Simulasi Pencarian Shortest Distance Path

Sukiman[1)] Jeffrey[2)]
Teknik Informatika, STMIK IBBI
Jl. Sei Deli No. 18 Medan, Telp. 061-4567111 Fax. 061-4527548
Email : Sukiman_liu@yahoo.com

Abstrak

Path finding merupakan salah satu masalah yang sering dijumpai dan banyak diterapkan, misalnya untuk penentuan jalur terpendek dalam suatu peta dimana diterapkan dalam beberapa kategori *game* seperti *real time strategy game*, analis pergerakan robot cerdas. Salah satu algoritma *path finding* yang terkenal adalah algoritma A* dimana algoritma tersebut mempunyai waktu komputasi yang cepat. Algoritma ini bekerja dengan penentuan terlebih dahulu dua buah titik misalnya titik A dan B dimana titik A merupakan titik asal sedangkan titik B merupakan titik tujuan. Biasanya antara titik A dan B tersebut terdapat penghalang yang akan menentukan jumlah belokan dan jalur terpendek atau terpanjang dari A ke B. Sehingga tujuan dari penelitian ini adalah untuk menjelaskan bagaimana cara kerja atau teknik dari Algoritma A* dalam menentukan jalur terpendek antara dua buah titik dan untuk merancang dan membangun suatu perangkat lunak yang mampu melakukan path finding dengan menggunakan algoritma A*. Metode pengembangan perangkat lunak ini menggunakn A* yang merupakan suatu "perkiraan heuristik" h(x) yang memberi nilai untuk setiap *node* x dengan memperkirakan rute terbaik yang dapat dilalui oleh *node*. A* mengunjungi *node* dengan tujuan untuk perkiraan heuristik ini. Algoritma A* merupakan contoh dari *best-first search*. Dalam penelitian ini diimplementasikan algoritma A* sehingga pencarian lintasan terpendek antara dua titik dapat dengan mudah diperoleh. Kemudian dengan adanya simulasi pencarian anak laki-laki terhadap anak perempuan, dapat menggambarkan lintasan terpendek yang dilalui.

Kata Kunci: Simulasi, *Shortest Path*

Abstract

Path finding is one problem that is common and widely applied, for example, for determining the shortest path in a map which is applied in several categories such as real-time strategy game game, analysts intelligent robot movement. One path finding algorithm that is well-known A algorithm where the algorithm has a fast computation time. The algorithm works by determining the first two points as points A and B where point A is the point of origin, while point B is the point of destination. Usually between points A and B are included barrier that will determine the amount of the bend and the shortest or longest path from A to B. So the purpose of this study is to explain how to work or technique of Algorithm A* to determine the shortest path between two points and to design and build a software that is able to perform path finding using A* algorithm. Software development method is to use your A* which is a "heuristic estimate" h (x) which gives the value for every node x to estimate the best service that can be traversed by the node. A* visiting the node for the purpose of this heuristic estimate. Algorithm A* is an example of a best-first search. In this research implemented the search algorithm A* so that the shortest path between two points can be easily obtained. Then with a simulated search boys against girls, can describe the shortest path traversed.*

Keywords: Simulation, Shortest Path

1. Pendahuluan

Pada saat ini perkembangan teknologi komputasi sedang menuju kepada sistem cerdas. Banyak penelitian tentang sistem cerdas dilakukan oleh para ilmuwan atau insinyur. Mobile robot adalah salah satu sistem cerdas yang sangat cepat perkembangannya. Mobile robot pada jaman sekarang, salah satunya, digunakan untuk ekplorasi, misal pada ekplorasi untuk menganalisis tempat yang belum disentuh manusia atau bahaya jika dilakukan manusia. Mobile robot harus dilengkapi oleh kemampuan *decision making* atau penentu keputusan. Misalnya untuk mendeteksi lingkungan, *Path Planning* (perencanaan jalan seperti jalur jalan raya), *Path Finding* (penentuan jalur antara dua buah titik), dan lain-lain. Karena

sistem *path planning* dan *path finding* merupakan proses *searching* maka diperlukan optimasi pada banyak bagian seperti algoritmanya, pola pencarian, data yang dicari dan lain-lain.

