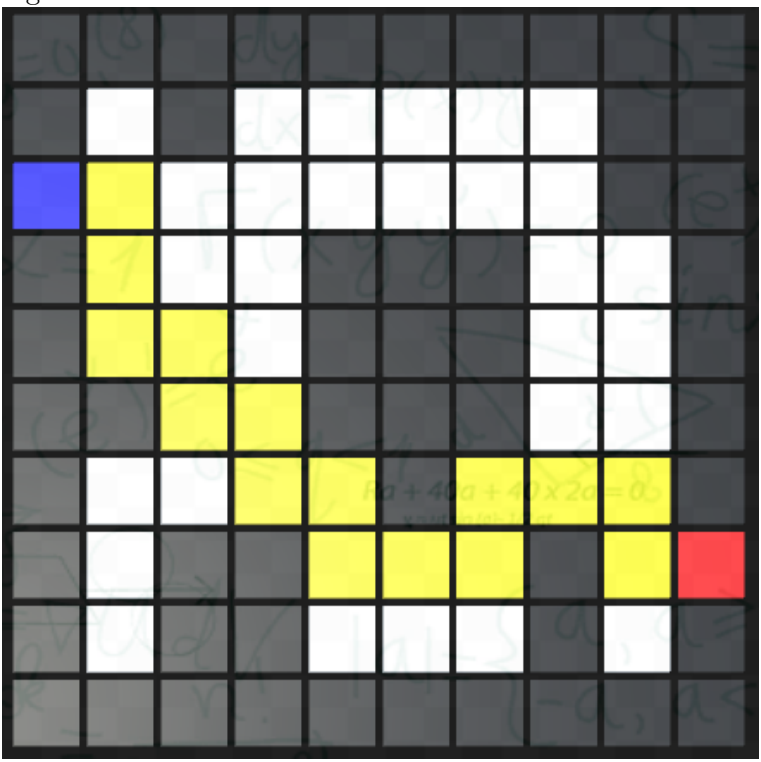
```
1  /*
2  Method ini ada pada class Pathfinding.cs
       difungsikan untuk menjalankan Algoritma A*
3  */
4  void FindPath(Vector3 a_StartPos, Vector3
       a_TargetPos)
5  {
6      Node StartNode = GridReference.
           NodeFromWorldPoint(a_StartPos);
7      Node TargetNode = GridReference.
           NodeFromWorldPoint(a_TargetPos);
8
9      List<Node> OpenList = new List<Node>();
10     HashSet<Node> ClosedList = new HashSet<Node
           >();
11
12     OpenList.Add(StartNode);//Add the starting
           node to the open list to begin the
           program
13
14     while (OpenList.Count > 0)
15     {
16         Node CurrentNode = OpenList[0];
17         for (int i = 1; i < OpenList.Count; i++)
18         {
19             if (OpenList[i].FCost < CurrentNode.
                   FCost || OpenList[i].FCost ==
                   CurrentNode.FCost && OpenList[i].
                   ihCost < CurrentNode.ihCost)
20             {
21                 CurrentNode = OpenList[i];
22             }
```

```
23          }
24          OpenList.Remove(CurrentNode);
25          ClosedList.Add(CurrentNode);
26          if (CurrentNode == TargetNode)//If the
                current node is the same as the
                target node
27          {
28              reachedEnd = true;
29              GetFinalPath(StartNode, TargetNode);
                    //Calculate the final path
30          }
31          foreach (Node NeighborNode in
                GridReference.GetNeighboringNodes(
                CurrentNode))
32          {
33              if (!NeighborNode.bIsWall ||
                    ClosedList.Contains(NeighborNode)
                    )
34              {
35                  continue;
36              }
37              int MoveCost = CurrentNode.igCost +
                    GetManhattenDistance(CurrentNode,
                     NeighborNode);
38
39              if (MoveCost < NeighborNode.igCost
                    || !OpenList.Contains(
                    NeighborNode))
40              {
41                  NeighborNode.igCost = MoveCost;
                        //Set the g cost to the f
                        cost
42                  NeighborNode.ihCost =
                        GetManhattenDistance(
                        NeighborNode, TargetNode);
43                  NeighborNode.ParentNode =
                        CurrentNode;
44                  if (!OpenList.Contains(
```

