IMPLEMENTASI ALGORITMA DYNAMIC WEIGHTING A* UNTUK PENCARIAN RUTE TERPENDEK PADA NPC DAN FISHER-YATES SHUFFLE UNTUK PENGATURAN KONTEN PADA GAME 3D FINDING DIAMOND

SKRIPSI

Oleh:
IBNU ZAKIFARDAN
NIM. 11650008

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2016

IMPLEMENTASI ALGORITMA DYNAMIC WEIGHTING A* UNTUK PENCARIAN RUTE TERPENDEK PADA NPC DAN FISHER-YATES SHUFFLE UNTUK PENGATURAN KONTEN PADA GAME 3D FINDING DIAMOND

SKRIPSI

Diajukan Kepada:
Fakultas Sains dan Teknologi
Universitas Islam Negeri (UIN)
Maulana Malik Ibrahim Malang
Untuk Memenuhi Salah Satu Persyaratan Dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh:
Ibnu Zakifardan
NIM. 11650008

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK BRAHIM MALANG
2016

IMPLEMENTASI ALGORITMA DYNAMIC WEIGHTING A* UNTUK PENCARIAN RUTE TERPENDEK PADA NPC DAN FISHER-YATES SHUFFLE UNTUK PENGATURAN KONTEN PADA GAME 3D FINDING DIAMOND

SKRIPSI

Oleh:
IBNU ZAKIFARDAN
NIM. 11650008

Telah disetujui oleh:

Dosen Pembimbing I	**Dosen Pembimbing II**
Dr. Muhammad Faisal, M.T	**Irwan Budi Santoso, M.Kom**
NIP. 19740510 200501 1 007	**NIP. 19770103 201101 1 004**

Tanggal, 16 Mei 2016

Mengetahui,
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

IMPLEMENTASI ALGORITMA DYNAMIC WEIGHTING A* UNTUK PENCARIAN RUTE TERPENDEK PADA NPC DAN FISHER-YATES SHUFFLE UNTUK PENGATURAN KONTEN PADA GAME 3D FINDING DIAMOND

SKRIPSI

Oleh:
IBNU ZAKIFARDAN
NIM. 11650008

Telah Dipertahankan di Depan Dewan Penguji Skripsi dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan Untuk Memperoleh Gelar Sarjana Komputer (S.Kom)
Tanggal 21 Juni 2016

Susunan Dewan Penguji		Tanda Tangan
1. Penguji Utama	: Hani Nurhayati, M.T NIP. 19780625 200801 2 006	( )
2. Ketua	: Yunifa Miftachul Arif, M.T NIP. 19830616 201101 1 004	( )
3. Sekretaris	: Dr. Muhammad Faisal, M.T NIP. 19740510 200501 1 007	( )
4. Anggota	: Irwan Budi Santoso, M.Kom NIP. 19770103 201101 1 004	( )

Mengetahui dan Mengesahkan
Ketua Jurusan Teknik Informatika

Dr.Cahyo Crysdian
NIP. 19740424 200901 1 008

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan di bawah ini:

Nama : IBNU ZAKIFARDAN

NIM : 11650008

Fakultas / Jurusan : Sains dan Teknologi / Teknik Informatika

Judul Penelitian : **IMPLEMENTASI ALGORITMA DYNAMIC WEIGHTING A* UNTUK PENCARIAN RUTE TERPENDEK PADA NPC DAN FISHER-YATES SHUFFLE UNTUK PENGATURAN KONTEN PADA GAME 3D FINDING DIAMOND**

Menyatakan dengan sebenar-benarnya bahwa hasil penelitian saya ini tidak terdapat unsur-unsur penjiplakan karya penelitian atau karya ilmiah yang pernah dilakukan atau dibuat oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata hasil penelitian ini terbukti terdapat unsur-unsur penjiplakan, maka saya bersedia untuk mempertanggungjawabkan, serta diproses sesuai peraturan yang berlaku.

Malang, 11 Mei 2016

Yang membuat pernyataan

Ibnu Zakifardan

NIM. 11650008

MOTO

"Sebaik-baik manusia adalah yang paling bermanfaat bagi sesama"

(HR. Ahmad)

UNIVERSITAS ISLAM NEGERI
MAULANA MALIK IBRAHIM
PUSAT PERPUSTAKAAN

PERSEMBAHAN

Karya ilmiah skripsi ini saya persembahkan untuk kedua orang tua, Ayahanda dan Ibunda tercinta. Yang selama ini telah membesarkan dan mendidik dengan penuh kasih sayang, kesabaran dan keikhlasan. Dan semoga Allah SWT membalas semua kebaikan mereka. Amin.

UNIVERSITAS ISLAM NEGERI
MAULANA MALIK IBRAHIM
PUSAT PERPUSTAKAAN

KATA PENGANTAR

Assalamu'alaikum Wr. Wb.

Segala puji bagi Allah SWT Tuhan semesta alam, karena atas segala rahmat dan karunia-Nya sehingga penulis mampu menyelesaikan skripsi dengan judul "Implementasi Algoritma *Dynamic Weighting A** Untuk Pencarian Rute Terpendek Pada NPC dan *Fisher-Yates Shuffle* Untuk Pengaturan Konten Pada *Game 3D Finding Diamond*" dengan baik dan lancar. Shalawat serta salam selalu tercurah kepada tauladan terbaik Nabi Muhammad SAW yang telah membimbing umatnya dari zaman kebodohan menuju Islam yang *rahmatan lil alamiin*.

Dalam penyelesaian skripsi ini, banyak pihak yang telah memberikan bantuan baik secara moril, nasihat dan semangat maupun materiil. Atas segala bantuan yang telah diberikan, penulis ingin menyampaikan doa dan ucapan terimakasih yang sedalam-dalamnya kepada:

1. Prof. Dr. H. Mudjia Raharjo, M.Si, selaku Rektor Universitas Islam Negeri Maulana Malik Ibrahim Malang beserta seluruh staf. Bakti Bapak dan Ibu sekalian terhadap UIN Maliki Malang turut membesarkan dan mencerdaskan penulis.
2. Dr. Hj. Bayyinatul Muchtaromah, drh., M.Si, selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang beserta seluruh staf. Bapak dan ibu sekalian sangat berjasa memupuk dan menumbuhkan semangat untuk maju kepada penulis.
3. Bapak Dr. Cahyo Crysdian, selaku Ketua Jurusan Teknik Informatika Universitas Islam Negeri Maulana Malik Ibrahim Malang, yang sudah memberi banyak pengetahuan, inspirasi dan pengalaman yang berharga.

4. Bapak Dr. Muhammad Faisal, M.T selaku dosen pembimbing I yang telah meluangkan waktu untuk membimbing, memotivasi, mengarahkan dan memberi masukan kepada penulis dalam pengerjaan skripsi ini hingga akhir.
5. Bapak Irwan Budi Santoso, M.Kom selaku dosen pembimbing II yang juga senantiasa memberi masukan dan nasihat serta petunjuk dalam penyusunan skripsi.
6. Bapak, Ibu, dan Adik-adik serta keluarga besar tercinta yang selalu memberi dukungan yang tak terhingga serta doa yang senantiasa mengiringi setiap langkah penulis.
7. Segenap Dosen Teknik Informatika yang telah memberikan bimbingan keilmuan kepada penulis selama masa studi.
8. Teman – teman seperjuangan Teknik Informatika 2011
9. Para peneliti yang telah mengembangkan *Game* dengan *Engine Unity* yang menjadi acuan penulis dalam pembuatan skripsi ini. Serta semua pihak yang telah membantu yang tidak bisa disebutkan satu satu. Terimakasih banyak.

Berbagai kekurangan dan kesalahan mungkin pembaca temukan dalam penulisan skripsi ini, untuk itu penulis menerima segala kritik dan saran yang membangun dari pembaca sekalian. Semoga apa yang menjadi kekurangan bisa disempurnakan oleh peneliti selanjutnya dan semoga karya ini senantiasa dapat memberi manfaat. Amin. *Wassalamualaikum Wr. Wb.*

Malang, 11 Mei 2016

Penulis

DAFTAR ISI

HALAMAN JUDUL ..... **i**
HALAMAN PENGAJUAN ..... **ii**
HALAMAN PERSETUJUAN ..... **iii**
HALAMAN PENGESAHAN ..... **iv**
PERNYATAAN KEASLIAN TULISAN ..... **v**
MOTO ..... **vi**
PERSEMBAHAN ..... **vii**
KATA PENGANTAR ..... **viii**
DAFTAR ISI ..... **x**
DAFTAR GAMBAR ..... **xii**
DAFTAR TABEL ..... **xiv**
ABSTRAK ..... **xv**

BAB I PENDAHULUAN **1**
1.1 Latar Belakang 1
1.2 Identifikasi Masalah 3
1.3 Batasan Masalah 3
1.4 Tujuan Penelitian 4
1.5 Manfaat Penelitian 4
1.6 Tahapan Penelitian 5

BAB II KAJIAN PUSTAKA **6**
2.1 Landasan Teori 6
2.1.1 Teori *Game* 6
2.1.2 *Genre Game* 7
2.1.3 *Non-Player Character* 10
2.1.4 *Finite State Machine* 11
2.1.5 Nabi dan Rasul 12
2.1.6 Algoritma A* 23
2.1.7 Algoritma *Dynamic Weighting A** 32
2.1.8 Algoritma *Fisher-Yates Shuffle* 39
2.2 Penelitian Terkait 41

BAB III PERANCANGAN GAME 45

3.1 Perancangan *Game* 45
3.1.1 Keterangan Umum *Game* 45
3.1.2 *Story Board Game* 47
3.1.3 Deskrispi Karakter 51
3.1.4 Deskripsi *Item* 52
3.2 Perancangan *Finite State Machine* 53
3.3 Perancangan Algoritma *Dynamic Weighting A** 54
3.4 Perancangan Algoritma *Fisher-Yates Shuffle* 74

BAB IV HASIL DAN PEMBAHASAN 77

4.1 Implementasi 77
4.1.1 Kebutuhan Perangkat Keras 77
4.1.2 Kebutuhan Perangkat Lunak 77
4.1.3 Implementasi Algoritma *Dynamic Weighting A** 78
4.1.4 Implementasi Algoritma *Fisher-Yates Shuffle* 84
4.1.5 Implementasi Aplikasi *Game* 86
4.2 Uji Coba 93
4.2.1 Uji Coba Algoritma *Dynamic Weighting A** 93
4.2.2 Uji Coba Algoritma *Fisher-Yates Shuffle* 104
4.2.3 Uji Coba Aplikasi *Game* 109
4.3 Integrasi Dalam Islam 110

BAB V PENUTUP 114

5.1 Kesimpulan 114
5.2 Saran 115

DAFTAR PUSTAKA 116

DAFTAR GAMBAR

Gambar 2.1 Masalah Pencarian Tute Terpendek Pada Suatu Graf 27
Gambar 2.2 Pencarian Rute Terpendek dengan Algoritma A* Langkah 1 27
Gambar 2.3 Pencarian Rute Terpendek dengan Algoritma A* Langkah 2 28
Gambar 2.4 Pencarian Rute Terpendek dengan Algoritma A* Langkah 3 29
Gambar 2.5 Pencarian Rute Terpendek dengan Algoritma A* Langkah 4 29
Gambar 2.6 Pencarian Rute Terpendek dengan Algoritma A* Langkah 5 30
Gambar 2.7 Pencarian Rute Terpendek dengan Algoritma A* Langkah 6 31
Gambar 2.8 Pencarian Rute Terpendek dengan Algoritma DWA* Langkah 1 34
Gambar 2.9 Pencarian Rute Terpendek dengan Algoritma DWA* Langkah 2 35
Gambar 2.10 Pencarian Rute Terpendek dengan Algoritma DWA* Langkah 3 35
Gambar 2.11 Pencarian Rute Terpendek dengan Algoritma DWA* Langkah 4 36
Gambar 2.12 Pencarian Rute Terpendek dengan Algoritma DWA* Langkah 5 37
Gambar 2.13 Pencarian Rute Terpendek dengan Algoritma DWA* Langkah 6 38
Gambar 3.1 Karakter Utama 51
Gambar 3.2 Karakter NPC *Enemy* 51
Gambar 3.3 Karakter NPC *Army* 52
Gambar 3.4 Koin Perak 52
Gambar 3.5 Koin Emas 53
Gambar 3.6 Berlian 53
Gambar 3.7 FSM NPC *Enemy* 54
Gambar 3.8 FSM NPC *Army* 54
Gambar 3.9 *Flowchart* Algoritma *Dynamyc Weighting A** 55
Gambar 3.10 Arena Untuk Simulasi Algoritma A* dan DWA* 56
Gambar 3.11 Indeks Pada Kotak (*node*) Saat Simulasi Algoritma A* dan DWA* 57
Gambar 3.12 Simulasi Penghitunagn Manual Algoritma DWA* Langkah 1 59
Gambar 3.13 Simulasi Penghitunagn Manual Algoritma DWA* Langkah 2 60
Gambar 3.14 Simulasi Penghitunagn Manual Algoritma DWA* Langkah 3 61
Gambar 3.15 Simulasi Penghitunagn Manual Algoritma DWA* Langkah 4 62
Gambar 3.16 Simulasi Penghitunagn Manual Algoritma DWA* Langkah 5 63
Gambar 3.17 Simulasi Penghitunagn Manual Algoritma DWA* Langkah 6 64
Gambar 3.18 Simulasi Penghitunagn Manual Algoritma A* Langkah 1 65
Gambar 3.19 Simulasi Penghitunagn Manual Algoritma A* Langkah 2 66
Gambar 3.20 Simulasi Penghitunagn Manual Algoritma A* Langkah 3 67
Gambar 3.21 Simulasi Penghitunagn Manual Algoritma A* Langkah 4 68
Gambar 3.22 Simulasi Penghitunagn Manual Algoritma A* Langkah 5 69
Gambar 3.23 Simulasi Penghitunagn Manual Algoritma A* Langkah 6 70
Gambar 3.24 Simulasi Penghitunagn Manual Algoritma A* Langkah 7 71
Gambar 3.25 Simulasi Penghitunagn Manual Algoritma A* Langkah 8 72
Gambar 3.26 Simulasi Penghitunagn Manual Algoritma A* Langkah 9 73
Gambar 3.27 Flowchart Algoritma *Fisher-Yates Shuffle* 75
Gambar 4.1 Hasil *Build* Aplikasi *Game Finding Diamond* 86

Gambar 4.2 Pengaturan *Graphics* Pada Aplikasi *Game Finding Diamond* 86
Gambar 4.3 Pengaturan *Input* Pada Aplikasi *Game Finding Diamond*..... 86
Gambar 4.4 *Unity Splash Screen*..... 87
Gambar 4.5 Menu Utama Aplikasi *Game Finding Diamond*..... 87
Gambar 4.6 Mulai Bermain *Game Finding Diamond*..... 88
Gambar 4.7 *Player* Mendapatkan Koin Perak..... 88
Gambar 4.8 *Player* Mendapatkan Koin Emas..... 89
Gambar 4.9 Skor Telah Mencapai Target 89
Gambar 4.10 *Player* Berhasil Menyelesaikan Misinya..... 90
Gambar 4.11 NPC *Army* Mengambil Koin Perak 90
Gambar 4.12 NPC *Enemy* Mengejar *Player*..... 91
Gambar 4.13 Kesehatan *Player* Habis..... 92
Gambar 4.14 Waktu *Player* Habis..... 92
Gambar 4.15 *Game Over Screen*..... 93
Gambar 4.16 Hasil Implementasi Algoritma DWA* Pada *Console Unity* 96
Gambar 4.17 Rute Terpendek yang Berhasil Ditemukan Pada *Console Unity* 97
Gambar 4.18 Rute Terpendek yang Berhasil Ditemukan Pada *Scene Unity*..... 97
Gambar 4.19 Jalur yang Dilalui Semua NPC *Army* Pada *Scene Unity* 98
Gambar 4.20 Hasil Implementasi Algoritma FYS Pada *Console Unity*..... 106
Gambar 4.21 Hasil Pengacakan Fitur *Random Unity* Pada *Console Unity* 106

PUSAT PERPUSTAKAAN

DAFTAR TABEL

Tabel 3.1 *Story Bord Game* 47
Tabel 3.2 Simulasi Penghitungan Manual Algoritma *Fisher-Yates Shuffle* 76
Tabel 4.1 Kebutuhan Perangkat Keras 77
Tabel 4.2 Kebutuhan Perangkat Lunak 78
Tabel 4.3 Keterangan Implementasi Algoritma *Dynamic Weighting A** 78
Tabel 4.4 Keterangan Implementasi Algoritma *Fisher-Yates Shuffle* 84
Tabel 4.5 Uji Coba Pembobotan Algoritma DWA* 98
Tabel 4.6 Perbandingan Jumlah Node yang Dibangkitkan Oleh DWA* dan A* 100
Tabel 4.7 Perbandingan Waktu Pencarian Rute Terpendek Pada DWA* dan A* 101
Tabel 4.8 Total Waktu yang Dibutuhkan NPC Untuk Menyelesaikan Misinya 102
Tabel 4.9 Uji Coba Urutan Baru yang Dihasilkan Algoritma *Fisher-Yates Shuffle* 104
Tabel 4.10 Perbandingan Pengacakan *Fisher-Yates Shuffle* dan Fitur *Random Unity* ... 107
Tabel 4.11 Uji Coba Aplikasi *Game Finding Diamond* 109
Tabel 4.12 Presentase Hasil Uji Coba Aplikasi *Game Finding Diamond* 110

UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG
PUSAT PERPUSTAKAAN

ABSTRAK

Zakifardan, Ibnu. 2016. **Implementasi Algoritma *Dynamic Weighting A**** **Untuk Pencarian Rute Terpendek Pada NPC dan *Fisher-Yates Shuffle* Untuk Pengaturan Konten Pada *Game 3D Finding Diamond***. Skripsi. Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Pembimbing: (I) Dr. Muhammad Faisal, M.T, (II) Irwan Budi Santoso, M.Kom

Kata Kunci: *Nabi dan Rasul, Fisher-Yates Shuffle, A*, Dynamic Weighting A**

Iman kepada nabi dan rasul Allah, merupakan rukun iman yang keempat, jadi keimanan ini harus dimiliki oleh setiap umat islam. Salah satu jalan untuk bisa mengimani nabi dan rasul adalah dengan mengenali nama-nama mereka dan mempelajari kisah-kisah kehidupan mereka. Selain melalui pendidikan, ada media yang lebih modern untuk mengenalkan nabi dan rasul yaitu dengan *game*. Maka dari itu peneliti tergerak untuk menciptakan sebuah aplikasi *adventure game* tiga dimensi bernama *Finding Diamond*, yang di dalamnya terdapat konten islami tentang nabi dan rasul.

Pada aplikasi *game* yang dibuat ada koin perak, koin emas dan berlian yang harus dikumpulkan *player*. Khusus untuk setiap koin emas yang diambil, akan ditampilkan satu pengetahuan tentang nabi dan rasul yang telah diacak urutannya menggunakan algoritma *Fisher-Yates Shuffle*. Sehingga pengetahuan yang ditampilkan kelihatan lebih bervariasi. Algoritma ini bisa memastikan setiap koin emas menampilkan materi pengetahuan yang berbeda. Tetapi jika menggunakan fitur *random* di *Unity Game Engine* ada kemungkinan satu materi pengetahuan yang sama di tampilkan beberapa kali dalam satu permainan.

Ada dua karakter NPC di dalam *game* yaitu NPC *Enemy* dan NPC *Army*. NPC *Enemy* perilakunya adalah selalu mengejar *player*. Sedangkan NPC *Army* mengambil semua koin perak di arena permainan dengan mengimplementasikan algoritma *Dynamic Weighting A*,* sehingga pergerakannya lebih efektif, karena NPC *Army* bergerak menuju target sesuai dengan rute terpendek yang dihasilkan oleh algoritma tersebut. Berdasarkan hasil uji coba pada penelitian ini algoritma *Dynamic Weighting A** membangkitkan lebih sedikit *node* daripada algoritma A* sehingga memori yang digunakan lebih kecil dan waktu untuk menemukan rute terpendek lebih cepat.

ABSTRACT

Zakifardan, Ibnu. 2016. **Implementation of Dynamic Weighting A* Algorithm to Finding the Shortest Path on NPC and Fisher-Yates Shuffle to Manage Content on 3D Game Finding Diamond**. Thesis. Informatics Engineering Department of Science and Technology Faculty Islamic State University Maulana Malik Ibrahim Malang.

Adviser: (I) Dr. Muhammad Faisal, M.T, (II) Irwan Budi Santoso, M.Kom

Keywords: *Prophets and Messengers, Fisher-Yates Shuffle, A*, Dynamic Weighting A**

Faith in the prophets and messengers of Allah, is the fourth pillar of faith, so this faith must be owned by all Muslims. One way to be able to believe in the prophets and messengers is recognize their names and learn the stories of their lives. In addition through education, there are more modern media to introduce the prophets and messengers are with the game. Thus the researchers moved to create an application of 3D adventure game called Finding Diamond, in which there is Islamic content of the prophets and messengers.

In gaming applications was created there are silver coins, gold coins and diamonds to be collected player. For each gold coin is taken, will be shown the knowledge of the prophets and messengers who had been randomized sequence using the Fisher-Yates Shuffle algorithm. So that the knowledge displayed more varied look. This algorithm can ensure every gold coin featuring different knowledge. But if using the random feature in Unity Game Engine there is the possibility same knowledge in the show several times in one game.

There are two NPC characters in this game, there is NPC Enemy and NPC Army. NPC Enemy behavior is always pursuing player. While NPC Army took all the silver coins in the game arena by implementing Dynamic Aeighting A* algorithm, so that the movement is more effective, because the NPC Army move towards the target in accordance with the route that is generated by the algorithm. Based on trial results on this study Dynamic Weighting A* algorithm generate fewer nodes than A* algorithm so that the memory used is smaller and the time to find the shortest route more quickly.

الملخص

زاكي فاردان، ابنو. ٢٠١٦. **تنفيذ الحيوي الترجيح خوارزمية *Aللبحث في أقصر هذه الوطنى وفيشر بيتس المراوغة لإعدادات المحتوى في لعبة 3Dالعثور الماس.** أطروحة. قسم المعلوماتية كلية العلوم والتكنولوجيا في جامعة الدولة الإسلامية مولانا مالك إبراهيم مالانج.

مؤدب: (١) الدكتور محمد فيصل، M.T، (٢) إيروان بودي سانتوسو، M.Kom

الكلمات المفتاحية: *الأنبياء والمرسلين، فيشر بيتس المراوغة، *A، الترجيح دينامية *A*

الإيمان الأنبياء والمرسلين من الله، هو الركن الرابع من أركان الإيمان، لذلك يجب أن تكون مملوكة نية من قبل جميع المسلمين .طريقة واحدة لتكون قادرة على الاعتقاد في الأنبياء والمرسلين هو الاعتراف أسمائهم ومعرفة قصص حياتهم .وبالإضافة إلى ذلك من خلال التعليم، وهناك وسائل الاعلام أكثر حداثة لتقديم الأنبياء والرسل هم مع اللعبة .وهكذا انتقل الباحثون إلى إنشاء تطبيق لعبة مغامرة ثلاثية الأبعاد تسمى العثور الماس، والتي لا يوجد محتوى الإسلامي من الأنبياء والرسل.

في الألعاب خلق تطبيقات أي لاعب من الفضة، الذهب والماس التي يتعين جمعها. لكل عملة ذهبية تم اتخاذها، وسوف يعرض معرفة الأنبياء والرسل الذين تم اختيارهم بصورة عشوائية تسلسل باستخدام خوارزمية فيشر بيتس المراوغة. ذلك أن عرض المعرفة نظرة أكثر تنوعا. يمكن هذه الخوارزمية ضمان كل قطعة ذهبية يضم المواد المعرفة المختلفة. ولكن في حالة استخدام ميزة عشوائية في الوحدة لعبة المحرك لا يوجد إمكانية للمادة نفسها المعرفة في المعرض عدة مرات في مباراة واحدة.

هناك نوعان من الشخصيات في اللعبة التي مجلس الشعب مجلس الشعب ومجلس الشعب جيش العدو . السلوك مجلس الشعب العدو دائما تسعى لاعب .بينما تولى الجيش الوطنى جميع القطع النقدية الفضية في الساحة لعبة من خلال تنفيذ الحيوي الترجيح خوارزمية *A، بحيث تكون الحركة أكثر فعالية، لأن الجيش الوطنى المضي قدما نحو الهدف وفقا للمسار الذي تم إنشاؤه بواسطة خوارزمية .بناء على نتائج المحاكمة في هذه الدراسة الترجيح خوارزمية الحيوي *Aتوليد العقد أقل من *Aخوارزمية ذلك أن الذاكرة المستخدمة هي أصغر والوقت للعثور على أقصر الطرق بسرعة أكبر.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Iman kepada nabi-rasul Allah merupakan salah satu rukun iman, yaitu rukun iman keempat. Keimanan seseorang itu tidak sah, sampai ia mengimani semua nabi dan rasul Allah dan membenarkan bahwa Allah telah mengutus mereka untuk menunjuki, membimbing dan mengeluarkan manusia dari kegelapan kepada cahaya kebenaran. Ditambah juga keharusan membenarkan bahwa mereka telah menyampaikan apa yang Allah turunkan kepada mereka dengan benar dan sempurna, dan mereka telah berjihad dengan sebenar-benarnya di jalan Allah.

Firman Allah yang menegaskan tentang kewajiban beriman kepada nabi dan rasul Allah ada beberapa ayat di dalam Al-Qur'an, salah satunya yaitu terdapat pada surat Al-Baqarah ayat 177 yang artinya, *"Bukanlah menghadapkan wajahmu ke arah timur dan barat itu suatu kebajikan, akan tetapi sesungguhnya kebajikan itu ialah beriman kepada Allah, hari kemudian, malaikat-malaikat, kitab-kitab, nabi-nabi dan memberikan harta yang dicintainya kepada kerabatnya, anak-anak yatim, orang-orang miskin, musafir (yang memerlukan pertolongan) dan orang-orang yang meminta-minta, dan (memerdekakan) hamba sahaya, mendirikan shalat, dan menunaikan zakat, dan orang-orang yang menepati janjinya apabila ia berjanji, dan orang-orang yang sabar dalam kesempitan, penderitaan dan dalam peperangan. Mereka itulah orang-orang yang benar (imannya), dan mereka itulah orang-orang yang bertakwa." (QS. Al-Baqarah: 177).*

Pada ayat tersebut di atas, Allah memerintahkan kaum mukminin untuk beriman kepada Allah, Rasul-Nya, Al-Qur'an dan kitab suci yang diturunkan sebelumnya. Karena pentingnya iman kepada nabi dan rasul Allah maka keimanan ini harus dimiliki oleh setiap umat islam, salah satu jalan untuk bisa mengimani adanya nabi dan rasul Allah adalah dengan mengenali nama-nama mereka dan mempelajari kisah-kisah kehidupan mereka. Hal tersebut bisa didapat salah satunya melalui pendidikan, baik yang pendidikan formal maupun nonformal. Selain melalui pendidikan tersebut, ada media yang lebih modern untuk mengenalkan nama-nama nabi dan rasul yaitu dengan *game*. Karena pada dasarnya bermain *game* itu menyenangkan dan hampir setiap orang pernah bermain *game*, maka dari itu peneliti tergerak untuk menciptakan sebuah aplikasi *game* modern yang bertujuan untuk memperkenalkan nabi dan rasul dengan media permainan komputer. Permainan ini bergenre *adventure game* yang bertujuan untuk semakin mendekatkan anak-anak dan para generasi muda terkait pengetahuan tentang nabi dan rasul dengan cara yang menyenangkan.

Pemanfaatan teknologi yang telah berkembang saat ini mengisyaratkan kesimpulan bahwa teknologi dapat dimanfaatkan sebagai media pembelajaran sekaligus pengenalan nilai-nilai keislaman khususnya pengetahuan tentang nabi dan rasul. Dalam penelitian ini, peneliti berharap agar seiring berkembangnya jaman dan teknologi, anak-anak dan generasi muda semakin peduli dan tetap menjaga dan mengamalkan ajaran islam. Sesungguhnya modernitas dapat dimanfaatkan sebagai media dalam memperkenalkan ajaran islam kepada masyarakat, bukan sebagai ajang maksiat dan terbawa arus negatif modernisasi.

1.2 Identifikasi Masalah

Berdasarkan latar belakang diatas identifikasi masalah yang dapat dirumuskan adalah sebagai berikut:

a. Bagaimana membuat *adventure game* tiga dimensi sebagai media pengenalan nabi dan rasul yang dimainkan di *personal computer*?
b. Bagaimana mengimplementasikan algoritma DWA* (*Dynamic Weighting A**) untuk pencarian rute terpendek pada NPC (*Non-Player Character*) yang ada di aplikasi *game* tersebut?
c. Apakah perbedaan antara pencarian rute terpendek dengan menggunakan algoritma DWA* yang merupakan salah satu modifikasi dari algoritma A* dan algoritma A* itu sendiri?
d. Bagaimana mengimplementasikan algoritma *Fisher-Yates Shuffle* untuk mengatur kemunculan konten islami tentang nabi dan rasul yang ditampilkan pada aplikasi *game* tersebut?
e. Apakah perbedaan antara pengacakan dengan menggunakan algoritma *Fisher-Yates Shuffle* dan pengacakan dengan menggunakan fitur *random* yang ada di *Unity Game Engine*?

1.3 Batasan Masalah

Batasan masalah pada penelitian ini adalah sebagai berikut:

a. Konten islami di dalam *game* adalah pengetahuan tentang nabi dan rasul
b. Berbasis *desktop* dan diimplementasikan pada *platform Windows*
c. Dimainkan oleh *single player*

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah sebagai berikut:

a. Membuat aplikasi *game* tiga dimensi yang berwawasan islam untuk pengenalan nabi dan rasul yang bisa dimainkan di *personal computer*
b. Mengimplementasikan algoritma DWA* (*Dynamic Weighting A**) untuk pencarian rute terpendek pada NPC (*Non-Player Character*) yang ada di aplikasi *game* tersebut
c. Melakukan uji coba untuk mengetahui perbedaan antara pencarian rute terpendek dengan menggunakan algoritma DWA* dan algortitma A*
d. Mengimplementasikan algoritma *Fisher-Yates Shuffle* untuk mengatur kemunculan konten islami tentang nabi dan rasul yang ditampilkan pada aplikasi *game* tersebut
e. Melakukan uji coba untuk mengetahui perbedaan antara pengacakan dengan menggunakan algoritma *Fisher-Yates Shuffle* dan pengacakan dengan menggunakan fitur *random* yang ada di *Unity Game Engine*

1.5 Manfaat Penelitian

Manfaat pembuatan aplikasi *game* ini adalah dengan disertakannya konten islami di dalam *game* maka bermain *game* akan lebih bermanfaat dan bisa menambah pengetahuan orang yang bermain *game* terutama yang dibahas dalam penelitian ini yaitu pengetahuan tentang nabi dan rasul. Jadi dengan bermain *game* yang dibuat pada penelitian ini secara tidak langsung orang tersebut telah mempelajari pengetahuan tentang nabi dan rasul. Suasana belajar jadi lebih mudah dan menyenangkan dengan menggunakan *game*.

1.6 Tahapan Penelitian

Berikut adalah tahapan-tahapan yang dilakukan dalam penelitian:

1. Studi literatur

   Studi literatur dilakukan proses pengumpulan dan pengkajian data-data yang diperlukan dalam pembuatan *game*, diantaranya informasi tentang kisah-kisah nabi dan rasul, pembuatan aplikasi *game* dengan *Unity Game Engine*, algoritma A*, *Dynamic Weigting A** dan *Fisher-Yates Shuffle.*

2. Perancangan *game*

   Proses ini merupakan proses perancangan, mulai dari perancangan *story board*, karakter, perancangan implementasi algoritma *Dynamic Weigting A** dan *Fisher-Yates Shuffle* pada *game* hingga *game environment*.

3. Pembuatan *game*

   Proses pembuatan *game* dibangun dengan memanfaatkan *Game Engine Unity 5.2.0f3 64-bit* dengan menggunakan bahasa C#. *Scripting* dilakukan dengan fasilitas *MonoDevelop*.

4. Uji coba dan evaluasi

   Uji coba dalam penelitian ini dilakukan pada *game* setelah diimplementasikan algoritma *Fisher-Yates Shuffle* dan karakter NPC *Army* yang sudah diimplementasikan algoritma *Dynamic Weighting A**.

5. Penyusunan laporan

   Penyusunan laporan akhir merupakan bagian dokumentasi dari keseluruhan pelaksanaan penelitian yang diharapkan dapat bermanfaat bagi penelitian selanjutnya.

BAB II

KAJIAN PUSTAKA

2.1 Landasan Teori

2.1.1 Teori *Game*

Game merupakan kata dari bahasa inggris yang artinya adalah permainan. Permainan adalah suatu yang dapat dimainkan dengan aturan tertentu untuk menyenangkan hati (dengan menggunakan alat-alat tertentu atau tidak) sehingga ada yang menang ada yang kalah (Kamus Besar Bahasa Indonesia).