Path finding merupakan salah satu masalah yang sering dijumpai dan banyak diterapkan, misalnya untuk penentuan jalur terpendek dalam suatu peta dimana diterapkan dalam beberapa kategori *game* seperti *real time strategy game*, analisis pergerakan robot cerdas. Salah satu algoritma *path finding* yang terkenal adalah algoritma A* dimana algoritma tersebut mempunyai waktu komputasi yang cepat. Algoritma ini bekerja dengan penentuan terlebih dahulu dua buah titik misalnya titik A dan B dimana titik A merupakan titik asal sedangkan titik B merupakan titik tujuan. Biasanya antara titik A dan B tersebut terdapat penghalang yang akan menentukan jumlah belokan dan jalur terpendek atau terpanjang dari A ke B.

Seperti kita ketahui bahwa masalah mengenai *shortest path* merupakan masalah yang sangat penting dalam kehidupan kita. Misalnya saja pada penentuan jalan antara kota A dan kota B, kita dapat memodelkan kedua kota tersebut dengan sebuah graf dimana simpul menyatakan kota dan sisi menyatakan hubungan terdapat jalan antara kedua kota terbsebut. sisi juga mempunyai tanda berupa bobot yang artinya jarak antar kedua buah kota. Model graf tersebut dapat kita gunakan untuk menyelesaikan masalah pencarian jalan terpendek (*shortest path*). Untuk mencari jalan terpendek antara dua simpul dibutuhkan sebuah algoritma yang bisa bekerja pada suatu model graf.

Disebabkan luasnya permasalahan yang ada, maka peneliti memberikan batasan masalah antara lain:

1. *Path finding* dilakukan terhadap dua titik yaitu A dan B dimana posisi keduanya dapat digeser dan ditentukan sendiri oleh *user*.
2. Aplikasi penelusuran jalan dibuat dengan penggambaran peta (*map*) yang dapat dirancang oleh *user* dan proses penelusuran juga berupa peta kecil (*minimap*).
3. Aplikasi yang dirancang mampu melakukan animasi pergerakan anak laki-laki yang mencari anak perempuan berdasarkan atas *path finding*.
4. Peta (*map*) yang sudah dirancang dapat disimpan *(save)* serta dapat ditampilkan kembali *(load)*.
5. Perancangan peta (*map*) terdiri dari jalan (bernilai 7), rumput (bernilai 10), pasir (bernilai 14) dan halangan/hambatan terdiri dari pohon, kotak bunga dan tanda.
6. Nilai yang dapat di-*input* dari *keyboard* adalah lebar dan tinggi peta, bentuk jalur (*terrain*), interval, dan ukuran langkah.
7. Perangkat lunak dirancang dengan menggunakan bahasa pemrograman *Visual Basic 2005*.

2. Metodologi Penelitian

Simulasi adalah proses mencontoh atau mempergunakan gambaran sebenarnya dari suatu sistem kehidupan nyata tanpa harus mengalaminya pada keadaan yang sesungguhnya. Kecerdasan buatan atau *artificial intelligence* merupakan salah satu bagian ilmu komputer yang membuat mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan oleh manusia. Pada awal diciptakannya, komputer hanya difungsikan sebagai alat hitung saja. Namun seiring dengan perkembangan zaman, maka peran komputer makin mendominasi kehidupan umat manusia. Komputer tidak lagi hanya digunakan sebagai alat hitung, lebih dari itu, komputer diharapkan untuk dapat diberdayakan untuk mengerjakan segala sesuatu yang bisa dikerjakan oleh manusia.

Path finding merupakan suatu bentuk kasus dalam bentuk algoritma bertujuan untuk mencari jalur tercepat dan terpendek dari suatu lokasi titik tertentu ke titik lainnya. *Path finding* merupakan suatu *tool* yang berguna dimana dapat diterapkan pada banyak jenis *game* dalam situasi-situasi yang berbeda.