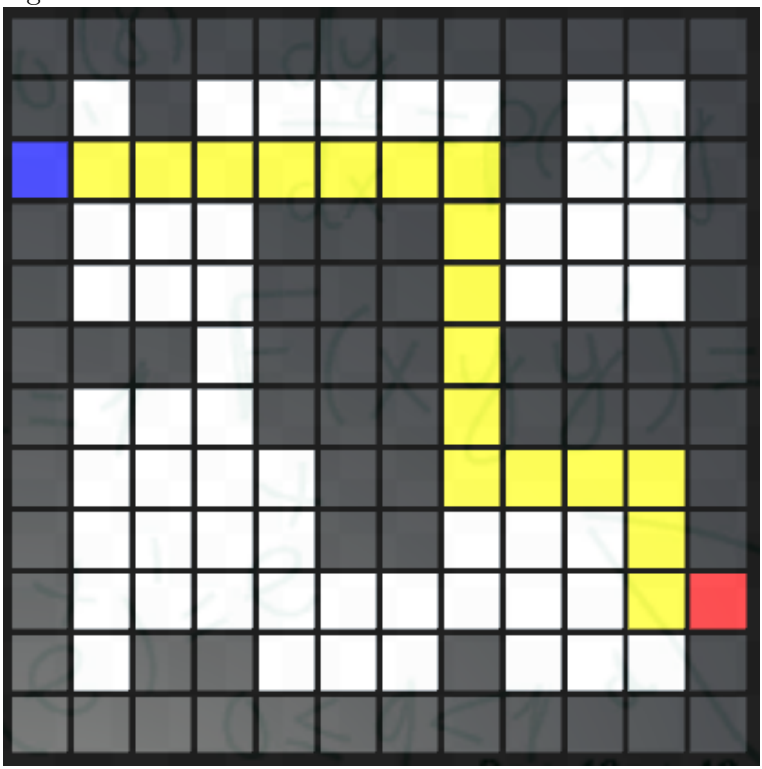
```
                        NeighborNode))
45                      {
46                          OpenList.Add(NeighborNode);
                                //Add it to the list
47                      }
48                  }
49              }
50          }
51  }
```