Game atau permainan adalah sesuatu yang dapat dimainkan dengan aturan tertentu sehingga ada yang menang dan ada yang kalah, biasanya dalam konteks tidak serius dengan tujuan refresing. Bermain *game* sudah dapat dikatakan sebagai salah satu gaya hidup masyarakat dimasa kini. Dimulai dari usia anak-anak hingga orang dewasa pun menyukai *video game*. Itu semua dikarenakan bermain *game* adalah hal yang menyenangkan. (Anggara, 2008).

Permainan terdiri atas sekumpulan peraturan yang membangun situasi bersaing dari dua atau lebih orang atau kelompok dengan memilih strategi yang dibangun untuk memaksimalkan kemenangan sendiri ataupun untuk meminimalkan kemenangan lawan. Peraturan-peraturan yang ada menentukan kemungkinan tindakan untuk setiap pemain, sejumlah keterangan diterima setiap pemain sebagai kemajuan bermain, dan sejumlah kemenangan atau kekalahan dalam berbagai situasi. (John von Neumann dan Oskar Morgenstern, 1944).

Permainan merupakan media hiburan yang sangat diminati hampir semua usia. Kualitas *game* ditentukan dari berbagai aspek, baik dari kecerdasan buatan, konten yang disajikan hingga *gameplay* dan lain sebagainya. Kecerdasan buatan diperlukan guna menciptakan efek realistis pada sistem permainan. Kecerdasan buatan umumnya diterapkan pada NPC. Permainan yang menarik merupakan salah satu terobosan yang mampu menjadikan sebuah *game* bernilai edukasi. Muatan edukasi bisa dijadikan sebagai media pembelajaran yang menyenangkan.

2.1.2 *Genre Game*

Dalam sebuah permainan, seringkali *game* diklasifikasikan ke dalam sebuah *genre* tertentu. Pengklasifikasian didasarkan pada gaya ataupun kumpulan karakteristik seperti *gameplay*, interaksi, tujuan dan lain lain. Beberapa contoh *genre game* meliputi *adventure games*, *action games*, *FPS (First Person Shooter), RPG (Role Playing Game), MMRPGs (Massively Multiplayer Online Role Playing Games), Puzzle Games, Racing Games, Educational Games* dan lain sebagainya. Dibawah ini akan dijelaskan rincian penjelasan dari beberapa *genre* yang ada di dalam *game*.

2.1.2.1 *Action Shooting*

Menembak, memukul, berkelahi adalah contoh cerita dari permainan *action shooting*, tergantung cerita dan tokoh dalam permainan tersebut. *Game* jenis ini sangat memerlukan kecepatan refleks, koordinasi mata-tangan, juga *timing*. Inti dari *game* jenis ini adalah menembak dan melempuhkan lawan. Contoh permainan: *Counter Strike* dan *Crysis*.

2.1.2.2 *Adventure Game*

Adventure menggambarkan nuansa petualangan. Berkeliling hutan, melompati bebatuan di antara lahar, bergelayutan dari pohon satu ke pohon lain, melawan naga sambil mencari kunci untuk membuka kuil tersembunyi, atau sekedar mencari petunjuk untuk mendapatkan misi berikutnya. Hal tersebut adalah beberapa hal yang akan dilakukan pemain dalam menikmati *game* bergenre *adventure*. Pengertian dari *Adventure Game* adalah permainan video yang mengasumsikan pemain merupakan karakter protagonis dalam sebuah cerita interaktif yang memiliki tujuan untuk mengeksplorasi cerita dan memecahkan puzzle (Andrew Rollings dan Ernest Adams, 2006).

2.1.2.3 Education

Game edukasi merupakan permainan yang memotivasi atau membantu siswa untuk melalui alur *game* secara teliti untuk mengembangkan kemampuannya. *Developer* yang membuatnya, harus memperhitungkan berbagai hal agar *game* ini benar-benar dapat mendidik, menambah pengetahuan dan meningkatkan keterampilan. Target segmentasi pemain harus pula disesuaikan dengan tingkat kesulitan, desain visual ataupun animasinya.

2.1.2.4 *Fighting*

Pertarungan adalah menu utama dalam permainan jenis ini. Karena nuansa yang tidak berbeda jauh dengan *action game*, ada juga yang mengelompokan *game fighting* bergenre *action*. Pendapat yang berbeda menyatakan bahwa jenis permainan ini memang memerlukan kecepatan refleks serta koordinasi mata dan

tangan, tetapi inti dari *game* ini adalah penguasaan jurus (hafal caranya dan lancar mengeksekusinya), pengenalan karakter dan *timing* eksekusi menjadi poin penting. Contoh: Mortal Kombat dan Tekken.

2.1.2.5 *Sport game*

Game ini mengadaptasi permainan olahraga dari kenyataan. Membutuhkan kelincahan dan juga strategi dalam memainkannya. *Game sport* umumnya berupa kompetisi antara dua pemain atau lebih, di mana pemain dapat berupa individual atau tim. Contoh *game* tipe ini antara lain: sepakbola, bola basket, tenis.

2.1.2.6 *Strategy*

Game strategi biasanya memberikan pemain akses kendali tidak hanya pada satu orang tapi minimal sekelompok orang dengan berbagai jenis tipe kemampuan, kendaraan, bahkan hingga pembangunan berbagai bangunan, pelatihan tempur, kerajaan atau apapun tergantung dari tema ceritanya. Kebanyakan *game* stategi adalah *game* bernuansa perang. Contoh *game* tipe ini antara lain: *Warcraft*.

2.1.2.7 *Puzzle*

Game jenis ini memiliki inti dan tujuan terkait pemecahan teka-teki. Baik menyusun balok, menyamakan warna bola, memecahkan perhitungan matematika atau melewati labirin. Semua hak tersebut termasuk dalam jenis *puzzle game*. Seringkali pula *game* jenis ini juga memiliki unsur *game* petualangan maupun *game* edukasi.

2.1.2.8 RPG (*Role Playing Game*)

G*ame* jenis ini sesuai dengan terjemahannya yakni bermain peran. Memiliki penekanan pada tokoh/peran yang dimainkan pemain di dalam *game*. Pemain biasanya berperan sebagai tokoh utama dimana seiring permainan berlangsung semakin lama terdapat plot cerita dan karakter pun dapat berubah dan mengalami perkembangan kemampuan. Perkembangan terjadi dalam berbagai parameter yang biasanya ditentukan dengan naiknya level, status kepintaran, kecepatan dan kekuatan karakter, senjata dan mahluk peliharaan yang semakin hebat dan lain-lain.

2.1.3 *Non-Player Character*

NPC atau *Non-Player Character* adalah jenis *otonomous agent* yang ditujukan untuk penggunaan komputer animasi dan media interaktif seperti *games* dan *virtual reality*. Agen ini mewakili tokoh dalam cerita atau permainan dan memiliki kemampuan untuk improvisasi tindakan mereka. Ini adalah kebalikan dari seorang tokoh dalam sebuah film animasi, yang tindakannya ditulis di muka, dan untuk "*avatar*" dalam sebuah permainan atau *virtual reality*, tindakan yang diarahkan secara *real time* oleh pemain. Dalam permainan, karakter *otonom* biasanya disebut *Non-Player Character* (NPC). (Yunifa Miftachul Arif , 2010).

Secara garis besar maka NPC dapat diartikan sebagai karakter pada *game* yang sepenuhnya dikendalikan komputer dan tidak dapat dimainkan oleh pemain. Pengendalian NPC umumnya menggunakan bidang ilmu kecerdasan buatan.

Kecerdasan buatan dimasukkan dengan tujuan agar NPC dapat melakukan pekerjaan seperti yang dapat dilakukan manusia. Beberapa macam bidang yang

menggunakan kecerdasan buatan antara lain sistem pakar, permainan komputer, logika fuzzy, jaringan syaraf tiruan dan robotika.

Kecerdasan buatan menjadikan NPC memiliki gerak variatif, mampu memainkan perannya sebagai penyempurna permainan dan menjadikan *gameplay* lebih asik dan menantang.

2.1.4 *Finite State Machine*

Dalam perancangan *Artificial Intelegent* untuk *game*, *state machine* merupakan teknik yang paling banyak digunakan untuk permasalahan *"decision making"* dan sekaligus dengan *scriptingnya* juga digunakan secara luas untuk merancang *system decision making* dalam *game*. *State machine* dikenal secara luas sebagai teknik untuk pemodelan fenomena atau kondisi berbasis *event*, termasuk penguraiannya, serta desain *interface*. *Finite State Machine* (FSM) atau juga disebut sebagai teknik yang secara luas dipergunakan dalam merancang AI dalam *game.* Teknik ini merupakan metodologi perancangan system untuk memodelkan perilaku (*behavior*) dari sistem atau objek yang kompleks dengan kondisi yang telah didefinisikan dalam satu set.

Finite State Machines (FSM) masuk dalam ranah *Decision Making* (pembuat keputusan) pada *Artificial Intelligence*. Dalam FSM masing-masing karakter menempati satu *state*. Biasanya, tindakan atau perilaku yang terkait dengan masing-masing *state*. Jadi selama karakter tetap dalam keadaan itu, ia akan terus melakukan tindakan yang sama. *State* terhubung bersama oleh *transition*. Setiap *transition* mengarah dari satu *state* ke *state* lain yang biasanya *state* tujuan *state* target ini disebut dengan *action* dan masing-masing memiliki

seperangkat kondisi yang terkait. Jika permainan menentukan bahwa kondisi transition terpenuhi, maka karakter berubah dari *state* ke *state* target (*action*) melalui *transition* itu. (Ian Millington, 2006).

FSM melacak himpunan *state* yang ada kemudian inputan masuk ke masing-masing *state*, serangkaian keadaan *transition* tetap. Setiap *transition* dapat diimplementasikan dengan kondisi yang sesuai. Pada setiap iterasi (biasanya setiap *frame*), fungsi update FSM digunakan. Ini memeriksa untuk melihat apakah ada perubahan *transition* dari kondisi saat dipicu oleh inputan. Kemudian menyusun daftar *action* dari kondisi yang sedang aktif. Jika *transition* telah menemukan *action* yang dituju, maka *transition* berhenti.

2.1.5 Nabi dan Rasul

Nabi dan rasul adalah hamba-hamba Allah pilihan yang menerima wahyu dan risalah dari Allah SWT. Nabi adalah hamba Allah pilihan yang menerima wahyu untuk dirinya sendiri dan tidak mempunyai kewajiban untuk menyampaikannya kepada umat manusia. Rasul adalah manusia pilihan yang menerima wahyu dan risalah dari Allah dan bertanggung jawab menyampaikannya kepada umat manusia. Setiap rasul adalah nabi, sedangkan setiap nabi belum tentu rasul. (Hanafi, 2011).

Sebagian ulama menyatakan bahwa definisi ini memiliki kelemahan, karena tidaklah wahyu disampaikan Allah ke bumi kecuali untuk disampaikan, dan jika nabi tidak menyampaikan maka termasuk menyembunyikan wahyu Allah. Kelemahan lain dari definisi ini ditunjukkan dalam hadits dari nabi *shallallahu ‘alaihi wa sallam*, “*Ditampakkan kepadaku umat-umat, aku melihat seorang nabi*

dengan sekelompok orang banyak, dan nabi bersama satu dua orang dan nabi tidak bersama seorang pun." (HR. Bukhori dan Muslim).

Hadits ini menunjukkan bahwa nabi juga menyampaikan wahyu kepada umatnya. Ulama lain menyatakan bahwa ketika nabi tidak diperintahkan untuk menyampaikan wahyu bukan berarti nabi tidak boleh menyampaikan wahyu. *Wallahu'alam*. Perbedaan yang lebih jelas antara nabi dan rasul adalah seorang rasul mendapatkan syari'at baru sedangkan nabi diutus untuk mempertahankan syari'at yang sebelumnya.

Jumlah nabi dan rasul sangat banyak. Namun yang tersebut dalam Al-Qur'an dan wajib kita imani berjumlah 25 orang. Mereka adalah Adam, Idris, Nuh, Hud, Shaleh, Ibrahim, Luth, Ismail, Ishaq, Ya'qub, Yusuf, Syuaib, Ayub, Zulkifli, Musa, Harun, Daud, Sulaiman, Ilyas, Ilyasa', Yunus, Zakariya, Yahya, Isa dan Muhammad SAW.

Banyak nabi dan rasul lainnya yang tidak dikisahkan dalam Al-Qur'an. Allah berfirman yang artinya, *"Dan (kami telah mengutus) rasul-rasul yang sungguh telah kami kisahkan tentang mereka kepadamu dahulu, dan rasul-rasul yang tidak Kami kisahkan tentang mereka kepadamu." (QS. An-Nisa': 164).* Walaupun dalam Al-Qur'an hanya disebut 25 nabi, maka kita tetap mengimani secara global adanya nabi dan rasul yang tidak dikisahkan dalam Al-Qur'an. Allah berfirman yang artinya, "*Dan sesungguhnya telah Kami utus beberapa orang rasul sebelum kamu, di antara mereka ada yang Kami ceritakan kepadamu dan di antara mereka ada yang tidak Kami ceritakan kepadamu." (QS. Al-Mu'min: 78).*

Setiap umat di dunia ini memiliki rasul. Allah mengutus setiap rasul-Nya untuk tiap umat sepanjang masa secara terus menerus. Tidak ada satu umat pun yang tidak punya rasul. Hal ini supaya setiap umat di muka bumi ini tetap beriman dan berbakti kepada Allah serta menghindarkan kerusakan yang dilakukan oleh umat tertentu. Karena rasul diutus dengan tujuan mengingatkan mereka yang lalai dan memberi kabar baik bagi yang ingat. Allah berfirman yang artinya, *"Demi Allah, sesungguhnya Kami telah mengutus rasul-rasul Kami kepada umat-umat sebelum kamu." (QS. An-Nahl: 63).* Dan di ayat lain Allah berfirman yang artinya, *"Dan tidak ada suatu umatpun melainkan telah ada padanya seorang pemberi peringatan." (QS. Fathir: 24).*

Konten islami yang akan dimasukkan di dalam *game* adalah pengenalan nama-nama nabi dan rasul beserta mukjizatnya yang akan ditampilkan setiap kali *player* mendapatkan satu koin emas yang ada di arena permainan. Di bawah ini rincian nama-nama nabi dan rasul beserta mukjizatnya atau beberapa cuplikan kisahnya yang akan disertakan di dalam aplikasi *game.*

2.1.5.1 Nabi Adam

Nabi Adam diyakini sebagai manusia pertama yang menginjakkan kaki di bumi. Sebagai pasangan Nabi Adam adalah Hawa yang diciptakan dari tulang rusuk kiri Nabi Adam. Mereka diturunkan ke bumi karena telah berbuat kesalahan akibat godaan iblis/syetan, Adam dan Hawa dikaruniai dua pasangan putra-putri yang bernama Qabil dan Iklima, kemudian Habil dan Labuda. Qabil bersifat kasar, sedangkan Labuda bersifat lembut, Kedua sifat inilah yang akhirnya menjadi cikal bakal dalam sifat-sifat dasar manusia.

2.1.5.2 Nabi Idris

Nabi Idris diyakini nabi pertama yang menulis dengan pena. Masyarakat terdahulu mempercayai pula bahwa ia dibawa ke surga tanpa mengalami kematian. Peristiwa itu terjadi ketika beliau berusia 82 tahun. Nabi Idris merupakan keturunan dari Qabil dan Iqlima (putra dan putri Nabi Adam) kepada keturununannya inilah Nabi Idris ditugaskan Tuhan mengajak kepada kebenaran. Nabi Idris adalah orang pertama yang menerima wahyu lewat Malaikat Jibril, ketika berumur 82 tahun. Nabi Idris menurut riwayat dalam hadits Bukhari adalah kakeknya bapak Nuh a.s. berarti Nabi Idris merupakan generasi ke enam dari Adam, mengingat Nuh sendiri sebagai keturunan ke sepuluh dari Nabi Adam.

2.1.5.3 Nabi Nuh

Nabi Nuh menyebarkan ajaran untuk menyembah Allah SWT namun masyarakat menolak dan menganggapnya gila, Nabi Nuh kemudian diberikan peringatan oleh Allah bahwa akan terjadi banjir besar yang akan melanda daerahnya. Oleh karena itu Nabi Nuh diperintahkan untuk membuat sebuah kapal, masyarakat sekitar tetap tidak mengindahkan peringatan yang disampaikan oleh Nabi Nuh, sehingga mereka akhirnya hanyut dalam banjir tersebut.

2.1.5.4 Nabi Hud

Nabi Hud tergolong dalam kaum Ad yang terhormat, kehidupan mereka serba maju dan berkecukupan, namun sayangnya mereka selalu berfoya-foya dan tenggelam dalam kehidupan fana. Nabi Hud mengingatkan mereka untuk bersyukur dan selalu memohon kepada Allah SWT, namun mereka menolak.

Akhirnya murka Allah datang dengan menurunkan azab berupa badai gurun selama 7 hari 7 malam. Kaum yang mendengarkan himbauan Nabi Hud selamat dengan berpindah ke kota Hadramaut.

2.1.5.5 Nabi Shalih

Mukjizat beliau yang paling dikenal adalah unta betina yang keluar dari batu setelah ia memukulkan telapak tangannya. Nabi Shalih meminta kepada penduduk setempat untuk tidak mengganggu unta tersebut dan susunya boleh diperah untuk memenuhi kebutuhan penduduk miskin. Namun kaum yang tidak menyukainya berusaha membunuh unta itu dan pada akhirnya mereka dijatuhi azab petir dan gempa.

2.1.5.6 Nabi Ibrahim

Nabi Ibrahim dikenal sebagai bapak para nabi. Dia dihormati oleh pemeluk 3 agama, yaitu Islam, Kristen dan Yahudi. Nabi Ibrahim adalah orang yang membangun Ka'bah di kota Makkah. Keyakinannya yang kuat terhadap Islam dimulai dari pencariannya akan Tuhan, dia sangat tidak menerima orang-orang disekitarnya yang menyembah berhala, sampai akhirnya dia dibakar hidup-hidup, namun Allah SWT menurunkan mukjizatnya dengan menyelamatkan Nabi Ibrahim dari kobaran api.

2.1.5.7 Nabi Ismail

Nabi Ismail dan keluarganya merupakan orang-orang yang terdahulu melaksanakan Haji. Suatu saat Nabi Ismail haus dan ibunya bolak-balik dari bukit Safa-Marwah untuk mencari air, hingga akhirnya keluar sebuah mata air zam-

zam. Dalam perjalanan menuju tempat penyembelihan, Nabi Ismail digoda oleh syaitan agar membatalakan niatnya. Namun Nabi Ismail tidak goyah dan melempar syaitan tersebut dengan batu, yang saat ini menjadi ritual ibadah haji, yaitu lempar jumrah. Seperti yang kita ketahui, saat akan disembelih jasad Nabi Ismail digantikan oleh seekor kambing, yang akhirnya menjadi cikal bakal ibadah Idul Adha.

2.1.5.8 Nabi Luth

Perjuangan Nabi Luth adalah menyeru kaum sodom untuk kembali ke jalan yang benar, yaitu meninggalkan homoseksual, kemudian menyembah Allah. Pada akhirnya Allah SWT berfirman agar Nabi Luth segera meninggalkan pemukimannya dan kemudian Allah SWT menurunkan azab yang pedih kepada kaum tersebut.

2.1.5.9 Nabi Ishaq

Nabi Ishaq banyak menemani bapaknya yaitu Nabi Ibrahim dalam berdakwah menyebarkan ajaran Islam. Ishaq adalah putra kedua Nabi Ibrahim setelah Ismail yang beribu Sarah dan merupakan orang tua dari Nabi Yaqub. Ishaq diutus untuk masyarakat Kana'an di wilayah Al-Khalil Palestina. Kisah Nabi Ishaq sangat sedikit diceritakan dalam Al-Qur'an. Nabi Ishaq disebutkan dalam Al-Qur'an sebanyak 15 kali. Sedangkan keutamaan Nabi Ishaq disebutkan 9 kali dan kenabian Ishaq 10 kali. Dikatakan bahwa ia memiliki 2 anak dan meninggal di Al-Khalil (Hebron) Palestina.

2.1.5.10 Nabi Ya'qub

Nabi Ya'qub adalah kakek moyang para rasul sebelum masa Nabi Muhammad. Sikap dan cara berpikirnya tentu berpengaruh kepada para rasul keturunannya, serta kaum Yahudi dan kemudian Nasrani penegak panji keesaan Allah sebelum era Nabi Muhammad SAW.

2.1.5.11 Nabi Yusuf

Nabi Yusuf dikisahkan dalam riwayatnya sebagai seorang pria yang sangat tampan dan sangat piawai dalam memimpin negaranya. Sejak kecil dia mendapat mimpi yang tidak biasa dan ketika besar dia dapat mentakwilkan mimpinya tersebut, sehingga dia sangat dihormati oleh masyarakat sekitarnya.

2.1.5.12 Nabi Syuaib

Nabi Syuaib menyebarkan ajaran Islam di daerah Madyan, namun masyarakat Madyan menolak ajaran tersebut hingga akhirnya Allah menurunkan azab berupa petir dan kilat yang menghanguskan mereka.

2.1.5.13 Nabi Ayub

Nabi Ayub dikenal seorang yang kaya raya dan sangat dermawan. Namun kesejahteraan ini tidak membuatnya sombong, ini yang mendorong iblis untuk menggodanya. Allah pun menentang iblis sekiranya dia dapat meruntuhkan iman Nabi Ayub. Ujian itu pun tiba, seluruh harta kekayaan yang dimiliki Nabi Ayub habis terbakar, setelah itu Nabi Ayub terserang penyakit kulit hingga 80 tahun lamanya. Namun dia dan istrinya yang setia, Rahmah, tetap bertawakal kapada Allah SWT. Sampai akhirnya Allah berfirman agar Nabi Ayub menapakkan

kakinya ditanah. kemudian dari tanah tersebut keluar air yang dapat menyembuhkan penyakit yang dideritanya selama 80 tahun.

2.1.5.14 Nabi Dzulkifli

Sejarah menyebutkan bahwa Nabi Dzulkifli adalah putra Nabi Ayub. Dikisahkan pula bahwa dia mewarisi sifat sabar ayahnya. Suatu saat beliau ditunjuk menjadi seorang raja setelah dapat memenuhi persyaratan yang diminta. Yaitu calon pengganti haruslah seorang yang sanggup berpuasa di siang hari, beribadah di malam hari, dan bukan seorang yang pemarah.

2.1.5.15 Nabi Musa

Kisah pertarungan Nabi Musa dengan Fir'aun merupakan salah satu kisah yang tersohor. Dikisahkan bahwa Fir'aun merasa terancam dengan keberadaan Nabi Musa yang menyebarkan ajaran untuk mengesakan Allah. Mereka bertarung dan Nabi Musa memenangkannya dengan bantuan tongkatnya, kemudian ia dan kaumnya dikejar oleh pengikut Fir'aun. Namun mereka berhasil lolos dengan bantuan tongkat Nabi Musa yang dapat membelah lautan. Nabi Musa mendapat mukjizat kitab Taurat, yang dikenal dengan perjanjian lama yang berisi ajaran pokok 10 perintah Allah SWT.

2.1.5.16 Nabi Harun

Nabi Harun disebut sebagai partner Nabi Musa. Dia adalah sosok yang cakap berdakwah, pandai berdiplomasi, dan penuh perhatian. Nabi Harun selalu mendampingi Nabi Musa dalam berdakwah, hingga suatu saat Nabi Musa memutuskan untuk beruzlah dan menitipkan pembinaan umatnya kepada Nabi

Harun. Nabi Harun juga sempat berjuang untuk memberantas penyembahan berhala yang dipimpin oleh Samiri, salah seorang tukang sihir kerajaan Fir'aun.

2.1.5.17 Nabi Daud

Figur Nabi Daud memuncak saat dia berhasil membunuh jalut, pemimpin kaum pemberontak Palestina. Nabi Daud kemudian menjadi seorang raja dan berlaku sangat adil. Di masa kerajaan Nabi Daud tumbuh kuat dan masyarakat menjadi makmur. Suatu saat Nabi Daud melarang para nelayan untuk tidak melaut di hari sabtu, namun peringatan tersebut dilanggar, sehingga terjadi bencana gempa yang menewaskan seluruh penduduk.

2.1.5.18 Nabi Sulaiman

Nabi Sulaiman yang paling menonjol adalah kemampuannya berkomunikasi dengan binatang seperti burung, semut, dan sebagainya. Dia juga merupakan raja yang sangat bijaksana dan kaya raya dengan istana yang megah berkilauan dan bertaburan permata. Dia juga berkuasa memerintah bangsa jin. Nabi Sulaiman dapat bepergian ke mana saja dengan mengendarai angin.

2.1.5.19 Nabi Ilyas

Nabi Ilyas tinggal di lembah sungai Yordan dimana penduduknya menyembah berhala, Nabi Ilyas menyuruh kepada mereka semua untuk meninggalkan berhala, namun mereka tidak mengindahkannya. Bahkan menantang agar Tuhan yang disembah Nabi Ilyas menurunkan bencana, dan akhirnya kekeringan melanda daerah tersebut. Setelah beberapa tahun, Nabi Ilyas dapat meyakinkan kaum tersebut untuk menyembah Allah SWT.

2.1.5.20 Nabi Ilyasa

Nabi Ilyasa merupakan kerabat dekat Nabi Ilyas. Setelah Nabi Ilyas meninggal, beliau melanjutkan perjuangan Nabi Ilyas untuk menghalau penyembahan berhala yang kembali merebak di lembah sungai Yordan. Namun kaum tersebut tidak mau mendengarkan sehingga terjadi bencana kekeringan kembali melanda mereka.

2.1.5.21 Nabi Yunus

Nabi yunus berusaha menyebarkan ajaran Allah, namun ia tidak mendapat sambutan baik dari masyarakat. Dalam perjalanannya menjauhi daerah tersebut karena khawatir akan dibunuh, kapal yang ia tumpangi diguncang topan dan diputuskan bahwa Nabi Yunus akan dikorbankan untuk ditenggelamkan ke laut demi keselamatan penumpang lainnya. Namun mukjizat Allah tiba, Nabi Yunus dimakan oleh seekor ikan yang kemungkinan adalah ikan paus, dan ditemukan masih hidup didalam perut ikan paus tersebut.

2.1.5.22 Nabi Zakaria

Nabi Zakaria dan istrinya, Isya, membaktikan diri untuk menjaga Baitul Maqdis – Rumah Ibadah peninggalan Nabi Sulaiman di Yerusalem. Nabi Zakaria dikaruniai keturunan oleh Allah SWT di saat usianya sudah cukup uzur, yaitu sekitar 100 tahun, anak tersebut adalah Nabi Yahya.

2.1.5.23 Nabi Yahya

Nabi Yahya adalah putra Nabi Zakaria dari perkawinannya dengan Isya. Beliau diutus menjadi nabi dan rasul kepada Bani Israil, melanjutkan risalah

ayahnya. Sejak kecil, Nabi Yahya terpelihara dari perbuatan syirik (menyekutukan Allah) dan maksiat. Beliau mengajarkan bahwa kebenaran harus ditegakkan dengan resiko apapun. Pada riwayatnya dicontohkan saat ia bersikeras melarang pernikahan antara seorang paman dengan keponakannya sendiri.

2.1.5.24 Nabi Isa

Nabi Isa adalah putra dari Bunda Maryam yang dilahirkan tanpa memiliki suami, Hal ini menimbulkan kontroversi dan hujatan bertubi-tubi kepada Maryam. Secara ajaib Nabi Isa yang saat itu masih bayi tiba-tiba berbicara dan menjelaskan apa yang sebenarnya terjadi. Bahwa penciptaan dirinya diawalai dari kedatangan Malaikat Jibril kepada ibunya. Nabi Isa juga memperlihatkan banyak mukjizat lainnya ketika ia tumbuh dewasa, diantaranya membentuk seekor burung hidup dari sebuah tanah liat, menghidupkan orang mati, menyembuhkan kebutaan dan mendatangkan makanan yang semula tidak ada dan menjadi ada. Penyelamatan Nabi Isa dari penyaliban juga merupakan salah satu bentuk mukjizat yang diberikan oleh Allah SWT.

2.1.5.25 Nabi Muhammad SAW

Nabi Muhammad adalah anak dari Abdullah bin Abdul Muthalib dan ibunya bernama Aminah. Beliau adalah rasul terakhir, sekaligus sebagai penutup para rasul-rasul sebelumnya. Dialah yang menyempurnakan ajaran-ajaran Islam. Mukjizat yang diturunkan Allah kepadanya sangat banyak, salah satu yang paling besar adalah Al-Qur'an, yang menjadi pedoman utama kehidupan manusia. Selain itu ada pula peristiwa Isra' Mi'raj yang membawanya bertemu dengan Allah.

2.1.6 Algoritma A*

Algoritma A* merupakan salah satu algoritma yang dimanfaatkan sebagai metode untuk menemukan rute (*pathfinding*). *Pathfinding* digunakan untuk menentukan arah pergerakan suatu objek dari satu tempat ke tempat lain berdasarkan keadaan peta dan objek lainnya. Algoritma yang digunakan untuk *pathfinding* sudah banyak yang ditemukan, misalnya *Bellman–Ford, Dijkstra, Floyd–Warshall, A Star* (A*), *Dynamic Weighting A** (DWA*) dan lain-lain.

Algoritma yang akan diimplementasikan pada *game* di penelitian ini adalah algoritma DWA*. Tetapi karena algoritma tersebut merupakan pengembangan dari algoritma A* maka terlebih dahulu dibahas tentang A*. Algoritma A* merupakan algoritma *Best First Search* yang menggabungkan *Uniform Cost Seach* dan *Gredy Best-First Seach*. Biaya yang diperhitungkan didapat dari biaya sebenarnya ditambah dengan biaya perkiraan. Dalam notasi matematika dituliskan sebagai:

$$f(n) = g(n) + h(n) \quad \text{.....(1)}$$

dengan:

f(n) = fungsi evaluasi

g(n) = biaya sebenarnya / jarak dari *start node* (implementasi di *game*)

h(n) = biaya perkiraan / jarak dari *end node* (implementasi di *game*)

Dengan perhitungan biaya seperti ini, algoritma A* adalah *complete* (selalu menemukan solusi jika solusinya ada) dan *optimal* (menemukan rute terpendek). (Suyanto, 2011).

A* adalah algoritma pencarian *graph*/pohon yang mencari jalur dari satu titik awal ke sebuah titik akhir yang telah ditentukan. Algoritma A* menggunakan pendekatan heuristik *h(n)* yang memberikan peringkat ke tiap-tiap titik *n* dengan cara memperkirakan rute terbaik yang dapat dilalui dari titik tersebut. Setelah itu tiap-tiap titk *n* tersebut dicek satu-persatu berdasarkan urutan yang dibuat dengan pendekatan heuristik tersebut. Maka dari itulah algoritma A* adalah contoh dari *Best-First Search*. Algoritma ini pertama kali ditemukan pada tahun 1968 oleh *Peter Hart, Nils Nilsson* dan *Bertram Raphael*. Dalam tulisan mereka, algoritma ini dinamakan algoritma A. Penggunaan algoritma ini dengan fungsi heuristik yang tepat dapat memberikan hasil yang optimal, maka algoritma inipun disebut A*. Beberapa terminologi dasar yang terdapat pada algoritma ini ketika diimplementasisan pada aplikasi *game* antara lain: *startnode*, *endnode*, simpul (*node*), *currentnode*, *openlist, closedlist,* harga/biaya (*cost*), halangan (*unwalkable*).

- *Startnode* adalah sebuah terminologi posisi awal sebuah benda.
- *Currentnode* adalah simpul yang sedang dijalankan algortima pencarian jalan terpendek.
- Simpul atau *node* adalah petak-petak kecil sebagai representasi dari area *pathfinding*. Bentuknya dapat berupa persegi, lingkaran, maupun segitiga.
- *Openlist* adalah tempat menyimpan data simpul yang mungkin diakses dari *startnode* maupun simpul yang sedang dijalankan.
- *Closedlist* adalah tempat menyimpan data simpul sebelum *currentnode* yang juga merupakan bagian dari jalur terpendek yang telah didapatkan.

- Harga (F) adalah nilai yang diperoleh dari penjumlahan nilai G, jumlah nilai tiap simpul dalam jalur terpendek dari *startnode* ke *currentnode*, dan H, jumlah nilai perkiraan dari sebuah simpul ke simpul tujuan.
- *Endnode* atau simpul tujuan yaitu simpul yang dituju.
- Rintangan (*unwalkable*) adalah sebuah atribut yang menyatakan bahwa sebuah simpul tidak dapat dilalui oleh *currentnode*.

