# IMPLEMENTASI ALGORITMA A* DAN GERAK LURUS BERUBAH BERATURAN (GLBB) SEBAGAI DASAR PERGERAKAN PADA *NON-PLAYABLE CHARACTER* (NPC) DALAM *GAME* 3D MAZE

**Skripsi**

**diajukan sebagai salah satu persyaratan untuk memperoleh gelar Sarjana Pendidikan Program Studi Pendidikan Teknik Informatika dan Komputer**


**Oleh**

**Diah Novianti**

**NIM.5302415018**

**PENDIDIKAN TEKNIK INFORMATIKA DAN KOMPUTER**

**JURUSAN TEKNIK ELEKTRO**

**FAKULTAS TEKNIK**

**UNIVERSITAS NEGERI SEMARANG**


**2020**

# PERSETUJUAN PEMBIMBING

**PERSETUJUAN PEMBIMBING**

Nama : Diah Novianti

NIM : 5302415018

Program Studi : Pendidikan Teknik Informatika dan Komputer, S1.

Judul : IMPLEMENTASI ALGORITMA A* DAN GERAK LURUS BERUBAH BERATURAN (GLBB) SEBAGAI DASAR PERGERAKAN PADA *NON-PLAYABLE CHARACTER* (NPC) DALAM *GAME* 3D *MAZE*

Skripsi ini telah disetujui oleh pembimbing untuk diajukan ke sidang panitia ujian Skripsi Pprogram Studi Pendidikan Teknik Informatika dan Komputer Fakultas Teknik Universitas Negeri Semarang.

Semarang, 13 Januari 2020
Pembimbing I,

Alfa Faridh Suni, S.T., M.T.
NIP. 198210192014041001

# PENGESAHAN

PENGESAHAN

Skripsi dengan judul Implementasi Algoritma A* dan Gerak Lurus Berubah Beraturan (GLBB) sebagai Dasar Pergerakan pada *Non-Playable Character* (NPC) dalam *Game* 3D *Maze* telah dipertahankan di depan sidang Panitia Ujian Skripsi Fakultas Teknik UNNES pada tanggal 21 bulan Januari tahun 2020.

Oleh

Nama : Diah Novianti
NIM : 5302415018
Program Studi : Pendidikan Teknik Informatika dan Komputer

Panitia:

Ketua

Ir. Ulfah Mediaty Arief, M.T. IPM.
NIP.196605051998022001

Sekretaris

Budi Sunarko, S.T., M.T., Ph.D
NIP.197101042006041001

Penguji 1

Dr. Hari Wibawanto, M.T.
NIP. 196501071991021001

Penguji 2

Ir. Ulfah Mediaty Arief, M.T., IPM.
NIP. 196605051998022001

Penguji 3/Pembimbing

Alfa Faridh Suni, S.T., M.T.
NIP.198210192014041001

Mengetahui,
Dekan Fakultas Teknik UNNES

Dr. Nur Qudus, M.T., IPM
NIP.196911301994031001



ii

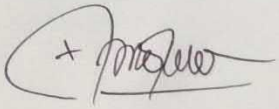
## PERNYATAAN KEASLIAN

PERNYATAAN KEASLIAN

Dengan ini saya menyatakan bahwa:

1. Skripsi/TA ini, adalah asli dan belum pernah diajukan untuk mendapatkan gelar akademik (sarjana, magister, dan/atau doktor), baik di Universitas Negeri Semarang (UNNES) maupun di perguruan tinggi lain.
2. Karya tulis ini adalah murni gagasan, rumusan, dan penelitian saya sendiri, tanpa bantuan pihak lain, kecuali arahan Pembimbing dan masukan Tim Penguji.
3. Dalam karya tulis ini tidak terdapat karya atau pendapat yang telah ditulis atau dipublikasikan orang lain, kecuali secara tertulis dengan jelas dicantumkan sebagai acuan dalam naskah dengan disebutkan nama pengarang dan dicantumkan dalam daftar pustaka.
4. Pernyataan ini saya buat dengan sesungguhnya dan apabila di kemudian hari terdapat penyimpangan dan ketidakbenaran dalam pernyataan ini, maka saya bersedia menerima sanksi akademik berupa pencabutan gelar yang telah diperoleh karena karya ini, serta sanksi lainnya sesuai dengan norma yang berlaku di perguruan tinggi ini.

Semarang, 13 Januari 2020
Yang membuat pernyataan,



Diah Novianti
NIM. 5302415018

iii

## MOTTO

AL JAZA MIN JINSIL A'MAL
"Setiap balasan adalah tergantung pada perbuatan"

## PERSEMBAHAN

Skripsi ini dipersembahkan untuk:

1. Bapak Siswanto (Ayah), Ibu Rokayah (Ibu), Yusril Subhanandi (Adik), Syifa Amalia Putri (Adik), Keluarga Besar Abah Sakrim dan Alhm. Emi.
2. Malika, Dila, Tiara, Maria, Ayunita, Dea, Maya, Edi, Dzakir, Dhuka, Ressa, Mahfud teman selama penelitian dan penyusunan.
3. Aswi, Debby dan keluarga kos Hamna binti Jahsy.
4. Teman-teman PTIK 2015.

# SARI

Diah Novianti.2020. Implementasi Algoritma A* dan Gerak Lurus Berubah Beraturan (GLBB) sebagai Dasar Pergerakan pada *Non-Playable Character* (NPC) dalam *Game* 3D *Maze*. Alfa Faridh Suni, S.T., M.T. Pendidikan Teknik Infomatika dan Komputer.

Perkembangan dunia *game* semakin pesat dari waktu ke waktu. Salah satu komponen penting dalam sebuah *game* adalah karakter musuh atau *Non-Playable Character*. Realitas sebuah *game* akan meningkat dengan menggunakan *Artificial Intelegence.* Salah satunya dengan mengimplementasikan algoritma pathfinding pada *Non-Playable Character* sebagai kecerdasan buatan dalam mencari rute menuju *player*.

Algoritma A* mampu menemukan rute terdekat dan tercepat dengan mempertimbangkan nilai heuristiknya. Fungsi heuristik adalah perkiraan biaya minimum dari setiap simpul n ke tujuan. Sangat penting untuk memilih fungsi heuristik yang baik. Pada penelitian ini fungsi heuristik yang digunakan adalah *Manhattan Distance* karena dapat diterapkan pada representsasi *grid* dan efektif dalam mempertimbangkan heuristik dalam algoritma A*. Jalur dihasilkan pada proses *pathfinding* algoritma A* merupakan rute yang harus dilewati *Non-Playable Character*. Pergerakan *Non-Playable Character* menuju *player* ini mempertimbangkan kecepatan *movement Non-Playable Character*, serta waktu tempuh yang dibutuhkan untuk mencapai titik tujuan. Penulis mengimplementasikan konsep Gerak Lurus Berubah Beraturan pada pergerakan NPC, untuk meningkatkan kecepatan NPC pada lintasan lurus.

Hasil percobaan menunjukan bahwa algoritma A* dapat diimplementasikan dengan baik sebagai *pathfinding*. Penambahan konsep pergerakan Gerak Lurus Berubah Beraturan pada *Non-Playable Character* mampu meningkatkan kecepatan *Non-Playable Character* pada setiap lintasan lurus dibandingkan tanpa penambahan rumus Gerak Lurus Berubah Beraturan. Waktu tempuh yang dibutuhkan *Non-Playable Character* untuk mencapai titik tujuan juga lebih cepat dengan menggunakan rumus Gerak Lurus Berubah Beraturan dibandingkan tidak menggunakannya.

**Kata kunci**: *Algoritma A*, Gerak Lurus Berubah Beraturan, Non-Playable Character*

# PRAKATA

Segala puji dan syukur penulis ucapkan kehadirat Allah SWT yang telah melimpahkan rahmat-Nya sehingga penulis dapat menyelesaikan Skripsi/TA yang berjudul Implementasi Algoritma A* dan Gerak Lurus Berubah Beraturan (GLBB) Sebagai Dasar Pergerakan Pada *Non-Playable Character* (NPC) dalam *Game* 3D *Maze*. Skripsi/TA ini disusun sebagai salah satu persyaratan meraih gelar Sarjana Pendidikan pada Program Studi S1 Pendidikan Teknik Informatika dan Komputer Universitas Negeri Semarang. Shalawat dan salam disampaikan kepada Nabi Muhammad SAW, mudah-mudahan kita semua mendapatkan safaat Nya di yaumil akhir nanti, Aamiin.