Suatu graph merupakan struktur data paling umum yang terdiri atas sekumpulan *node* yang berisi data dan tepi yang menghubungkan *node* satu sama lainnya. *Path finding* memungkinkan penggunaan untuk memeriksa apakah terdapat kemungkinan untuk mencapai satu *node* ke *node* yang lain secara cepat dan paling pendek. Sebagai tambahan dapat dipertimbangkan jika hanya terdapat dua *node* yang terhubung tetapi juga terdapat beberapa jarak dan biaya untuk bergerak dari satu titik ke titik satunya lagi jika dilakukan proses penelusuran, maka akan dilakukan pencarian hanya untuk jalur paling dan yang paling murah, cepat ataupun aman.

2.1 Model

Algoritma lintasan terpendek (*shortest path*) yang paling terkenal adalah algoritma A* (dibaca A Star). Algoritma A* diterapkan untuk mencari lintasan terpendek pada graf berarah. Namun, algoritma ini tetap benar untuk graf yang tak-berarah.

Algoritma A* mencari lintasan terpendek dalam sejumlah langkah. Algoritma ini menerapkan strategi *greedy* dalam pengerjaannya. Penerapan strategi *greedy* dalam algoritma A* terlihat pada deskripsi berikut:

Pada setiap langkah, ambil sisi yang berbobot minimum yang menghubungkan sebuah simpul yang telah dipilih dengan sebuah simpul lain yang belum terpilih. Lintasan dari simpul asal ke simpul yang baru haruslah merupakan lintasan yang terpendek diantara semua lintasannya ke simpul-simpul yang belum terpilih.

Misalkan kita tentukan S adalah simpul awal dan T adalah simpul akhir, akan dicari lintasan terpendek (*shortest path*) antara simpul S dan simpul T. Dari deskripsi di atas langkah – langkah yang digunakan oleh algoritma A*. Di bawah ini merupakan representasi klasik dari algoritma A*:

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

$g(n)$ merupakan jarak total yang telah ditempuh untuk mendapatkan posisi awal hingga ke posisi tujuan.

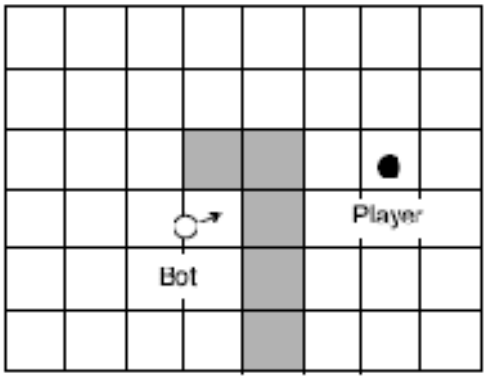
$h'(n)$ merupakan jarak yang diestimasi dari posisi saat ini hingga ke posisi tujuan. Suatu fungsi heuristik digunakan untuk membentuk estimasi ini berkaitan dengan seberapa jauhnya suatu karakter mengambil jalur untuk mencapai ke tujuan.

$f'(n)$ merupakan jumlah dari $g(n)$ dan $h'(n)$. Ini merupakan estimasi dari jalur terpendek. $f(n)$ merupakan jalur terpendek yang benar dimana tidak akan ditemukan apabila algoritma A* belum selesai.

Ilustrasi berikut ini merupakan pola dasar pemikiran dari algoritma A*. Misalkan suatu keluarga akan melakukan rekreasi dimana Ayah, Ibu dan dua orang anak semuanya berada dalam satu mobil. Setelah 5 menit perjalanan, anak-anak tersebut mulai risih dan bertanya pertanyaan yang mengganggu. Salah satu pertanyaannya adalah "Seberapa jauh hingga kita sampai di sana?". Untuk menjawab rasa ingin tahu anak-anak tersebut, sang Ayah memberikan suatu terkaan kasar "sekitar 300 mil lagi. Jadi masih jauh, kenapa kalian tidak tidur dulu?". Jika keluarga tersebut telah mengendarai sejauh 100 mil pada titik tersebut, maka akan direpresentasi dengan $g(n)$, yang merupakan jarak yang telah dikunjungi sejauh ini. Jarak 300 mil direpresentasi dengan $h'(n)$, dan dapat dikira berapa jauh lagi sekarang ini. Jadi $f'(n)$ dapat dinyatakan 100 + 300 = 400 mil.