Prinsip algoritma ini adalah mencari jalur terpendek dari sebuah simpul awal (*startnode*) menuju simpul tujuan dengan memperhatikan harga (F) terkecil. A* memperhitungkan *cost* dari *currentnode* ke tujuan dengan fungsi *heuristic*. Algoritma ini juga mempertimbangkan *cost* yang telah ditempuh selama ini dari *initial node* ke *currentnode*. Jadi jika ada jalan yang telah ditempuh sudah terlalu panjang dan ada jalan lain yang *cost*-nya lebih kecil tetapi memberikan posisi yang sama dilihat dari *goal*, jalan yang lebih pendek yang akan dipilih. Algoritma A* pada aplikasi *game* dijelaskan dengan *pseudocode* dibawah ini:

1. Masukan *startnode* ke *openlist*
2. *Loop* langkah-langkah di bawah ini :

    a. Cari *node(n)* dengan nilai *f(n)* yang paling rendah dalam *openlist*, *node* ini sekarang menjadi *currentnode*.

    b. Keluarkan *currentnode* dari *openlist* dan masukan ke *closedlist.*

    c. Untuk setiap tetangga dari *currentnode* lakukan berikut :

    • Jika tidak dapat dilalui atau sudah ada dalam *closelist*, maka abaikan.

• Jika belum ada di *openlist* . Buat *currentnode parent* dari *node* tetangga ini. Simpan nilai *f,g* dan *h* dari *node* ini.

• Jika sudah ada di *openlist*, cek bila *node* tetangga ini lebih baik, menggunakan nilai *g* sebagai ukuran. Jika lebih baik ganti *parent* dari *node* ini di *openlist* menjadi *currentnode*, lalu kalkulasi ulang nilai *g* dan *f* dari *node* ini.

d. Hentikan *loop* jika :

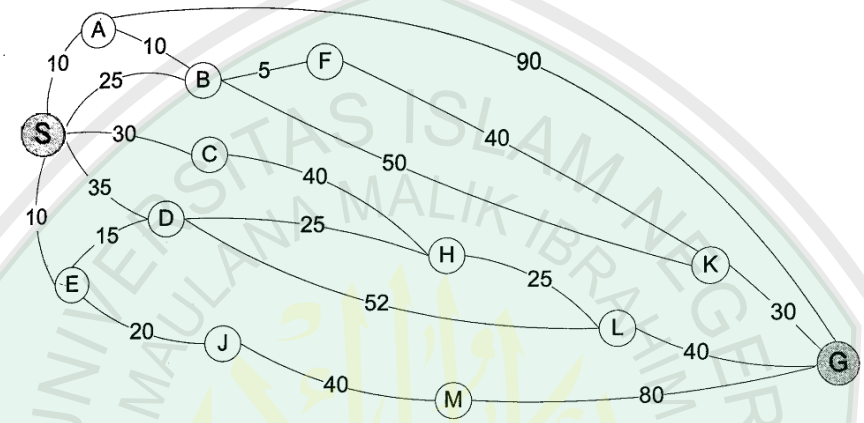
• *Node* tujuan telah ditambahkan ke *openlist*, yang berarti rute telah ditemukan.

• Belum menemukan *node goal*/*endnode* sementara *openlist* kosong atau berarti tidak ada rute.

3. Simpan rute, secara *backward*, urut mulai dari *node goal* ke *parent*-nya terus sampai mencapai *node* awal sambil menyimpan *node* ke dalam array.

Untuk lebih jelasnya, perhatikan sebuah masalah pencarian rute terpendek dari kota S (*initial state*) ke kota G (*goal state*) pada Gambar 2.1 dibawah ini. Terdapat 13 kota yang dinyatakan oleh simpul-simpul dalam suatu *graph* dua arah. Setiap angka pada busur menyatakan biaya sebenarnya antara satu kota dengan kota lainnya. Misalnya biaya disini adalah jarak antar kota dalam satuan kilometer. Nilai *h(n)* adalah fungsi heuristik, yaitu jarak garis lurus dari simpul *n* menuju simpul G dalam satuan kilometer. Pada langkah-langkah penyelesaian masalah rute terpendek dengan algoritma A* kali ini, ada sedikit perbedaan dengan *pseudocode* diatas. Simpul-simpul yang sudah ada di *closed* tidak diabaikan, tetapi masih harus di cek lagi apakah perlu atau tidak untuk mengganti

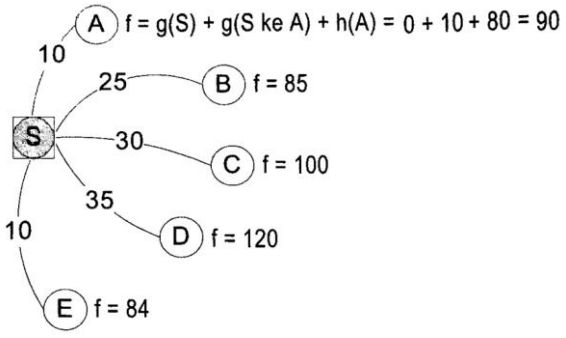
parent-nya, karena biaya sebenarnya atau *g(n)* pada masalah ini nilainya berbeda-beda disetiap jarak antar simpulnya. Sedangkan ketika diimplementasikan pada *game* di *Unity* (dibahas pada bab 3), *g(n)* memiliki nilai yang sama yaitu selalu +10 untuk gerakan horizontal atau vertikal dan +14 untuk gerakan diagonal.



n	S	A	B	C	D	E	F	G	H	J	K	L	M
h(n)	80	80	60	70	85	74	70	0	40	100	30	20	70

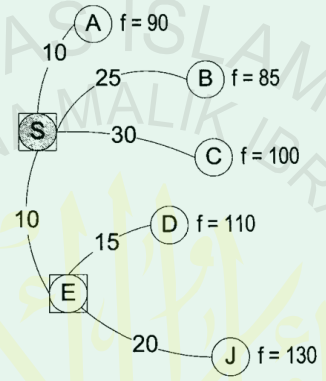
Gambar 2.1 Masalah Pencarian Rute Terpendek Pada Suatu Graf

Untuk memperjelas pemahaman tentang algoritma A*, di bawah ini akan dijelaskan penyelesaian masalah diatas menggaunakan algoritma A*. Langkah-langkah pencarian rute berdasarkan algoritma tersebut adalah sebagai berikut:



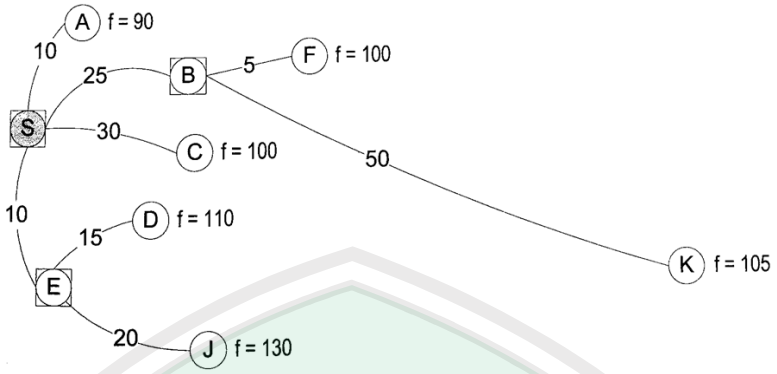
Gambar 2.2 Pencarian Rute Terpendek dengan Algoritm A* Langkah 1

Langkah pertama, karena di *open* hanya terdapat satu simpul (yaitu S), maka S terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Kemudian dibangkitkan semua suksesor S, yaitu: A, B, C, D dan E. Karena kelima suksesor tidak ada di *open* maupun *closed*, maka kelimanya dimasukkan ke *open*. Langkah pertama ini menghasilkan *open* = [A,B,C,D,E] dan *closed* = [S].



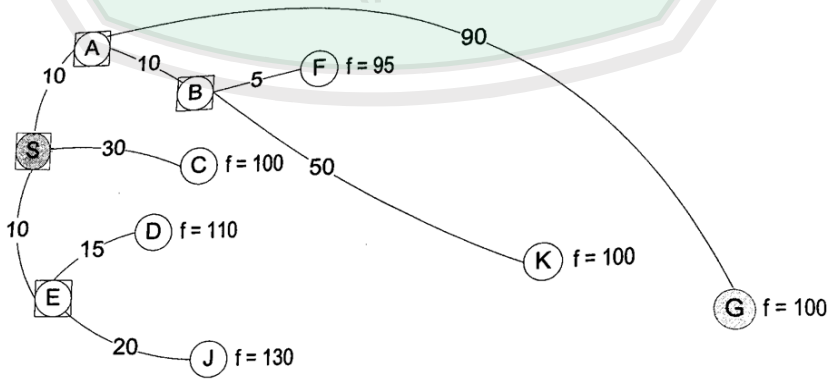
Gambar 2.3 Pencarian Rute Terpendek dengan Algoritm A* Langkah 2

Langkah kedua, E dengan biaya terkecil (yaitu 84) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor E dibangkitkan, yaitu: D dan J. Karena belum pernah ada di *open* maupun *closed*, maka J dimasukkan ke *open*. Sedangkan simpul D sudah ada di *open*, maka harus dicek, apakah *parent* dari D perlu diganti atau tidak. Ternyata biaya dari S ke D melalui E (yaitu 10+15 = 25) lebih kecil daripada biaya S ke D (yaitu 35). Oleh karena itu, *parent* D harus diubah, yang semula S menjadi E. Dengan perubahan *parent* ini maka nilai *g* dan *f* pada D juga diperbarui (nilai *g* yang semula 35 menjadi 25, dan nilai *f* dari 120 menjadi 110). Akhir dari langkah kedua ini menghasilkan *open* = [A,B,C,D,J] dan *closed* = [S,E].



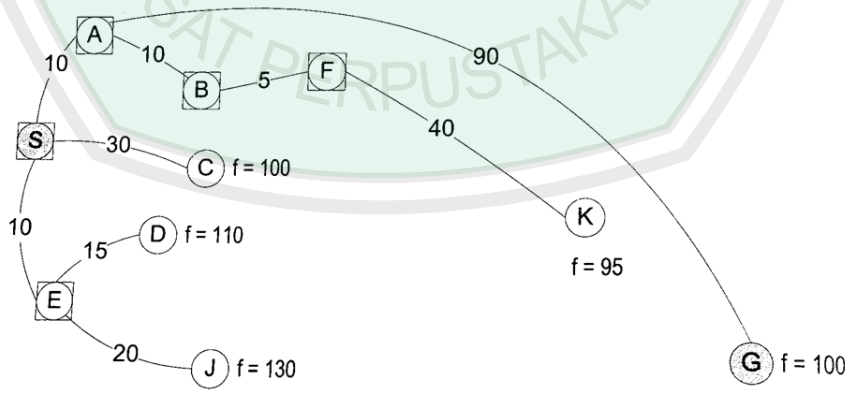
Gambar 2.4 Pencarian Rute Terpendek dengan Algoritm A* Langkah 3

Langkah ketiga, B dengan biaya terkecil (yaitu 85) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor B dibangkitkan, yaitu: A, F dan K. Karena belum pernah ada di *open* maupun *closed*, maka F dan K dimasukkan ke *open*. Sedangkan simpul A sudah ada di *open*, maka harus dicek, apakah *parent* dari A perlu diganti atau tidak. Ternyata biaya dari S ke A melalui B (yaitu 25+10 = 35) lebih besar daripada biaya S ke A (yaitu 10). Oleh karena itu, *parent* dari A tidak perlu diubah (tetap S). Akhir dari langkah ketiga ini menghasilkan *open* = [A,C,D,F,J,K] dan *closed* = [S,E,B].



Gambar 2.5 Pencarian Rute Terpendek dengan Algoritm A* Langkah 4

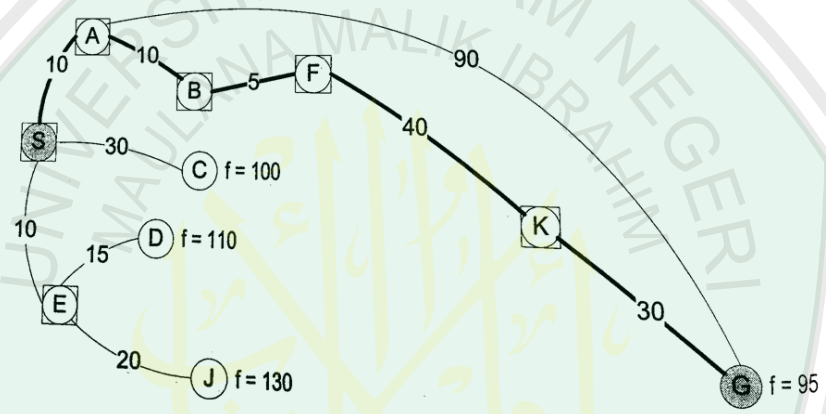
Langkah keempat, A dengan biaya terkecil (yaitu 90) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor A dibangkitkan, yaitu: B dan G. Karena belum pernah ada di *open* maupun *closed*, maka G dimasukkan ke *open*. Sedangkan simpul B sudah ada di *closed*, maka harus dicek, apakah *parent* dari B perlu diganti atau tidak. Ternyata biaya dari S ke B melalui A (yaitu 10+10 = 20) lebih kecil daripada biaya S ke B (yaitu 25). Oleh karena itu, *parent* dari B harus diubah, yang semula S menjadi A. Dengan perubahan *parent* ini maka nilai *g* dan *f* pada B juga diperbarui (nilai *g* yang semula 25 menjadi 20, dan nilai *f* dari 85 menjadi 80). Nilai *g* dan *f* pada suksesor-suksesor B juga harus diperbarui menngunakan penelusuran *Dept First Search* (DFS). Dalam kasus ini, B hanya mempunyai dua anak, yaitu F dan K. Nilai *g*(F) yang semula 30 diubah menjadi 25, dan nilai *f*(F) dari 100 menjasi 95. Nilai *g*(K) yang semula 75 diubah menjadi 70, dan nilai *f*(K) dari 105 menjadi 100. Akhir langkah keempat ini menghasilkan *open* = [C,D,F,G,J,K] dan *closed* = [S,E,B,A].



Gambar 2.6 Pencarian Rute Terpendek dengan Algoritm A* Langkah 5

Langkah kelima, F dengan biaya terkecil (yaitu 95) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor F dibangkitkan,

yaitu: K. Karena K sudah ada di *open*, maka harus dicek, apakah *parent* dari K perlu diganti atau tidak. Ternyata biaya dari S ke K melalui F lebih kecil daripada biaya S ke K melalui parent lama (B). Oleh karena itu, *parent* dari K harus diubah, yang semula B menjadi F. Selanjutnya, nilai *g*(K) yang semula 70 diubah menjadi 65, dan nilai *f*(K) dari 100 menjadi 95. Akhir dari langkah kelima ini menghasilkan *open* = [C,D,G,J,K] dan *closed* = [S,E,B,A,F].



Gambar 2.7 Pencarian Rute Terpendek dengan Algoritm A* Langkah 6

Langkah keenam, K dengan biaya terkecil (yaitu 95) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor K dibangkitkan, yaitu: G. Karena G sudah ada di *open*, maka harus dicek, apakah *parent* dari G perlu diganti atau tidak. Ternyata biaya dari S ke G melalui K lebih kecil daripada biaya S ke G melalui *parent* lama (A). Oleh karena itu, *parent* dari G harus diubah, yang semula A menjadi K. Selanjutnya, nilai *g*(G) yang semula 100 diubah menjadi 95, dan nilai *f*(G) dari 100 menjadi 95. Pada akhir langkah keenam ini menghasilkan *open* = [C,D,G,J] dan *closed* = [S,E,B,A,F,K].

Selanjutnya, G dengan biaya terkecil (yaitu 95) terpilih sebagai *bestnode*. Karena *bestnode* sama dengan *goal*, berarti solusi sudah ditemukan. Rute dan total biaya bisa ditelusuri balik dari G menuju S karena setiap simpul hanya memiliki satu *parent*. Penelusuran balik menghasilkan rute S-A-B-F-K-G dengan total jarak sama dengan 95 kilometer. Rute ini merupakan rute terpendek yang ada di *graph* tersebut. Jadi algoritma A* adalah *optimal* (menemukan rute terpendek) dan *complete* (selalu menemukan solusi jika solusinya ada). (Suyanto, 2011).

2.1.7 Algoritma *Dynamic Weighting A**

Algoritma DWA* merupakan pengembangan dari algoritma A*. Algoritma DWA* merupakan salah jenis algoritma yang menerapkan sistem pencarian solusi dengan memperhitungkan nilai heuristik. Perbedaan dengan algoritma A* adalah algoritma ini memberikan sebuah nilai bobot yang dinamis terhadap fungsi heuristik *h*. Dengan menggunakan bobot yang dinamis ini, bisa diasumsikan bahwa pada awal iterasi, sebaiknya pencarian dilakukan ke segala arah, namun begitu akan mencapai *goal state*, proses pencarian bisa difokuskan ke arah yang lebih spesifik.

Untuk dapat melakukan hal ini, diperlukanlah suatu nilai *weight* yang dinamis, dimana nilai tersebut akan semakin kecil apabila sudah mendekati *goal*. Algorima ini juga masih menggunakan fungsi heuristik, dalam notasi matematika dituliskan sebagai berikut:

$f(n) = g(n) + w(n)*h(n)$ .....(2)

dengan :

$f(n)$ = fungsi evaluasi

g(n) = biaya sebenarnya / jarak dari *start node* (implementasi di *game*)

h(n) = biaya perkiraan / jarak dari *end node* (implementasi di *game*)

w(n) = bobot dinamis

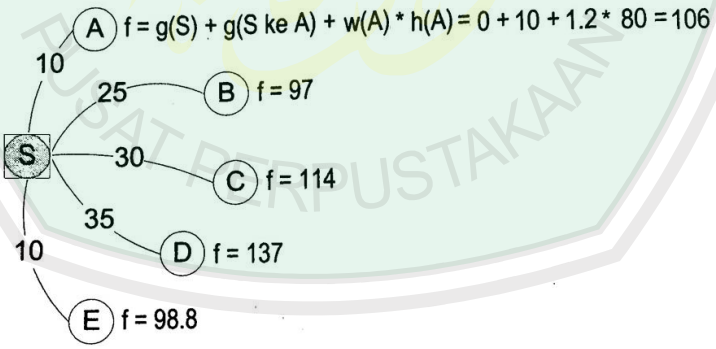
Pada perumusan fungsi heuristik tersebut, nilai *w* haruslah lebih besar dari satu. Pada awal iterasi, sebaiknya digunakan nilai *w* yang sangat besar, dan pada iterasi berikutnya, nilai *w* dapat diturunkan secara bertahap. Pada saat proses akan mencapai *goal*, maka nilai *w* yang dipakai harus semakin mendekati nilai 1. Dari sini bisa dilihat bahwa pada awal iterasi nilai *h(n)* dianggap penting untuk diperhitungkan, sedangkan pada saat akan mencapai *goal*, proses pencarian lebih banyak dipengaruhi oleh nilai sebenarnya *g(n).*

Variasi seperti ini akan sangat terasa kegunaannya pada permasalahan dengan fungsi heuristik menghasilkan nilai yang bervariasi. Dalam kata lain, bisa terdapat biaya perkiraan yang sangat jauh lebih kecil dibandingkan biaya sebenarnya dan ada juga biaya perkiraan yang sudah sangat mendekati biaya sebenarnya. Hal ini lah yang membedakannya dengan algoritma A*, dimana pada algoritma A* masih ada kemungkinan terjadinya kesalahan dalam membangkitkan simpul yang disebabkan bobot yang dipakai semua simpul sama.

Algoritma DWA* dapat memberikan solusi yang tepat dan waktu eksekusi yang cepat dan penggunaan memory yang kecil. Waktu eksekusi DWA* dipengaruhi oleh jumlah *node* yang ada pada ruang masalah. Waktu eksekusi akan bertambah seiring dengan pertambahan jumlah *node*. Nilai yang dinamis pada DWA* sangat mempengaruhi dalam menemukan *goal state*, nilai *w* yang tepat

akan mengarah pada solusi yang tepat, sedangkan jika nilai *w* yang tidak tepat akan cenderung membuat pencarian menjadi salah arah.

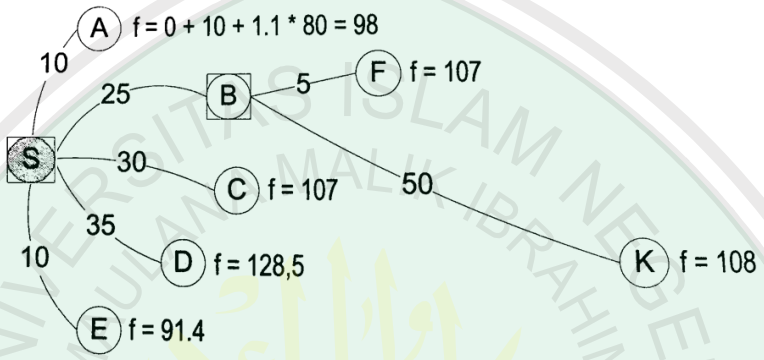
Implementasi algoritma DWA* sama dengan A*, tetapi ada mekanisme pengubahan *w* sehingga fungsi heuristiknya menjadi dinamis. Untuk lebih memahami DWA*, pada Gambar 2.3 mengilustrasikan langkah-langkah DWA* untuk menyelesaikan masalah pada Gambar 2.1 diatas. Seperti halnya pada langkah-langkah penyelesaian masalah rute terpendek dengan algoritma A*, simpul yang sudah ada di *closed* pada kasus ini tidak diabaikan, tetapi masih harus di cek lagi apakah perlu atau tidak untuk mengganti *parent*-nya, karena biaya sebenarnya atau *g(n)* pada masalah ini nilainya berbeda-beda disetiap jarak antar simpulnya. Sedangkan ketika diimplementasikan pada *game* di *Unity* (dibahas pada bab 3), *g(n)* memiliki nilai yang sama yaitu selalu +10 untuk gerakan horizontal atau vertikal dan +14 untuk gerakan diagonal.



Gambar 2.8 Pencarian Rute Terpendek dengan Algoritm DWA* Langkah 1

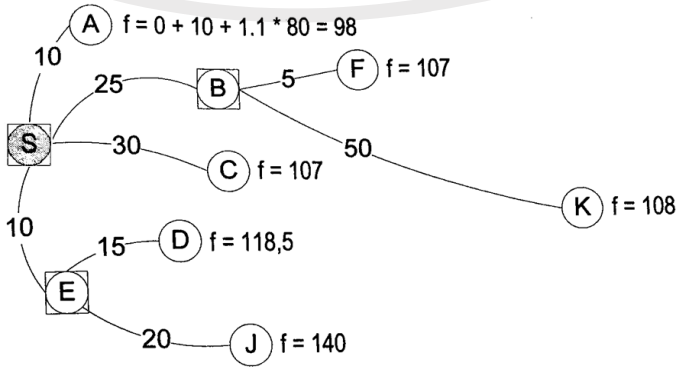
Langkah pertama, karena di *open* hanya terdapat satu simpul (yaitu S), maka S terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Kemudian dibangkitkan semua suksesor S, yaitu: A, B, C, D dan E. Pada langkah ini,

misalkan digunakan *w* = 1,2. Maka hasil penghitungan nilai *f* seperti pada Gambar 2.8. Karena kelima suksesor tidak ada di *open* maupun *closed*, maka kelimanya dimasukkan ke *open*. Langkah pertama ini menghasilkan *open* = [A,B,C,D,E] dan *closed* = [S].



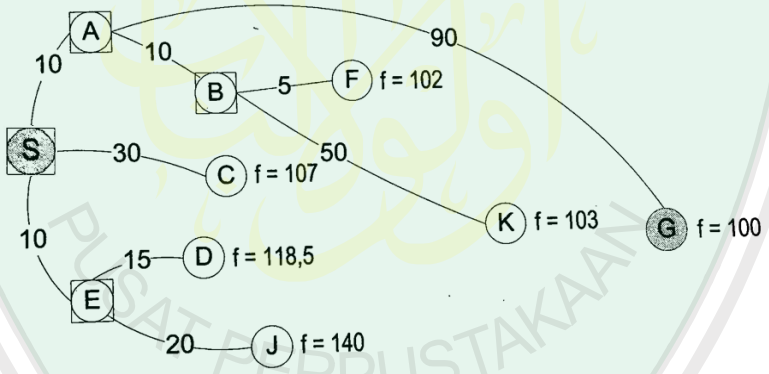
Gambar 2.9 Pencarian Rute Terpendek dengan Algoritm DWA* Langkah 2

Langkah kedua, misalkan nilai *w* diturunkan menjadi 1,1. Simpul B dengan biaya terkecil (yaitu 97) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor B dibangkitkan, yaitu: F dan K. Karena belum pernah ada di *open* maupun *closed*, maka F dan K dimasukkan ke *open*. Langkah kedua ini menghasilkan *open* = [A,C,D,E,F,K] dan *closed* = [S,B].



Gambar 2.10 Pencarian Rute Terpendek dengan Algoritm DWA* Langkah 3

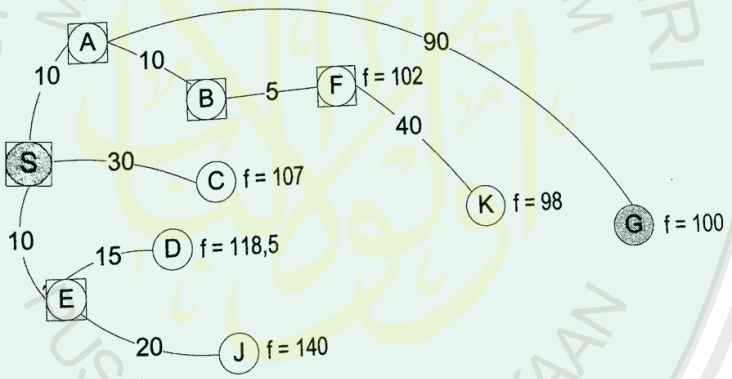
Langkah ketiga, misalkan nilai *w* tetap 1,1. Simpul E dengan biaya terkecil (yaitu 84) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor E dibangkitkan, yaitu: D dan J. Karena belum pernah ada di *open* maupun *closed*, maka J dimasukkan ke *open*. Sedangkan simpul D sudah ada di *open*, maka harus dicek, apakah *parent* dari D perlu diganti atau tidak. Ternyata biaya dari S ke D melalui E (yaitu 10+15 = 25) lebih kecil daripada biaya S ke D (yaitu 35). Oleh karena itu, *parent* dari D harus diubah, yang semula S menjadi E. Dengan perubahan *parent* ini maka nilai *g* dan *f* pada D juga diperbarui (nilai *g* yang semula 35 menjadi 25, dan nilai *f* dari 128,5 menjadi 118,5). Akhir dari langkah ketiga ini menghasilkan *open* = [A,C,D,F,J,K] dan *closed* = [S,B,E].



Gambar 2.11 Pencarian Rute Terpendek dengan Algoritm DWA* Langkah 4

Langkah keempat, misalkan nilai *w* tetap 1,1. Simpul A dengan biaya terkecil (yaitu 98) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor A dibangkitkan, yaitu: B dan G. Karena belum pernah ada di *open* maupun *closed*, maka G dimasukkan ke *open*. Sedangkan simpul B sudah ada di *closed*, maka harus dicek, apakah *parent* dari B perlu diganti atau tidak. Ternyata biaya dari S ke B melalui A (yaitu 10+10 = 20) lebih

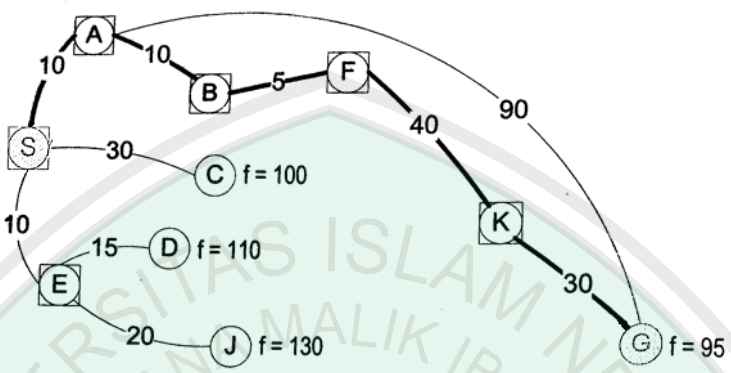
kecil daripada biaya S ke B (yaitu 25). Oleh karena itu, *parent* dari B harus diubah, yang semula S menjadi A. Dengan perubahan *parent* ini maka nilai *g* dan *f* pada B juga diperbarui (nilai *g* yang semula 25 menjadi 20, dan nilai *f* dari 97 menjadi 80). Nilai *g* dan *f* pada suksesor-suksesor B juga harus diperbarui menngunakan penelusuran *Dept First Search* (DFS). Dalam kasus ini, B hanya mempunyai dua anak, yaitu F dan K. Nilai *g*(F) yang semula 30 diubah menjadi 25, dan nilai *f*(F) dari 107 menjadi 102. Nilai *g*(K) yang semula 75 diubah menjadi 70, dan nilai *f*(K) dari 108 menjadi 103. Akhir dari langkah keempat ini menghasilkan *open* = [C,D,F,G,J,K] dan *closed* = [S,E,B,A].



Gambar 2.12 Pencarian Rute Terpendek dengan Algoritm DWA* Langkah 5

Langkah kelima, misalkan nilai *w* tetap 1,1. Simpul F dengan biaya terkecil (yaitu 102) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor F dibangkitkan, yaitu: K. Karena K sudah ada di *open*, maka harus dicek, apakah *parent* dari K perlu diganti atau tidak. Ternyata biaya dari S ke K melalui F lebih kecil daripada biaya S ke K melalui *parent* lama (simpul B). Oleh karena itu, *parent* dari K harus diubah, yang semula B menjadi F. Selanjutnya, nilai *g*(K) yang semula 70 diubah menjadi 65, dan nilai *f*(K) dari

103 menjadi 98. Akhir dari langkah kelima ini menghasilkan *open* = [A,C,D,F,G,J,K] dan *closed* = [S,E,B,A,F].



Gambar 2.13 Pencarian Rute Terpendek dengan Algoritm DWA* Langkah 6

Langkah keenam, misalkan nilai *w* diturunkan menjadi 1. Simpul K dengan biaya terkecil (yaitu 98) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor K dibangkitkan, yaitu: G. Karena G sudah ada di *open*, maka harus dicek, apakah *parent* dari G perlu diganti atau tidak. Ternyata biaya dari S ke G melalui K lebih kecil daripada biaya S ke G melalui parent lama (simpul A). Oleh karena itu, *parent* dari G harus diubah, yang semula A menjadi K. Selanjutnya, nilai *g*(G) yang semula 100 diubah menjadi 95, dan nilai *f*(G) dari 100 menjadi 95. Pada akhir langkah keenam ini menghasilkan *open* = [C,D,G,J] dan *closed* = [S,E,B,A,F,K].

Selanjutnya, G dengan biaya terkecil (yaitu 95) terpilih sebagai *bestnode*. Karena *bestnode* sama dengan *goal*, berarti solusi sudah ditemukan. Rute dan total biaya bisa ditelusuri balik dari G menuju S karena setiap simpul hanya memiliki satu *parent*. Penelusuran balik menghasilkan rute S-A-B-F-K-G dengan total jarak

sama dengan 95 kilometer. Rute ini merupakan rute terpendek yang ada di *graph* tersebut. Jadi algoritma DWA* adalah *optimal* (menemukan rute terpendek) dan *complete* (selalu menemukan solusi jika solusinya ada). (Suyanto, 2011).

2.1.8 Algoritma *Fisher-Yates Shuffle*

Konsep diambil dari nama *Ronald Fisher* dan *Frank Yates* atau juga dikenal dengan nama *Knuth shuffle* (diambil dari nama *Donald Knuth*) merupakan sebuah algoritma untuk menghasilkan suatu permutasi acak dari suatu himpunan yang terhingga, atau dengan kata lain untuk mengacak suatu himpunan tertentu. Apabila diimplementasikan secara benar, maka akan mendapatkan hasil yang tidak akan berat sebelah, sehingga setiap permutasi akan memiliki kemungkinan yang sama.

Pemakaian *Fisher-Yates Shuffle* bisa melalui dua cara yaitu: *original method* dan *modern method*. Menurut *Pavel Micka* (2011) *original method* dipublikasikan pada tahun 1938, pada metode ini dilakukan dengan cara penarikan secara berulang dari unsur daftar masukan kemudian menuliskannya ke daftar keluaran kedua. Pendekatan ini dilakukan oleh manusia dengan secarik kertas dan sebuah pensil.

Pada *modern method* dijabarkan untuk penggunaan komputerisasi yang dikenalkan oleh *Richard Durstenfield* pada tahun 1964. *Modern method* dikenalkan karena lebih optimal dibandingkan dengan *original method*. Algoritma yang modern berbeda dari yang sebelumnya, sangat komputasi dan matematis. Prosesnya angka terakhir akan dipindahkan ke angka yang ditarik keluar dan mengubah angka yang ditarik keluar menjadi angka akhir yang tidak ditarik lagi

untuk setiap kali penarikan dan berlanjut untuk iterasi berikutnya. Hal ini dilakukan dalam *O*(1) waktu dan ruang. Degan demikian, waktu dan ruang kompleksitas algoritmanya *O(n),* yang *optimal*.

Menurut Vinay Signh (2014) penggunaan algoritma *Fisher-Yates* yang modern oleh Richard Durstenfeld dapat mengurangi kompleksitas algoritma menjadi *O(n),* dibandingkan dengan mengacak menggunakan metode yang lain seperti menggunakan *sorting* yang sangat tidak efisien karena adanya *loop* bersarang.