Penyelesaian karya tulis ini tidak lepas dari bantuan berbagai pihak, oleh karena itu pada kesempatan ini penulis menyampaikan ucapan terima kasih serta penghargaan kepada:

1. Prof. Dr. Fathur Rokhman, M.Hum, Rektor Universitas Negeri Semarang atas kesempatan yang diberikan kepada penulis untuk menempuh studi di Universitas Negeri Semarang.
2. Dr. Nur Qudus, MT, IPM, Dekan Fakultas Teknik, Ir. Ulfah Mediaty Arief, M.T., Ketua Jurusan Teknik Elektro, Budi Sunarko S.T, M.T, Ph.D, Koordinator Program Studi Pendidikan Teknik Informatika dan Komputer atas fasilitas yang disediakan bagi mahasiswa.
3. Alfa Faridh Suni S.T, MT., Pembimbing 1 yang penuh perhatian dan atas perkenaan memberi bimbingan dan dapat dihubungi sewaktu-waktu disertai kemudahan menunjukkan sumber-sumber yang relevan dengan penulisan karya ini.
4. Dr. Hari Wibawanto M.T., Ir. Ulfah Mediaty Arief, M.T., Penguji 1 dan 2 yang telah memberi masukan yang sangat berharga berupa saran, ralat, perbaikan, pertanyaan, komentar, tanggapan, menambah bobot dan kualitas karya tulis ini.
5. Semua dosen Jurusan Teknik Elektro FT. UNNES yang telah memberi bekal pengetahuan yang berharga.
6. Berbagai pihak yang telah memberi bantuan untuk karya tulis ini yang tidak dapat disebutkan satu persatu.

Penulis berharap semoga Skripsi/TA ini dapat bermanfaat untuk pelaksanaan penelitian selanjutnya.

Semarang, 13 Januari 2020

Penulis

# DAFTAR ISI

Halaman

Halaman

# DAFTAR TABEL

Halaman

# DAFTAR GAMBAR

Halaman

Halaman

Halaman

## DAFTAR LAMPIRAN

Halaman

# BAB I

# PENDAHULUAN

## 1.1 Latar Belakang

Industri *game* merupakan perwujudan industri kreatif yang ada pada saat ini. Seiring berkembangnya teknologi, *game* sebagai sarana hiburan juga memiliki variasi dan inovasi yang semakin luas. Di Indonesia, aplikasi *game* sangat digemari dan hampir tidak terhitung jumlah *game* yang telah dikembangkan. Pada awalnya teknologi gambar yang digunakan pada *game* masih 2 Dimensi dengan pergerakan yang terbatas. Namun, pada saat ini banyak *game* yang dibangun dengan teknologi 3 Dimensi, dimana detil sebuah citra akan lebih baik jika divisualisasikan dengan pendekatan 3 Dimensi (Tayal, 2012).

Pada dasarnya, dalam sebuah *game* memiliki beberapa komponen penting yaitu skenario (alur cerita), level (tingkatan), skor (nilai), karakter, dan *obstacle* (halangan) (Atmaja, Siahaan, & Kuswardayan, 2016). Karakter dalam sebuah *game* ini meliputi *player character* dan *Non-Playable Character* (NPC). Agen *Non-Playable Character* (NPC) merupakan objek karakter yang tidak dapat dikendalikan oleh pemain. NPC pada *game* dapat meningkatkan realitas *game* dengan menambahkan *Artificial Intelligence* (AI) atau kecerdasan buatan. Terdapat banyak metode AI yang dapat diimplementasikan pada NPC, diantaranya yaitu *pathfinding* dan pergerakan atau *movement* NPC. Algoritma pencarian rute atau *pathfinding* dibutuhkan NPC untuk menemukan jalan menuju target atau *player* dalam *game* (Febliama, 2018). Algoritma *pathfinding*

berkaitan dengan menemukan jalur terpendek dari titik awal menuju titik tujuan dengan memperhatikan hambatan (Kallem, 2012). Terdapat beberapa algoritma pencarian untuk menemukan solusi pencarian jalur tercepat, diantaranya adalah algoritma Djikstra, *Best First Search*, A*, dan lain lain.

Algoritma A* merupakan algoritma pencarian jalur yang populer dalam *Game Artificial Intelligence* (AI) (Cui & Shi, 2011). Algoritma pencari jalan yang lain, seperti Djikstra mampu memastikan bahwa jalan yang dipilih adalah yang terpendek. Namun pada penerapannya Djikstra dinilai lamban (Zainulhayat, 2017). Algoritma lain, yaitu BFS (*Best Fisrt Search*) mampu mencari jalan lebih cepat, namun bukan jalur terpendek (Putrady, 2009). Algoritma A* menggabungkan kedua unsur dari Djikstra dan BFS, dimana A* mampu mencari jalur terpendek dengan waktu yang hampir secepat BFS. Hal ini karena algoritma A* mempertimbangkan nilai heuristik, yaitu nilai optimasi yang menjadikan algoritma A* lebih baik dari pada algoritma lainnya (Sharma & Pal, 2015).

Pencarian rute terpendek dengan algoritma A* telah diterapkan di berbagai dibidang untuk mengoptimasi kinerja suatu sistem, baik untuk meminimalkan biaya atau mempercepat jalannya suatu proses. Algoritma ini juga yang paling sering digunakan untuk menemukan jalur optimal pada sebuah jalan yang menggunakan fungsi heuristik khususnya dalam hal pengembangan dan pemeriksaan *node* pada peta (Witanti, 2013). Dalam terminologi standar algoritma A* yakni, g(n) yang mewakili biaya dari jarak antara titik awal ke titik (n) dan h(n) yang mewakili biaya perkiraan heuristik dari titik (n) ke titik tujuan.

Kecepatan serta akurasi dalam *pathfinding* algoritma A* sangat ditentukan oleh penggunaan fungsi heuristik dan biaya perpindahan (Harabor & Grastien, 2014). Fungsi heuristik pada algoritma A* menjadi dasar pertimbangan mengestimasi jarak terdekat untuk mencapai titik tujuan (Syukriyah, 2016). Fungsi heuristik yang banyak digunakan yaitu *Manhattan Distance*. Fungsi ini dapat diterapkan pada simulasi *grid* dan efektif dalam mempertimbangkan heuristik dalam algoritma A* (Attoyibi, 2019).

Algoritma A* adalah algoritma yang paling baik dalam pencarian jalur pada *game grid maze* (Dian, Permana, Bayu, Bintoro, & Arifitama, 2018). Jalur yang ditemukan hasil dari *pathfinding* algoritma A* menjadi rute yang harus ditempuh NPC menuju titik tujuan yaitu *player*. Dalam hal ini, NPC bergerak melalui 8 arah direksi pada representasi *grid*. Pergerakan NPC dalam mengejar dipengaruhi oleh kecepatan, percepatan, jarak serta waktu tempuh yang diperlukan untuk mencapai target. NPC dalam mengejar *player* dapat lebih cerdas dengan adanya variasi perubahan kecepatan pada kondisi tertentu. Pada Gerak Lurus Berubah Beraturan (GLBB) dimana objek pada lintasan lurus dengan percepatan tetap, maka kecepatannya akan berubah secara teratur seiring waktu (Shin, Jang, & Lee, 2012). Konsep pergerakan ini akan diimplementasikan pada saat NPC mulai mengejar *player* dalam *game*. Hasil dari proses *pathfinding* algoritma A*, jalur yang membentuk lintasan lurus dan arah serta percepatan yang tetap, maka NPC akan bergerak semakin semakin cepat. Sedangkan pada lintasan yang tidak lurus maka NPC akan kembali ke kecepatan awal.

Berdasarkan penjelasan pada latar belakang tersebut, maka penulis melaksanakan penelitian yang berjudul “Implementasi Algoritma A* dan Gerak Lurus Berubah Beraturan (GLBB) Sebagai Dasar Pergerakan Pada *Non-Playable Character* (NPC) dalam *Game* 3D Maze**“**. Penelitian ini akan mengimplementasikan algoritma A* sebagai solusi *pathfinding* pada NPC dalam *game* 3 Dimensi yang akan dibangun serta penerapan GLBB dalam *movement* NPC untuk meningkatkan performa kecepatan dalam mengejar *player* menjadi adaptif.

## 1.2 Identifikasi Masalah

Berdasarkan penjabaran latar belakang yang telah dipaparkan, maka dapat diidentifikasikan masalah dalam penelitian ini adalah sebagai berikut:

1.2.1 Dalam pengembangan *game* ini akan dibahas bagaimana penggunaan algoritma A* pada pergerakan NPC terhadap *player* dalam *game* yang dapat mengatur pencarian rute terpendek dengan waktu yang singkat.