LAMPIRAN K

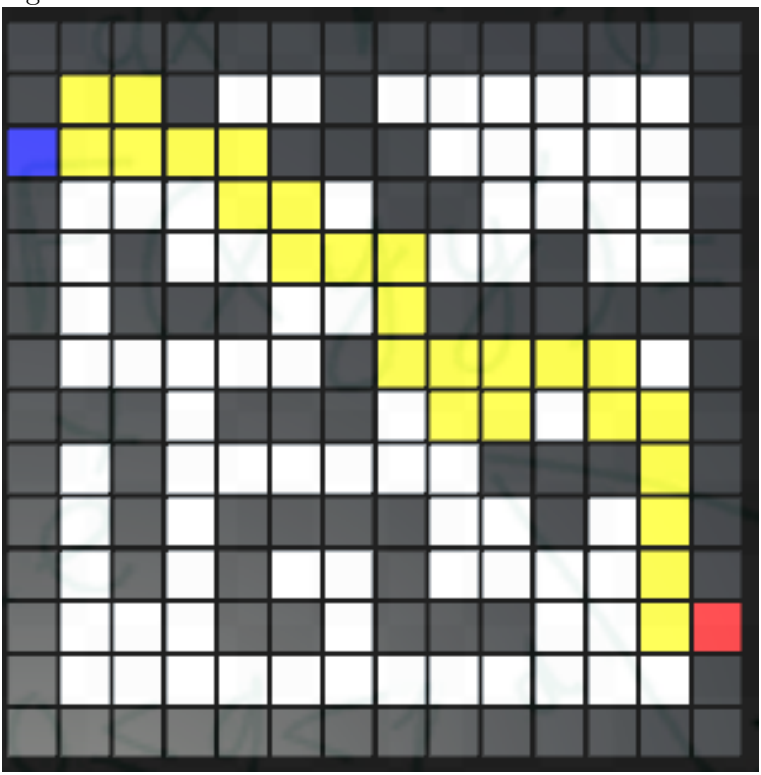
- Algoritma Genetika Permasalahan 1



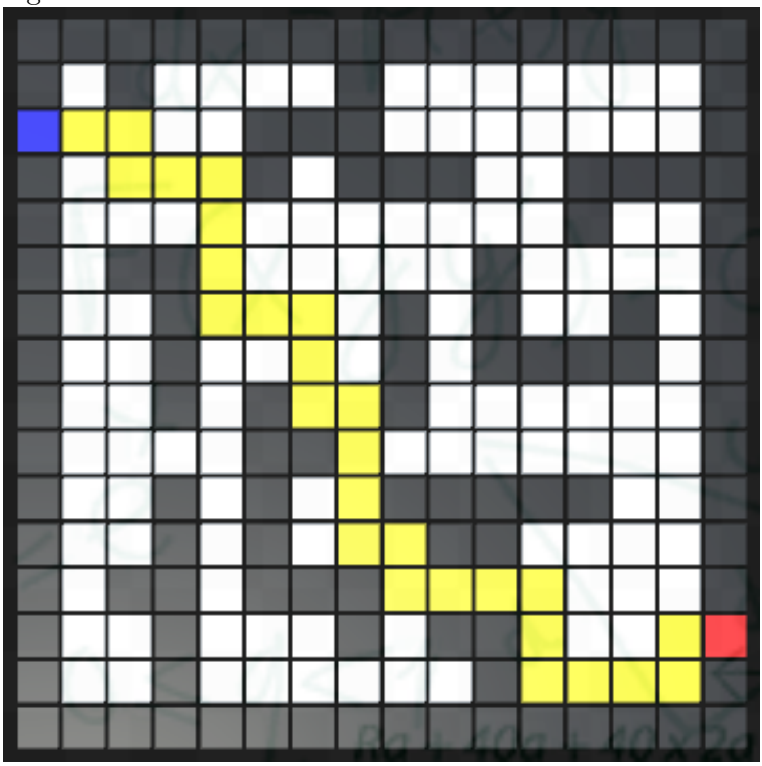
- Algoritma Genetika Permasalahan 2



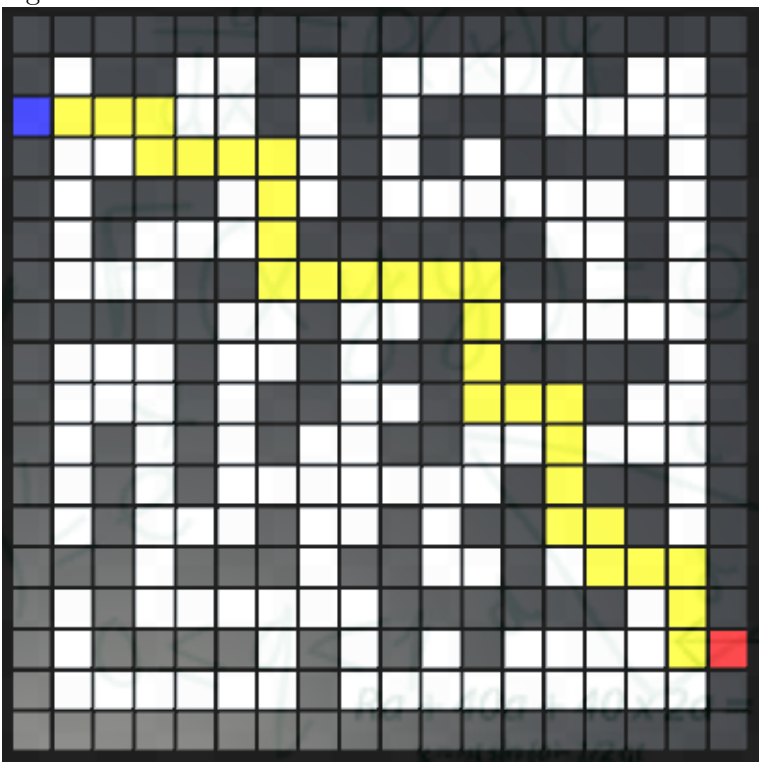
- Algoritma Genetika Permasalahan 3



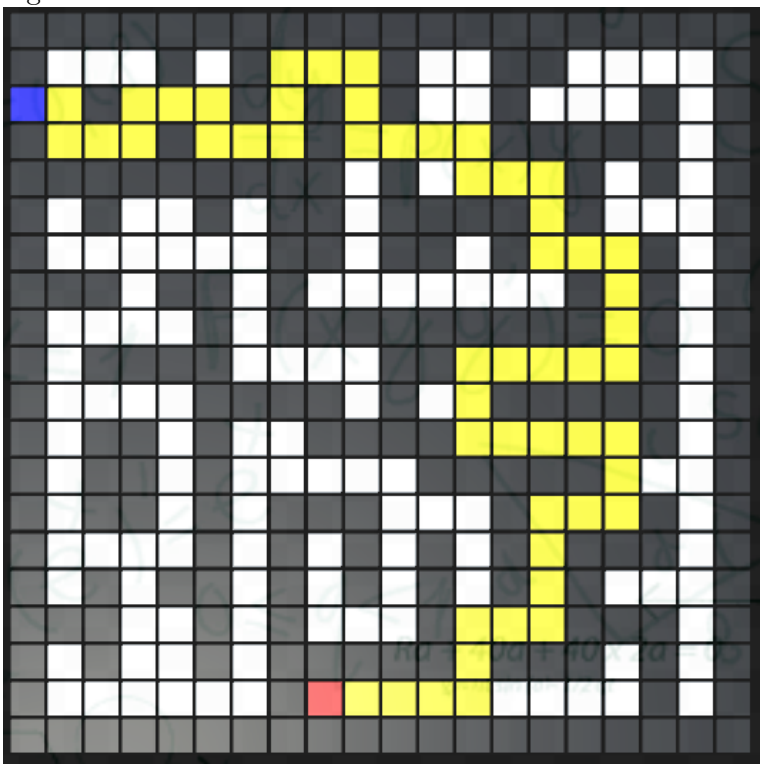
- Algoritma Genetika Permasalahan 4



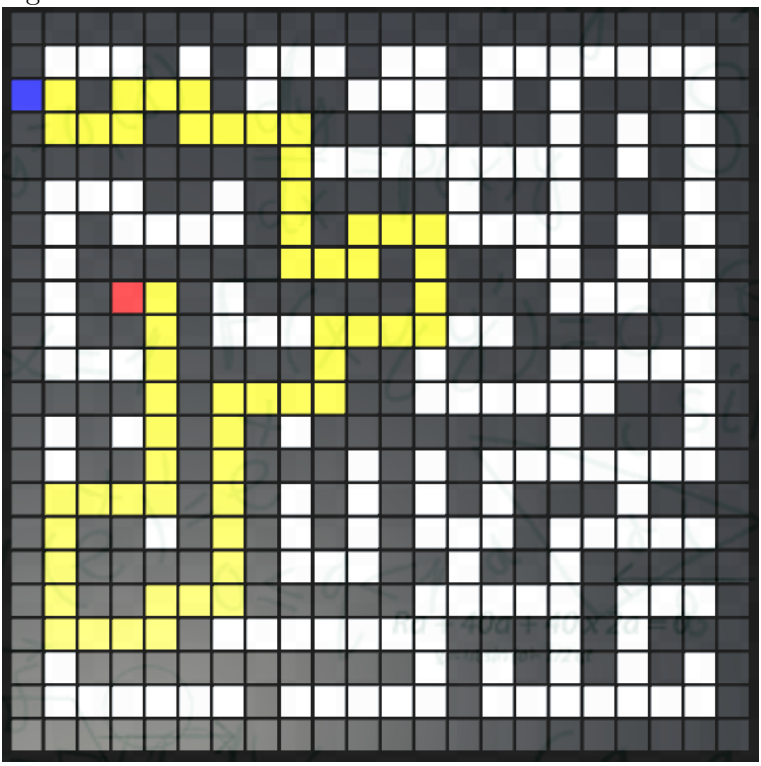