2.2 **Analisis**

Jika terdapat kondisi lingkungan bersifat *flat* atau datar dan tidak terdapat rintangan maka *path finding* bukanlah menjadi suatu masalah. *Bot* dapat bergerak secara garis lurus langsung ke tujuannya. Tetapi ketika rintangan muncul antara titik A dan titik B, maka masalah tidaklah sesederhana lagi. Sebagai contoh, pada Gambar 2.8, jika *bot* hanya bergerak lurus menuju ke pemain, maka terakhir *bot* akan menabrak dinding dan tidak mampu bergerak lagi.



Gambar 1 Kondisi *Bot* dan Player dengan Penghalang Berupa Dinding

Masalah ini bukanlah solusi terbaik: *bot* (ditandai dengan lingkaran berwarna putih) dapat bergerak lurus menuju ke *player* (lingkaran hitam) bahkan jika terdapat penghalang di antara mereka.

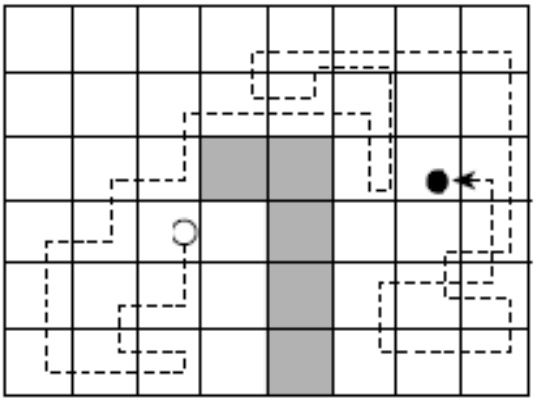
Tingkah laku demikian mungkin tidak menjadi masalah dalam beberapa jenis *game* karena biasanya ini bukanlah merupakan masalah apa yang dilakukan oleh *bot*. Tetapi bila dalam bentuk *game* 3D maka terlihat dalam perspektif pemain bahwa *bot* hanya kelihatan menunggu di balik dinding.

Bagi tampilan perspektif mata burung ini merupakan masalah yang serius. path dinding secara semu diperlukan untuk *game-game* dalam bentuk tampilan mata burung, seperti *game real-time* strategi

seperti *The Sims*. Pada *game-game* seperti ini *bot* berupa lawan biasanya dapat mencari jalan sendiri meskipun terhalang oleh bukit, dinding, ataupun rintangan lainnya.

Selain itu *bot* yang cukup cerdik mencari jalur di sekitar rintangan membuatnya lebih menarik bagi pemain dan membuat kesulitan bagi pemain untuk menipu *bot* dalam melakukan sesuatu ataupun bergerak ke arah tertentu.

Beberapa teknik dapat digunakan untuk mengelilingi rintangan ini. Suatu algoritma acak dapat menemukan solusi secara acak. Teknik ini membuat *bot* berjalan secara acak seperti terlihat pada Gambar 2 memperlihatkan *bot* menemukan apa yang dicarinya.

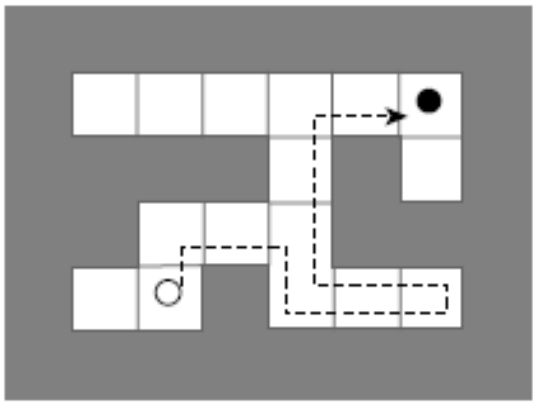


Gambar 2 Pola Pergerakan *Bot* dengan Algoritma Acak

Tentunya algoritma acak tidak memberikan hasil yang terbaik ataupun merupakan algoritma untuk pencarian jalur terpendek. Algoritma acak ini dapat dimodifikasi untuk bergerak secara acak hanya pada beberapa keadaan dan pada waktu yang lain dibuat *bot* mampu bergerak menuju ke *player* secara langsung dimana ini akan membuat pergerakan *bot* sedikit lebih cerdas.