1.2.2 Pada pengimplementasiannya bagaimana algoritma A* mampu berjalan dengan *obstacle* atau halangan.

1.2.3 Dalam penerapannya, bagaimana NPC mampu meningkatkan kecepatan bergerak pada kondisi ketika lintasan lurus pada *scene* permainan menggunakan konsep Gerak Lurus Berubah Beraturan (GLBB).

## 1.3 Batasan Masalah

Pembatasan masalah pada penelitian ini dilakukan untuk memfokuskan pada permasalahan penelitian yang diteliti sehingga tidak keluar atau meluas dari permasalahan diluar penelitian. Adapun batasan masalah pada penelitian ini adalah:

1.3.1 *Game* yang dibangun adalah *game* 3D (3 dimensi) bergenre *maze* atau labirin dengan misi mengumpulkan 7 poin untuk menang.

1.3.2 Penelitian ini membahas tentang implementasi algoritma A* sebagai teknik pencarian rute NPC (*Non-Playable Character*) terhadap *player* dalam *game*.

1.3.3 Metode heuristik dalam algoritma A* yang digunakan adalah metode *Mahanttan Distance* dengan mengijinkan rute diagonal.

1.3.4 Rute jalan yang dilalui NPC (*Non-Playable Character*) pada *game* direpresentasikan dengan *grid*.

1.3.5 Jarak yang ditempuh NPC (*Non-Playable Character*) diasumsikan kedalam banyaknya *grid* yang dilalui jalur.

1.3.6 Ukuran *grid* pada penelitian ini adalah 30 x 30.

1.3.7 NPC (*Non-Playable Character*) pada satu *scene* permainan berjumlah 1 (satu).

1.3.8 *Obstacle* atau halangan yang dibuat pada *game* adalah hambatan statis yang sudah ditentukan pada area permainan berjumlah 10 (sepuluh) dalam satu *scene* permainan.

1.3.9 Modifikasi diterapkan pada perubahan kecepatan *movement* NPC. Dengan mengimplementasikan juga konsep Gerak Lurus Berubah Beraturan (GLBB) dipercepat untuk mengatur kecepatan NPC agar menjadi adaptif, yaitu dengan mempertimbangkan .

1.3.10 Pada penelitian ini, pembahasan yang diteliti pada perancangan *game* sampai tahap *testing* oleh developer untuk mengetahui hasil implementasi algoritma A* dan perubahan kecepatan serta waktu tempuh NPC menuju target.

## 1.4 Rumusan Masalah

Berdasarkan pokok permasalahan diatas, rumusan masalah dari penelitian ini yaitu:

1. Bagaimana implementasi algoritma A* dan Gerak Lurus Berubah Beraturan (GLBB) pada pergerakan *Non-Playable Character* (NPC) menuju *player* dengan adanya *obstacle* dalam *game* 3D *maze* ?
2. Sejauh mana tingkat optimalisasi diterapkannya rumus GLBB (Gerak Lurus Berubah Beraturan) terhadap peningkatan kecepatan dan mempercepat waktu tempuh pada pergerakan NPC menuju *player* ?

## 1.5 Tujuan Penelitian

Penelitian ini bertujuan untuk:

1. Mengimplementasikan algoritma A* dan Gerak Lurus Berubah Beraturan (GLBB) pada pergerakan *Non-Playable Character* (NPC) menuju *player* dengan adanya *obstacle* dalam *game* 3D *maze*.
2. Mengetahui sejauh mana optimalisasi diterapkannya rumus GLBB (Gerak Lurus Berubah Beraturan) dipercepat terhadap peningkatan kecepatan dan mempercepat waktu tempuh pada pergerakan NPC menuju *player*.

## 1.6 Manfaat Penelitian

Penelitian ini diharapkan dapat memberikan manfaat antara lain sebagai berikut:

### 1.6.1 Manfaat Teoritis

Memberikan gambaran jelas implementasi algoritma A* sebagai pencarian rute terpendek oleh NPC (*Non-Playable Character*) dan optimalisasi pergerakan NPC dengan diterapkannya rumus GLBB (Gerak Lurus Berubah Beraturan) dipercepat pada *game* 3D *maze*.

### 1.6.2 Manfaat Praktis

a. Bagi Peneliti

Penelitian ini diharapkan dapat menambah wawasan, pengetahuan tentang pencarian rute terpendek dan penerapan algoritma A* serta pengimplementasian rumus GLBB untuk pergerakan NPC pada aplikasi *game* 3D *maze*.

b. Bagi Universitas Negeri Semarang

Penelitian ini diharapkan dapat digunakan sebagai bahan referensi untuk nantinya dapat dijadikan sebagai acuan pada penelitian selanjtnya serta dapat dijadikan tolak ukur keberhasilan akademik dalam mendidik dan memberikan ilmu pengetahuan.

# BAB II

# KAJIAN PUSTAKA DAN LANDASAN TEORI

## 2.1 Kajian Pustaka

Penelitian yang berjudul "Penerapan Algortitma A* (Star) untuk Mencari Rute Tercepat dengan Hambatan" oleh Yenie Yukriyah, Falahah, dan Herni Solihin pada tahun 2016. Penelitian berisi simulasi yang dilakukan dengan bahasa pemrograman Java. Tujuan utama penelitian ini mempelajari cara kerja algoritma A* dalam mencari jarak tercepat, yang disimulasikan seperti kondisi ketika seorang mencari rute dalam keadaan jalanan macet. Simulasi ini memberikan gambaran yang lebih realistis terhadap perilaku algoritma A* dalam pencarian jarak tercepat. Hasil pengujian dikatakan bahwa rute yang ditemukan merupakan rute yang terbaik dengan nilai f(n) terkecil dibandingkan dengan jalur-jalur lainnya.

Penelitian yang berjudul "Pencarian Jalur Terpendek Pada Permainan Pacman Menggunakan Algoritma A*" oleh Darmawan Aditama, Nuniek Fahriani dan Putri Aiyiyah Rakhma Devi pada tahun 2018. Pada penelitian ini mengimplementasikan algoritma A* pada *enemy* atau NPC *game* dengan *genre Arcade* yaitu Pacman. Hasil penelitian ini menyebutkan bahwa kemampuan NPC yang diberi algoritma A* dapat menemukan *player* semakin akurat dan efektif. Namun, belum ditentukan pembatasan jarak pada sensor kecerdasan buatan A* sehingga *player* masih dapat memenangkan permainan dengan mudah.

Penelitian yang berjudul "Penerapan Algoritma A Star (A*) pada *Game* Petualangan Labirin Berbasis Android" oleh Imam Ahmad dan Wahyu Widodo pada tahun 2017. Penelitian ini membahas implementasi algoritma A* pada *game* 2D labirin. Algoritma A* dibutuhkan untuk menemukan jalur tercepat menuju pintu keluar labirin, dengan metode pencarian heuristik yaitu *euclidean distance*. Hasil penelitian menunjukan efisiensi penggunaan algoritma A* dalam mencari jalan keluar pada *game* Labirin tersebut tanpa ada musuh atau mengejar target bergerak lainnya.

Penelitian yang berjudul "*Comparative Analysis of Pathfinding Algorithms A*, Dijkstra , and BFS on Maze Runner Game*" oleh Silvester Dian Handy Permana, Ketut Bayu Yogha Bintoro, Budi Arifitama, Ade Syahputra pada tahun 2018 mengenai perbandingan penggunaan algoritma *pathfinding* A*, Djikstra dan *Best First Search* (BFS) pada *game maze*. Pada penelitian tersebut membandingkan panjang *path* yang ditemukan dan waktu komputasi pencarian rute, tanpa pembahasan bagaimana objek dapat melewati rute tersebut. Hasil percobaannya menunjukan bahwa algoritma A* adalah algoritma *pathfinding* terbaik, khususnya pada *game maze* atau *grid*. Hal itu didukung dengan waktu komputasi yang relatif rendah.

Berdasarkan pada kajian penelitian yang relevan diatas, maka penelitian ini akan membahas implementasi algoritma A* pada *game maze* 3 Dimensi. NPC (*Non-Playable Character*) dapat mengejar *player* dengan menerapkan pergerakan Gerak Lurus Berubah Beraturan pada kondisi tertentu dalam *game*.