- Algoritma Genetika Permasalahan 5



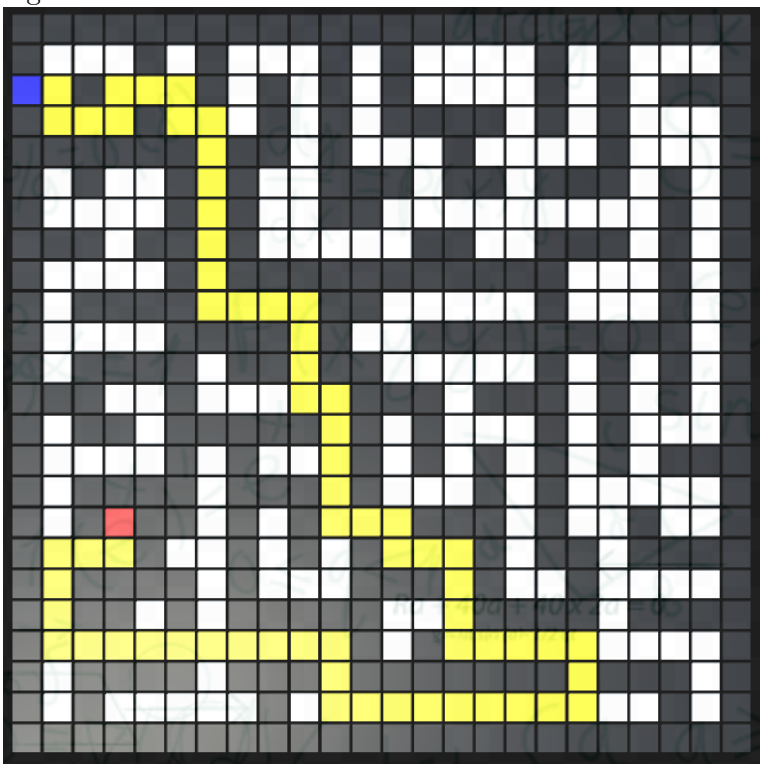
- Algoritma Genetika Permasalahan 6



- Algoritma Genetika Permasalahan 7

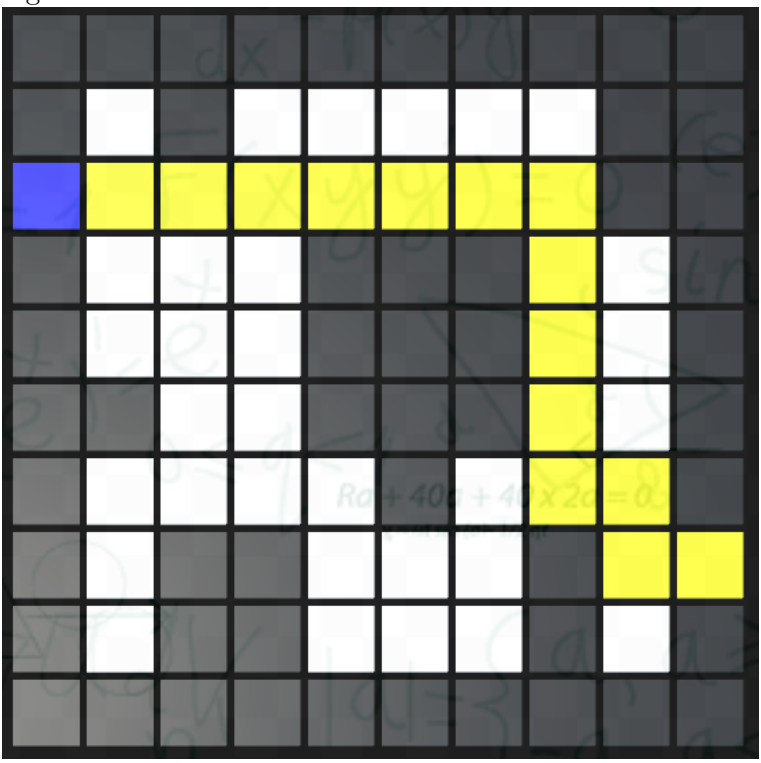


- Algoritma Genetika Permasalahan 8

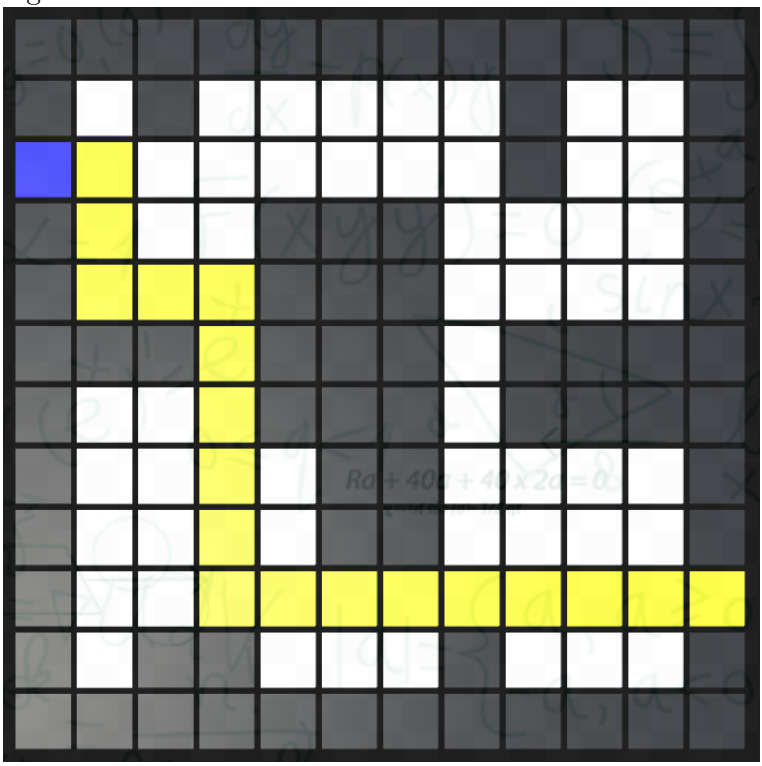


LAMPIRAN L

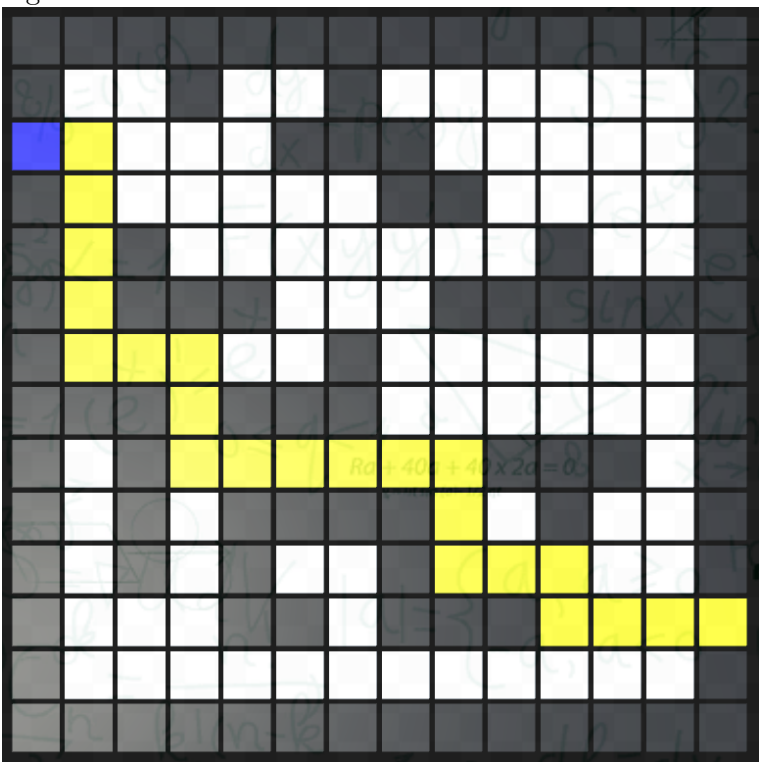
- Algoritma A* Permasalahan 1