Solusi lain yang dapat ditemukan ketika lingkungan berupa jaringan dengan tidak mempunyai *loop* seperti terlihat pada Gambar 3. Di sini, algoritma *path finding* yang diterapkan disebut algoritma "sebelah kanan dinding" (ataupun algoritma "sebelah kiri dinding). Hanya dengan terus bergerak pada sisi kanan dari dinding sewaktu menelusuri jaringan ini, maka sekali kebetulan *bot* akan menemukan tujuannya.

Bot (lingkaran berwarna putih) bergerak dalam suatu jaringan *maze*, dengan tetap bergerak ke sisi kanan dari dinding hingga ia menemukan pemain (lingkaran berwarna hitam).



Gambar 3 Proses Penelusuran *Path* dengan Algoritma ”Sebelah kanan dinding”

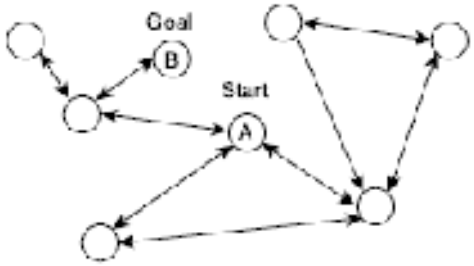
Jika *bot* mengikuti algoritma "sisi kanan" secara terus menerus, maka *bot* akan sangat jarang menemukan jalur terpendek dari A ke B. Sebagai alternatif, path dapat dikalkulasi terlebih dahulu, sebelum *bot* membuat setiap pergerakan. Kemudian setiap *backtracking* (seperti ketika melakukan spekulasi pada bagian jalur kanan bawah pada gambar) dapat dihilangkan dari *path*, sehingga hanya perlu mencari solusi untuk jalur terpendek.

Namun, algoritma ini bekerja hanya untuk kasus-kasus sederhana dengan jaringan maze tanpa adanya *loop*. Jika terdapat suatu *loop* dalam *path*, *bot* tidak akan pernah mencapai tujuannya dengan hanya selalu bergerak ke sisi kanan. Juga, jika ruangan ataupun lingkungan bertambah luas, *bot* mungkin

tidak akan pernah menemukan destinasinya untuk kasus destinasi dalam ruangan tengah karena *bot* akan selalu selalu menempel ke dinding. Idealnya, dalam *game-game* tidak akan membentuk lingkungan interaktif seperti ini.

Selain memperlakukan lingkungan sebagai suatu *grid*, maka pada lingkungan dapat dilihat sebagai suatu *graph*, seperti terlihat pada Gambar

Suatu *graph* sederhananya dapat berupa sekumpulan *node-node* yang dikelompokkan dengan tepi-tepi yang berbeda.



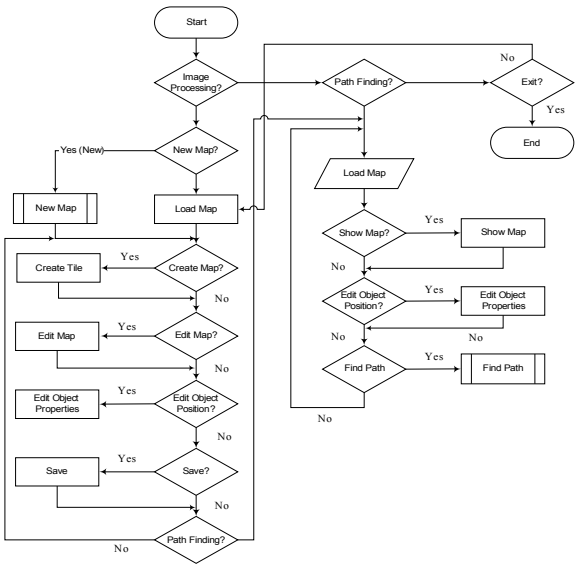
Gambar 4 Suatu Graph Sederhana

Setiap *node* dari graph dapat berupa apapun. Sebagai contoh, suatu *node* dapat berupa suatu sel dalam *grid* 2D ataupun suatu *node* dapat berupa "kota" dengan tepi-tepi dari *graph* merepresentasikan jalur tol. Dan perlu diingat, bahwa ketika menemukan *path* dari A ke B, setiap *node* dapat menjadi *node* awal ataupun *node* tujuan.