## 2.2 Landasan Teori

### 2.2.1 Permainan/*Game*

*Game* atau permainan merupakan aktifitas yang biasanya bertujuan untuk hiburan dan juga sebagai sarana pendidikan. Karakteristik *game* yang menyenangkan, memotovasi, membuat kecanduan dan kolaboratif membuat aktifitas ini digemari banyak orang (Wahono, R.S., 2009). Berdasarkan grafis, permainan dibagi menjadi 2 yaitu *game* 2D dan 3D. Permainan/ *Game* 2D biasanya termasuk permainan yang ringan dan tidakmembani sistem. Kelemahan pada grafis 2D ini adalah kualitas gambar yang kurang enak dilihat jika dibandingkan dengan permainan 3D. Pada permainan bertipe 3D memiliki grafis yang lebih baik dalam penggambaran sehingga mirip dengan realita. Biasanya dalam permainan grafis 3D memiliki sudut pandang hingga 360 derajat sehingga kita bisa melihat keseluruhan dunia dalam permainan tersebut (Akbar, 2012). Dengan demikian, perancangan *game* yang akan dibangun dalam penelitian ini menggunakan grafis 3 Dimensi.

### 2.2.2 Algoritma *Pathfinding*

*Pathfinding* merupakan salah satu materi yang sangat penting didalam *Artificial Intelligence*. *Pathfinding* biasanya digunakan untuk menyelesaikan masalah pada sebuah *graph*. *Edge* yang menghubungkan setiap *node* merupakan suatu *vektor* yang memiliki arah dan besaran tertentu. Untuk dapat menemukan jalan dari *node* awal menuju *node* tujuan, dilakukan penelusuran terhadap *graph* tersebut. Graf pada *game* merupakan graf berarah dan berbobot. Penelusuran

biasanya dilakukan dengan mengikuti arah *edge* yang menghubungkan antar *node* (Setiawan, 2018).

Secara umum algoritma *pathfinding* digolongkan menjadi dua jenis :

1. Algoritma *Un-infromed Pathfinding*

   Algoritma *Un-infromed Pathfinding* adalah algoritma yang tidak memiliki keterangan tentang jarak atau biaya dari path dan tidak memiliki pertimbangan akan path mana yang lebih baik, seperti algoritma *Breadth-First-Search*.

2. Algoritma *Informed Search*

   Algoritma *Informed Search* adalah algoritma yang memiliki keterangan tentang jarak atau biaya dari path dan memiliki pertimbangan berdasarkan pengetahuan akan path mana yang lebih baik, seperti algoritma *Djikstra* dan algoritma A*.

### 2.2.3 Pencarian Heuristik

Kata heuristik berasal dari sebuah kata kerja bahasa Yunani, *heuriskein*, yang berarti “Mencari atau Menemukan”. Dalam dunia pemrograman, sebagian orang menggunakan kata *heuristic* sebagai lawan kata dari algoritma, dimana kata heuristik ini diartikan sebagai suatu proses yang mungkin dapat menyelesaikan suatu masalah tetapi tidak ada jaminan bahwa solusi yang dicari selalu dapat ditentukan. Didalam mempelajari metode-metode pencarian ini, kata heuristik diartikan sebagai suatu fungsi yang memberikan suatu nilai berupa biaya perkiraan (estimasi) dari suatu solusi. Teknik pencarian heuristik *(heuristic*

*searching)* merupakan suatu strategi untuk melalukan proses pencarian secara selektif dan dapat memandu proses pencarian yang memiliki kemungkinan sukses paling besar, namun dengan kemungkinan mengorbankan kelengkapan *(completeness).* Menerapkan pencarian heuristik diperlukan suatu fungsi heuristik. Fungsi heuristik adalah aturan-aturan yang digunakan untuk mendapatkan solusi yang diinginkan (Setiawan, 2018). Fungsi heuristik yang digunakan pada algoritma A* meliputi *Manhattan Distance*, *Diagonal Distance*, dan *Euclidean Distance* (Patel, Amit J., 2007, p1) :

#### 2.2.3.1 *Manhattan Distance*

Metode *Manhattan Distance* merupakan standar nilai heuristik yang digunakan dalam *pathfinding*. Metode heuristik ini memiliki proses yang cepat dibandingkan dengan metode heuristik yang lain. Pada representasi *map Grid*, metode heuristik yang paling baik adalah metode *manhattan distance* karena mampu menemukan jalur terpendek dengan maksimal. Rumus umum metode *Manhattan Distance* sebagai berikut :

h(n) = abs(Xn – Xgoal) + abs (Yn – Ygoal) ..... (1)

Dimana:

- Xn adalah koordinat X dari *node* pertama pada *Grid*
- Xgoal adalah koordinat X dari *final node*
- Yn adalah koordinat dari *node* pertama pada *Grid*
- Ygoal adalah koordinat Y dari *final node*

Kelebihan menggunakan metode heuristik *Manhattan Distance* yaitu semua jalur dapat ditemukan. Hal ini karena pada setiap penambahan nilai g(n), pada perhitungan nilai heuristiknya. Sehingga dengan penambahan nilai g(n),

tidak mempengaruhi nilai pencarian jalur. Dengan menggunakan fungsi heuristik *manhattan distance* didapatkan nilai iterasi dan jumlah langkah yang paling kecil dibanding dengan menggunakan fungsi heuristik yang lain.

#### 2.2.3.2 *Euclidean Distance*

*Euclidean Distance* adalah fungsi heuristik yang digunakan pada aplikasi yang dapat bergerak ke segala arah/sudut. Pada algoritma A* yang berupa representasi graf/point maka penggunaan fungsi ini *straight line distance* menjadi tepat karena akan menghasilkan *cost* yang lebih kecil dibanding dengan *manhattan* distance namun membutuhkan waktu pencarian lebih lama karena memiliki iterasi yang lebih banyak (Harabor & Grastien, 2014). Rumus umum *euclidean distance* adalah:

$h(n) = sqrt((Xn - Xgoal)^2 + (Yn – Ygoal)^2)$ ..... (2)

Dimana:

- Xn adalah koordinat X dari *node* pertama pada *Grid*
- Xgoal adalah koordinat X dari final *node*
- Yn adalah koordinat dari *node* pertama pada *Grid*
- Ygoal adalah koordinat Y dari final *node*

#### 2.2.3.3 *Diagonal Distance*

*Diagonal Distance* adalah fungsi heuristik yang digunakan pada aplikasi memiliki delapan arah gerakan. Rumus *diaginal distance* adalah:

h(n) = d* max abs ((Xn – Xgoal), abs (Yn – Ygoal)) ..... (3)

Pencarian jalur dengan *Diagonal Distance* dapat menemukan semua jalur. Jumlah iterasi dan jumlah langkah yang didapat pada pengujian dengan fungsi ini lebih sedikit dibanding dengan fungsi *straight line distance,* tetapi lebih besar dibanding dengan fungsi *manhattan distance* .

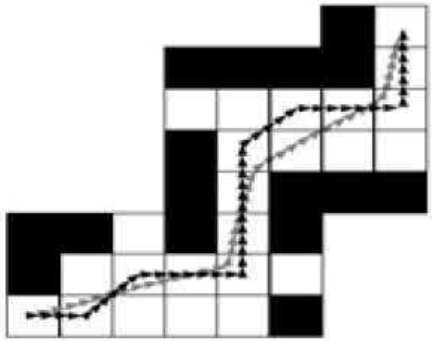
### 2.2.4 Representasi *Map*

*Map* atau peta dalam *game* membantu *Artificial Intelligent* (AI) bekerja lebih baik dalam keputusan dengan memberikan informasi tentang lingkungan, pemilihan representasi *map* menentukan kualitas *game* yang dibuat (Sabri, Haizan, Radzi, Samah, & Games, 2018). Representasi *map* dibutuhkan oleh algoritma *pathfinding* untuk menentukan *node* yang akan dihitung dalam proses algoritma pencarian rute tersebut (Sazaki, Satria, Primanita, & Syahroyni, 2018). Terdapat beberapa metode yang digunakan untuk merepresentasikan *map* pada permainan diantaranya adalah *Waypoint graph*, *Navigation mesh* dan *Grids*. Namun pada penelitian ini digunakan representasi *grid*.

#### 2.2.4.1 *Grid*

Representasi *Grid* sering disebut juga sebagai *tile graph*. *Grid* tersebut digunakan untuk membagi *map* menjadi *cell* yang teratur berbentuk kotak,

segitiga, atau heksagonal (Millington & Funge, 2009). *Map* direpresentasikan melalui *array* berdasarkan *cell grid* tertutup dan terbuka.