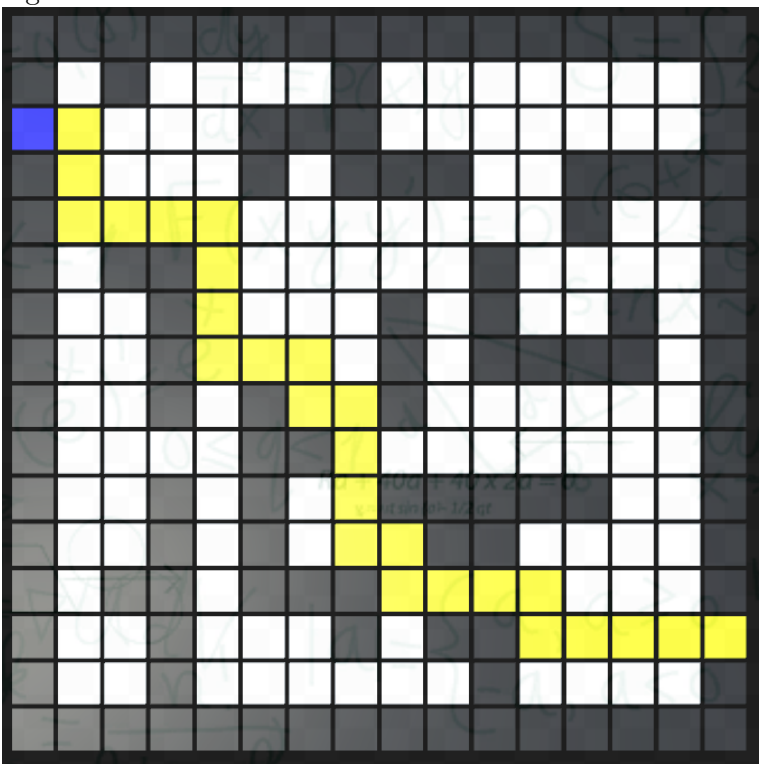
- Algoritma A* Permasalahan 2



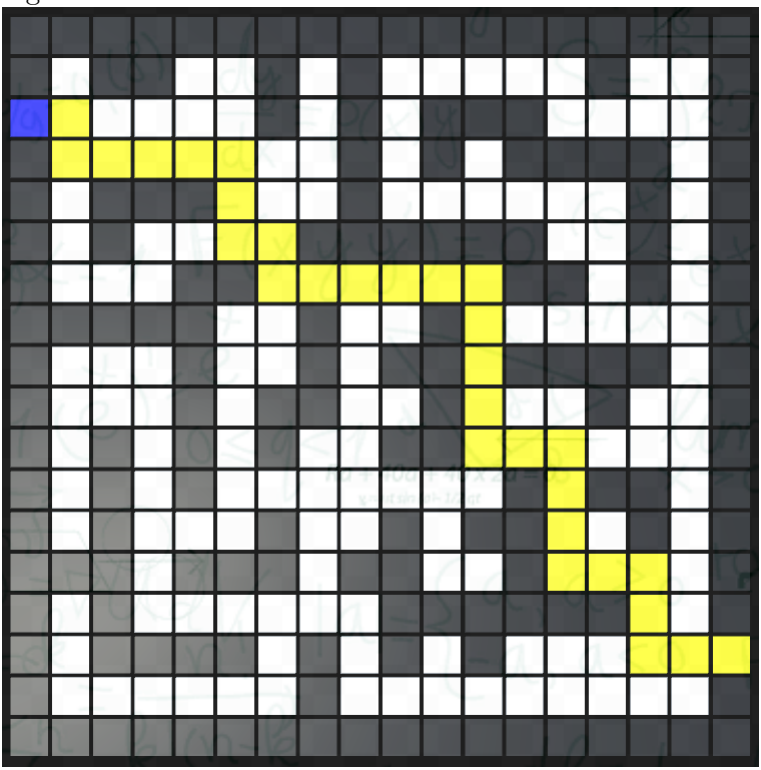
- Algoritma A* Permasalahan 3



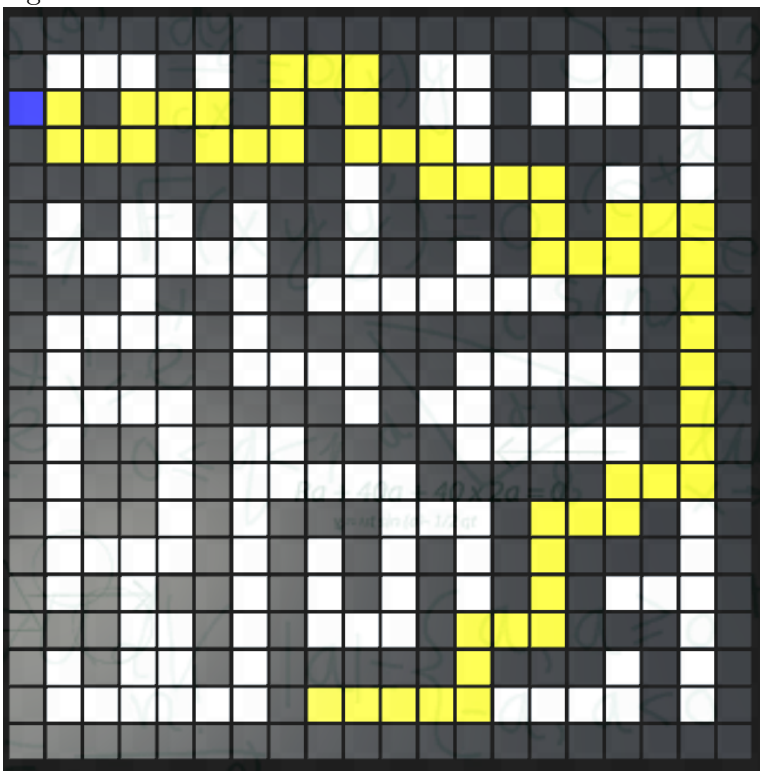
- Algoritma A* Permasalahan 4



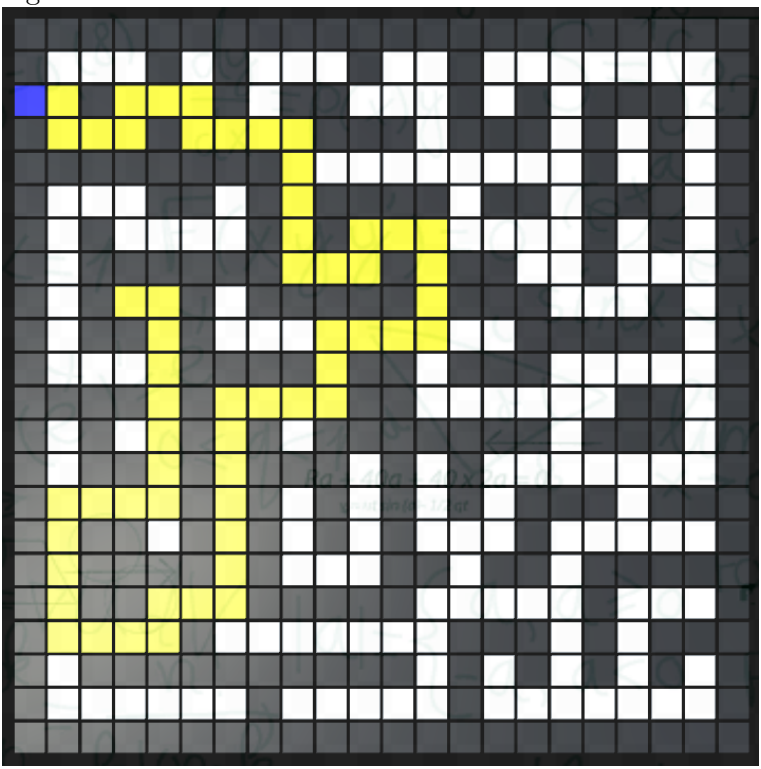
- Algoritma A* Permasalahan 5



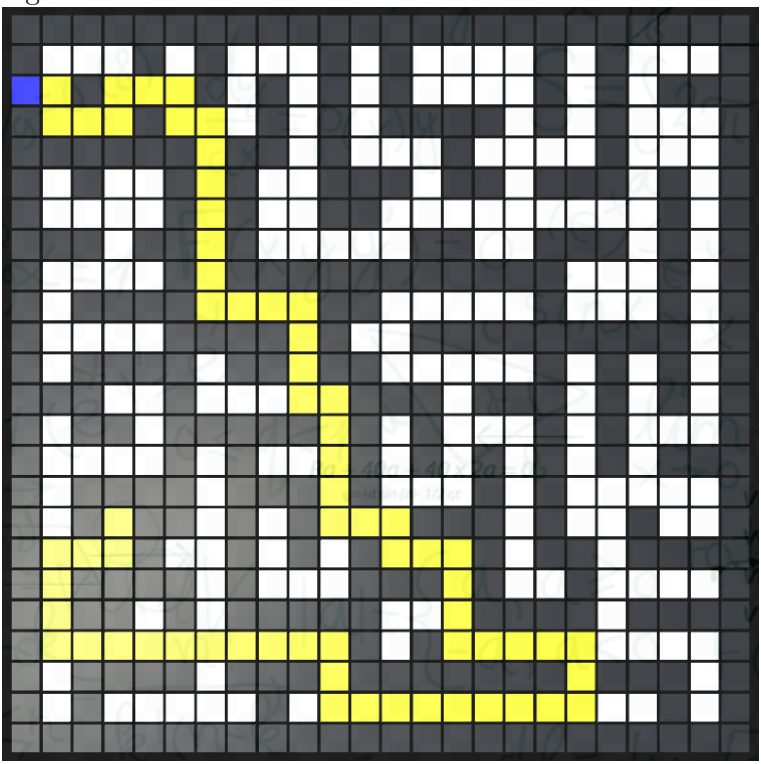
- Algoritma A* Permasalahan 6



- Algoritma A* Permasalahan 7