Algoritma *Fisher-Yates* dipilih karena algoritma ini merupakan metode pangacakan yang lebih baik atau dapat dikatakan sesuai untuk pengacakan angka, dengan waktu eksekusi yang cepat serta tidak memerlukan waktu yang lama untuk melakukan suatu pengacakan. Algoritma *Fisher-Yates* terdiri dari dua metode yakni, metode orisinal dan metode modern. Namun dalam pengembangan aplikasi ini algoritma ini diterapkan dengan menggunakan metode modern. Metode modern dipilih karena metode ini memang khusus digunakan untuk pengacakan dengan sistem komputerisasi, dikarenakan hasil pengacakan bisa lebih variatif.

Metode yang digunakan untuk menghasilkan suatu permutasi acak untuk angka 1 sampai N adalah sebagai berikut:

1. Tuliskan angka dari 1 sampai N.
2. Pilih sebuah angka acak K diantara 1 sampai dengan jumlah angka yang belum dicoret.
3. Dihitung dari bawah, coret angka K yang belum dicoret, dan tuliskan angka tersebut di lain tempat.

4. Ulangi langkah 2 dan langkah 3 sampai semuat angka sudah tercoret.
5. Urutan angka yang dituliskan pada langkah 3 adalah permutasi acak dari angka awal.

Pada versi modern yang digunakan sekarang, angka yang terpilih tidak dicoret, tetapi posisinya ditukar dengan angka terakhir dari angka yang belum terpilih. Metode modern yang digunakan untuk menghasilkan suatu permutasi acak untuk angka 1 sampai N adalah sebagai berikut:

1. Masukkan array yang akan diacak urutannya
2. Hitung jumlah elemen dari 1sampai N
3. Ambil elemen secara acak dari elemen yang tersisa
4. Tukar dengan elemen ke-N
5. Ulangi selama masih ada elemen yang tersisa
6. Tampilkan array baru yang telah diacak urutannya

Pengacakan suatu hal yang sangat penting dalam pembuatan banyak aplikasi. Meskipun terlihat mudah, namun pada dasarnya jika tidak dilakukan dengan baik maka pengacakan itu dapat berdampak buruk untuk suatu aplikasi.

2.2 Penelitian Terkait

2.2.1 Penentuan Lokasi Parkir Pada *Smart Parking System* menggunakan *Dynamic Weighting A** (DWA*)

Aplikasi yang dibuat mampu menentukan lokasi parkir terdekat dari pintu masuk dan memberikan lintasan terdekat menuju lokasi parkir menggunakan metode pencarian *heuristic* yaitu *Dynamic Weighting A** (DWA*), dengan menggunakan algoritma ini sistem akan dapat memberikan informasi yang dapat

mempermudah proses parkir jika keadaan di lokasi parkir sangat padat dan tidak teratur. Waktu eksekusi algoritma ini pada ruang masalah area parkir yang cukup besar membutuhkan waktu yang relatif singkat, dibandingkan dengan menggunakan algoritma Dijkstra yang memiliki waktu eksekusi yang jauh lebih lama. Sedangkan untuk penggunaan memory, algoritma *Dynamic Weighting A** (DWA*) menggunakan memory lebih kecil karena jumlah node yang dibangkitkan sedikit. (Rendi Christian et al, 2011).

2.2.2 Analisis Perbandingan Algoritma DWA* dan Algoritma F2L Pada Permasalahan Kubus Rubik 3x3x3

Perbandingan algoritma ini dilakukan untuk menganalisa algoritma manakah yang lebih baik dalam pemecahan masalah Rubik, baik dilihat dari kompleksitas waktu asimptotiknya secara teori, maupun dengan mengukur tingkat optimasi solusi yang dihasilkan. Dari penelitian tersebut didapatkan bahwa algoritma DWA* mempunyai waktu eksekusi dan ruang memory yang dibutuhkan yang cenderung naik seiring bertambahnya jumlah acakan pada *initial state*, sedangkan algoritma F2L mempunyai waktu eksekusi yang cenderung stabil. Kompleksitas waktu asimtotik pada algoritma DWA* akan bernilai ekponensial seiring bertambahnya jumlah acakan pada *initial state*, sedangkan algoritma *First Two Layers* (F2L) bernilai konstan. Dari panjang solusi yang dihasilkan, *Dynamic Weighting A** (DWA*) menghasilkan solusi yang lebih optimal dibandingkan *First Two Layers* (F2L). (Ferico Elmanus et al, 2012).

2.2.3 Penerapan Algoritma A* (A *Star*) Pada Game Edukasi *The Maze Island* Berbasis Android

Aplikasi *game* yang dibuat pada penelitian ini berhasil menerapkan algortima A*, algoritma tersebut telah berjalan dengan baik karena musuh dapat mendeteksi posisi pemain. Kemudian musuh juga dapat mengejar pemain dengan memilih rute terpendek melalui nilai heuristik terkecil dari setiap simpul yang dipilih. Musuh akan selalu mengejar pemain jika musuh masih ada di dalam area pengejaran musuh. (Agung Pamungkas et al, 2014)

2.2.4 Implementasi *Pathfinding* dengan Algoritma A* pada *Game Funny English* Menggunakan *Unity 3D* Berbasis *Graf*.

Game Funny English dikembangkan dengan *Unity 3D 4.34* bertujuan mempermudah pengenalan kosakata bahasa Inggris yang disajikan dalam bentuk objek 3D serta audio pengucapan objek tersebut. Algoritma A* diimplementasikan pada NPC untuk mencari rute menuju ke suatu tempat. Dari seluruh pengujian yang telah dilakukan, dapat diambil kesimpulan bahwa seluruh rute yang diambil oleh NPC telah sesuai dengan rute optimal yang harus ditempuh secara logika manusia. Rata-rata waktu yang dibutuhkan untuk pengambilan rute juga terjadi dalam waktu yang singkat (≤0.2 detik). (Riwinoto dan Alfian, 2014).

2.2.5 Penerapan Algoritma *Fisher-Yates Shuffle* Pada Aplikasi *The Lost Insect* Untuk Pengenalan Jenis Serangga Berbasis *Unity 3D.*

Aplikasi yang dibangun adalah *The Lost Insect* bertemakan pengenalan serangga dengan metode petualangan pada habitat serangga dan pengacakan soal

dengan menggunakan algoritma *Fisher-Yates Shuffle*. Aplikasi ini juga dibangun dengan menggunakan metodologi RAD (*Rapid Application Development*) dikarenakan waktu pengerjaan aplikasi yang relatif singkat. Berdasarkan dari hasil pengujian dengan persentase tingkat aplikasi dapat dengan mudah digunakan dengan perolehan persentase mencapai 66,25% dan kepuasan penggunaanya mencapai 80%. (Ryan Nugraha et al, 2014).

2.2.6 Penerapan Algoritma *Fisher-Yates Shuffle* pada *Edugame Guess Caculation* Berbasis *Android*

Aplikasi yang dibangun adalah *edugame Guess Calculation* berbasis Android bertemakan edukasi atau edugame dengan metode perhitungan logika matematika sederhana yang menggunakan bahasan waktu dan pengacakan *puzzle* berbasis *Fisher-Yates Shuffle*.Selain itu aplikasi *edugame Guess Calculation* dibangun untuk menggali minat belajar anak-anak terhadap pelajaran matematika. Aplikasi ini juga dibangun dengan menggunakan metodologi RAD (*Rapid Application Development*) dikarenakan waktu pengerjaan aplikasi yang relatif singkat. Hasil pengujian membuktikan bahwa aplikasi dapat memenuhi tujuan awal yaitu berdasarkan hasil pengujian melalui kuesioner yang ada, aplikasi dapat membantu anak-anak dalam pembelajaran matematika dengan persentase tingkat kepuasan pemakainya mencapai 98%, dan aplikasi dapat dengan mudah digunakan dengan perolehan persentase mencapai 97%. (Supriyanto et al, 2014).

BAB III

PERANCANGAN GAME

3.1 Perancangan *Game*

3.1.1 Keterangan Umum *Game*

Game ini adalah *game* yang bergenre *adventure game* dengan konten islami tentang nabi dan rasul yang dimainkan secara *single player*. Pada *game* ini terdapat karakter *player* sebagai pemain utama yang akan dijalankan oleh pengguna. Dan ada dua karakter NPC yaitu NPC *Army* dan NPC *Enemy* yang merupakan karakter lawan akan dijalankan secara otomatis oleh komputer sesuai dengan yang telah di programkan. *Game* ini lebih bersifat bermain sambil belajar dan dapat memberikan pembelajaran untuk mengenal nama-nama nabi dan rasul beserta mukjizatnya.

Game ini mengisahkan perjalanan *player* dalam menyelesaikan misi untuk meningkatkan pengetahuannya dalam mengenal nama-nama nabi dan rasul beserta mukjizatnya. Misi utama *player* adalah untuk mendapatkan berlian yang ada di arena permainan. Tetapi ketika permainan dimulai berlian tersebut masih berada di dalam tembok pelindung yang tidak bisa dilewati oleh *player*. Supaya tembok tersebut runtuh dan *player* bisa mengambil berlian tersebut, terlebih dahulu *player* harus mengumpulkan koin emas dan koin perak yang ada di arena permainan. Setiap koin yang didapat akan menambah poin *player*. Jika poin *player* sudah mencukupi sesuai dengan level yang dimainkan, maka tembok pelindung berlian akan runtuh dan *player* bisa mengambil berlian tersebut.

Selama perajalan mengumpulkan koin emas, koin perak, dan berlian ada dua karakter NPC yang mengganggu perjalanan *player*. NPC *Enemy* akan bergerak mengikuti kemanapun *player* pergi. *Player* harus berlari untuk menghindari NPC *Enemy*, jika *player* terkena NPC *Enemy* maka kesehatan *player* berkurang satu poin dan NPC *Enemy* mati, tetapi masih ada NPC *Enemy* lain yang harus dihindari oleh *player*. Jumlah dari NPC *Enemy* yang mengejar *player* berbeda-beda sesuai level yang sedang dimainkan oleh *player*.

Karakter NPC yang lain yaitu NPC *Army* merupakan satu pasukan NPC yang akan mencuri atau mengambil semua koin perak yang ada di arena permainan. Sehingga *player* harus dengan segera jika ingin mengambil koin perak untuk menambah jumlah poinnya, supaya tidak terlebih dahulu dihabiskan oleh pasukan NPC *Army*. Pergerakan dari NPC *Army* ini mengimplementasikan algoritma *Dynamic Weighting A** (DWA*). Ketika permainan dimulai setiap NPC *Army* akan mencari rute terdekat menuju satu koin perak yang menjadi tujuannya. Setelah rute terpendek berhasil ditemukan, NPC *Army* akan bejalan menuju koin sesuai dengan rute terpendek yang telah ditemukan. Setelah sampai tujuannya, NPC *Army* langsung mengambil koin tersebut dan tugasnya selesai. Untuk selanjutnya NPC *Army* tetap diam di tempat dia mengambil koin, yang juga menjadi tanda untuk *player* bahwa di tempat tersebut pernah ada koin tetapi telah lebih dulu diambih oleh NPC *Army*.

Satu koin perak yang didapat *player* akan mendapatkan lima poin, sedangkan untuk koin emas sepuluh poin. Tetapi untuk setiap satu koin emas yang didapat, *player* akan mendapatkan satu pengetahuan tentang nabi dan rasul.

Materi pengetahuan tersebut telah disiapkan sebelumnya dan ketika permainan dimulai, beberapa pengetahuan tersebut diacak menggunakan algoritma *Fisher-Yates Shuffle*. Sehingga urutannya berubah setiap kali perbaiman baru dimulai. Jadi nantinya pengetahuan yang ditampilkan menjadi kelihatan bervariasi dan setiap koin emas pasti akan menampilkan materi pengetahuan yang berbeda. Ditambah lagi selain ditampilkan dalam berntuk teks, pengetahuan tersebut juga akan dibacakan dalam bentuk audio supaya lebih mudah diterima oleh pengguna.

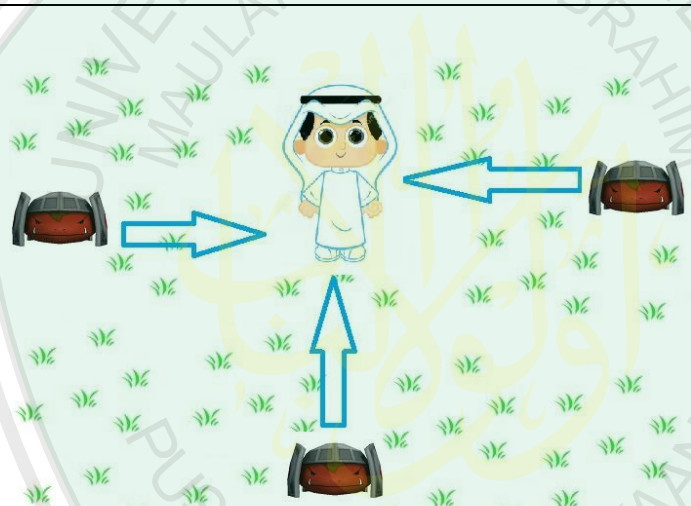
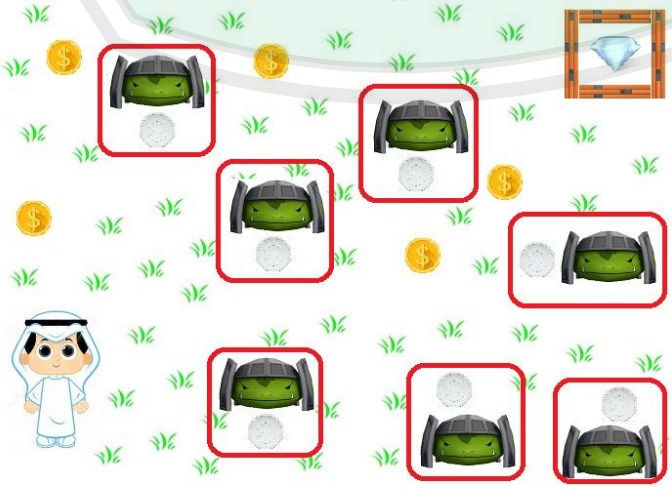
Jika poin *player* dari hasil mengumpulkan koin emas dan koin perah telah mencukupi sesuai dengan level yang dimainkan, maka tembok pelindung berlian akan runtuh, sehingga *player* bisa mengambil berlian yang ada di dalamnya. Ketika *player* telah berhasil mengambil berlian tersebut, berarti *player* telah berhasil menyelesaikan misi pada permainan ini. *Player* dinyatakan menang dan bisa bermain di level selanjutnya.

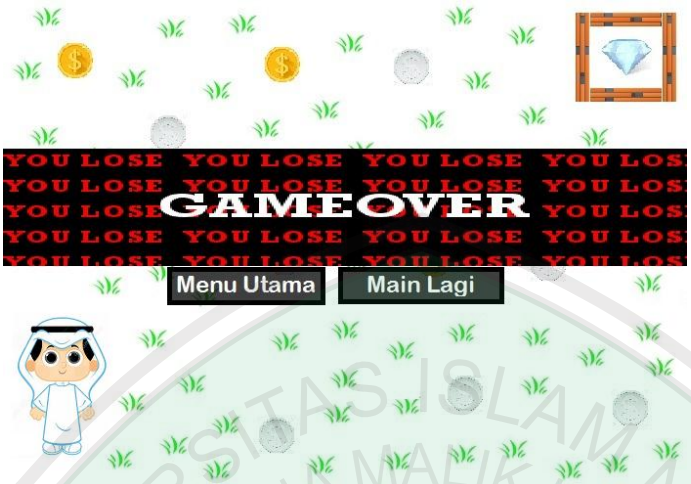
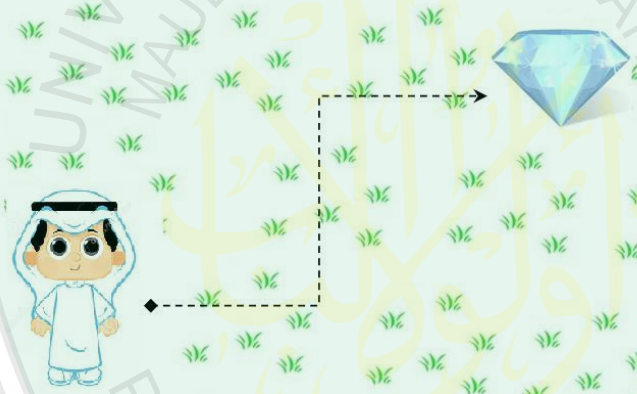
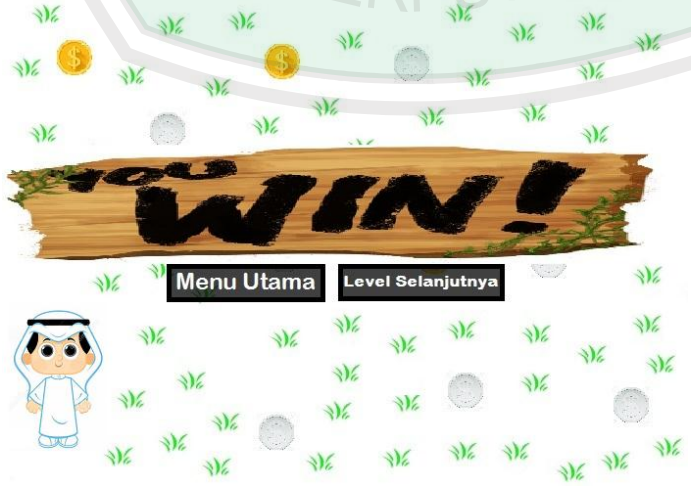
3.1.2 *Story Board Game*

Tabel 3.1 *Story Board Game*

No	Gambar	Keterangan
1		Kondisi awal ketika *game* dimulai, *player* siap untuk bermain, selain *player* di arena permainan terdapat koin emas, koin perak, berlian, NPC *Army* dan NPC *Enemy.*

2		Misi *player* adalah untuk menemukan sebuah berlian di arena permainan, tetapi berlian tersebut terhalang oleh sebuah tembok pelindung yang tidak bisa dilewati oleh *player.*
3		*Player* harus mengum pulkan koin emas dan koin perak dengan jumlah tertentu sesuai dengan level yang dimainkan untuk membuat tembok pelindung berlian hilang dan bisa mengambil berliannya.
4		Disaat *player* berhasil mendapatkan satu koin emas maka *player* mendapat 10 poin dan mendapatkan satu penge tahuan tentang nabi dan rasul, pengetahuan yang ditampilkan selalu berbeda setiap kali *player* mendapat koin emas karena telah diacak dengan algoritma *Fisher-Yates Shuffle.*

5		Disaat *player* berhasil mendapatkan satu koin perak maka *player* mendapatkan 5 poin.
6		Selama perajalan meng umpulkan koin emas, koin perak, dan berlian ada NPC *Enemy* yang ber gerak mengikuti kemana pun *player* pergi. *Player* harus berlari menghindari NPC *Enemy,* jika *player* tertabrak oleh NPC *Enemy* maka kesehatan *player* ber kurang satu poin dan NPC *Enemy* mati, tetapi masih ada NPC *Enemy* lain yang harus dihindari oleh *player.*
7		Karakter NPC yang lain yaitu NPC *Army* meru pakan satu pasukan NPC yang akan mencuri atau mengambil semua koin perak yang ada di arena permainan. Jadi *player* harus dengan segera jika ingin mengambil koin perak untuk mena mbah poinnya sebelum dihabiskan NPC *Army*. Pergerakan NPC *Army* ini merupakan imple mentasi dari algoritma *Dynamic Weighting A**.

8		Jika waktu yang dimiliki *player* habis atau kesehatan *player* habis, maka *player* dinyatakan kalah atau *game over.*
9		Misalkan untuk level satu poin yang harus dikumpukan *player* adalah 100 poin, jika *player* berhasil mendapatkannya maka tembok pelindung berlian akan runtuh dan *player* bisa untuk segera mengambil berlian tersebut.
10		Ketika *player* telah ber hasil mengambil berlian, berarti *player* telah ber hasil menyelesaikan misi pada permainan ini. *Player* dinyatakan men ang dan bisa bermain di level selanjutnya.

3.1.3 Deskrispi Karakter

Pada permainan ini terdapat satu karakter utama atau *player* yang pergerakannya sesuai perintah orang yang bermian *game* dan dua karakter NPC yaitu NPC *Enemy* dan NPC *Army*.

a. Karakter Utama

Karakter utama atau *player* adalah karakter berpakaian baju muslin bernama Adam. Merupakan karakter di dalam *game* yang dimainkan oleh *user*, pergerakannya sesuai dengan perintah *user*.

Gambar 3.1 Karakter Utama

b. Karakter NPC *Enemy*

Karakter NPC *Enemy* adalah karakter pengganggu yang terus mengejar kemanapun *player* pergi. Jika berhasil mengenai *player* maka kesehatan *player* berkurang satu poin dan NPC *Enemy* akan mati.

Gambar 3.2 Karakter NPC *Enemy*

c. Karakter NPC *Arny*

Karakter NPC *Army* adalah satu pasukan NPC yang akan mencuri atau mengambil semua koin perak yang ada di arena permainan. Sehingga *player* harus dengan segera jika ingin mengambil koin perak untuk menambah jumlah poinnya, supaya tidak terlebih dahulu dihabiskan oleh pasukan NPC *Army*. Pergerakan dari NPC *Army* ini mengimplementasikan algoritma *Dynamic Weighting A** (DWA*) untuk mencari rute terdekat ke koin perak yang menjadi tujuan dari setiap NPC *Army* dan kemuadian bergerak untuk mengambilnya sesuai dengan rute yang telah ditemukan.

Gambar 3.3 Karakter NPC *Army*

3.1.4 Deskripsi *Item*

Pada permainan ini terdapat tiga *item* utama yang harus didapatkan oleh *player* untuk bisa menyelaikan misi dalam permainan ini.

a. Koin Perak

Setiap kali mendapatkan satu koin perak pada permainan ini maka poin *player* bertambah lima poin.

Gambar 3.4 Koin Perak

b. Koin Emas

Setiap kali mendapatkan satu koin emas pada permainan ini maka poin *player* bertambah sepuluh poin dan satu pengetahuan tentang nabi dan rasul ditampilkan. Sehingga bisa menambah wawasan pengetahuan *player* tentang nabi dan rasul.

Gambar 3.5 Koin Emas

c. Berlian

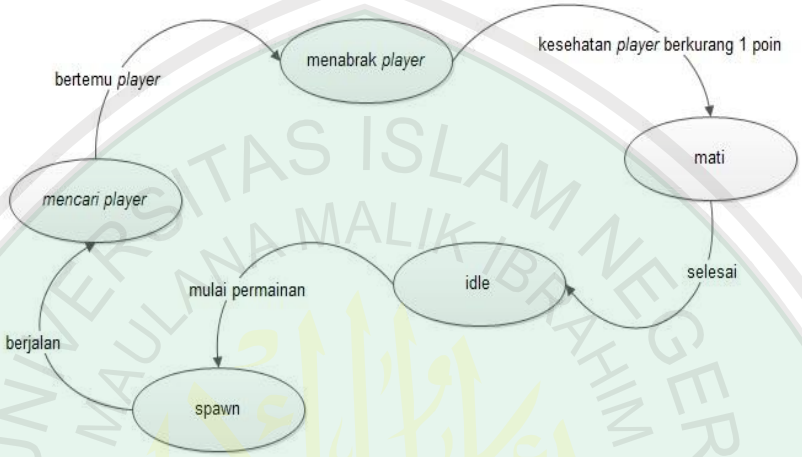
Jika poin yang dikumpulkan *player* dari hasil mengumpulkan koin emas dan koin perak teah mencukupi sesuai dengan target poin yang harus diperoleh pada level yang dimainkan, maka *player* harus segera mencari dan mengambil item berlian ini untuk bisa menyelesaikan permainan.

Gambar 3.6 Berlian

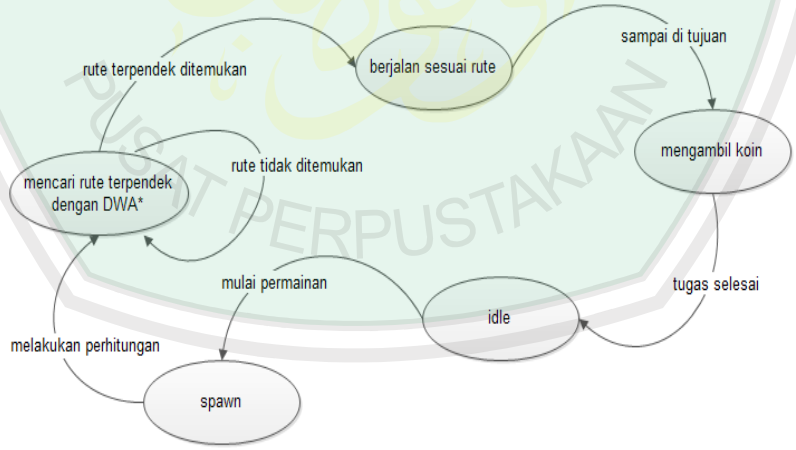
3.2 Perancangan *Finite State Machine*

Implementasi FSM di *game* ini adalah untuk mengatur perilaku NPC. Sedangkan perilaku karakter utama mengikuti perintah orang yang bermain *game*. Pada permainan ini terdapat dua jenis NPC, yaitu NPC *Enemy* dan NPC *Army*.

Jumlah NPC *Enemy* dan NPC *Army* pada *game* berbeda-beda sesuai dengan level yang dimainkan. Pergerakan NPC otomatis sesuai dengan yang diprogramkan, FSM berikut ini menggambarkan perilaku NPC *Enemy* dan NPC *Army*.



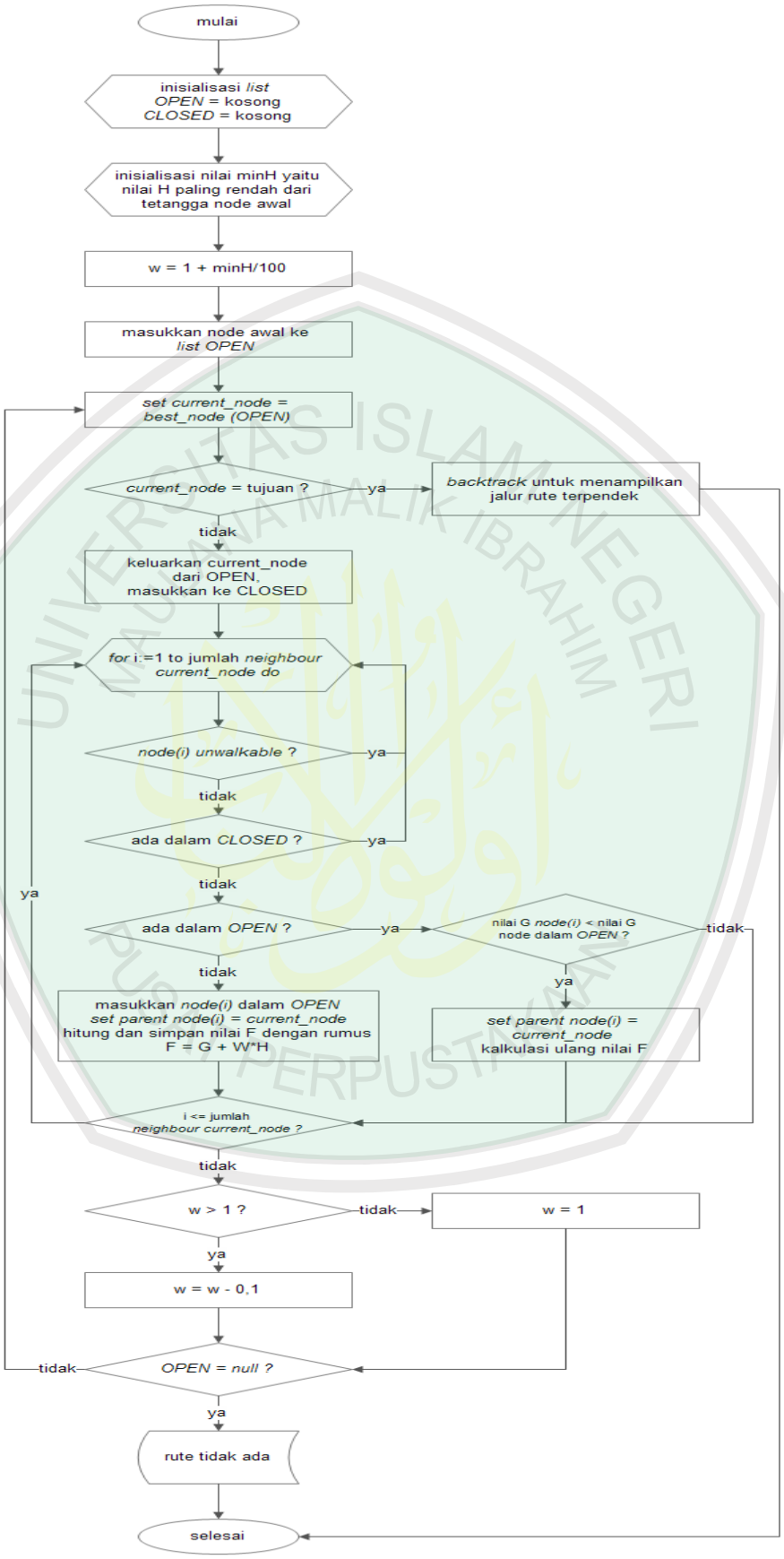
Gambar 3.7 FSM NPC *Enemy*



Gambar 3.8 FSM NPC *Army*

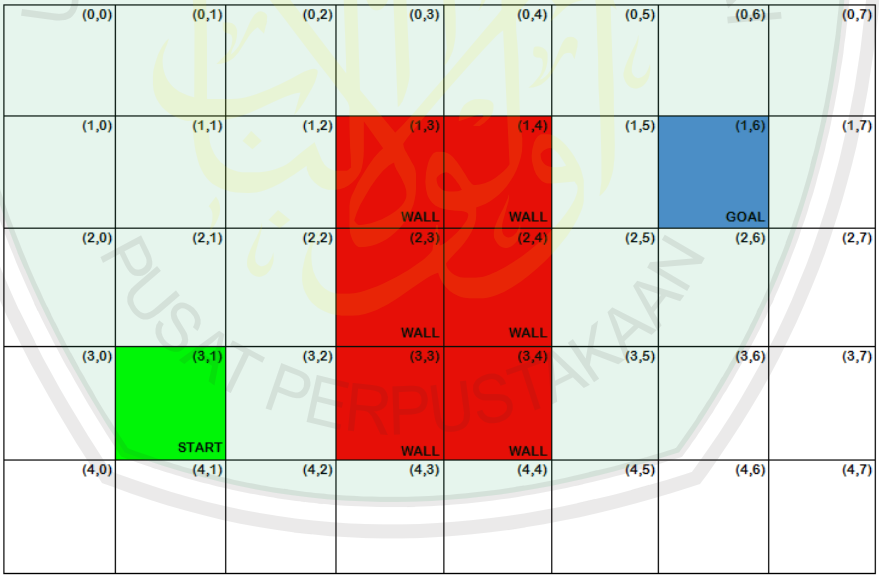
3.3 Perancangan Algoritma *Dynamic Weighting A**

Implementasi algoritma *Dynamic Weighting A** terdapat pada NPC *Army*. Pada sub bab ini terdapat *flowchart* dan penghitungan manual algoritma tersebut.



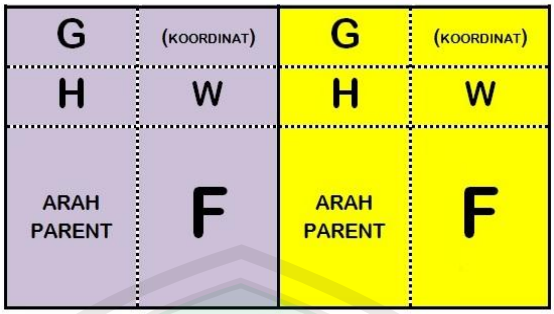
Gambar 3.9 *Flowchart* Algoritma *Dynamic Weighting A**

Gambar 3.9 merupakan langkah-langkah penyelesaian masalah pencarian rute terpendek dengan menggunakan algoritma DWA* dalam bentuk *flowchart* yang akan diimplementasikan pada aplikasi *game* di penelitian ini. Untuk lebih jelasnya tentang pengimplementaian algoritma tersebut ke dalam aplikasi *game*, pada sub bab ini akan dilakukan simulasi penghitungan manual algoritma DWA*. Berikut adalah contoh arena yang digunakan untuk simulasi perhitungan manual pencarian rute terdekat dengan DWA*. Warna hijau adalah *startingpoin/ startnode*, warna biru adalah *goal/endpoint/endnode*, dan merah adalah penghalang/*unwalkable*. Tujuan dari simulasi ini adalah mencari rute terpendek dari kotak hijau ke kotak biru tanpa melewati penghalang/kotak warna merah.