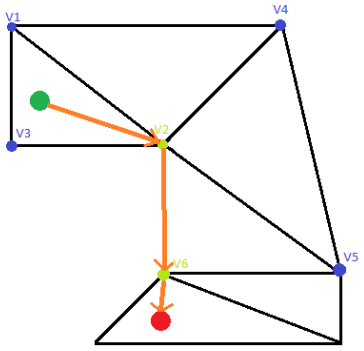


**Gambar 2.1 Representasi *Grid***

Pada proses pencarian rute dengan representasi *grids*, *node* dilakukan disetiap koordinat *cell grids*. Sehingga *grid* yang dibuat harus mencakup keseluruhan area atau *map* pada permainan (Sazaki et al., 2018). Perpindahan objek pada representasi *grid* bernilai 1, namun jika pepindahan diagonal dizinkan maka setiap perpindahannya benilai 1.4 atau √2. Jika perpindahan diagonal tidak diijinkan maka *grid* akan menggunakan 4 arah direksi, namun jika diijinkan maka *grid* akan menggunakan 8 arah direksi. *Grid map* populer karena beberapa alasan: (i) mudah dimengerti dan mudah diterapkan (ii) peta dapat direpresentasikan sebagai matriks bit dan disimpan secara efisien (iii) masing-masing node dapat diperbarui atau ditanyakan dalam waktu yang konstan (Harabor & Grastien, 2014). Maka dalam penelitian ini, akan digunakan representasi *grid* dengan 8 arah direksi pergerakan.

### 2.2.5 Algoritma A*

Algoritma A* (dibaca "*A star*") adalah algoritma komputer yang digunakan secara luas dalam mencari jalur (*pathfinding*) dan grafik melintang (*graph traversal*), proses *plotting* sebuah jalur melintang secara efisien antara titik-titik, disebut *node*. Algoritma A* juga merupakan algoritma yang paling populer untuk masalah *pathfinding* atau pencarian rute. Algoritma ini dikenalkan oleh Hart, Nilsson dan Raphael pada tahun 1967 melalui penelitian tentang *graph* (Me-cse, 2015). Algoritma A* menyelesaikan masalah yang menggunakan graf untuk perluasan ruang statusnya. Dengan kata lain digunakan untuk menyelesaikan permasalah yang bisa direpresentasikan dengan graf.




**Gambar 2.2** Representasi *Graf* Algoritma A*

Metode A* adalah metode yang merupakan hasil pengembangan dari metode dasar *Best First Search*. Metode ini mengevaluasi setiap titik dengan mengkombinasikan dengan g(n), nilai untuk mencapai titik n dari titik awal, dan h(n), nilai perkiraan untuk mencapai tujuan dari titik n tersebut (Sazaki et al., 2018). Algoritma ini menggunakan fungsi *distance plus cost* (biasanya di notasikan dengan f(n)) untuk menentukan urutan kunjungan pencarian *node* di dalam *tree*. Gabungan *distance plus cost* merupakan penjumlahan dari dua fungsi,

yaitu fungsi *path cost* (selalu dinotasikan dengan g(n), dimungkinkan bernilai heuristik ataupun tidak), dan sebuah kemungkinan penerimaan atas "perkiraan heuristik" jarak ke titik tujuan (dinotasikan dengan h(n)). Fungsi *path cost* g(n) adalah jumlah biaya yang harus dikeluarkan dari *node* awal menuju *node* tujuan dengan terlebih dahulu mencari rute yang tampaknya mempunyai kemungkinan besar untuk menuju ke arah tujuan, algoritma ini mengambil jarak perjalanan ke arah tujuan (dimana g(n) bagian dari heuristik adalah biaya dari awal).

Secara matematis, nilai fungsi evaluasi heuristik sebuah titik pada algoritma A* diberikan oleh:

**f(n) = g(n) + h(n)** ..... (4)

keterangan,

f(n) = Nilai *node* tiap simpul dari penjumlahan g(n) dan h(n).

g(n) = Biaya dari titik awal (*start node*) ke titik n.

h(n) = Estimasi biaya dari titik n ke titik tujuan (*goal node*).

n = Titik atau *node* ke-n

Beberapa terminologi dasar yang terdapat pada algoritma ini adalah:

a. *Starting point* adalah terminologi untuk posisi awal sebuah benda.
b. A adalah simpul yang sedang dijalankan dalam algoritma pencarian jalan terpendek.
c. Simpul adalah petak-petak kecil sebagai representasi dari area *pathfinding*. Bentuknya dapat berupa persegi, lingkaran, maupun segitiga.

d. *Open list* adalah tempat menyimpan data simpul yang mungkin diakses dari starting point maupun simpul yang sedang dijalankan.

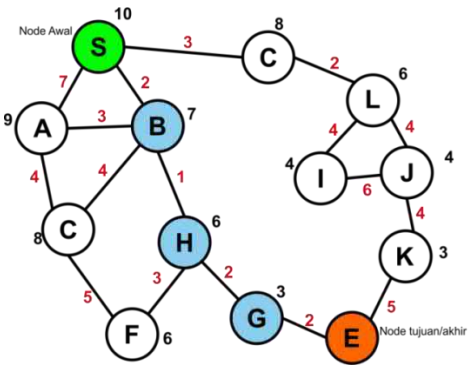
e. *Closed list* adalah tempat menyimpan data simpul sebelum A yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan.

f. Harga (*cost*) adalah nilai yang diperoleh dari penjumlahan, jumlah nilai tiap simpul dalam jalur terpendek dari starting point ke A, dan jumlah nilai perkiraan dari sebuah simpul ke simpul tujuan.

g. Simpul tujuan yaitu simpul yang dituju.

h. Halangan (*unwalkable*) adalah sebuah atribut yang menyatakan bahwa sebuah simpul tidak dapat dilalui oleh A.

Secara konseptual, berikut cara kerja algoritma A* menurut (Syukriyah, 2016) :




*Open list*

| Tujuan | Harga | Via |
|---|---|---|
| G | 8 | H |
| C | 11 | S |
| F | 12 | H |
| A | 14 | B |
| D | 14 | B |

*Closed list*

| Tujuan | Harga | Via |
|---|---|---|
| S | 10 | S |
| B | 9 | S |
| H | 9 | B |
| | | |

**Gambar 2.3** Contoh Pencarian Jalur pada *Graph* Algoritma A*

Dimana:

- Angka warna hitam adalah nilai heuristik (h(n))
- Angka warna merah adalah jarak antar *node* tetangga (g(n))
- *Node* S (berwarna hijau) adalah *starting point* atau titik awal

- *Node* E (berwarna jingga) adalah titik tujuan
- *Node* B-H-G (berwarna biru) adalah jalur yang ditemukan antara *node* S ke *node* E

Prinsip algoritma ini adalah mencari jalur terpendek dari sebuah simpul awal (*starting point*) menuju simpul tujuan dengan memperhatikan harga (f) terkecil. Pada Gambar 2.3, diawali dengan menempatkan S pada *starting point*, memasukkan seluruh simpul yang bertetangga dan tidak memiliki atribut rintangan dengan S ke dalam *open list*, mencari nilai h(n) terkecil dari simpul-simpul dalam *open list* tersebut, memindahkan S ke simpul yang memiliki nilai h(n) terkecil. Simpul sebelum S disimpan sebagai *parent* dari S dan dimasukkan ke dalam *closed list*. Jika terdapat simpul lain yang bertetangga dengan S (yang sudah berpindah) namun belum termasuk kedalam anggota *open list*, maka masukkan simpul-simpul tersebut ke dalam *open list*. Setelah itu, bandingkan nilai g(n) yang ada dengan nilai g(n) sebelumnya. Jika nilai g(n) sebelumnya lebih kecil maka S kembali ke posisi awal. Sehingga jalur yang ditemukan pada *graph* adalah S-B-H-G-E merupakan jalur tercepat. Simpul yang pernah dicoba dimasukkan ke dalam *closed list*. Hal tersebut dilakukan berulang-ulang hingga terdapat solusi atau tidak ada lagi simpul lain yang berada pada *open list* (Syukriyah, 2016). Algoritma inilah yang akan digunakan sebagai dasar pencarian rute terpendek dan cepat oleh NPC (*Non-Playable Character*) terhadap *player* dalam pengembangan *game* 3D *maze*.