- Algoritma A* Permasalahan 8



BIODATA PENULIS

Nama lengkap penulis Dimas Ari Nugroho, lahir di Tanjung Pinang pada tanggal 2 Desember 1997. Sebelum menempuh bangku perkuliahan, penulis memulai pendidikan formal mulai dari SD AL-Akbar (2004-2010), SMP Negeri 1 Mojokerto (2010-2013), SMAN 1 Sooko (2013-2016). Pada tahun 2016, penulis meneruskan pendidikan ke jenjang S1 di Departemen Matematika Institut Teknologi Sepuluh Nopember (ITS), melalui jalur SBMPTN dengan nomor registrasi peserta (NRP) 06111640000081. Pada masa perkuliahan penulis aktif dalam berbagai organisasi antara lain Himpunan Mahasiswa Matematika sebagai staff *Internal Affair* (2017-2018), Kepala Divisi *internal* (2018-2019), Penanggung Jawab OMITS *region* Bogor (2018), Staf Perlengkapan OMITS(2018), dan beberapa kali mengikuti kepanitiaan FMIPA.(2016-2017).

Selama di Departemen Matematika, Penulis saat menjalani kuliah lebih dominan mengambil Bidang Studi Ilmu Komputer, dan pada saat tugas akhir penulis mengambil Komputasi. Adapun informasi lebih lanjut mengenai Tugas Akhir ini dapat ditujukan ke penulis melalui email dimas.ira121@gmail.com.