Gambar 3.10 Arena Untuk Simulasi Algoritma A* dan DWA*

Ketika penghitungan dimulai, warna ungu adalah *open list* dan warna kuning adalah *closed list*. Didalam satu kotak atau *node* ada beberapa parameter, diantaranya: G, H, W, F, koordinat *node* dan arah/*direction* yang ditunjuk merupakan *parent node* tersebut. Untuk lebih jelasnya lihat gambar dibawah ini.



Gambar 3.11 Indeks Pada Kotak (*node*) Saat Simulasi Algoritma A* dan DWA*

Skor atau biaya disetiap *node* dilambangkan dengan F. Pada algoritma DWA* nilai F dapat diperoleh dengan rumus berikut ini:

F = G + W*H .....(2)

dimana:

F = fungsi evaluasi

G = biaya sebenarnya / jarak dari *start node* (implementasi di *game*)

H = biaya perkiraan / jarak dari *end node* (implementasi di *game*)

W = bobot dinamis

Untuk nilai G diasumsikan setiap langkah dari hijau adalah legal baik vertikal, horizontal, maupun diagonal dengan catatan tidak membentur tembok. Setiap langkah yang diizinkan kita berikan nilai G dimana G adalah *cost* atau biaya dalam setiap langkah. Dalam kasus ini kita akan berikan nilai 10 untuk setiap langkah vertikal maupun horizontal, dan 14 untuk diagonal. Nilai 14 kita dapatkan dari perhitungan pitagoras dimana 14,1421 = *sqrt(sqr(10)+sqr(10)).*

Untuk nilai H adalah jumlah nilai perkiraan dari sebuah simpul ke simpul tujuan. Untuk perhitungan nilai H digunakan fungsi heuristik, metode yang digunakan di dalam contoh ini adalah metode Manhattan dimana perhitungan

jumlah *node* hanya yang bergerak secara vertical dan horizontal menuju tujuannya serta mengabaikan penghalang, yang kemudian nilainya dikalikan dengan 10. Atau: H = *10*(abs(currentX-targetX) + abs(currentY-targetY)).*

Nilai W merupakan bobot dinamis yang digunakan untuk penghitungan nilai F. Untuk menentukan bobot awal yang akan digunakan pada penghitungan algoritma ini adalah dengan mencari nilai H terkecil dari tetangga *startnode*, jika dituliskan dalam bentuk persamaan adlah sebagai berikut:

BA = 1 + minH/100 .....(3)

dimana:

BA = bobot awal

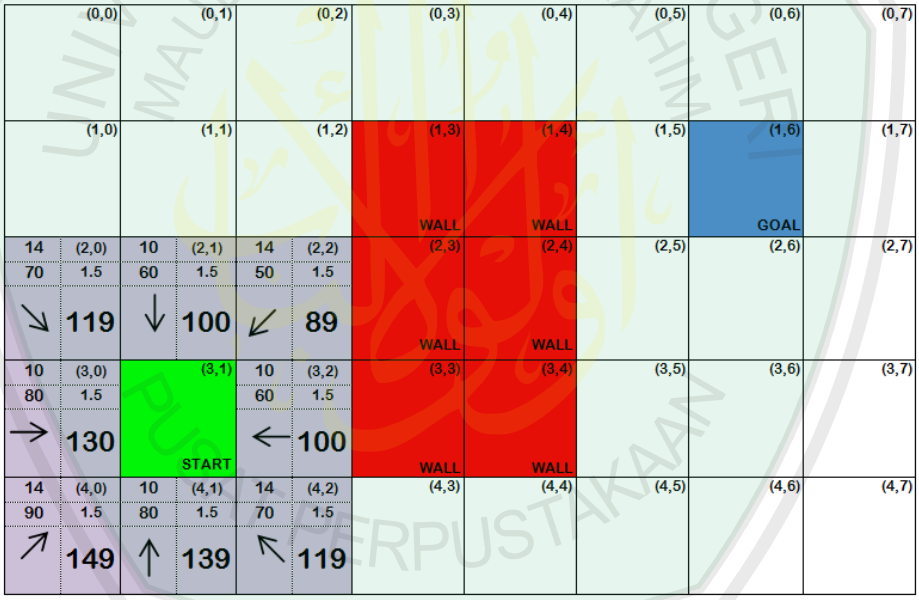
minH = nilai H terkecil dari tetangga *startnode*

Nilai yang diperoleh dari BA akan dijadikan nilai W yang pertama kali digunakan dalam penghitungan. Dan untuk selanjutnya nilai W diturunkan sebanyak 0,1 secara bertahap setiap kali menghitung nilai F dari tetangga *bestnode*. Nilai W akan terus diturunkan sampai *goal* ditemukan atau sampai dengan nilai W lebih dari sama dengan 1. Jika pada perulangan tersebut nilai W kurang dari 1, tetapi *goal* belum ditemukan, maka untuk selanjutnya nilai W tetap sama dengan 1 sampai *goal* ditemukan, karena nilai W tidak boleh kurang dari 1.

Berikut adalah simulasi penghitungan manual algoritma DWA*. Pada simulasi kali ini masalahnya adalah untuk mencari rute terpendek dari koordinat (3,1) atau kotak hijau yang merupakan *startnode* menuju koordinat (1,6) kotak biru yang merupakan *endnode/goal*. Penjelasan lengkap ada pada gambar dan keterangan langkah-langkah penghitungan manual algoritma DWA* dibawah ini.

Pertama-tama yang dicari dahulu adalah nilai bobot awal. Untuk mendapatkan nilai bobot awal, cari semua nilai H dari tetangga *startnode*. Selanjutnya tentukan nilai variabel minH, yaitu nilai H terkecil dari tetangga *startnode*. Pada gambar di bawah ini bisa dilihat bahwa nilai H terkecil dari tetangga *startnode* (minH) adalah 50. Maka bobot awalnya (BA) adalah:

BA = 1 + minH/100
= 1 + 50/100
= 1 + 0,5
= 1,5



Gambar 3.12 Simulasi Penghitungan Manual Algoritma DWA* Langkah 1

Langkah pertama, nilai *w* yang digunakan sama dengan BA yaitu 1,5. karena di *open* hanya terdapat satu *node* (yaitu *node* (3,1)), maka *node* (3,1) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Kemudian dibangkitkan semua suksesor *node* (3,1) yaitu *node*: (2,0), (2,1), (2,2), (3,0), (3,2), (4,0), (4,1) dan (4,2). Karena kedelapan suksesor tersebut belum pernah ada di *open* maupun

closed, maka semuanya dimasukkan ke *open*. Penghitung nilai F dari setiap suksesor *bestnode* adalah sebagai berikut:

Node (2,0)
F = 14 + 1.5 * 70
F = 119

Node (2,1)
F = 10 + 1.5 * 60
F = 100

Node (2,2)
F = 14 + 1.5 * 50
F = 89

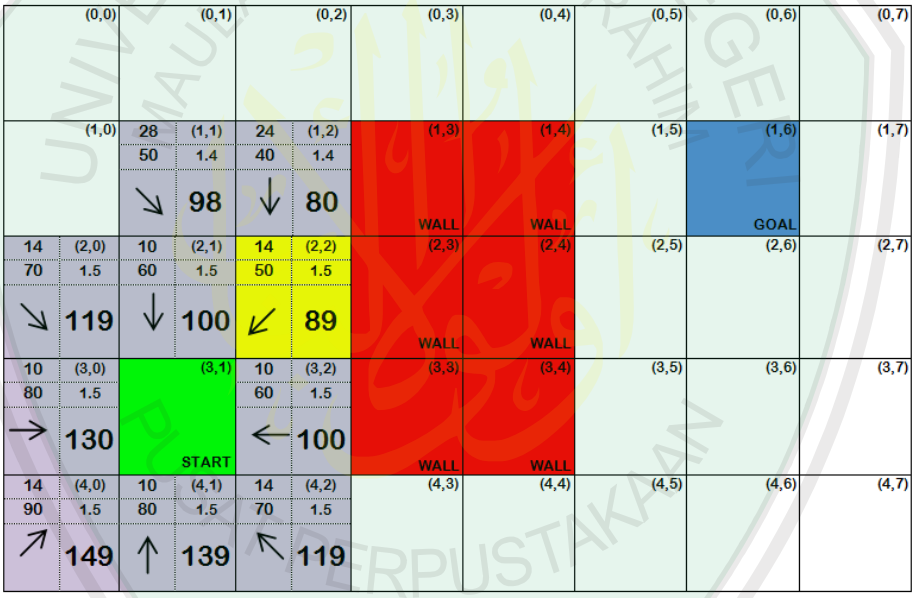
Node (3,0)
F = 10 + 1.5 * 80
F = 130

Node (3,2)
F = 10 + 1.5 * 60
F = 100

Node (4,0)
F = 14 + 1.5 * 90
F = 149

Node (4,1)
F = 10 + 1.5 * 80
F = 130

Node (4,2)
F = 14 + 1.5 * 70
F = 119



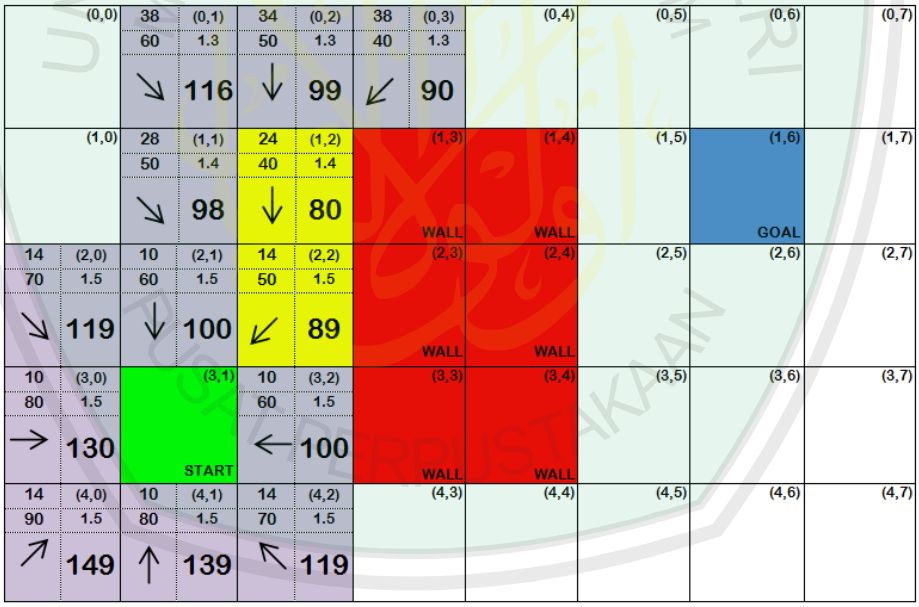
Gambar 3.13 Simulasi Penghitungan Manual Algoritma DWA* Langkah 2

Selanjutnya pada langkah kedua nilai *w* diturunkan menjadi 1,4. *Node* (2,2) dengan biaya terkecil (yaitu 89) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor *node* (2,2) dibangkitkan, yaitu *node*: (1,1), (1,2), (2,1),(3,1) dan (3,2). Karena belum pernah ada di *open* maupun *closed*, maka *node* (1,1) dan (1,2) dimasukkan ke *open*. *Node* (3,1) sudah ada di *closed* maka dibiarkan saja. Sedangkan *node* (2,1) dan (3,2) sudah ada di *open*, maka

harus dicek apakah *parent node* tersebut perlu diganti atau tidak. Biaya dari *startnode* ke *node* (2,1) yaitu 10, lebih kecil daripada biaya dari *startnode* ke *node* (2,1) melalui *node* (2,2) yaitu 24, maka *parent node* (2,1) tidak perlu diubah. Dan juga biaya dari *startnode* ke *node* (3,2) yaitu 10, lebih kecil daripada biaya dari *startnode* ke *node* (3,2) melalui *node* (2,2) yaitu 24, maka *parent node* (3,2) tidak perlu diubah. Sedangkan penghitung *node-node* yang baru dibangkitkan adalah sebagai berikut:

Node (1,1)
F = 28 + 1.4 * 50
F = 98

Node (1,2)
F = 24 + 1.4 * 40
F = 80



Gambar 3.14 Simulasi Penghitungan Manual Algoritma DWA* Langkah 3

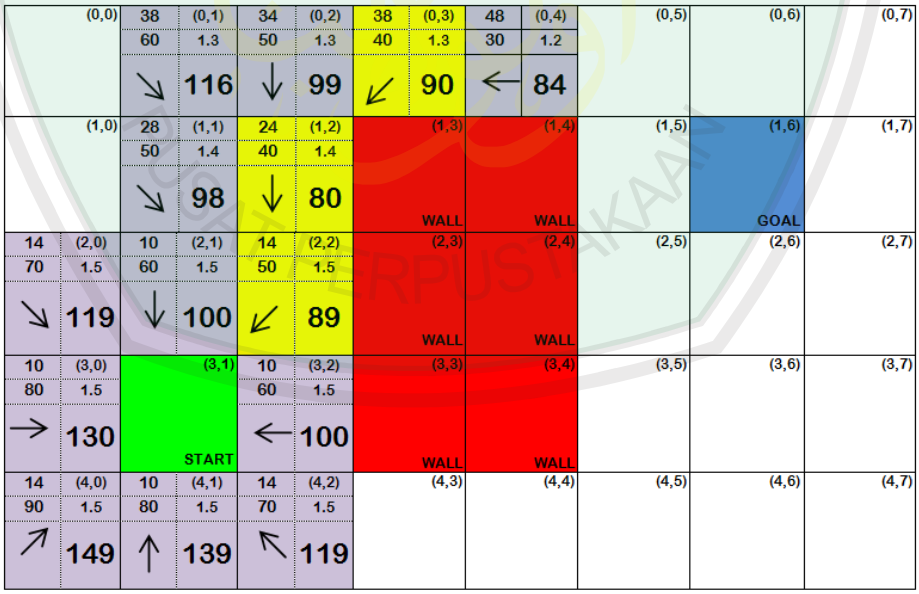
Selanjutnya pada langkah ketiga nilai *w* diturunkan menjadi 1,3. *Node* (1,2) dengan biaya terkecil (yaitu 80) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor *node* (1,2) dibangkitkan, yaitu *node*: (0,1), (0,2), (0,3), (1,1), (2,1) dan (2,2). Karena belum pernah ada di *open* maupun

closed, maka *node* (0,1), (0,2) dan (0,3) dimasukkan ke *open*. *Node* (2,2) sudah ada di *closed* maka dibiarkan saja. Sedangkan *node* (1,1) dan (2,1) sudah ada di *open*, maka harus dicek apakah *parent node* tersebut perlu diganti atau tidak. Biaya dari *startnode* ke *node* (1,1) melalui *node* (2,2) yaitu 28 lebih kecil daripada biaya dari *startnode* ke *node* (1,1) melalui *node* (1,2) yaitu 34, maka *parent node* (1,1) tidak perlu diubah. Dan juga biaya dari *startnode* ke *node* (2,1) yaitu 10 lebih kecil daripada biaya dari *startnode* ke *node* (2,1) melalui *node* (1,2) yaitu 38 maka *parent* node (2,1) tidak perlu diubah. Sedangkan penghitung *node-node* yang baru dibangkitkan adalah sebagai berikut:

Node (0,1)
F = 38 + 1.3 * 60
F = 116

Node (0,2)
F = 34 + 1.3 * 50
F = 99

Node (0,3)
F = 38 + 1.3 * 40
F = 90

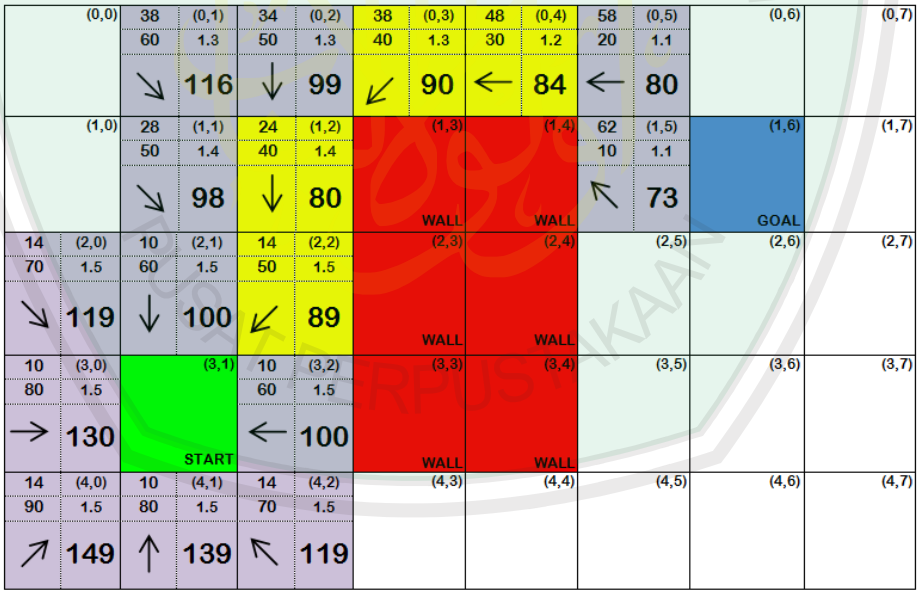


Gambar 3.15 Simulasi Penghitungan Manual Algoritma DWA* Langkah 4

Selanjutnya pada langkah keempat nilai *w* diturunkan menjadi 1,2. *Node* (0,3) dengan biaya terkecil (yaitu 90) terpilih sebagai *bestnode* dan dipindahkan

ke *closed*. Selanjutnya, semua suksesor *node* (0,3) dibangkitkan, yaitu *node*: (1,2), (0,2) dan (0,4). Karena belum pernah ada di *open* maupun *closed*, maka *node* (0,4) dimasukkan ke *open*. *Node* (1,2) sudah ada di *closed* maka dibiarkan saja. Sedangkan *node* (0,2) sudah ada di *open*, maka harus dicek apakah *parent node* tersebut perlu diganti atau tidak. Biaya dari *startnode* ke *node* (0,2) melalui *node* (1,2) yaitu 34 lebih kecil daripada biaya dari *startnode* ke *node* (0,2) melalui *node* (0,3) yaitu 48, maka *parent node* (0,2) tidak perlu diubah. Sedangkan penghitungan *node-node* yang baru dibangkitkan adalah sebagai berikut:

Node (0,4)
F = 48 + 1.2 * 30
F = 84



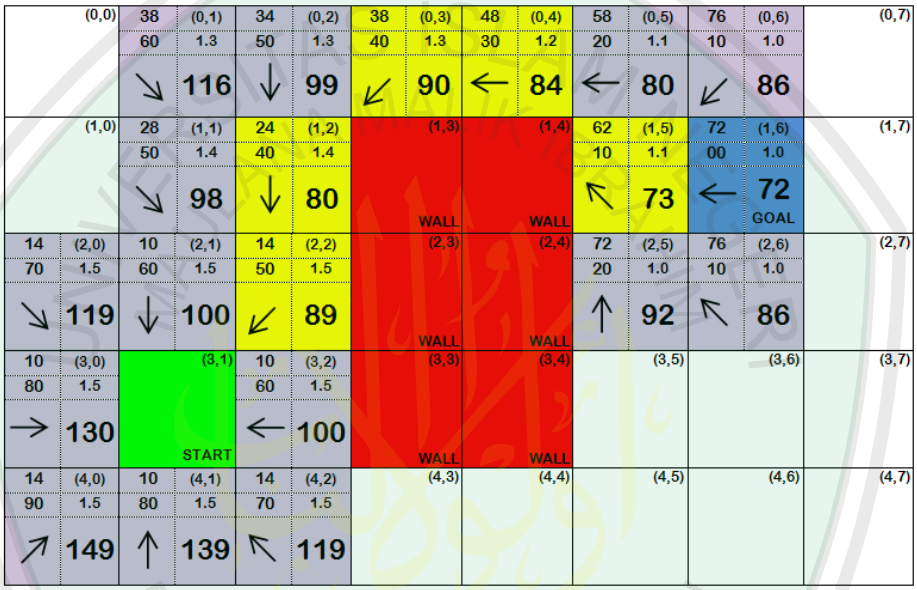
Gambar 3.16 Simulasi Penghitungan Manual Algoritma DWA* Langkah 5

Selanjutnya pada langkah kelima nilai *w* diturunkan menjadi 1,1. *Node* (0,4) dengan biaya terkecil (yaitu 84) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor *node* (0,4) dibangkitkan, yaitu *node*: (0,3),

(0,5) dan (1,5). *Node* (0,3) sudah ada di *closed* maka dibiarkan saja. Karena belum pernah ada di *open* maupun *closed*, maka *node* (0,5) dan (1,5) dimasukkan ke *open*. Penghitungan *node-node* yang baru dibangkitkan adalah sebagai berikut:

Node (0,5)
F = 58 + 1.1 * 20
F = 80

Node (1,5)
F = 62 + 1.1 * 10
F = 73



Gambar 3.17 Simulasi Penghitungan Manual Algoritma DWA* Langkah 6

Selanjutnya pada langkah keenam nilai *w* diturunkan menjadi 1,0. *Node* (1,5) dengan biaya terkecil (yaitu 73) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor *node* (1,5) dibangkitkan, yaitu *node*: (0,4), (0,5), (0,6), (1,6), (2,5) dan (2,6). Karena belum pernah ada di *open* maupun *closed*, maka *node* (0,6), (1,6), (2,5) dan (2,6) dimasukkan ke *open*. *Node* (0,4) sudah ada di *closed* maka dibiarkan saja. Sedangkan *node* (0,5) sudah ada di *open*, maka harus dicek apakah *parent node* tersebut perlu diganti atau tidak. Biaya dari *startnode* ke *node* (0,5) melalui *node* (0,4) yaitu 58 lebih kecil daripada biaya dari

startnode ke *node* (0,5) melalui *node* (1,5) yaitu 72, maka *parent node* (0,5) tidak perlu diubah. Sedangkan penghitungan *node-node* yang baru dibangkitkan adalah sebagai berikut:

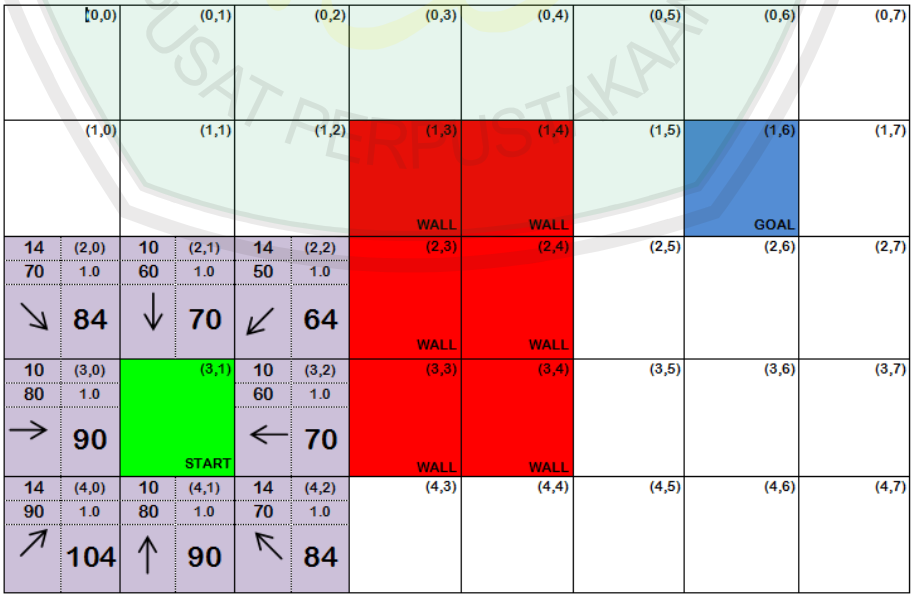
Node (0,6)
F = 76 + 1.0 * 10
F = 86

Node (2,5)
F = 72 + 1.0 * 20
F = 92

Node (2,6)
F = 76 + 1.0 * 10
F = 86

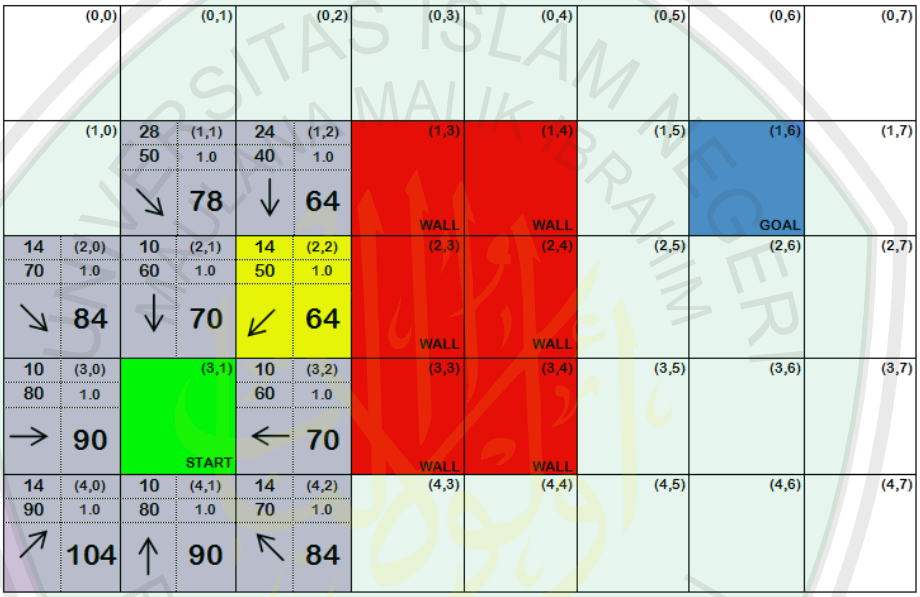
Node (1,6)
F = 72 + 1.0 * 0
F = 72

Selanjutnya *node* (1,6) dengan biaya terkecil yaitu 72 terpilih sebagai *bestnode*. Karena *bestnode* sama dengan *goal*, berarti solusi sudah ditemukan. Rute bisa ditelusuri balik dari *node* (1,6) menuju (3,1) karena setiap simpul atau *node* hanya memiliki satu *parent*. Hasil penelusuran balik menghasilkan rute (3,1)-(2,2)-(1,2)-(0,3)-(0,4)-(1,5)-(1,6). Rute ini merupakan rute terpendek yang ada di simulasi ini. Selanjutnya akan dilakukan juga penghitungan manual dengan studi kasus yang sama, tetapi penyelesaiannya menggunakan algoritma A*.



Gambar 3.18 Simulasi Penghitungan Manual Algoritma A* Langkah 1

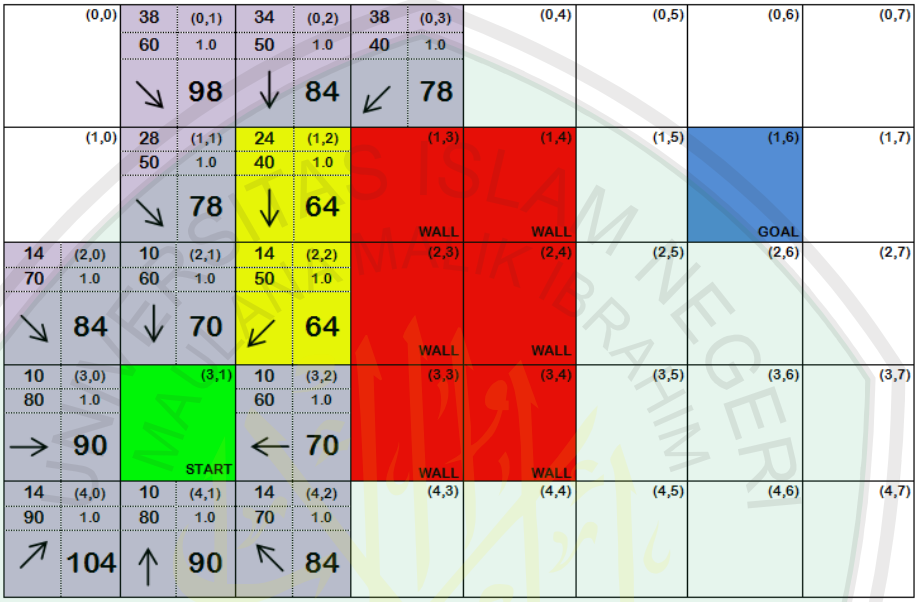
Langkah pertama, karena di *open* hanya terdapat satu *node* (yaitu *node* (3,1)), maka *node* (3,1) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Kemudian dibangkitkan semua suksesor *node* (3,1) yaitu *node*: (2,0), (2,1), (2,2), (3,0), (3,2), (4,0), (4,1) dan (4,2). Karena kedelapan suksesor tersebut belum pernah ada di *open* maupun *closed*, maka semuanya dimasukkan ke *open*.



Gambar 3.19 Simulasi Penghitungan Manual Algoritma A* Langkah 2

Langkah kedua, n*ode* (2,2) dengan biaya terkecil (yaitu 64) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor *node* (2,2) dibangkitkan, yaitu *node*: (1,1), (1,2), (2,1),(3,1) dan (3,2). Karena belum pernah ada di *open* maupun *closed*, maka *node* (1,1) dan (1,2) dimasukkan ke *open*. *Node* (3,1) sudah ada di *closed* maka dibiarkan saja. Sedangkan *node* (2,1) dan (3,2) sudah ada di *open*, maka harus dicek apakah *parent node* tersebut perlu diganti atau tidak. Biaya dari *startnode* ke *node* (2,1) yaitu 10, lebih kecil daripada biaya dari *startnode* ke *node* (2,1) melalui *node* (2,2) yaitu 24, maka

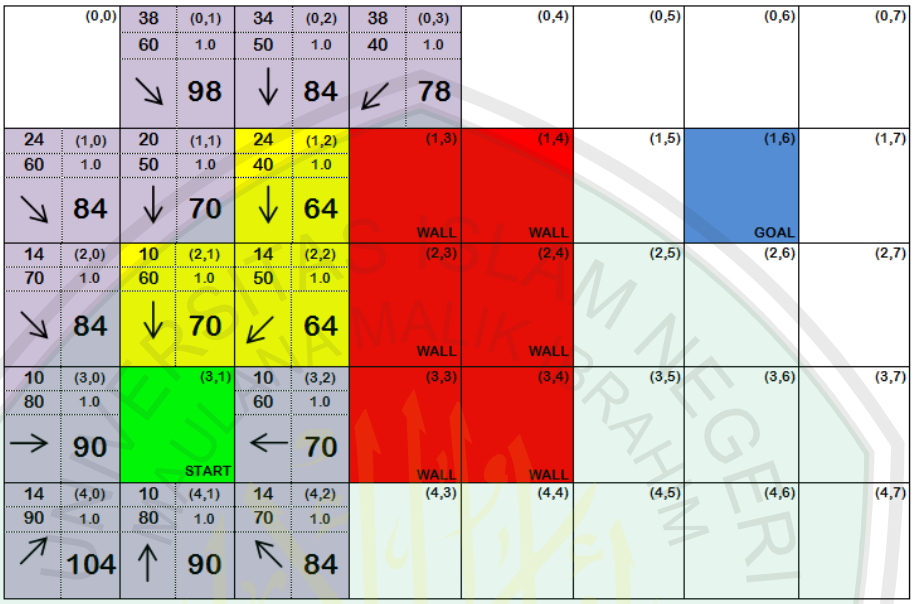
parent node (2,1) tidak perlu diubah. Dan juga biaya dari *startnode* ke *node* (3,2) yaitu 10, lebih kecil daripada biaya dari *startnode* ke *node* (3,2) melalui *node* (2,2) yaitu 24, maka *parent node* (3,2) tidak perlu diubah.



Gambar 3.20 Simulasi Penghitungan Manual Algoritma A* Langkah 3

Langkah ketiga, *node* (1,2) dengan biaya terkecil (yaitu 64) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor *node* (1,2) dibangkitkan, yaitu *node*: (0,1), (0,2), (0,3), (1,1), (2,1) dan (2,2). Karena belum pernah ada di *open* maupun *closed*, maka *node* (0,1), (0,2) dan (0,3) dimasukkan ke *open*. *Node* (2,2) sudah ada di *closed* maka dibiarkan saja. Sedangkan *node* (1,1) dan (2,1) sudah ada di *open*, maka harus dicek apakah *parent node* tersebut perlu diganti atau tidak. Biaya dari *startnode* ke *node* (1,1) melalui *node* (2,2) yaitu 28 lebih kecil daripada biaya dari *startnode* ke *node* (1,1) melalui *node* (1,2) yaitu 34, maka *parent node* (1,1) tidak perlu diubah. Dan juga biaya dari *startnode* ke *node* (2,1) yaitu 10 lebih kecil daripada biaya dari

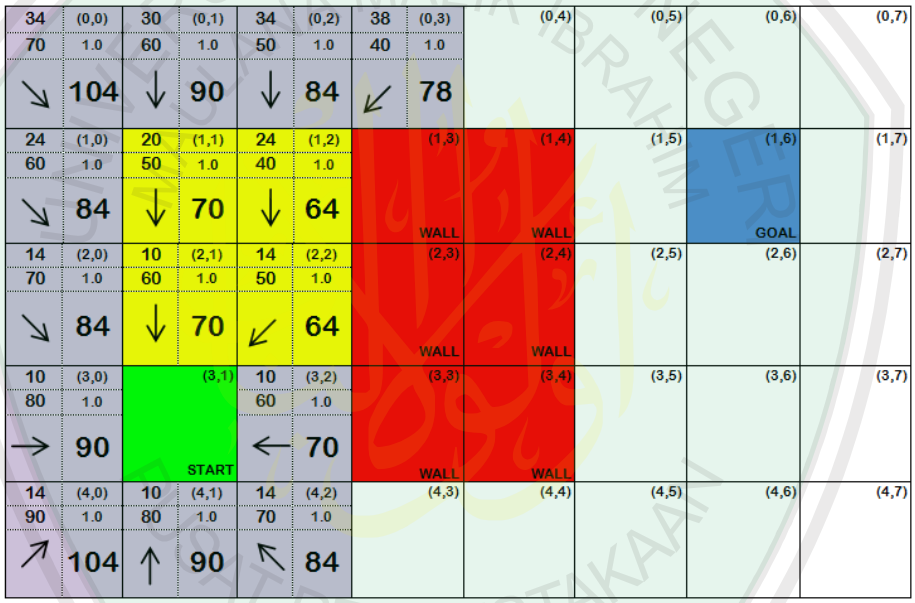
startnode ke *node* (2,1) melalui *node* (1,2) yaitu 38 maka *parent* node (2,1) tidak perlu diubah.