#### 2.2.5.1 Sifat Algoritma A*

Pada dasarnya algoritma A* merupakan algoritma dengan struktur data graf berbobot (*weighted directed graph*), dimana setiap langkah atau perpindahan memiliki arah dan nilai bobot. Medan atau area yang dibangun dalam *pathfinding* dapat berupa area datar/*flat area* dan medan pegunungan. Medan area tersebut sangat memengaruhi kecepatan dalam proses *pathfinding*. Dalam pencariannya, algoritma A* mengatur kecepatan dengan memberi beban *cost* yang berbeda pada area *walkable*. (Patel, Amit J., 2007)

Besarnya waktu yang dibutuhkan sebuah algoritma untuk menyelesaikan suatu permasalahan selalu bergantung pada jumlah masukan yang harus diproses. Semakin besar data maka semakin besar pula waktu yang dibutuhkan. Tetapi, pada keadaan sebenarnya nilai dari suatu fungsi algoritma tergantung pada banyak faktor, misalnya kecepatan komputer, kualitas compiler dan kualitas program itu sendiri (Weiss, Mark Allen, 1996, p149). Pada algoritma A*, nilai heuristik juga sangat berpengaruh pada kompleksitas waktu yang digunakan dalam mencari jalur. Algoritma A* juga dapat menjamin keoptimalannya untuk sembarang heuristik, yang berarti bahwa tidak ada satupun algoritma lain yang mempergunakan heuristik yang sama ketika ada beberapa solusi parsial dimana h (nilai heuristik) dapat dengan tepat memprediksi ongkos jalur minimal.

#### 2.2.5.2 Metode Heuristik pada Algoritma A*

Algoritma A* tanpa fungsi heuristik yang baik akan memperlambat pencarian dan dapat menghasilkan rute yang tidak tepat. Fungsi heuristik juga sangat berpengaruh ke kecepatan *pathfinding* (Ferguson, Likhachev, & Stentz,

2005). Menurut Amit pada tahun 2010, fungsi heuristik sangat berpengaruh terhadap kelakuan algoritma A* , antara lain:

1. Apabila h(n) selalu bernilai 0, maka hanya g(n) yang akan berperan, dan A* berubah menjadi Algoritma Dijkstra, yang menjamin selalu akan menemukan jalur terpendek.
2. Apabila h(n) selalu lebih rendah atau sama dengan ongkos perpindahan dari titik n ke tujuan, maka A* dijamin akan selalu menemukan jalur terpendek. Semakin rendah nilai h(n), semakin banyak titik-titik yang diperiksa A*, membuatnya semakin lambat.
3. Apabila h(n) tepat sama dengan ongkos perpindahan dari n ke tujuan, maka A* hanya akan mengikuti jalur terbaik dan tidak pernah memeriksa satupun titik lainnya, membuatnya sangat cepat. Walaupun hal ini belum tentu bisa diaplikasikan ke semua kasus, ada beberapa kasus khusus yang dapat menggunakannya.
4. Apabila h(n) kadangkala lebih besar dari ongkos perpindahan dari n ke tujuan, maka A* tidak menjamin ditemukannya jalur terpendek, tapi prosesnya cepat.
5. Apabila h(n) secara relatif jauh lebih besar dari g(n), maka hanya h(n) yang memainkan peran, dan A* berubah menjadi BFS.

Pada penelitian ini, algoritma A* menggunakan metode *Manhattan Distance*. Perhitungan nilai heuristik untuk *node* ke-n menggunakan *manhattan distance* adalah h(n) = (abs(n(x)-goal(x))+ abs(n(y)-goal))). Untuk mendekati

kenyataan, *cost* untuk perpindahan *node* secara diagonal dan orthogonal dibedakan. *Cost* diagonal adalah 1,4 kali *cost* perpindahan secara orthogonal.

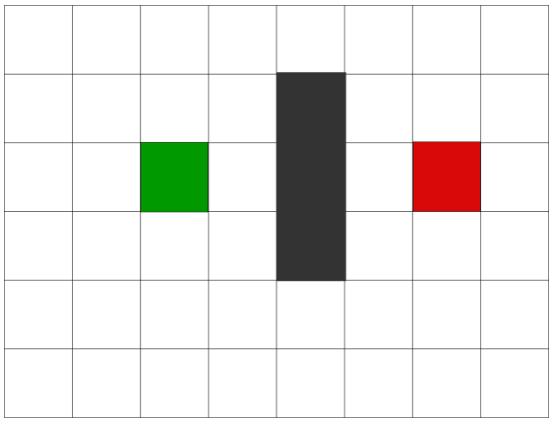
#### 2.2.5.3 Pseodocode Algoritma A*

1. Masukan *node* awal ke *openlist*.
2. Loop langkah-langkah dibawah ini:
   a) Cari *node* (n) dengan nilai f(n) yang paling rendah dalam *openlist*. *Node* ini sekarang menjadi *current node* .
   b) Keluarkan current *node* dari openlist dan masukan ke close list.
   c) Untuk setiap tetangga dari current *node* dilakukan berikut:
      - Jika tidak dapat dilalui atau sudah ada dalam close list, abaikan.
      - Jika belum ada di *open list*. Buat current *node* *parrent* dari *node* tetangga ini. Simpan nilai f,g dan h dari *node* ini.
      - Jika sudah ada di open list, cek bila *node* tetangga ini lebih baik, menggunakan nilai g sebagai ukuran. Jika lebih baik ganti *parrent* dari *node* ini di openlist menjadi current *node*, 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 sementara openlist kosong atau berarti tidak adda rute.

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

### 2.2.6 Cara Kerja Algoritma A*

Menurut Patrick Lester (2005, p1) cara kerja algoritma A* dapat digambarkan sebagai berikut, misalkan seseorang ingin berjalan dari *node* A ke *node* B, dimana diantaranya terdapat hambatan, lihat pada Gambar 2.4. Dimana *node* A ditunjukan dengan kotak berwarna hijau, sedangkan titik B ditunjukan dengan kotak berwarna merah, dan kotak berwarna biru mewakili hambatan/tembok yang memisahkan kedua *node* tersebut.



**Gambar 2.4** Tampilan Awal

Perlu diperhatikan bahwa, area pencarian dibagi ke dalam bentuk *node* seperti yang bisa dilihat pada Gambar 2.4. Menyederhanakan area pencarian seperti yang telah dilakukan adalah langkah awal dalam pencarian jalur. Dengan fungsi ini dapat menyederhanakan area pencarian menjadi *array* dua dimensi yang sederhana. Tiap nilai dalam *array* merepresentasikan satu *node* pada area

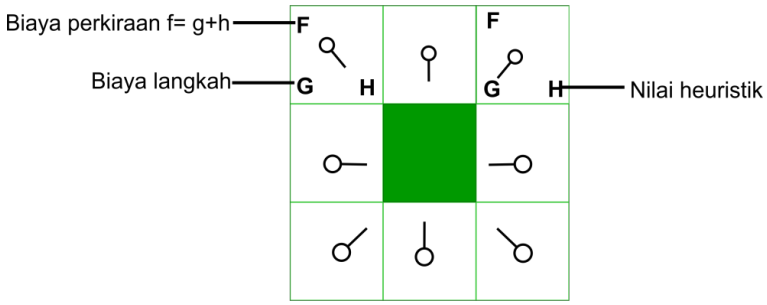
pencarian, dan statusnya disimpan sebagai “yang bisa dilalui” atau “yang tidak bisa dilalui”. Jalur ditemukan dengan menentukan *node* mana saja yang dilalui untuk mencapai *node* B ke *node* A. Ketika jalur ditemukan maka akan berpindah dari satu *node* ke *node* yang lain sampai ke tujuan.

Ketika area pencarian sudah disederhanakan ke dalam beberapa *node*. Seperti yang telah dilakukan diatas, langkah berikutnya adalah melakukan pencarian untuk mencari jalur terpendek. Dalam pencarian jalur algoritma A*, dimulai dari *node* A, memeriksa *node* yang berdekatan dan secara umum mencari kesebelah sampai tujuan ditemukan. Pencarian dilakukan dengan tahap sebagai berikut:

1. Dimulai dari *start node* A dan *start node* tersebut ditambahkan ke sebuah *openlist* dari *node-node* yang akan diperiksa. List tersebut berisi *node-node* yang mungkin dilalui pada jalur yang ingin dicari, atau mungkin juga tidak, jadi *list* tersebut berisi *node-node* yang perlu diperiksa.
2. Lihatlah semua *node-node* yang dapat dilalui yang terhubung dengan *start node*. Hindari *node-node* yang merupakan penghalang-penghalang. Tambahkan ke dalam *open list,* untuk tiap-tiap *node*, *node* A merupakan *node parent, node* ini berguna ketika ingin mengikuti jalur.
3. Buang *node* A dari *open list,* kemudian tambahkan *node* A ke dalam *closed list,* dimana pada *list* ini tidak perlu lagi memeriksa *node-node* yang ada didalamnya.