Suatu *graph* mirip dengan *tree*, kecuali selain setiap *node* mempunyai dua anak, setiap *node* dapat mempunyai jumlah atas yang tidak terbatas, ataupun tetangga. Beberapa *graph* merupakan jenis *directed*, yang berarti suatu sisi antara dua *node* dapat ditarik hanya dengan satu arah. Perhatikan contoh pada Gambar 4 di atas, semua tepi-tepi bersifat *bidirectional* kecuali hanya satu yang tidak mempunyai dua arah.

2.3 **Perancangan**

Secara garis besar, sistem ini terdiri dari dua bagian utama, yaitu bagian pertama adalah bagian pengolahan animasi dan bagian kedua yaitu proses pencarian rute terpendek dengan algoritma A*. Sebelum dapat dilakukan pencarian rute terpendek, harus tersedia citra yang sudah diubah menjadi *graph*bobot sesuai kebutuhan dan ketentuan yang ada.



Gambar 5 *Flowchart* Kerja Aplikasi

Bagian pengolahan animasi mempunyai dua pilihan, yaitu pembuatan peta baru atau pemasukkan peta yang sudah ada ke dalam aplikasi. Jika peta sudah aktif, dapat dilakukan berbagai proses, yaitu: pembuatan jalan, pengubah citra, pengubahan keterangan dari objek-objek yang ada, penyimpanan peta dan dapat pula melanjutkan ke pencarian rute.

Bagian pencarian rute mempunyai tiga pilihan, yaitu melihat animasi, pengubahan keterangan pada objek-objek yang ada dan pencarian rute. Pengubahan keterangan pada jalan dapat dilakukan pada saat pengolahan animasi ataupun pada saat pencarian rute terbaik. Keterangan jalan yang dapat diubah adalah jarak, tingkat kepadatan objek.

2.4 Implementasi

Setelah proses tahapan kompilasi telah dilakukan dan program telah diinstall maka langkah pertama untuk menjalankan program ini adalah melalui Tombol *Start→Path Finding→Path Finding*. Selain itu untuk lebih mudah, *user* dapat mengklik langsung pada *icon* ataupun *filePathFinding.exe*. Setelah itu program akan dieksekusi dan di-*load* ke memori setelah selesai maka pada layar akan muncul sebuah *window* menu seperti terlihat pada Gambar 6.

Gambar 6 Tampilan Proses Penentuan Jalur

Pada tampilan di atas proses untuk penentuan jalur dilakukan dengan melakukan pembuatan jalur dengan menyertakan halangan berupa rumput, jalan, pasir, dan tiga halangan seperti kotak bunga, tanda, dan pohon. Khusus untuk halangan seperti kotak bunga, tanda dan pohon tidak mempunyai *cost* karena bagian ini merupakan bagian yang tidak dapat dilalui. Sedangkan untuk jalur seperti rumput mempunyai nilai *cost* sebesar 10, jalan mempunyai nilai *cost* 7 dan pasir mempunyai nilai *cost* 14.

Pada penentuan jalur ini jika terdapat kemungkinan jalur yang harus dilalui apakah merupakan rumput dan jalan maka karakter akan memilih jalur jalan dikarenakan mempunyai nilai *cost* 7.

Setiap perpindahan dari karakter, maka akan dilakukan perhitungan ulang untuk semua kemungkinan jalur menuju ke target dari karakter titik awal. Setelah itu akan dilakukan pergerakan animasi karakter.

Untuk membuat peta baru, maka langkah pertama yang harus dilakukan melalui akses dari menu Arsip → Baru atau melalui *shortcut* Ctrl + N. Ketika menu baru dipilih maka program akan memunculkan sebuah kotak dialog untuk menentukan lebar dan tinggi dari peta yang ada. Untuk setiap tinggi dan lebar peta hanya diperbolehkan hingga 25 *point*.