Gambar 3.21 Simulasi Penghitungan Manual Algoritma A* Langkah 4

Langkah keempat, *node* (2,1) dengan biaya terkecil (yaitu 70) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor *node* (2,1) dibangkitkan, yaitu *node*: (1,0),(1,1),(1,2), (2,0),(2,2), (3,0),(3,1) dan (3,2). Karena belum pernah ada di *open* maupun *closed*, maka *node* (1,0) dimasukkan ke *open*. *Node* (2,2),(1,2) dan (3,1) sudah ada di *closed* maka dibiarkan saja. Sedangkan *node* (1,1),(2,0),(3,2) dan (3,0) sudah ada di *open*, maka harus dicek apakah *parent node* tersebut perlu diganti atau tidak. Biaya dari *startnode* ke *node* (2,0) yaitu 14, lebih kecil daripada biaya dari *startnode* ke *node* (2,0) melalui *node* (2,1) yaitu 20, maka *parent node* (2,0) tidak perlu diubah. Biaya dari *startnode* ke *node* (3,0) yaitu 10, lebih kecil daripada biaya dari *startnode* ke *node* (3,0) melalui *node* (2,1) yaitu 24, maka *parent node* (3,0) tidak perlu diubah. Biaya dari

startnode ke *node* (3,2) yaitu 10, lebih kecil daripada biaya dari *startnode* ke *node* (3,2) melalui *node* (2,1) yaitu 24, maka *parent node* (3,2) tidak perlu diubah. Ternyata biaya dari *startnode* ke *node* (1,1) melalui *node* (2,1) lebih kecil daripada biaya *startnode* ke *node* (1,1) melalui *parent* lama (*node* (2,2)). Oleh karena itu, *parent* dari *node* (1,1) harus diubah, yang semula *node* (2,2) menjadi *node* (2,1). Selanjutnya, nilai G *node* (1,1) yang semula 28 diubah menjadi 20, dan nilai F *node* (1,1) dari 78 menjadi 70.



Gambar 3.22 Simulasi Penghitungan Manual Algoritma A* Langkah 5

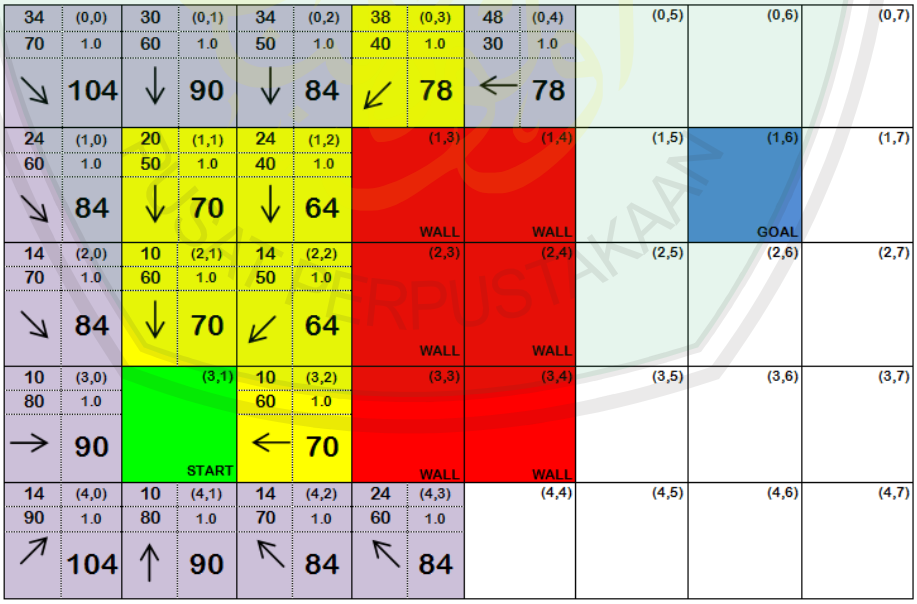
Langkah kelima, *node* (1,1) dengan biaya terkecil (yaitu 70) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor *node* (1,1) dibangkitkan, yaitu *node*: (0,0),(0,1),(0,2), (1,0),(1,2),(2,0),(2,1) dan (2,2). Karena belum pernah ada di *open* maupun *closed*, maka *node* (0,0) dimasukkan ke *open*. *Node* (1,2),(2,1) dan (2,2) sudah ada di *closed* maka dibiarkan saja. Sedangkan *node* (0,1),(0,2), (1,0) dan (2,0) sudah ada di *open*, maka harus dicek

apakah *parent node* tersebut perlu diganti atau tidak. Biaya dari *startnode* ke *node* (0,2) melalui *node* (1,2) yaitu 34, sama dengan biaya dari *startnode* ke *node* (0,2) melalui *node* (1,1) yaitu 34, maka *parent node* (0,2) tidak perlu diubah. Biaya dari *startnode* ke *node* (1,0) melalui *node* (2,1) yaitu 24, lebih kecil daripada biaya dari *startnode* ke *node* (1,0) melalui *node* (1,1) yaitu 30, maka *parent node* (1,0) tidak perlu diubah. Biaya dari *startnode* ke *node* (2,0) yaitu 14, lebih kecil daripada biaya dari *startnode* ke *node* (2,0) melalui *node* (1,1) yaitu 34, maka *parent node* (2,0) tidak perlu diubah. Ternyata biaya dari *startnode* ke *node* (0,1) melalui *node* (1,1) lebih kecil daripada biaya *startnode* ke *node* (0,1) melalui *parent* lama (*node* (1,2)). Oleh karena itu, *parent* dari *node* (0,1) harus diubah, yang semula *node* (1,2) menjadi *node* (1,1). Selanjutnya, nilai G *node* (0,1) yang semula 38 diubah menjadi 30, dan nilai F *node* (0,1) dari 98 menjadi 90.

34 (0,0) 70 1.0 ↘ 104		30 (0,1) 60 1.0 ↓ 90		34 (0,2) 50 1.0 ↓ 84		38 (0,3) 40 1.0 ↙ 78		(0,4)	(0,5)	(0,6)	(0,7)
24 (1,0) 60 1.0 ↘ 84		20 (1,1) 50 1.0 ↓ 70		24 (1,2) 40 1.0 ↓ 64		(1,3) WALL		(1,4) WALL	(1,5)	(1,6) GOAL	(1,7)
14 (2,0) 70 1.0 ↘ 84		10 (2,1) 60 1.0 ↓ 70		14 (2,2) 50 1.0 ↙ 64		(2,3) WALL		(2,4) WALL	(2,5)	(2,6)	(2,7)
10 (3,0) 80 1.0 → 90		(3,1) START		10 (3,2) 60 1.0 ← 70		(3,3) WALL		(3,4) WALL	(3,5)	(3,6)	(3,7)
14 (4,0) 90 1.0 ↗ 104		10 (4,1) 80 1.0 ↑ 90		14 (4,2) 70 1.0 ↖ 84		24 (4,3) 60 1.0 ↖ 84		(4,4)	(4,5)	(4,6)	(4,7)

Gambar 3.23 Simulasi Penghitungan Manual Algoritma A* Langkah 6

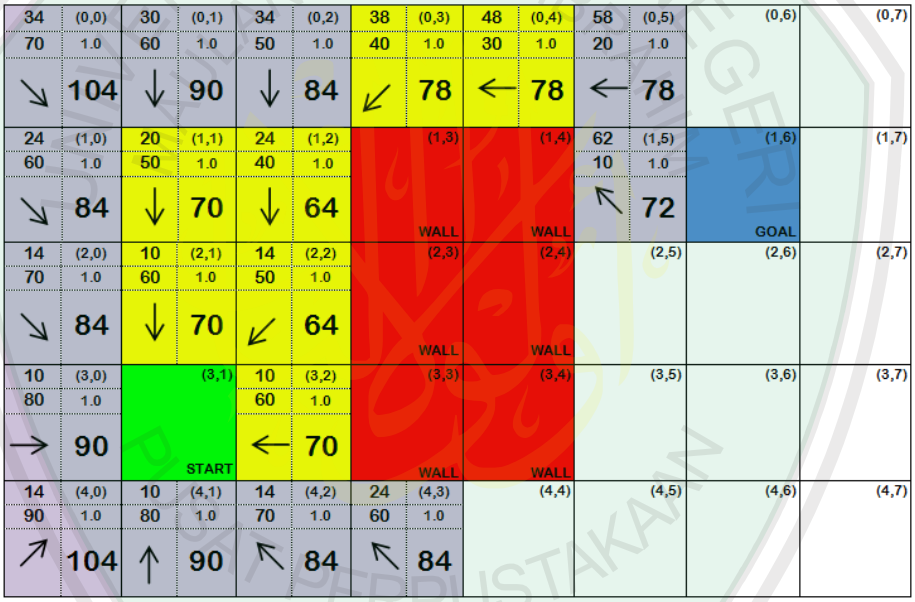
Langkah keenam, *node* (3,2) dengan biaya terkecil (yaitu 70) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor *node* (3,2) dibangkitkan, yaitu *node*: (2,1),(2,2),(3,1),(4,1),(4,2) dan (4,3). Karena belum pernah ada di *open* maupun *closed*, maka *node* (4,3) dimasukkan ke *open*. *Node* (2,1),(2,2) dan (3,1) sudah ada di *closed* maka dibiarkan saja. Sedangkan *node* (4,1) dan (4,2) sudah ada di *open*, maka harus dicek apakah *parent node* tersebut perlu diganti atau tidak. Biaya dari *startnode* ke *node* (4,1) yaitu 10, lebih kecil dari pada biaya dari *startnode* ke *node* (4,1) melalui *node* (3,2) yaitu 24, maka *parent node* (4,1) tidak perlu diubah. Biaya dari *startnode* ke *node* (4,2) yaitu 14, lebih kecil daripada biaya dari *startnode* ke *node* (4,2) melalui *node* (3,2) yaitu 24, maka *parent node* (4,2) tidak perlu diubah.



Gambar 3.24 Simulasi Penghitungan Manual Algoritma A* Langkah 7

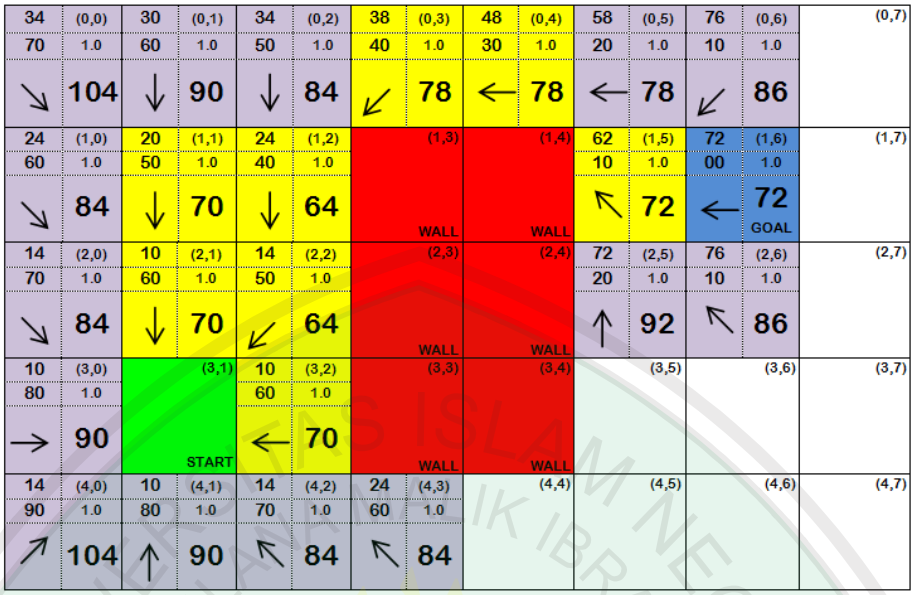
Langkah ketujuh, *node* (0,3) dengan biaya terkecil (yaitu 78) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor *node*

(0,3) dibangkitkan, yaitu *node*: (0,2),(0,4) dan (1,2). Karena belum pernah ada di *open* maupun *closed*, maka *node* (0,4) dimasukkan ke *open*. *Node* (1,2) sudah ada di *closed* maka dibiarkan saja. Sedangkan *node* (0,2) sudah ada di *open*, maka harus dicek apakah *parent node* tersebut perlu diganti atau tidak. Biaya dari *startnode* ke *node* (0,2) melalui node (1,2) yaitu 34, lebih kecil dari pada biaya dari *startnode* ke *node* (0,2) melalui *node* (0,3) yaitu 48, maka *parent node* (0,2) tidak perlu diubah.



Gambar 3.25 Simulasi Penghitungan Manual Algoritma A* Langkah 8

Langkah kedelapan, *node* (0,4) dengan biaya terkecil (yaitu 78) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor *node* (0,4) dibangkitkan, yaitu *node*: (0,3),(0,5) dan (1,5). Karena belum pernah ada di *open* maupun *closed*, maka *node* (0,5) dan (1,5) dimasukkan ke *open*. *Node* (0,3) sudah ada di *closed* maka dibiarkan saja.



Gambar 3.26 Simulasi Penghitungan Manual Algoritma A* Langkah 9

Langkah kesembilan, *node* (1,5) dengan biaya terkecil (yaitu 72) terpilih sebagai *bestnode* dan dipindahkan ke *closed*. Selanjutnya, semua suksesor *node* (1,5) dibangkitkan, yaitu *node*: (0,4), (0,5), (0,6), (1,6), (2,5) dan (2,6). Karena belum pernah ada di *open* maupun *closed*, maka *node* (0,6), (1,6), (2,5) dan (2,6) dimasukkan ke *open*. *Node* (0,4) sudah ada di *closed* maka dibiarkan saja. Sedangkan node (0,5) sudah ada di *open*, maka harus dicek apakah *parent node* tersebut perlu diganti atau tidak. Biaya dari *startnode* ke *node* (0,5) melalui *node* (0,4) yaitu 58 lebih kecil daripada biaya dari *startnode* ke *node* (0,5) melalui *node* (1,5) yaitu 72, maka *parent node* (0,5) tidak perlu diubah.

Selanjutnya *node* (1,6) dengan biaya terkecil yaitu 72 terpilih sebagai *bestnode*. Karena *bestnode* sama dengan *goal*, berarti solusi sudah ditemukan. Rute bisa ditelusuri balik dari *node* (1,6) menuju (3,1) karena setiap simpul atau *node* hanya memiliki satu *parent*. Hasil penelusuran balik menghasilkan rute

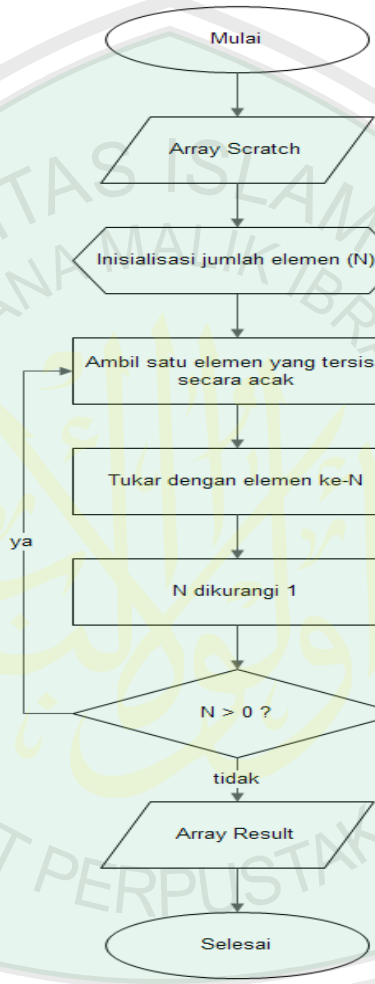
(3,1)-(2,2)-(1,2)-(0,3)-(0,4)-(1,5)-(1,6). Rute tersebut merupakan rute terpendek yang ada pada simulasi ini. Rute yang dihasilkan algoritma A* sama dengan rute yang dihasilkan algoritma DWA*. Tetapi algoritma DWA* lebih baik karena DWA* bisa menemukan *goal* dengan hanya 6 langkah sedangkan dengan kasus yang sama jika diselesaikan dengan algoritma A* membutuhkan 9 langkah untuk bisa menemukan *goal*. Dan juga jumlah *node* yang dibangkitkan oleh algoritma DWA* lebih sedikit daripada jumlah *node* yang dibangkitkan oleh algoritma A*.

3.4 Perancangan Algoritma *Fisher-Yates Shuffle*

Implementasi algortima *Fisher-Yates Shuffle* pada *game* ini digunakan untuk mengacak konten islami tentang nabi dan rasul. Setiap satu teks pengetahuan terdapat nomor urut yang kemudian nomor tersebut dimasukkan kedalam sebuah array untuk diacak urutannya menggunakan algoritma *Fisher-Yates Shuffle*. Misalkan ada 50 teks pengetahuan tentang nabi dan rasul yang masing-masing diberi nomor urut, semua nomor urut tersebut dimasukkan ke dalam sebuah array, setelah metode dijalankan hasilnya adalah array baru yang telah di acak. Sehingga urutan kelimapuluh pengetahuan tersebut berubah dan siap untuk ditampilkan. Jika dalam permainan terdapat sejumlah *n* koin emas, maka setiap kali pemain mendapatkan satu koin emas akan ditampilkan pengetahuan tentang nabi dan rasul sesuai dengan nomor yang telah diacak mulai dari urutan pertama sampai dengan *n*. Hal ini membuat pengetahuan yang ditampilkan kelihatan bervariasi dan selalu berbeda setiap kali pemain mendapatka koin emas.

Algoritma *Fisher-Yates Shuffle* dipilih karena algoritma ini merupakan metode pangacakan yang baik atau dapat dikatakan sesuai untuk pengacakan

angka, dengan waktu eksekusi yang cepat serta tidak memerlukan waktu yang lama untuk melakukan suatu pengacakan. Di bawah ini adalah *flowchart* langkah-langkah pengacakan dengan menggunakan algoritma *Fisher-Yates Shuffle*.



Gambar 3.27 *Flowchart* Algoritma *Fisher-Yates Shuffle*

Perhitungan manual implementasi algoritma *Fisher-Yates Shuffle* di *game* ini, misalnya ada delaapan teks pengetahuan yang masing-masing di beri nomor 1, 2, 3, 4, 5, 6, 7, 8. Setelah diacak menggunakan algoritma *Fisher-Yates Shuffle* urutannya berubah menjadi 6, 1, 7, 2, 8, 4, 3, 5. Jadi ketika pemain mendapatkan koin emas pertama yang ditampilkan adalah pengetahuan nomor enam dan

demikian seterusnya. Detail perhitungan dalam pengacakan ada pada Tabel 3.2 di bawah ini. *Range* adalah jumlah angka yang belum terpilih, *roll* adalah angka acak yang terpilih, *scratch* adalah daftar angka yang belum terpilih, *result* adalah hasil permutasi yang akan didapatkan.

Tabel 3.2 Simulasi Penghitungan Manual Algoritma *Fisher-Yates Shuffle*

Range	***Roll***	***Scratch***	***Result***
		1 2 3 4 5 6 7 8	
1-8	5	1 2 3 4 **8** 6 7	**5**
1-7	3	1 2 **7** 4 8 6	**3** 5
1-6	4	1 2 7 **6** 8	**4** 3 5
1-5	5	1 2 7 6	**8** 4 3 5
1-4	2	1 **6** 7	**2** 8 4 3 5
1-3	3	**1** 6	**7** 2 8 4 3 5
1-2	1	**6**	**1** 7 2 8 4 3 5
Hasil Pengacakan			**6 1 7 2 8 4 3 5**

Pengacakan suatu hal yang sangat penting dalam pembuatan banyak aplikasi, salah satunya pada aplikai *game* yang akan dibuat. Meskipun terlihat mudah, namun pada dasarnya jika tidak dilakukan dengan baik maka pengacakan itu dapat berdampak buruk untuk suatu aplikasi. Untuk itulah diperlukan sebuah algoritma yang baik terutama dalam hal pengacakan. Dalam hal ini pengacakan menggunakan algoritma *Fisher-Yates Shuffle* dapat dijadikan referensi untuk diterapkan dalam sebuah aplikasi yang menggunakan metode pengacakan. *Fisher-Yates Shuffle* merupakan cara yang optimal dengan waktu eksekusi yang efisien, serta dengan ruang penyimpanan memori yang tidak terlalu besar.

BAB IV

HASIL DAN PEMBAHASAN

4.1 Implementasi

Bab ini membahas mengenai implementasi dari perencanaan yang telah diajukan. Selain itu pada bab ini dilakukan pengujian terhadap aplikasi *game* untuk mengetahui apakah aplikasi *game* tersebut telah berjalan sesuai dengan tujuan penelitian yang ingin dicapai.

4.1.1 Kebutuhan Perangkat Keras

Perangkat keras yang digunakan untuk menguji perangkat lunak dari aplikasi *game* ini adalah sebagai berikut:

Tabel 4.1 Kebutuhan Perangkat Keras

No	Perangkat Keras	Spesifikasi
1	*Processor*	*Intel® Core™ i5 CPU U 470 @1,33GHz (4 CPUs)*
2	RAM	4096 MB
3	VGA	*Intel® HD Graphics*
4	HDD	320 GB
5	*Monitor*	11,6”
6	*Speaker*	*On*
7	*Mouse & Keyboard*	*On*

4.1.2 Kebutuhan Perangkat Lunak

Perangkat keras yang diperlukan untuk mengimplementasikan perangkat lunak dari aplikasi *game* ini adalah sebagai berikut:

Tabel 4.2 Kebutuhan Perangkat Lunak

No	Perangkat Lunak	Spesifikasi
1	Sistem Operasi	*Windows 7 64 Bit*
2	*Game Engine*	*Unity 5.2.0f3*
3	*Image Editor*	*Adobe Photoshop CS3*
4	Desain 3D	*Blender 2.7*
5	*Script Writer*	*Mono Develop*

4.1.3 Implementasi Algoritma *Dynamic Weighting A**

Implementasi kecerdasan buatan pada penelitian ini diterapkan pada NPC *Army* dengan menggunakan algoritma DWA* untuk mencari rute terpendek dari posisi NPC ke tujuan / koin perak dan kemudian bergerak untuk mengambil semua koin perak di arena permainan. Algoritma DWA* ini diimplementasikan menggunakan bahasa C#, *method*/fungsi yang digunakan adalah sebagai berikut:

Tabel 4.3 Keterangan Implementasi Algoritma *Dynamic Wieghting A**

No	***Method* / Fungsi**	**Keterangan**
1	`Grid grid; private bool cek; PathRequestManager request Manager; private double minH,bbawal,bbawalnew,bbdinamis,bbdinamisnew,angka;`	Deklarasi variabel
2	`void Awake() { requestManager= GetComponent <PathRequestManager>(); grid = GetComponent<Grid>();}`	Memanggil komponen yang dibutuhkan dalam penghitungan algoritma DWA*
3	`void Start(){ angka = 0; cek = false;}`	Nilai pada variabel pada saat permainan dimulai

4	`IEnumerator iWeight(Vector3 aa, Vector3 bb) {`	Mulai menghitung bobot awal yang akan digunakan
	`Node sNode = grid.NodeFromWorldPoint (aa);` `Node tNode = grid.NodeFromWorldPoint (bb);`	Mengambil koordinat *node* awal dan koordinat *node* tujuan, kemudian memasukkan ke variabel *sNode* dan *tNode*
	`if (sNode.walkable && tNode.walkable) {`	Cek apakah lokasi awal dan lokasi tujuan dapat dilalui
	`List<Node> opens = new List<Node> ();`	Membuat variabel untuk menampung *open node*
	`foreach (Node tetangga in grid.GetNeighbours(sNode)) { tetangga.hCost = GetDistance (tetangga, tNode); opens.Add (tetangga);}`	Menghitung nilai H setiap tetangga dari *startnode* dan memasukkannya ke *open node*
	`if (opens.Count > 0) {` `Node cNode = opens [0];`	Cek *open node* sudah terisi, kemudian menetapkan nilai *cNode* adalah *node open* indeks pertama
	`for (int i = 1; i < opens.Count; i ++) { if (opens [i].hCost < cNode.hCost) { cNode = opens [i];}} minH = cNode.hCost;`	Melakukan perulangan untuk mencari nilai H terkecil dari tetangga *startnode,* dan setelah ketemu simpan di variabel minH

	`bbawal = 1 + minH/100;`
	`bbawalnew = bbawal + 0.10;`
	`cek = true;}`
		Mengubah minH menjadi 1.xx untuk dijadikan bobot awal, kemudian mengubah nilai variabel cek menjadi *true* yang menunjukkan kalu bobot awal sudah ditemukan

	`}yield return null;}`	Penutup *method iWeight* untuk mencari bobot awal
5	`IEnumerator FindPath(Vector3 startPos, Vector3 targetPos) {`	Mulai pencarian rute terpendek dengan input lokasi awal dan lokasi tujuan

	`Vector3[] waypoints = new Vector3[0];`
	
	`bool pathSuccess = false;`
		Membuat variabel untuk menyimpan rute dan variabel untuk memberikan status apakah rute rudah ditemukan

	`Node startNode = grid.NodeFromWorldPoint(startPos);`
	
	`Node targetNode = grid.NodeFromWorldPoint(targetPos);`
		Mengambil lokasi awal dan lokasi target/tujuan, kemudian memasukkan ke variabel *startNode* dan *targetNode*

	`if (startNode.walkable && targetNode.walkable) {`	Cek apakah lokasi awal dan lokasi tujuan dapat dilalui

	`List<Node> openSet = new List<Node>();`
	
	`HashSet<Node> closedSet = new HashSet<Node>();`
		Membuat variabel *openSet* untuk menampung *open node* dan *closedSet* untuk menampung *closed node*

	`openSet.Add(startNode);`	Pertama-tama memasukkan *start node* ke *open*
	`while (openSet.Count > 0) {`	Mulai perulangan untuk mencari rute terpendek, perulangan akan berhenti jika rute sudah ditemukan atau sampai tidak ada *node* di dalam *open*

	`Node currentNode = openSet[0];`
	
	`for (int i = 1; i < openSet.Count; i ++) {`
	
	`if (openSet[i].fCost < currentNode.fCost || openSet[i].fCost == currentNode.fCost && openSet[i].hCost < currentNode.hCost) {`
	
	`currentNode = openSet[i];`
	
	`}}`
		Membuat variabel *currentNode*, merupakan *node* yang berada di *open* dengan nilai F paling kecil. Perulangan disamping adalah untuk mendapatkan *node* yang akan dijadikan sebagai *bestNode* atau yang akan dimasukkan ke variabel *currentNode* dari *node-node* yang sudah ada di dalam *open/openSet*

	`openSet.Remove(currentNode);`
	
	`closedSet.Add(currentNode);`
		Memindahkan *node* yang terpilih sebagai *bestNode* dari *open* ke *closed*

	`angka = angka + 0.10;`
	
	`bbdinamis = bbawalnew - angka;`
	
	`if(bbdinamis > 1){`
	
	`bbdinamisnew = bbdinamis;`
	
	`} else {`
	`bbdinamisnew = 1;`
		Setelah *current node* dimasukkan ke *closed*, untuk penghitungan berikutnya nilai *w* (bobot dinamis) diturunkan 0,1
		Jika bobot baru lebih dari 1 akan digunakan untuk menghitung nilai F, jika tidak maka bobot adalah 1 sampai *goal* ditemukan

	`if (currentNode == targetNode) {`
	`pathSuccess = true;`
	`break;`
	`}`
		Jika *currentnode* adalah *node* tujuan maka rute telah ditemukan, hentikan perulangan dan ubah nilai variabel *pathSuccess* manjadi *true*

	`foreach (Node neighbour in grid.GetNeighbours(currentNode)) {`
	
	`if (!neighbour.walkable || closedSet.Contains(neighbour)) {`
	`continue;`
	`}`
		Lakukan penghitungan nilai F untuk setiap tetangga dari *currentnode*, jika tetangga tidak dapat dilalui atau node tetangga sudah ada di *closed* maka abaikan dan lanjutkan ke tetangga berikutnya

	`double newMovementCostToNeighbour = currentNode.gCost + GetDistance(currentNode, neighbour);`	Menghitung nilai G baru dari tetangga *current node*

	`if (newMovementCostToNeighbour < neighbour.gCost || !openSet.Contains(neighbour){`
	`neighbour.gCost = newMovementCostToNeighbour;`
	`neighbour.hCost = GetDistance(neighbour,`
		Jika *node* tetangga belum ada di *open* maka hitung nilai G, H, W, dan F kemudian *set parent node* tetangga adalah *currentnode*

	targetNode); neighbour.wCost = bbdinamisnew; neighbour.parent = currentNode;	
	if (!openSet.Contains(neighbour)) openSet.Add(neighbour);	Jika *node* tetangga belum ada di *open* maka masukkan *node* tetangga ke dalam *open*
	}}}} yield return null; if (pathSuccess) { waypoints = RetracePath(startNode,targetN ode);}	Jika *pathSuccess* bernilai *true* yang artinya rute sudah ditemukan, maka telusuri kembali rute dari lokasi awal sampai lokasi tujuan dan simpan ke dalam variabel *waypoints*

6	public double fCost {
	
	get {
	
	return gCost + hCost*wCost;
	
	}}
		Penghitungan nilai F pada algoritma DWA* yaitu
		F = G + W*H

7	public void
	StartFindPath(Vector3
	startPos, Vector3 targetPos)
	{
	StartCoroutine(iWeight(startP
	os,targetPos));
	if(cek){
	StartCoroutine(FindPath(start
	Pos,targetPos));
	}
	}
		Method yang digunakan untuk memulai peghitungan, pertama-tama yang dijalankan adalah method *iWeight* untuk menghitung bobot awal, jika bobot awal sudah ditemukan maka nilai variabel cek adalah true, jika nilai cek *true* maka mulai jalankan *method* untuk mencari rute terpendek.