Pada saat ini, harus dilakukan seperti yang terlihat pada Gambar 2.5. Pada gambar dibawah *node* berwarna hijau di tengah-tengah adalah *start node*. *Node*

yang sisinya berwarna biru adalah *node* yang telah dimasukan ke dalam *closed list,* semua *node* yang bersebelahan dengan *node* pusat yang akan diperiksa dimasukan ke dalam *open list*, dan sisinya yang berwarna hijau. Tiap petunjuk yang berwarna hijau ke *node parent* –nya, yang merupakan *start node*.




**Gambar 2.5** Set Parent

Selanjutnya dipilih salah satu *node* yang berhubungan dalam *open list* lalu dilakukan berulang-ulang seperti langkah yang akan dijelaskan dibawah ini:

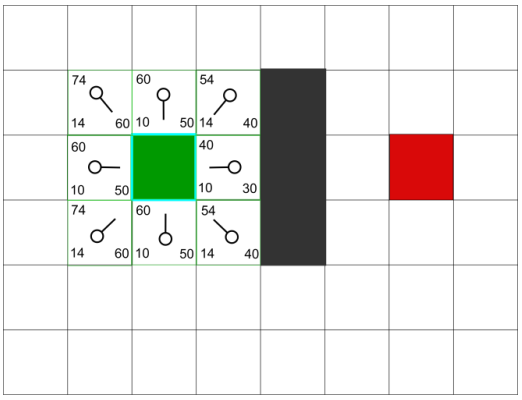
Persamaan untuk pemberian nilai pada *node* adalah f(n) = g(n) = h(n)

Dimana g(n) adalah nilai yang dibutuhkan untuk bergerak dari *start node* A ke sebuah *node* pada area tersebut, mengikuti jalur yang ditentukan untuk menuju kesana. h(n) adalah nilai perkiraan untuk bergerak dari suatu *node* pada area ke final *node* B. *Node* yang dipilih untuk tujuan selanjutnya adalah *node* yang memiliki nilai f(n) terendah.

Jalur yang dibuat adalah jalur yang dibangun secara berulang-ulang dengan menentukan *node-node* yang mempunyai f(n) terendah pada *open list.* Seperti yang telah dikatakan diatas, g(n) adalah nilai yang dibutuhkan untuk bergerak dari *start node* ke final *node* menggunakan jalur yang dibuat untuk kesana. Akan diberi nilai 10 untuk tiap pergerakan horizontal atau vertikal, dan nilai 14 untuk tiap pergerakan diagonal. Nilai 10 dan 14 digunakan untuk

penyerhanaan, dihindari perhitungan decimal dan pengakaran. Cara untuk menentukan nilai g(n) adalah dengan menghitung nilainya terhadap *node parent*-nya, dengan menambahkan 10 atau 14 tergantung apakah *node* tersebut diagonal atau orthogonal (non-diagonal) terhadap *parent node.* Fungsi ini diperlukan apabila didapatkan suatu *node* berjarak lebih dari satu *node* terhadap *start node*.

Nilai heuristik atau h(n) dapat diukur dengan berbagai macam cara. Cara yang digunakan adalah fungsi *Manhattan distance*, dimana dihitung jumlah total *node* yang bergerak horizontal atau vertikal untuk mencapai *final node* dari *node* sekarang. Lalu dikalikan dengan 10. Ini dinamakan fungsi *Manhattan Distance* karena ini seperti menghitung jumlah blok-blok *node* satu tempat ke tempat lain, dimana tidak dapat memotong suatu blok secara diagonal. Ketika menghitung h(n), harus mengacuhkan rintangan apapun seperti tembok, air, dan lain-lain. Hal ini karena h(n) merupakan perhitungan perkiraan dari titik awal ke titik tujuan, bukan jarak nyatanya. Hitung nilai f(n) dengan menambahkan g(n) dan h(n). Hasilnya dapat dilihat pada Gambar 2.6, dimana nilai f(n) ditulis dari kiri ke atas, g(n) kiri bawah dan h(n) di kanan bawah pada tiap-tiap *node*.



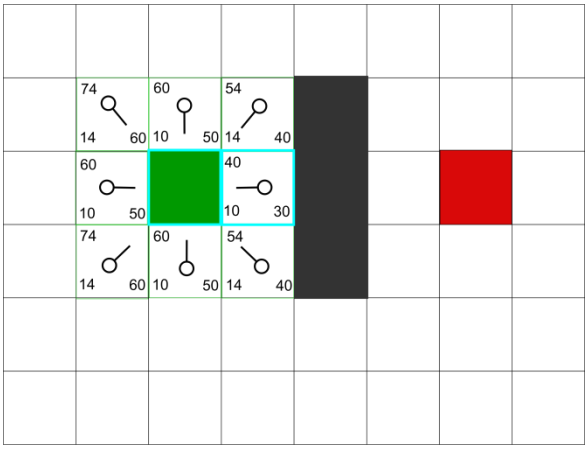

**Gambar 2.6** Masukan Ke *Closed List*

Diberi nilai g(n) sama dengan 10, pada *node* bagian atas, bawah, kiri, dan kanan sedangkan *node-node* diagonal diberi nilai 14, karena *node-node* tersebut bersebelahan dengan *start node.*

Nilai h(n) ditentukan dengan menggunakan fungsi *Manhattan Distance*, dimana ditentukan jarak antara *node* tersebut dengan final *node* (*grid* merah), dengan bergerak kedelapan arah. Selanjutnya dalam *pathfinding*, dipilih *node* yang nilai f(n)-nya paling rendah dalam *open list* , ketika dipilih *node* tersebut lalu dilakukan:

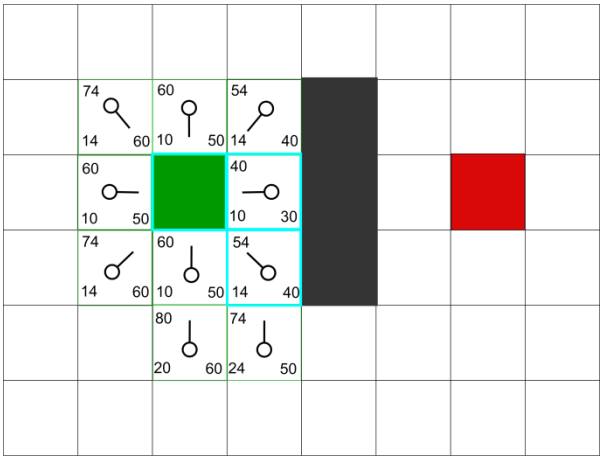
1. Keluarkan *node* tersebut dari *open list* lalu dimasukan ke *closed list*.
2. Periksa semua *node* yang berhubungan, kecuali *node* yang sudah masuk ke *closed list* atau *node* yang tidak dapat dilalui (seperti dinding, air, dan lain-lain). Tambahkan *node* tersebut ke *open list* tersebut. Jadikan *node* yang dipilih tadi sebagai *parent node* bagi *node* baru tersebut.
3. Jika *node* yang terhubung sudah masuk ke *open list*, periksa apakah nilai g(n) *node* tersebut lebih kecil.
4. Jika tidak jangan lakukan apa-apa, jika benar *parent node* harus diganti lalu dihitung ulang nilai f(n) dan g(n).

Seperti contoh pada Gambar 2.7 ada sembilan *node*, dimana 8 (delapan) *node* masuk *open list* dan *start node* sudah masuk *closed list*, lalu *node* dengan nilai f(n) terendah yaitu 40. Dimasukkan ke *closed list*, karena itu diberi warna biru pada sisinya.