3. Hasil dan Analisis

Dari hasil pembahasan dilakukan, maka dapat disimpulkan bahwa aplikasi penelusuran jalan telah berjalan dan sesuai dengan yang diharapkan. Pada bagian penilaian aplikasi ini akan dilakukan penilaian secara kualitatif.

Penilaian secara kualitatif dilakukan dengan cara membagikan angket kepada beberapa responden. Tujuan pembagian angket ini untuk mendapatkan penilaian subjektif seseorang setelah melihat citra asli dan citra yang telah disisipi dengan informasi. Penilaian yang dipakai menggunakan lima skala nilai yaitu: sama persis (= 5), sama (=4), agak sama (=3), berbeda (=2) dan sangat berbeda (=1).

Data yang terkumpul dikelompokkan berdasarkan citra dan penilaiannya, kemudian dihitung nilai *Mean Opinion Score* (MOS) seperti terlihat pada Tabel 4.2 berikut. Responden sebanyak 30 orang, dipilih secara acak dan terdiri atas: mahasiswa 73,33%, karyawan 23,33%, dan pelajar 3,34%.

Tabel 1 Hasil Penilaian Kualitatif

NamaMap	Nilai Skala	5	4	3	2	1	MOS
Map 1	12	14	4	0	0	4,27	4,27
Map 2	10	15	3	2	0	4,10	4,10
Map 3	13	9	7	1	0	4,13	4,13
Total	35	38	14	3	0		

4. Kesimpulan dan Saran

4.1. Kesimpulan

Berdasarkan pembahasan yang telah dilakukan maka disimpulkan sebagai berikut:

1. Dalam aplikasi ini dimana dua buah titik dapat berpindah maka setiap terjadi perpindahan titik awal dan titik target maka perhitungan akan dihitung kembali karena pada jenis animasi antara karakter dibuat dengan cara jika terdapat perpindahan karakter maka jalur pencarian akan dihitung ulang.
2. Metode yang digunakan ini bukanlah tanpa masalah. Jika terdapat kasus nilai *cost* yang sama, algoritma dengan menggunakan metode ini akan mengikuti .jalur yang pertama ditemuinya dengan mengacu pada sumbu *x* kanan.

4.2. Saran

Untuk pengembangan lebih lanjut program *path finding* ini, maka dapat diberikan beberapa saran sebagai berikut:

1. Proses *zooming* yang dapat dilakukan untuk beberapa bentuk tampilan sehingga bentuk rintangan dapat diperlihatkan secara jelas.
2. Proses penempatan halangan dapat disimulasikan sesuai dengan keinginan *user* sehingga *user* dapat mensimulasikan suatu lingkungan dengan halangantertentu untuk mencari jalur terpendek.
3. Menambahkan jenis metode *path finding* lain sehingga dapat dilakukan perbandingan kecepatan proses dan jalur terpendek yang dihasilkan.

Daftar Pustaka

[1] Kuswadi, S., *Kendali Cerdas: Teori dan Aplikasi Praktisnya*, Penerbit Andi, Yogyakarta, 2007.

[2] Leyden, M., Toal, D., Flanagan, C., *A Fuzzy Logic Based Navigation System for a Mobile Robot*, Department of Electronic & Computer Engineering, University of Florida.

[3] Pitowarno, E., *Robotika: Desain, Kontrol, dan Kecerdasan Buatan*, Penerbit Andi, Yogyakarta, 2006.

[4] *Fuzzy Logic – An Introduction*, http://www.seattlerobotics.org/encoder/ mar98/fuz/flindex.html/Limerick, mark.leyden@ul.ie, daniel.toal@ul.ie.

[5] Lester, P.,, *A* Pathfinding for Beginners*, http:// www. policyalmanac. org/games/aStarTutorial.htm, tanggal akses 22 Mei 2012.

[6] N. N, *Path Finding*, Chapter 12, 1592730051c.Pdf, 2006, tanggal *download* 22 Mei 2012.

[7]Society of Robots, *Introduction to Microcontrollers*, http://www.societyofrobots.com/micro-controller_tutorial.shtml/

[8] Sudirman, I., *Perkembangan Software Komputer*, Kuliah Pengantar Ilmu Komputer.com, 2003.