8	IEnumerator FollowPath() { Vector3 direction = target.position - transform.position;	*Method* yang digunakan untuk menjalankan objek, di dalam *game* ini adalah NPC, pertama-tama NPC akan dibuat menghadap ke target, supaya

	`transform.rotation = Quaternion.LookRotation (direction);`	jalannya nanti menghadap ke depan atau menghadap ke target

	`Vector3 currentWaypoint = path[ 0];`
	
	`while (true) {`
	
	`if (transform.position == currentWaypoint) { targetIndex ++; if (targetIndex>=path.Length) {yield break;} currentWaypoint = path[targetIndex];}`
	
	`transform.position = Vector3.MoveTowards(transform .position,currentWaypoint,spe ed * Time.deltaTime);`
	
	`yield return null;}}`
		Fungsi untuk membuat NPC berjalan sesuai dengan rute yang telah ditemukan menggunakan algoritma DWA*

9	`void Start() {`
	
	`PathRequestManager.RequestPat h(transform.position,target.p osition, OnPathFound);`
	
	`}`
		Ketika permainan dimulai terlebih dahulu mengambil lokasi awal, dalam *game* ini NPC *Army* dan lokasi tujuan

10	`public void OnPathFound(Vector3[] newPath, bool pathSuccessful) {if (pathSuccessful) {`
	
	`path = newPath;`
	
	`StopCoroutine("FollowPath");`
	
	`StartCoroutine("FollowPath"); }`
	
	`}`
		Method yang digunakan untuk menentukan kapan NPC akan dijalankan, yaitu jika *pathSuccessful* bernilai *true* atau rute terpendek telah ditemukan maka mulai jalankan *method* untuk menjalankan NPC *Army*

4.1.4 Implementasi Algoritma *Fisher-Yates Shuffle*

Implementasi algoritma *Fisher-Yates Shuffle* pada aplikasi *game* ini adalah untuk mengacak urutan konten islami tentang nabi dan rasul yang disertakan di dalam aplikasi *game*. Sehingga konten islami yang ditampilkan akan selalu berbeda setiap kali *player* mendapatkan satu koin emas. Algoritma *Fisher-Yates Shuffle* dalam aplikasi *game* ini akan diimplementasikan menggunakan bahasa C#, *method*/fungsi yang digunakan adalah sebagai berikut:

Tabel 4.4 Keterangan Implementasi Algoritma *Fisher-Yates Shuffle*

No	*Method* / Fungsi	Keterangan
1	`private int[] a = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,30,41,42,43,44,45,46,47,48,49,50};`	Variabel nomor urut konten yang akan diacak
2	`private int a1,a2,a3,a4,a5, a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,a18,a19,a20,a21,a22,a23, a24,a25,a26,a27,a28,a29,a30,a31,a32,a33,a34,a35,a36,a37,a38,a39,a40, a41,a42,a43,a44,a45,a46,a47,a48,a49,a50, x;` `private string y;`	Variabel untuk menyimpan urutan array baru yang sudah diacak

4	`void Start () {`
	`Shuffle (a);`
	`}`
		Method untuk mengacak array langsung dijalankan ketika permainan dimulai

5	`void Update (){`
	`hitung_emas = ItemsManager.hitung_emasx + 1;`
	`}`
		Method untuk melakukan *update* jumlah koin emas yang di dapat oleh *player* selama bermain

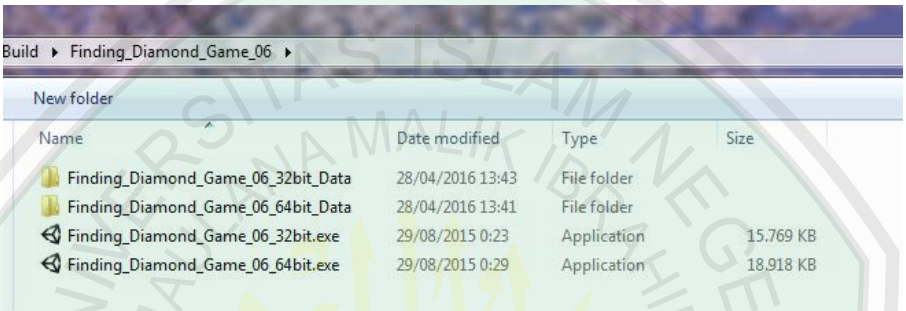
6	`void Shuffle(int[] a)`
	`{`
	`for (int i = a.Length-1; i > 0; i--){`
	`int rnd = Random.Range(0,i);`
	`int temp = a[i];`
	`a[i] = a[rnd];`
	`a[rnd] = temp;`
	`}`
		Mulai pengacakan array menggunakan algoritma *Fisher-Yates Shuffle*

	`a1=a[0]; a2=a[1]; a3=a[2];`
	`a4=a[3]; a5=a[4]; a6=a[5];`
	`a7=a[6]; a8=a[7]; a9=a[8];`
	`. . .`
	`a48=a[47]; a49=a[48]; a50=a[49];}`
		Menyimpan hasil array yang telah diacak ke variabel baru

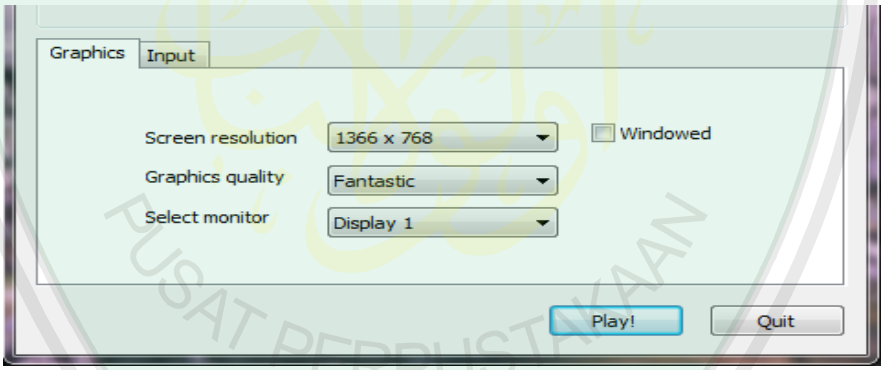
7	`public void Konten()`
	`{`
	`if (hitung_emas==1){x=a1;`
	`} else if (hitung_emas==2){x=a2;`
	`} else if (hitung_emas==3){x=a3;`
	`. . .`
	`} else if (hitung_emas==48){x=a48;`
	`} else if (hitung_emas==49){x=a49;`
	`} else {x=a50;}`
	`if (x==1){y="Nabi Adam...";`
	`} else if (x==2){y="Nabi ...";`
	`} else if (x==3){y="Nabi Idris ...";`
	`. . .`
	`} else if (x==48){y="Nabi Isa ...";`
	`} else if (x==49){y="Nabi Muhammad SAW ...";`
	`} else { y="Nabi Muhammad SAW ...";`
	`}`
		Method untuk menam pilkan konten islami tentang nabi dan rasul sesuai dengan urutan baru atau berdasarkan array baru yang telah diacak, setiap nomor urut tersebut memiliki isi konten islami yang berbeda-beda, *method* ini akan mengubah nomor urut tersebut menjadi konten islami yang akan ditampilkan.

4.1.5 Implementasi Aplikasi *Game*

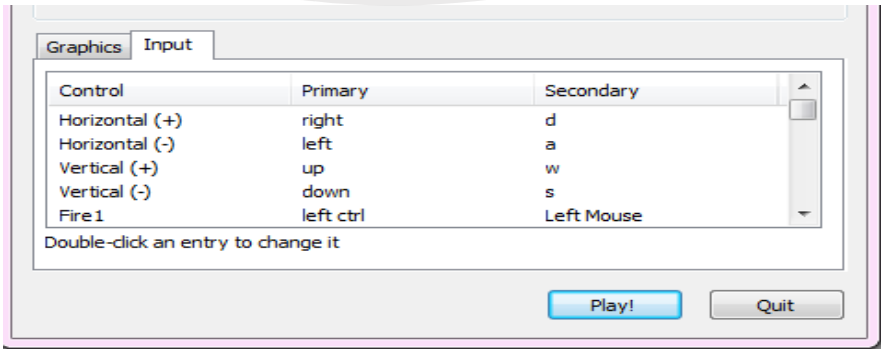
Aplikasi *game* yang telah selesai dibuat dengan *Unity* kemudian dibuat menjadi *stand alone*. *Stand alone* akan membuat aplikasi *game* dapat dijalankan tanpa harus menggunakan *Unity Game Engine*. Berikut ini adalah file *game* di *Unity* yang telah berhasil di *build* menjadi *.exe* versi *32bit* dan *64bit.*



Gambar 4.1 Hasil *Build* Aplikasi *Game Finding Diamond*



Gambar 4.2 Pengaturan *Graphics* Pada Aplikasi *Game Finding Diamond*



Gambar 4.3 Pengaturan *Input* Pada Aplikasi *Game Finding Diamond*

Jika file *.exe* pada Gambar 4.1 dijalankan, baik yang *32bit* atau *64bit* sesuai dengan komputer yang digunakan, muncul halaman pengaturan grafis dan pengaturan input yang akan digunakan ketika menjalankan aplikasi *game*. Setelah selesai dengan pengaturan tersebut, klik *Play!* untuk masuk ke aplikasi *game*.

Gambar 4.4 *Unity Splash Screen*

Gambar 4.4 adalah tampilan *unity splash screen* yang menunjukkan bahwa aplikasi *game* ini dibuat dengan menggunkan *game engine unity*. *Unity* yanng digunakan untuk membuat aplikasi *game* ini adalah *Unity 5.2*.

Gambar 4.5 Menu Utama Aplikasi *Game Finding Diamond*

Gambar 4.5 adalah tampilan menu utama, terdapat dua tombol yaitu *play game* untuk memulai permainan dan *quit game* untuk menutup aplikasi *game*.

Gambar 4.6 Mulai Bermain *Game Finding Diamond*

Gambar 4.6 adalah tampilan ketika permainan dimulai. Di pojok kiri atas ada informasi yang harus diketahui *player*. Di pojok kanan atas ada tampilan *game top view*. Di pojok kanan bawah terdapat tombol *Exit*. Di bagian bawah terdapat tulisan yang akan menampilkan konten islami tentang nabi dan rasul.

Gambar 4.7 *Player* Mendapatkan Koin Perak

Gambar 4.7 adalah tampilan ketika *player* berhasil mengambil satu koin perak. Skor *player* bertambah lima poin, koleksi koin perak yang jadi milik *player* bertambah satu, dan persediaan koin perak berkurang satu.

Gambar 4.8 *Player* Mendapatkan Koin Emas

Gambar 4.8 adalah tampilan ketika *player* berhasil mengambil satu koin emas. Skor *player* bertambah sepuluh poin, koleksi koin emas bertambah satu, dan persediaan koin emas berkurang satu. Dan setiap kali *player* mengambil koin emas, di bagian bawah *screen game* akan ditampilkan konten islami tentang nabi dan rasul berupa teks dan juga audio yang akan membacakan isi dari teks tersebut.

Gambar 4.9 Skor Telah Mencapai Target

Gambar 4.9 adalah tampilan ketika poin yang dikumpulkan *player* telah mencapai target. Pada permainan ini target poin adalah 100 poin, contoh gambar

diatas poin *player* adalah 105. Maka akan muncul peringatan supaya *player* segera mencari dan mengambil berlian untuk bisa menyelesaikan permainan.

Gambar 4.10 *Player* Berhasil Menyelesaikan Misimya

Gambar 4.10 adalah tampilan ketika *player* telah mengambil berlian yang artinya *player* berhasil menyelesaikan misi pada permainan ini dan *player* dinyatakan menang, setelah itu akan ditampilakan halaman menu utama.

Gambar 4.11 NPC *Army* Mengambil Koin Perak

Gambar 4.11 adalah tampilan ketika salah satu NPC *Army* akan mengambil koin perak. Selain *player*, pada permainan ini ada NPC *Army* yang

bisa mengambil semua koin perak yang ada di arena permainan. Jadi *player* harus dengan segera untuk mengambil koin perak supaya tidak kehabisan. Sedangkan unutk koin emas tidak diambil oleh NPC *Army*, supaya *player* lebih banyak belajar tentang nabi dan rasul setiap kali mendapatkan satu koin emas. NPC *Army* ini merupakan sebuah pasukan NPC, pada aplikasi *game* yang dibuat, jumlah pasukan mereka ada 20 NPC. Dibagian selanjutnya akan dilakukan uji coba menentukan kecepatan NPC ini, supaya tidak terlalu cepat atau terlalu lambat untuk menyelesaikan misinya, yaitu untuk menghabiskan koin perak.

Gambar 4.12 NPC *Enemy* Mengejar *Player*

Gambar 4.12 adalah tampilan ketika NPC *Enemy* mengejar *player*, jika *player* bertabrakan dengan NPC *Enemy* maka NPC *Enemy* akan mati dan kesehatan *player* berkurang sepuluh poin. *Player* bisa terus berlari untuk menghindari NPC *Enemy*, karena kecepatan berlari *player* lebih besar daripada kecepatan NPC *Enemy* dalam mengejar *player*. Tetapi jika *player* diam atau sering berhenti selama permainan, maka peluang NPC *Enemy* untuk bisa menabrak *player* semakin besar.

Gambar 4.13 Kesehatan *Player* Habis

Gambar 4.13 adalah tampilan ketika semua NPC *Enemy* telah menabrak *player* sehingga membuat kesehatan *player* habis. Maka dalam hal ini *player* gagal menyelesaikan misi pada permainan ini dan *player* dinyatakan kalah.

Gambar 4.14 Waktu *Player* Habis

Gambar 4.14 adalah tampilan ketika waktu yang dimiliki *player* untuk menyelesaikan permainan telah habis. Maka dalam hal ini *player* gagal menyelesaikan misi pada permainan ini dan *player* dinyatakan kalah.

Gambar 4.15 *Game Over Screen*

Gambar 4.15 adalah tampilan halaman *game over*, halaman ini muncul setelah *player* dinyatakan kalah karena kesehatan habis atau karena waktu habis. Halaman *game over* ini akan ditampilkan selama lima detik, setelah itu akan ditampilkan kembali halaman menu utama supaya bisa mulai permainan baru.

4.2 Uji Coba

Pada subbab ini membahas tentang uji coba yang telah dilakukan terhadap aplikasi *game* yang telah dibuat. Terdapat tiga uji coba yakni uji coba imlementasi algoritma *Dynamic Weighting A**, uiji coba algoritma *Fisher-Yates Shuffle* dan uji coba aplikasi *game*. Berikut ini pembahasan uji coba tersebut.

4.2.1 Uji Coba Algoritma *Dynamic Weighting A**

Uji coba algoritma *Dynamic Weighting A** dilakukan untuk melihat hasil dari implementasi algoritma tersebut ke dalam *game* yang telah dibuat menggunakan *Unity*. Gambar 4.16 adalah tampilan *tab console* pada *Unity* yang menampilkan hasil-hasil dari penghitungan algoritma DWA* dan pada Gambar

4.18 adalah tampilan *tab scene* pada *Unity* yang menampilkan *grid*, setiap kotaknya merupkan *node*, digunakan untuk penghitungan algortima DWA*. Ditampilkan juga rute terpendek yang berhasil ditemukan algoritma tersebut.

Node(60,76), Parent=Node(60,75), G=38, H=102, W=2, F=242
UnityEngine.Debug:Log(Object)

Node(61,74), Parent=Node(60,75), G=42, H=100, W=2, F=242
UnityEngine.Debug:Log(Object)

Node(61,75), Parent=Node(60,75), G=38, H=96, W=2, F=230
UnityEngine.Debug:Log(Object)

Node(61,76), Parent=Node(60,75), G=42, H=92, W=2, F=226
UnityEngine.Debug:Log(Object)

[4] BESTNODE YANG DIPINDAH KE CLOSED ADALAH NODE(61,76)
UnityEngine.Debug:Log(Object)

Node(60,77), Parent=Node(61,76), G=56, H=98, W=1.9, F=242.2
UnityEngine.Debug:Log(Object)

Node(61,77), Parent=Node(61,76), G=52, H=88, W=1.9, F=219.2
UnityEngine.Debug:Log(Object)

Node(62,76), Parent=Node(61,76), G=52, H=82, W=1.9, F=207.8
UnityEngine.Debug:Log(Object)

Node(62,77), Parent=Node(61,76), G=56, H=78, W=1.9, F=204.2
UnityEngine.Debug:Log(Object)

[5] BESTNODE YANG DIPINDAH KE CLOSED ADALAH NODE(62,77)
UnityEngine.Debug:Log(Object)

Node(61,78), Parent=Node(62,77), G=70, H=84, W=1.8, F=221.2
UnityEngine.Debug:Log(Object)

Node(62,78), Parent=Node(62,77), G=66, H=74, W=1.8, F=199.2
UnityEngine.Debug:Log(Object)

Node(63,76), Parent=Node(62,77), G=70, H=72, W=1.8, F=199.6
UnityEngine.Debug:Log(Object)

Node(63,77), Parent=Node(62,77), G=66, H=68, W=1.8, F=188.4
UnityEngine.Debug:Log(Object)

Node(63,78), Parent=Node(62,77), G=70, H=64, W=1.8, F=185.2
UnityEngine.Debug:Log(Object)

[6] BESTNODE YANG DIPINDAH KE CLOSED ADALAH NODE(63,78)
UnityEngine.Debug:Log(Object)

Node(62,79), Parent=Node(63,78), G=84, H=70, W=1.7, F=203
UnityEngine.Debug:Log(Object)

Node(63,79), Parent=Node(63,78), G=80, H=60, W=1.7, F=182
UnityEngine.Debug:Log(Object)

Node(64,77), Parent=Node(63,78), G=84, H=58, W=1.7, F=182.6
UnityEngine.Debug:Log(Object)

Node(64,78), Parent=Node(63,78), G=80, H=54, W=1.7, F=171.8
UnityEngine.Debug:Log(Object)

Node(64,79), Parent=Node(63,78), G=84, H=50, W=1.7, F=169
UnityEngine.Debug:Log(Object)

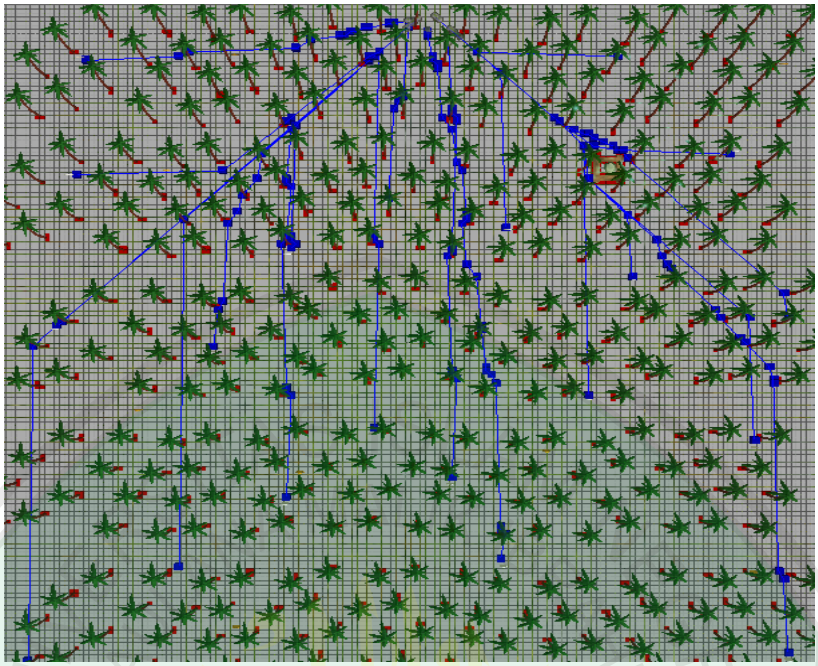
[7] BESTNODE YANG DIPINDAH KE CLOSED ADALAH NODE(64,79)
UnityEngine.Debug:Log(Object)

Gambar 4.16 Hasil Implementasi Algoritma DWA* Pada *Console Unity*

Gambar 4.17 Rute Terpendek yang Berhasil Ditemukan Pada *Console Unity*

Gambar 4.18 Rute Terpendek yang Berhasil Ditemukan Pada *Scene Unity*

Uji coba algoritma DWA* diatas merupakan salah satu contoh pergerakan NPC *Army* untuk mengambil satu koin perak. Sedangkan pada aplikasi *game* yang telah dibuat NPC *Army* akan mengambil semua koin perak yang ada di arena permainan. Gambar dibawah menampilkan semua NPC *Army* beserta rute terpendek yang akan mereka lalui untuk menghabiskan semua koin perak.



Gambar 4.19 Jalur yang Dilalui Semua NPC *Army* Pada *Scene Unity*

Pada algoritma DWA* terdapat bobot dinamis, hal inilah yang membedakannya dengan algoritma A*. Dibawah ini akan dilakukan beberapa uji coba untuk mengetahuai pengaruh penambahan bobot dinamis tersebut, sehingga sehingga bisa terlihat bahwa algoritma DWA* mempunyai bebeberapa perbedaan dengan algoritma asalnya yaitu algoritma A*.

Tabel 4.5 Uji Coba Pembobotan Algoritma DWA*

PK	*Start Node*		*Goal Node*		Min H	Bobot Awal	Jumlah *Node* Dibangkitkan		Penurunan Bobot	Bobot Akhir
	X	Y	X	Y			*Open*	*Closed*		
1	52	98	62	66	346	4,46	86	33	31x	1,36
2	52	98	76	91	254	3,54	63	25	23x	1,24
3	52	98	77	59	476	5,76	123	40	38x	1,96
4	52	98	89	77	440	5,40	107	38	36x	1,80

5	52	98	96	53	612	7,12	168	51	49x	2,22
6	52	98	71	41	632	7,32	153	58	56x	1,72
7	52	98	92	34	786	8,86	197	65	63x	2,56
8	52	98	61	17	832	9,32	183	82	80x	1,32
9	52	98	96	4	1102	12,02	263	95	93x	2,72
10	52	98	55	29	688	7,88	146	70	68x	1,08
11	52	98	45	36	634	7,34	144	63	61x	1,24
12	52	98	26	48	590	6,90	150	51	49x	2,00
13	52	98	35	26	774	8,74	178	73	71x	1,64
14	52	98	22	16	926	10,26	217	83	81x	2,16
15	52	98	3	2	1142	12,42	279	97	95x	2,92
16	52	98	35	62	414	5,14	108	37	35x	1,64
17	52	98	47	70	286	3,86	74	29	27x	1,16
18	52	98	7	52	626	7,26	173	48	46x	2,66
19	52	98	10	91	434	5,34	99	43	41x	1,24
20	52	98	8	74	522	6,22	130	45	43x	1,92

Tabel 4.5 menunjukkan hasil uji coba pembobotan pada algoritma DWA* dengan 20 rute yang berbeda. Judul pada kolom pertama PK yaitu percobaan ke-x. Dari tabel tersebut terlihat bahawa implementasi algoritma DWA* pada *Unity* telah berjalan dengan baik. Khususnya untuk masalah pembobotan, mulai dari penentuan bobot awal yang bernilai besar, ada penurunan bobot dinamis yang terus diturunkan sebanyak 0,1 sampai *goal* ditemukan, dan akhirnya diketahui nilai bobot yang terakhir digunakan. Pada bobot akhir tersebut ada yang nilainya lebih dari dua, karena memang rutenya panjang sehingga bobot awalnya sangat besar dan ketika bobot tersebut terus diturunkan, *goal* telah ditemukan sebelum

bobot dinamis tersebut sangat dekat dengan satu. Tetapi hal ini sudah sesuai dengan konsep pada algoritma DWA* karena nilai tersebut sudah jauh lebih mendekati satu dari pada bobot awalnya. Dan untuk membuktikan bahwa pembobotan tersebut sudah baik, pada uji coba selanjutnya algoritma DWA* akan dibandingkan dengan algoritma A* untuk menyelesaikan masalah yang sama.

Tabel 4.6 Perbandinagan Jumlah Node yang Dibangkitkan Oleh DWA* dan A*

PK	*Start Node*		*Goal Node*		Jumlah *Node* yang Dibangkitkan Oleh Algoritma DWA*			Jumlah *Node* yang Dibangkitkan Oleh Algoritma A*			Selisih Total *Node*	Yang Lebih Baik
	X	Y	X	Y	*Open*	*Closed*	Total	*Open*	*Closed*	Total		
1	52	98	62	66	86	33	119	110	159	269	150	DWA*
2	52	98	76	91	63	25	88	63	25	88	0	Sama
3	52	98	77	59	123	40	163	127	89	216	53	DWA*
4	52	98	89	77	107	38	145	107	38	145	0	Sama
5	52	98	96	53	168	51	219	165	186	351	132	DWA*
6	52	98	71	41	153	58	211	193	263	456	245	DWA*
7	52	98	92	34	197	65	262	209	161	370	108	DWA*
8	52	98	61	17	183	82	265	214	486	700	444	DWA*
9	52	98	96	4	263	95	358	275	191	466	108	DWA*
10	52	98	55	29	146	70	216	173	223	396	180	DWA*
11	52	98	45	36	144	63	207	175	262	437	230	DWA*
12	52	98	26	48	150	51	201	163	120	283	82	DWA*
13	52	98	35	26	178	73	251	247	438	685	434	DWA*
14	52	98	22	16	217	83	300	305	570	875	575	DWA*
15	52	98	3	2	279	97	376	280	100	380	4	DWA*
16	52	98	35	62	108	37	145	119	106	225	80	DWA*
17	52	98	47	70	74	29	103	79	80	159	56	DWA*

18	52	98	7	52	173	48	221	173	49	222	1	DWA*
19	52	98	10	91	99	43	142	123	251	374	232	DWA*
20	52	98	8	74	130	45	175	130	45	175	0	Sama

Untuk lebih jelas mengenai pengaruh pembobotan pada algoritma DWA* pada Tabel 4.6 dilakukan uji coba perbandingan *node* yang harus dibangkitkan untuk menemukan rute terpendek pada algoritma DWA* dan A*. Ada beberapa rute yang berhasil ditemukan oleh kedua algoritma tersebut dengan jumlah *node* yang dibangkitkan adalah sama. Tetapi dalam banyak masalah penemuan rute lainnya, algoritma DWA* membangkitkan lebih sedikit *node* dari pada A*, jadi algoritma DWA* membutuh memori lebih kecil dari pada algoritma A*.

Tabel 4.7 Perbandingan Waktu Pencarian Rute Terpendek Pada DWA* dan A*

No	*Start Node*		*Goal Node*		Waktu (ms) yang Dibutuhkan Algoritma DWA* Untuk Menemukan Rute Terpendek					Waktu (ms) yang Dibutuhkan Algoritma A* Untuk Menemukan Rute Terpendek					Yang Lebih Baik
	X	Y	X	Y	1	2	3	4	5	1	2	3	4	5	
1	52	98	62	66	7	7	8	8	7	8	8	9	9	8	DWA*
2	52	98	76	91	7	7	7	8	7	7	7	8	7	7	Sama
3	52	98	77	59	7	7	8	7	7	8	8	8	8	8	DWA*
4	52	98	89	77	7	7	8	8	7	7	7	8	8	7	Sama
5	52	98	96	53	8	7	8	8	8	9	9	9	9	9	DWA*
6	52	98	71	41	7	8	8	8	8	10	10	10	10	10	DWA*
7	52	98	92	34	8	8	8	8	8	8	9	8	9	9	DWA*
8	52	98	61	17	8	8	8	8	8	12	12	12	12	12	DWA*
9	52	98	96	4	9	9	9	8	9	9	9	9	9	9	DWA*
10	52	98	55	29	8	8	8	8	8	9	9	10	9	9	DWA*

11	52	98	45	36	8	8	7	7	7	9	9	10	10	10	DWA*
12	52	98	26	48	7	8	8	8	7	8	8	8	8	9	DWA*
13	52	98	35	26	8	8	8	8	8	12	11	11	12	11	DWA*
14	52	98	22	16	8	8	9	9	8	13	13	13	13	13	DWA*
15	52	98	3	2	8	9	8	8	8	8	8	8	9	8	Sama
16	52	98	35	62	8	7	7	7	7	8	8	8	8	8	DWA*
17	52	98	47	70	7	7	7	7	7	9	8	7	9	8	DWA*
18	52	98	7	52	8	7	7	7	8	8	7	7	7	8	Sama
19	52	98	10	91	7	7	7	7	7	9	9	10	9	9	DWA*
20	52	98	8	74	7	7	7	7	7	7	7	7	8	7	DWA*

Selanjutnya pada Tabel 4.7 dilakukan uji coba perbandingan waktu yang dibutuhkan untuk menemukan rute terpendek pada algoritma DWA* dan A*. Waktu yang dibutuhkan oleh kedua algoritma tersebut untuk menemukan rute terpendek kurang lebih sama hanya selisih beberapa *milisecond*. Algoritma DWA* mempunyai catatan waktu yang sedikit lebih baik dari pada algoritma A*. Algoritma DWA* memang sangat memungkinkan untuk menemukan rute lebih cepat, karena jumlah *node* yang dibangkitkan lebih sedikit, sehingga algoritma DWA* bisa beberapa langkah lebih cepat dari pada algoritma A*.

Tabel 4.8 Total Waktu Yang Dibutuhkan NPC Untuk Menyelesaikan Misinya

No	Kecepatan NPC	Waktu NPC Menyelesaikan Misi			
		PK1 (s)	PK1 (s)	PK1 (s)	Kesimpulan (s)
1	0,5	230.8657	230.9549	231.0307	231
2	1,0	116.1809	116.2376	116.2976	116
3	1,5	78.20097	78.27197	78.33861	78

4	2,0	58.32705	58.38465	58.44361	58
5	2,5	46.71588	46.77452	46.83284	47
6	3,0	38.65874	38.70577	38.75236	39
7	3,5	34.20966	34.28745	34.35868	34
8	4,0	30.18334	30.25813	30.32731	30
9	4,5	26.27105	26.32603	26.39093	26
10	5,0	24.17509	24.24307	24.30688	24
11	5,5	22.14423	22.1608	22.18513	22
12	6,0	20.15484	20.17484	20.20017	20
13	6,5	18.35335	18.36333	18.38772	18
14	7,0	17.03831	17.05833	17.08216	17
15	7,5	16.15789	16.1698	16.19371	16
16	8,0	15.2787	15.28926	15.31423	15
17	8,5	14.15382	14.16382	14.18753	14
18	9,0	13.35573	13.37566	13.39967	13
19	9,5	12.69236	12.71041	12.73616	13
20	10,0	12.15683	12.16812	12.19387	12

Setelah dilakukan beberapa uji coba algoritma DWA*, pada Tabel 4.8 merupakan uji coba total waktu yang dibutuhkan NPC *Army* pada *game* untuk menyelesaikan misinya yaitu menghabiskan semua koin perak (20 koin) yang ada di arena permainan. Kecepatan NPC *Army* diubah-ubah mulai dari 0,5 sampai dengan 10. Semakin mendekati 10 kecepatan gerak NPC *Army* maka waktu yang dibutuhkan untuk menyelesaikan misi akan semakin cepat. Teapi jika pada *game* ini kecepatan NPC *Army* diatur mendekati 10, maka waktu yang dimiliki *player* untuk bisa mengambil koin perak terlalu singkat. Jadi pada *game* ini kecepatan

NPC *Army* akan diberi nilai satu. Tidak terlalu cepat juga tidak terlalu lama, *player* mempunyai waktu kurang lebih 116 detik dari total waktu 180 detik yang dimiliki *player* untuk menyelesaikan misinya. Sehingga *player* masih punya cukup waktu untuk mengambil beberapa koin perak di dalam *game*.

4.2.2 Uji Coba Algoritma *Fisher-Yates Shuffle*

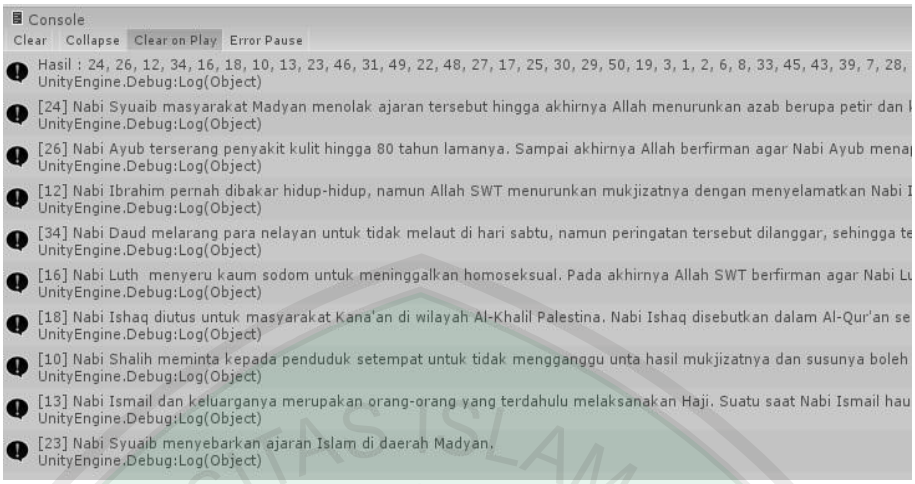
Uji coba algoritma *Fisher-Yates Shuffle* dilakukan untuk melihat hasil dari implementasi algoritma tersebut ke dalam *game* yang telah dibuat menggunakan *Unity*. Algoritma ini akan dilakukan uji coba beberapa kali untuk mengacak 50 angka yang merupkan nomor urut konten islami tentang nabi dan rasul.

Tabel 4.9 Uji Coba Urutan Baru yang Dihasilkan Algoritma *Fisher-Yates Shuffle*

PK	Hasil Urutan Baru
1	29, 18, 13, 28, 26, 37, 4, 16, 21, 15, 12, 31, 6, 17, 25, 23, 35, 19, 30, 32, 34, 41, 48, 50, 20, 49, 33, 8, 7, 14, 47, 38, 11, 2, 1, 5, 27, 36, 24, 39, 9, 10, 44, 46, 42, 3, 22, 43, 30, 45
2	6, 39, 4, 22, 7, 23, 14, 21, 37, 19, 17, 30, 43, 45, 25, 44, 27, 1, 8, 31, 15, 18, 41, 32, 33, 35, 29, 12, 24, 50, 5, 3, 49, 26, 13, 11, 30, 20, 48, 47, 16, 36, 9, 10, 28, 2, 46, 42, 38, 34
3	30, 37, 34, 46, 45, 11, 6, 28, 50, 21, 43, 10, 9, 4, 39, 29, 3, 5, 32, 36, 15, 49, 8, 30, 47, 24, 16, 44, 14, 27, 19, 26, 41, 2, 1, 42, 18, 12, 31, 22, 20, 23, 35, 13, 33, 25, 38, 17, 48, 7
4	44, 1, 7, 12, 47, 49, 17, 30, 18, 46, 10, 38, 32, 30, 43, 19, 29, 34, 50, 27, 15, 33, 35, 8, 42, 31, 11, 6, 24, 3, 9, 22, 16, 48, 37, 45, 25, 39, 26, 5, 2, 41, 14, 21, 28, 36, 20, 13, 4, 23
5	44, 31, 34, 27, 28, 25, 14, 16, 29, 26, 21, 50, 30, 9, 43, 12, 46, 38, 13, 3, 8, 37, 1, 48, 30, 6, 24, 45, 22, 19, 32, 39, 4, 10, 33, 18, 42, 23, 41, 36, 35, 20, 49, 11, 47, 15, 2, 7, 5, 17
6	16, 32, 4, 31, 12, 28, 5, 34, 30, 33, 27, 24, 2, 36, 18, 23, 19, 17, 26, 15, 25, 45, 22, 50, 7, 44, 47, 20, 37, 6, 21, 29, 42, 41, 10, 49, 38, 35, 46, 1, 9, 11, 39, 48, 3, 14, 43, 30, 8, 13
7	9, 22, 35, 31, 48, 18, 33, 15, 30, 21, 37, 47, 44, 10, 7, 28, 3, 38, 13, 19, 16, 11, 34, 2, 12, 41, 24, 1, 23, 45, 6, 20, 36, 26, 39, 5, 4, 32, 50, 17, 30, 46, 8, 43, 25, 14, 27, 42, 29, 49
8	6, 36, 23, 49, 12, 31, 18, 33, 29, 5, 4, 45, 38, 27, 11, 47, 48, 34, 43, 32, 44, 6, 37, 1, 17, 50, 42, 8, 25, 7, 30, 35, 9, 16, 14, 3, 20, 41, 26, 2, 10, 28, 21, 22, 39, 19, 30, 13, 24, 15
9	28, 22, 7, 21, 34, 49, 46, 15, 26, 14, 6, 42, 39, 11, 31, 30, 44, 10, 25, 43, 32, 50, 19, 37, 4, 16, 30, 27, 13, 5, 47, 35, 20, 9, 33, 17, 36, 24, 48, 45, 18, 1, 29, 12, 2, 23, 38, 41, 8, 3

10	7, 8, 34, 39, 11, 29, 3, 15, 5, 14, 43, 4, 20, 35, 18, 49, 16, 9, 2, 24, 27, 12, 30, 37, 38, 31, 25, 32, 50, 48, 10, 36, 21, 26, 33, 13, 17, 42, 46, 28, 22, 30, 6, 45, 19, 23, 1, 44, 47, 41
11	49, 43, 29, 39, 17, 33, 12, 4, 32, 15, 3, 27, 22, 36, 34, 9, 14, 26, 18, 45, 35, 10, 47, 28, 37, 50, 8, 48, 6, 25, 13, 20, 46, 24, 11, 19, 38, 5, 42, 30, 30, 23, 16, 1, 41, 7, 2, 21, 31, 44
12	31, 42, 4, 43, 1, 34, 35, 29, 37, 24, 32, 47, 11, 15, 5, 33, 14, 30, 36, 39, 49, 25, 30, 46, 17, 6, 16, 20, 12, 44, 26, 9, 21, 8, 27, 48, 23, 19, 41, 7, 45, 28, 50, 22, 18, 13, 38, 2, 3, 10
13	18, 46, 23, 16, 43, 42, 13, 30, 31, 35, 45, 4, 3, 10, 26, 24, 34, 28, 9, 48, 32, 11, 2, 29, 33, 36, 14, 8, 39, 38, 44, 22, 41, 5, 21, 20, 12, 37, 17, 1, 27, 50, 19, 49, 15, 6, 30, 7, 25, 47
14	22, 19, 46, 30, 29, 33, 45, 10, 41, 14, 21, 4, 5, 27, 12, 8, 38, 36, 44, 3, 1, 20, 9, 26, 15, 6, 48, 13, 7, 30, 16, 34, 42, 49, 18, 11, 50, 43, 17, 39, 2, 28, 35, 37, 31, 32, 24, 23, 47, 25
15	13, 5, 12, 20, 11, 14, 2, 25, 6, 43, 4, 41, 44, 26, 28, 22, 42, 33, 17, 37, 35, 36, 1, 10, 49, 8, 3, 34, 50, 9, 23, 16, 32, 19, 7, 45, 30, 21, 18, 38, 24, 29, 39, 27, 15, 47, 31, 30, 46, 48
16	19, 22, 25, 6, 27, 45, 39, 26, 28, 3, 41, 14, 17, 38, 44, 2, 12, 47, 15, 11, 35, 5, 36, 18, 32, 7, 34, 20, 13, 42, 23, 31, 9, 24, 30, 4, 48, 8, 16, 33, 1, 30, 50, 37, 21, 43, 10, 49, 46, 29
17	46, 38, 25, 7, 30, 20, 13, 33, 5, 14, 35, 10, 45, 15, 21, 6, 43, 27, 36, 30, 24, 16, 9, 50, 12, 42, 19, 11, 23, 18, 34, 3, 39, 17, 22, 8, 1, 48, 32, 37, 4, 41, 47, 49, 31, 2, 44, 28, 29, 26
18	22, 31, 24, 50, 38, 35, 4, 28, 21, 49, 30, 1, 18, 37, 12, 20, 23, 33, 32, 30, 26, 43, 25, 41, 9, 46, 13, 6, 7, 44, 29, 10, 8, 17, 16, 34, 39, 15, 5, 45, 36, 27, 11, 42, 48, 47, 2, 19, 3, 14
19	19, 13, 34, 25, 30, 41, 2, 36, 14, 9, 12, 15, 4, 5, 7, 30, 35, 11, 6, 26, 49, 23, 16, 18, 17, 27, 31, 50, 24, 46, 44, 10, 38, 43, 39, 3, 48, 28, 45, 20, 47, 22, 37, 21, 1, 8, 33, 29, 32, 42
20	43, 8, 31, 21, 2, 37, 4, 20, 41, 44, 17, 10, 34, 29, 3, 9, 28, 36, 18, 11, 46, 5, 1, 14, 48, 15, 26, 30, 16, 35, 30, 7, 12, 47, 42, 45, 49, 19, 27, 6, 32, 50, 24, 23, 13, 38, 22, 33, 39, 25

Tabel 4.9 merupakan hasil uji coba urutan baru yang dihasilkan oleh algoritma *Fisher-Yates Shuffle*. Uji coba ini dilakukan untuk menunjukkan bahwa algoritma ini selalu menghasilkan urutan baru yang berbeda setiap kali dijalankan, pada tabel tersebut menunjukkan hasil uji coba algoritma tersebut sebanyak 20 kali. Kolom pertama berjudul PK atau percobaan ke-x, dan pada kolom kedua hasil urutan baru setalah program dijalankan. Hasil uji coba pada Tabel 4.9 menujukkan bahwa tidak ada satupun urutan baru yang dihasilkan oleh algoritma tersebut sama dengan urutan yang lainnya.



Gambar 4.20 Hasil Implementasi Algoritma FYS Pada *Console Unity*

Gambar 4.20 di atas adalah tampilan *tab console di Unity* ketika *game* dimainkan. Baris pertama menunjukkan hasil dari array yang telah diacak urutannya dengan algoritma *Fisher-Yates Shuffle*. Hasil tersebut langsung muncul ketika permainan dimulai. Hasil urutan terbaru tersebut akan digunkan sebagai urutan konten islami yang akan ditampilkan setiap kali *player* mendapat satu koin emas. Misalkan pada percobaan ini hasil urutannya mulai dari urutan pertama adalah 24, 26, 12, 34, 16 dan seterusnya sampai urutan ke 50. Maka pada koin emas pertama yang di dapat *player* akan ditampilkan pengetahuan nomor 26, koin emas kedua akan ditampilkan pengetahuan nomor 24, koin emas ketiga akan ditampilkan pengetahuan nomor 12 dan begitu juga seterusnya. Mulai baris kedua dan setelahnya pada *tab console Unity* akan muncul satu persatu setiap kali *player* mendapatkan satu koin emas.

Dalam setiap kali permainan tidak semua konten yang berjumlah 50 itu akan ditampilkan. Pada aplikasi *game* yang telah dibuat terdapat sepuluh koin emas, jadi pengetahuan yang akan ditampilkan adalah pengetahuan dengan

nomor-nomor yang sesuai dengan sepuluh angka pertama pada urutan baru yang dihasilkan. Dan semua pengetahuan yang ditampilkan tersebut pasti selalu berbeda, karena algoritma *Fisher-Yates Shuffle* adalah mengacak urutan, berbeda dengan fitur *random* yang ada di *Unity Game Engine* hanya mengacak angka.

Gambar 4.21 Hasil Pengacakan Fitur *Random Unity* Pada *Console Unity*

Gambar 4.21 merupakan hasil dari tiga kali percobaan *random* angka *Unity* pada *tab console Unity* yanga diantara kesepuluh angka tersebut ada angka yang sama. Hal seperti ini tidak selalu terjadi, bisa saja kesepuluh angka tersebut berbeda. Lebih jelasnya akan dilakukan 20x uji coba dan hasilnya dibandingkan dengan sepuluh angka pertama hasil pengacakan algoritma *Fisher-Yates Shuffle.*

Tabel 4.10 Perbandingan Pengacakan *Fisher-Yates Shuffle* dan Fitur *Random Unity*

PK	***Fisher-Yates Shuffle***	**Fitur *Random Unity***
1	29, 18, 13, 28, 26, 37, 4, 16, 21, 15	5, 15, 44, 35, 18, 49, 19, 47, 45, 37
2	6, 39, 4, 22, 7, 23, 14, 21, 37, 19	**2, 16, 11, 16, 4, 45, 15, 1, 11, 45**
3	30, 37, 34, 46, 45, 11, 6, 28, 50, 21	12, 35, 13, 44, 28, 18, 20, 49, 28, 31

4	44, 1, 7, 12, 47, 49, 17, 30, 18, 46	**4, 7, 25, 44, 30, 4, 37, 47, 39, 16**
5	44, 31, 34, 27, 28, 25, 14, 16, 29, 26	30, 21, 32, 23, 31, 14, 33, 20, 1, 41
6	16, 32, 4, 31, 12, 28, 5, 34, 30, 33	**18, 9, 46, 27, 11, 27, 40, 15, 22, 35**
7	9, 22, 35, 31, 48, 18, 33, 15, 30, 21	**43, 44, 45, 38, 16, 14, 18, 10, 34, 45**
8	6, 36, 23, 49, 12, 31, 18, 33, 29, 5	**8, 4, 30, 17, 20, 7, 11, 25, 29, 4**
9	28, 22, 7, 21, 34, 49, 46, 15, 26, 14	7, 27, 22, 32, 19, 13, 49, 9, 33, 23
10	7, 8, 34, 39, 11, 29, 3, 15, 5, 14	2, 28, 49, 38, 11, 5, 9, 45, 15, 4
11	49, 43, 29, 39, 17, 33, 12, 4, 32, 15	**9, 17, 9, 13, 13, 22, 38, 11, 12, 46**
12	31, 42, 4, 43, 1, 34, 35, 29, 37, 24	39, 44, 40, 18, 49, 23, 29, 6, 30, 25
13	18, 46, 23, 16, 43, 42, 13, 30, 31, 35	20, 48, 22, 24, 41 , 47, 13, 39, 26, 20
14	22, 19, 46, 30, 29, 33, 45, 10, 41, 14	2, 9, 18, 13, 43, 3, 6, 44, 2, 46
15	13, 5, 12, 20, 11, 14, 2, 25, 6, 43	20, 1, 35, 10, 26, 29, 24, 22, 27, 36
16	19, 22, 25, 6, 27, 45, 39, 26, 28, 3	**10, 8, 12, 30, 6, 28, 20, 6, 29, 20**
17	46, 38, 25, 7, 30, 20, 13, 33, 5, 14	27, 16, 33, 46, 17, 13, 5, 25, 42, 17
18	22, 31, 24, 50, 38, 35, 4, 28, 21, 49	**30, 33, 17, 31, 23, 17, 39, 35, 40, 8**
19	19, 13, 34, 25, 30, 41, 2, 36, 14, 9	**30, 1, 43, 42, 35, 14, 30, 17, 28, 5**
20	43, 8, 31, 21, 2, 37, 4, 20, 41, 44	**29, 28, 13, 20, 18, 1, 40, 41, 5, 1**

Tabel 4.10 merupakan uji coba perbandingan hasil pengacakan dengan menggunakan algoritma *Fisher-Yates Shuffle* dan fitur *random* di *Unity Game Engine.* Dari tabel tersebut terlihat bahwa pada setiap kali percobaan, sepuluh angka yang dihasilkan oleh algoritma *Fisher-Yates Shuffle* selalu berbeda. Tetapi pada pengcakan angka dengan menggunakan fitur *random* di *Unity* pada sepuluh angka yang dihasilkan masih ada kemungkinan muncul angka yang sama. Pada tabel tersebut telah ditandai dengan *font* tebal *(bold).* Sehingga ada kemungkinan satu konten yang sama akan di tampilkan beberapa kali dalam satu permainan.

4.2.3 Uji Coba Aplikasi *Game*

Uji coba ini dilakukan untuk mengetahui apakah *game* yang telah dibuat dapat diimplementasikan di *personal computer* atau laptop. Berikut hasil pengujian yang disajikan dalam tabel.

Tabel 4.11 Uji Coba Aplikasi *Game*

No	Versi OS	Spesifikasi Perangkat	Keterangan
1	*Windows 7 64bit*	*- Acer Aspire 1830T* *- Processor Intel® Core™ i5 CPU U 470 @1,33GHz* *- RAM 4096MB* *- VGA Intel® HD Graphics* *- Display 11,6”*	Tampilan menu berjalan dengan baik. Tombol berfungsi dengan baik. Tampilan *game* berjalan dengan baik.
2	*Windows 7 32bit*	*- Lenovo G40* *- Processor AMD A6-6310 APU @1,8GHz* *- RAM 2048MB* *- VGA AMD Radeon™ R4 Graphics* *- Display 14”*	Tampilan menu berjalan dengan baik. Tombol berfungsi dengan baik. Tampilan *game* berjalan dengan baik.
3	*Windows 8.1 64bit*	*- Asus A43SA* *- Processor Intel® Core™ i3-2330M @2,2GHz* *- RAM 2048MB* *- VGA AMD Radeon™ HD 6730M (2 GB)* *- Display 14”*	Tampilan menu berjalan dengan baik. Tombol berfungsi dengan baik. Tampilan *game* berjalan dengan baik.
4	*Windows 10 64bit*	*- HP Pavilion 14* *- Processor Intel® Core™ i5 @2,4GHz* *- RAM 4096MB* *- VGA NVIDIA GeForce GT840M (2GB)* *- Display 14”*	Tampilan menu berjalan dengan baik. Tombol berfungsi dengan baik. Tampilan *game* berjalan dengan baik.

Dari pengujian yang dilakukan sebanyak 4 kali pada berbagai *platform windows* yang memiliki sistem operasi dan spesifikasi berbeda. Dapat diketahui presentase hasil pengujian aplikasi *game* pada Tabel 4.12 di bawah ini.

Tabel 4.12 Persentase Hasil Uji Coba Aplikasi *Game*

No	Jenis Pengujian	Baik	
		Jumlah	%
1	Sistem	4	(4/4) x 100 = 100%
2	Tombol	12	(12/12) x 100 = 100%
3	Tampilan	4	(4/4) x 100 = 100%

Tabel 4.12 di atas merupakan tabel yang berisi hasil pengujian *game* terhadap 4 sistem operasi *windows desktop* yang telah dijelaskan pada Tabel 4.11. Hasil persentase yang didapatkan dari pengujian adalah 100% *game* dapat berjalan dengan baik pada *device* tersebut.

4.3 Integrasi Dalam Islam

Mempelajari sejarah dan peradaban islam merupakan bagian penting yang tidak mungkin dipisahkan dari kehidupan kaum muslimin dari masa ke masa. Karena dengan jalan memahami sejarah dengan baik dan benar, kaum muslimin bisa bercermin untuk mengambil banyak pelajaran dan membenahi kekurangan atau kesalahan mereka guna meraih kejayaan dan kemuliaan dunia dan akhirat.

Sebaik-baik kisah sejarah yang dapat diambil pelajaran dan hikmah berharga darinya adalah kisah-kisah yang terdapat dalam ayat-ayat Al-Qur'an dan

hadits-hadits yang shahih dari Rasulullah SAW. Karena kisah-kisah tersebut disamping sudah pasti benar, bersumber dari wahyu Allah yang Maha Benar, juga karena kisah-kisah tersebut memang disampaikan oleh Allah untuk menjadi pelajaran bagi orang-orang yang berakal sehat. Allah berfirman :

لَقَدْ كَانَ فِي قَصَصِهِمْ عِبْرَةٌ لِّأُولِي الْأَلْبَابِ ۗ مَا كَانَ حَدِيثًا يُفْتَرَىٰ وَلَٰكِن تَصْدِيقَ الَّذِي بَيْنَ يَدَيْهِ وَتَفْصِيلَ كُلِّ شَيْءٍ وَهُدًى وَرَحْمَةً لِّقَوْمٍ يُؤْمِنُونَ ﴿١١١﴾

Artinya: Sesungguhnya pada kisah-kisah mereka (para Nabi dan umat mereka) itu terdapat pelajaran bagi orang-orang yang mempunyai akal (sehat). Al-Qur'an itu bukanlah cerita yang dibuat-buat, akan tetapi membenarkan (kitab-kitab) yang sebelumnya dan menjelaskan segala sesuatu, serta sebagai petunjuk dan rahmat bagi orang-orang yang beriman. (QS. Yusuf: 111).

. Kisah-kisah yang menggambarkan keadaan para nabi dan umat mereka tersebut, serta yang menjelaskan kemuliaan orang-orang yang beriman dan kebinasaan orang-orang kafir yang mendustakan seruan para nabi, berisi pelajaran bagi orang-orang yang beriman untuk memantapkan keimanan mereka dan menguatkan ketakwaan mereka kepada Allah dengan menjalankan perintah-Nya dan menjauhi larangan-Nya. Kisah-kisah sejarah para nabi adalah termasuk sebab utama untuk mengokohkan dan menyempurnakan keimanan dalam hati orang-orang yang beriman. Allah berfirman :

وَكُلًّا نَّقُصُّ عَلَيْكَ مِنْ أَنبَاءِ الرُّسُلِ مَا نُثَبِّتُ بِهِ فُؤَادَكَ ۚ وَجَاءَكَ فِي هَٰذِهِ الْحَقُّ وَمَوْعِظَةٌ وَذِكْرَىٰ لِلْمُؤْمِنِينَ ﴿١٢٠﴾

Artinya: Dan semua kisah dari rasul-rasul Kami ceritakan kepadamu, ialah kisah-kisah yang dengannya Kami teguhkan hatimu, dan dalam surat ini telah datang kepadamu kebenaran serta pengajaran dan peringatan bagi orang-orang yang beriman. (QS. Hud: 120).

Dalam ayat ini jelas sekali menunjukkan bahwa kisah-kisah dalam Al-Qur'an tentang ketabahan dan kesabaran para nabi dalam memperjuangkan dan mendakwahkan agama Allah sangat berpengaruh dalam meneguhkan hati dan keimanan orang-orang yang beriman di jalan Allah.

Mempelajari kisah para nabi juga merupakan implementasi dari rukun iman yang keempat yaitu iman kepada nabi dan rasul Allah. Keimanan seseorang itu tidak sah, sampai ia mengimani semua nabi dan rasul Allah dan membenarkan bahwa Allah telah mengutus mereka untuk menunjuki, membimbing dan mengeluarkan manusia dari kegelapan kepada cahaya kebenaran. Ditambah juga keharusan membenarkan bahwa mereka telah menyampaikan apa yang Allah turunkan kepada mereka dengan benar dan sempurna, dan mereka telah berjihad dengan sebenar-benarnya di jalan Allah. Allah berfirman:

لَّيْسَ ٱلْبِرَّ أَن تُوَلُّوا۟ وُجُوهَكُمْ قِبَلَ ٱلْمَشْرِقِ وَٱلْمَغْرِبِ وَلَٰكِنَّ ٱلْبِرَّ مَنْ ءَامَنَ بِٱللَّهِ وَٱلْيَوْمِ ٱلْءَاخِرِ وَٱلْمَلَٰٓئِكَةِ وَٱلْكِتَٰبِ
وَٱلنَّبِيِّۦنَ وَءَاتَى ٱلْمَالَ عَلَىٰ حُبِّهِۦ ذَوِى ٱلْقُرْبَىٰ وَٱلْيَتَٰمَىٰ وَٱلْمَسَٰكِينَ وَٱبْنَ ٱلسَّبِيلِ وَٱلسَّآئِلِينَ وَفِى ٱلرِّقَابِ وَأَقَامَ
ٱلصَّلَوٰةَ وَءَاتَى ٱلزَّكَوٰةَ وَٱلْمُوفُونَ بِعَهْدِهِمْ إِذَا عَٰهَدُوا۟ ۖ وَٱلصَّٰبِرِينَ فِى ٱلْبَأْسَآءِ وَٱلضَّرَّآءِ وَحِينَ ٱلْبَأْسِ ۗ أُو۟لَٰٓئِكَ ٱلَّذِينَ
صَدَقُوا۟ ۖ وَأُو۟لَٰٓئِكَ هُمُ ٱلْمُتَّقُونَ ﴿١٧٧﴾

Artinya: Bukanlah menghadapkan wajahmu ke arah timur dan barat itu suatu kebajikan, akan tetapi sesungguhnya kebajikan itu ialah beriman kepada Allah, hari

kemudian, malaikat-malaikat, kitab-kitab, nabi-nabi dan memberikan harta yang dicintainya kepada kerabatnya, anak-anak yatim, orang-orang miskin, musafir (yang memerlukan pertolongan) dan orang-orang yang meminta-minta dan (memerdekakan) hamba sahaya, mendirikan shalat, dan menunaikan zakat dan orang-orang yang menepati janjinya apabila ia berjanji, dan orang-orang yang sabar dalam kesempitan, penderitaan dan dalam peperangan. Mereka itulah orang-orang yang benar (imannya) dan mereka itulah orang-orang yang bertakwa. (QS. Al-Baqarah: 177).

Salah satu yang diperintahkan kepada orang-orang yang beriman dan bertaqwa dari ayat ini adalah untuk beriman kepada nabi dan rasul Allah. Untuk bisa beriman atau meyakini adanya nabi dan rasul Allah salah satu caranya adalah dengan mempelajari kisah-kisah kehidupan mereka. Banyak keutamaan dan manfaat yang bisa diperoleh dari mempelajari sejarah para nabi, diantaranya telah disebutkan pada bagian awal sub bab ini. Oleh karena itu melalui aplikasi *game* yang dibuat pada peneletian ini, diharapkan pengguna mampu mengenal nama-nama nabi dan rasul beserta mukjizatnya serta dapat memetik hikmah dalam setiap ilmu yang diberikan. Tentunya tidak semua kisah mereka disertakan pada aplikasi *game* ini, hanya yang pokok atau yang populer dari kisah nabi dan rasul. Tetapi dengan dimasukkannya konten islami di dalam *game*, menjadikan bermain *game* tidak hanya main-main saja tetapi ada nilai tambah, yaitu bisa menigkatkan pengetahuan keislaman khususnya tentang nabi dan rasul bagi siapapun yang bermain *game* ini. Semoga banyaknya peminat *game* edukasi atau *game-game* dengan konten islami selaras dengan kian bertambahnya *developer game* untuk memperkaya *game* edukasi sebagai media pembelajaran yang inovatif dan mengenalkan nilai-nilai islam kepada semua orang.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan hasil dari implementasi dan pengujian yang dilakukan terhadap aplikasi *game* yang telah dibuat pada penelitian ini, maka dapat ditarik beberapa kesimpulan diantaranya adalah sebagai berikut:

1. Algoritma DWA* berhasil diterapkan pada aplikasi *game Finding Diamond* sebagai pembangkit perilaku pencarian pada NPC *Army*. Semua NPC *Army* berhasil menemukan rute terpendek menuju setiap koin perak yang menjadi tujuannya, kemudian berjalan untuk mengambil koin perak tersebut serta bisa melewati semua halangan yang ada selama perjalanan.
2. Pemberian bobot dinamis pada algortima DWA* telah berjalan dengan baik dan sesuai konsep dari algoritma tersebut.
3. Salah satu kelebihan dari algortima DWA* jika dibandingkan dengan algoritma A* adalah algoritma DWA* membangkitkan lebih sedikit *node* daripada algoritma A* sehingga memori yang digunakan algoritma DWA* lebih kecil daripada algoritma A*.
4. Waktu yang dibutuhkan algoritma DWA* untuk menemukan rute terpendek relatif lebih cepat daripada algoritma A*.
5. Algoritma *Fisher-Yates Shuffle* berhasil diterapkan pada aplikasi *game Finding Diamond* untuk mengatur kemunculan konten islami tentang nabi dan rasul, sehingga konten yang ditampilkan kelihatan lebih bervariasi.

6. Salah satu kelebihan dari penggunaan algoritma *Fisher-Yates Shuffle* jika dibandingkan dengan fitur *random* di *Unity Game Engine* adalah algoritma *Fisher-Yates Shuffle* bisa memastikan semua konten yang ditampilkan dalam satu permainan selalu berbeda sedangkan jika hanya menggunakan fitur *random* di *Unity Game Engine* ada kemungkinan satu konten yang sama akan di tampilkan beberapa kali dalam satu permainan.
7. *Game Finding Diamond* telah diuji cobakan kepada beberapa komputer dengan spesifikasi dan sistem operasi berbeda pada *platform Windows*. Pada berbagai *device* tersebut *game* ini menunjukkan tingkat keberhasilan 100% pada uji coba sistem, tampilan dan tombol.

5.2 Saran

Peneliti yakin dengan penuh kesadaran bahwa dalam pembuatan permainan ini masih banyak kekurangan yang nantinya sangat perlu untuk dilakukan pengembangan demi sumbangsih terhadap ilmu pengetahuan, diantaranya:

1. Permainan ini tidak hanya disajikan dalam *platform desktop* saja, namun juga bisa dikembangkan pada *platform smartphone* khususnya *Android*.
2. Menambah jumlah level permainan dan materi pembelajaran serta aturan untuk kenaikan level sehingga permainan menjadi lebih menarik.
3. Mengimplementasikan berbagai modifikasi algoritma A* lainnya seperti IDA*, SMA*, BDA*, MBDA* pada aplikasi *game*, kemuadian dibandingkan hasilnya dengan algoritm DWA*.
4. Mengimplementasikan berbagai algoritma pengacakan atau *random* misalnya *Naive Shuffle* dan MCRNG pada aplikasi *game*.

DAFTAR PUSTAKA

Agung Pamungkas, Eka Puji Widiyanto dan Renni Angreni. 2014. *Penerapan Algoritma A* (A Star) Pada Game Edukasi The Maze Island Berbasis Android.* http://eprints.mdp.ac.id/1191/ diakses pada 19 November 2015.

Alif, Ifa. 2014. *3D Wayang Adventure Game Untuk Pengenalan Budaya Wayang Nusantara Menggunakan A* Pathfinding Algorithm Sebagai Pembangkit Perilaku Pencarian Pada NPC*. Skripsi. Malang: UIN Maulana Malik Ibrahim Malang.

Anggara. 2008. *Memahami Teknik Dasar Pembuatan Game Berbasis Flash*. Yogyakarta: Gave Media.

Arif, Yunifa Miftachul. 2010. *Strategi Menyerang pada Game FPS Menggunakan Hierarchy Finite State Machine dan Logika Fuzzy*. Thesis. Surabaya: Pasca Sarjana Teknik Elektro ITS.

Bronstring, Marek. 2012. *What are adventure games?*. www.adventure gamers.com /articles/view/17547 diakses pada 15 Desember 2015.

Durstenfeld, Richard. (July 1964). "Algorithm 235: Random permutation". Communications of the ACM 7 (7): 420. doi:10.1145/364520.364540.

Fadila, Juniardi Nur. 2014. *Aplikasi Permainan Meteor Shooter Menggunakan MCRNG dan A* Sebagai Algoritma Randoming Spawn dan Pencarian User Berbasis Mobile*. Skripsi. Malang: UIN Maulana Malik Ibrahim Malang.

Ferico Elmanus, Fazmah Arief Yulianto dan Bedy Purnama. 2012. *Analisis Perbandingan Algoritma DWA* dan Algoritma F2L Pada Permasalahan Kubus Rubik 3x3x3.* https://repository.telkomuniversity.ac.id/pustaka/95113/analisis-perbandingan-algoritma-dwa-dan-algoritma-f2l-pada-permasalahan-kubus-rubik-3x3x3.html diakses pada 2 Desember 2015.

Fisher, Ronald A., Yates, Frank. 1938. Statistical tables for biological, agricultural and medical research (3rd). London: Oliver & Boyd. pp. 26–27. OCLC

14222135. Note: the 6th edition, ISBN 0-02-844720-4, is available on the web, but gives a different shuffling algorithm by C. R. Rao.

Hanafi. 2011. *Kisah 25 Nabi & Rasul*. Jakarta: Bintang Indonesia.

Micka, Pavel. 2011. *Fisher-Yates-shuffle Algorithm: Founder and administrator of web encyclopedia Algoritmy.net.[online]*. Diakses 15 Maret 2016. http://en.algoritmy.net/article/43676/Fisher-Yates-shuffle

Millington, Ian. 2006. *Artificial Intelligence for Games*. California: Morgan Kaufmann Publishing.

Neumann, J. Von dan O. Morgenstern. 1944. *Theory of Games and Economic Behavior*. Princeton New Jersey: Princeton University Press.

Nilwan, Agustinus. 1998. *Pemrograman Animasi dan Game Profesional 4*. Jakarta: Elex Media Komputindo.

Peter Hart, Nils Nilsson dan Bertram Raphael. 1968. *A Formal Basic for the Heuristic Determination of MinimumCost Paths.* http://ai.stanford.edu/~nilsson/OnlinePubs-ils/PublishedPapers/astar.pdf diakses pada 15 Maret 2016.

Rendi Christian, Deni Saepudin dan Bedy Purnama. 2011. *Penentuan Lokasi Parkir Pada Smart Parking Systemmenggunakan Dynamic Weighting A* (DWA*).* https://repository.telkomuniversity.ac.id/pustaka/95458/penentuan-lokasi-parkir-pada-smart-parking-system-menggunakan-dynamic-weighting- a-dwa-.html diakses pada 2 Desember 2015.

Riyadi, Puanta Della Maharani. 2009. *Algoritma Pencarian A* dengan Fungsi Heuristik Jarak Manhattan*. Makalah. Bandung: Intitut Teknologi Bandung.

Riwinoto dan Alfian. 2014. *Implementasi Pathfinding dengan Algoritma A* pada Game Funny English Menggunakan Unity 3D Berbasis Graf.* http://p2m.polibatam.ac.id/wp-content/uploads/2015/01/Alfian.pdf diakses pada 19 November 2015.

Rollings, Andrew dan Ernest Adams. 2006. *Fundamental of Game Design (Game Design and Development Series)*. Unite States: Prentice Hall

Ryan Nugraha, Edo Exridores dan Hendri Sopryadi. 2014. *Penerapan Algoritma Fisher-Yates Shuffle Pada Aplikasi The Lost Insect Untuk Pengenalan Jenis Serangga Berbasis Unity 3D.* http://eprints.mdp.ac.id/1369/ diakses pada 11 November 2015.

Seno, dkk. 2014. *Mudah Membuat Game 3 Dimensi Menggunakan Unity 3D*. Yogyakarta: Andi.

Singh, Vinay. 2014. *Shuffle an array by modern Fisher–Yates method.* http://www.vinaysingh.info/fisher-yates-shuffle/ diakses 15 Maret 2016.

Supriyanto, Berry Priangga dan Yoannita. 2014. *Penerapan Algoritma Fisher-Yates Shuffle pada Edugame Guess Caculation Berbasis Android*. http://eprints.mdp.ac.id/1254/ diakses pada 11 November 2015.

Suyanto. 2011. *Artificial Intelligence – Serching, Reasoning, Planning dan Learning*. Bandung: Informatika.

Tim Penyusun Kamus Besar Bahasa Indonesia. 1990. *Kamus Besar Bahasa Indonesia*. Jakarta: Balai Pustaka.

Xiaoxun Sun, Mare J. Druzdzel dan Changhe Yuan. 2004. *Dynamic Weighting A* Search-based MAP Algorithm for Bayesian Networks.* http://www.pitt.edu/~druzdzel/psfiles/pgm06c.pdf diakses pada 1 Desember 2015.

https://almanhaj.or.id/3026-iman-kepada-para-rasul-allah.html diakses pada 2 November 2015.

https://almanhaj.or.id/3833-pentingnya-belajar-dari-sejarah.html diakses pada 2 November 2015.

http://ceritaislami.net/category/cerita-nabi/ diakses pada 2 November 2015.

https://kisahmuslim.com/category/kisah-nyata/kisah-nabi-dan-rasul diakses pada 2 November 2015.

http://laksmitadewi.esy.es/?page_id=83 diakses pada 2 November 2015.

http://www.bimbingan.org/25-mukjizat-nabi.htm diakses pada 2 November 2015.

http://www.hermantolle.com/class/docs/unity-3d-game-engine/ diakses pada 15 Desember 2015

http://www.masuk-islam.com/nama-nama-25-nabi-lengkap-dengan-riwayat-dan-kisahnya.html diakses pada 2 November 2015.