**Gambar 2.7** Pemilihan *Closed List*

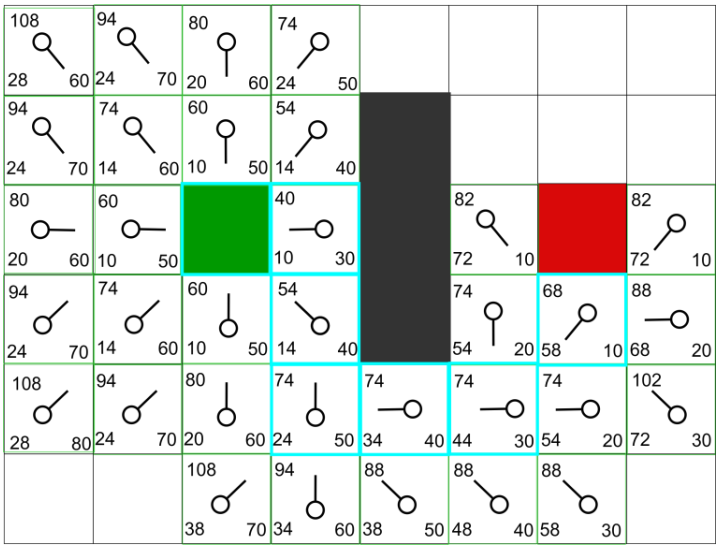
Semua *node* yang berhubungan dengan *node* tersebut diperiksa, *start node* tidak dianggap karena sudah masuk ke *closed list*, dan *node* hambatan. 4 (empat) *node* lain yang berhubungan semuanya sudah masuk ke *open list* maka harus diperiksa, apakah nilai g(n) yang dibutuhkan untuk mencapai *node* tersebut melalui *node* yang dipilih lebih kecil daripada menggunakan *node* lain, seperti contoh di atas (Gambar 2.7) didapatkan bahwa apabila ingin ke bawah atau ke atas dari *node* yang dipilih ternyata membutuhkan g(n) sama dengan 10, sedangkan apabila diambil arah diagonal dari *start node* hanya membutuhkan g(n) sama dengan 14, maka tidak dilakukan apa-apa. Saat ini pada *open list* terdapat 7 *node*, dilakukan pencarian rute kembali dengan mencari nilai f(n) terendah setiap *node* yang menjadi *neighbor*. Terdapat 2 *node* yang punya nilai f(n) yang sama terlihat pada Gambar 2.8, dapat dipilih yang mana saja, tapi untuk mempercepat dapat dipilih yang terakhir masuk ke *open list*. Jadi dipilih yang bawah, maka akan tampak seperti Gambar 2.8.




**Gambar 2.8** Pemilihan *Closed List* Kedua

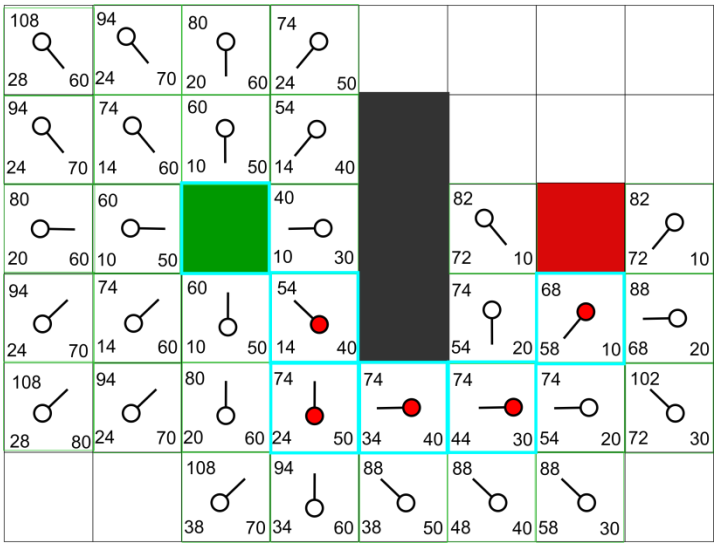
Periksa kembali *node* yang dipilih, masukkan *node* yang berhubungan ke *open list*, karena tidak dapat langsung dari *node* sekarang ke *node* tersebut tanpa memotong bagian pojok dari dinding diatasnya, jadi harus turun dulu ke bawah (aturan memotong sudut adalah pilihan).

Setelah itu dilakukan proses yang sama seperti yang diatas, lalu ditentukan *node* yang akan dipilih dengan membandingkan nilai f(n). Proses tersebut diulangi sampai final *node* ke dalam *open list* ditambahkan, terlihat pad Gambar 2.9.




**Gambar 2.9** Final Node Masuk ke Closed List

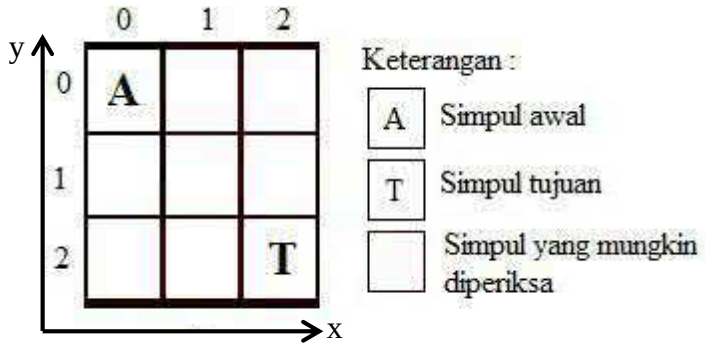
Pada Gambar 2.9, perhatikan *node* kedua dibawah *start node*, pada diagram awal nilai g(n) sama dengan 28 dan menunjuk pada *node* kanan atasnya sekarang bernilai g(n) sama dengan 20 dan menunjuk pada *node* atasnya, hal ini terjadi karena pemeriksaan nilai g(n) dimana nilainya lebih rendah dengan menggunakan jalur yang baru, sehingga *parent node* harus digantu dan niali g(n) dan f(n) harus dihitung ulang. Lalu setelah selesai ditandai dn proses telah diselesaikan maka ditentukan jalurnya dengan menggunakan fungsi *backtrack* dengan menelusuri dari final *node* mengikuti anak pada *node* tersebut hingga sampai ke *start node*. Hasilnya akan terlihat seperti pada Gambar 2.10.




**Gambar 2.10** *Backtrack* Hasil *Pathfinding* Algoritma A*

### 2.2.7 Perhitungan Algoritma A*

Perhitungan yang dilakukan dengan algoritma A* dengan kondisi tanpa penghalang dengan ordo(X,Y) yaitu 3x3, terlihat pada gambar 2.11.




**Gambar 2.11** Contoh Kondisi Tanpa Penghalang Pada Pencarian Algoritma A*

Posisi simpul awal = Ax : 0, Ay : 0

Posisi simpul tujuan = goal x : 2, goal y :2

(1) Langkah ke satu:

Hitung nilai pada *node* yang menjadi *neighbor BestNode* dari *node* atau simpul awal. Terdapat 3 (tiga) *node neighbor*, yaitu *node*(0,1), *node*(1,1), dan *node* (1,0).

- **Node (0,1)**

g(0,1) = 1

h(0,1) = (abs(A(x) - goal(x)) + abs(A(y) - goal(y)))

= (abs (0 – 2) + abs (1 – 2))

= 3

f(0,1) = g(0,1) + h(0,1)

= 1 + 3

= 4

- **Node (1,1)**

g(1,1) = 1,4

h(1,1) = (abs(A(x) - goal(x)) + abs(A(y) - goal(y)))

= (abs (1 – 2) + abs (1– 2))

= 2

f(1,1) = g(1,1) + h(1,1)

= 1,4 + 2

= 3,4

- **Node (1,0)**

g(1,0) = 1

h(1,0) = (abs(A(x) - goal(x)) + abs(A(y) - goal(y)))

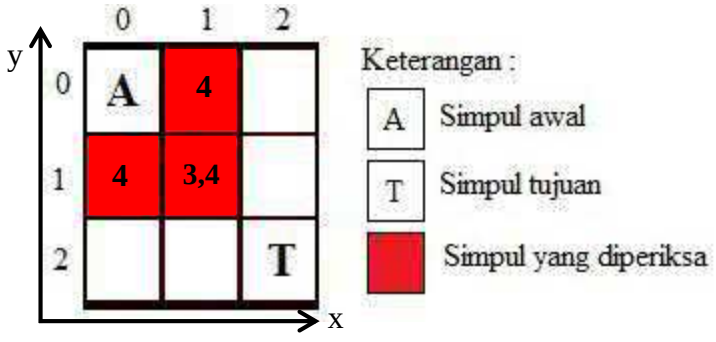
= (abs (1 – 2) + abs (0 – 2))

= 3

f(1,0) = g(1,0) + h(1,0)

= 1 + 3

= 4




**Gambar 2.12** Langkah Pertama Pencarian *BestNode*

Pada gambar 2.12, *node* yang diperiksa memiliki nilai masing-masing yaitu *node* (1,0) dengan f(n) = 4, (1,1) dengan f(n) =3,4 dan *node* (0,1) dengan f(n)= 4. Dari ketiga simpul yang diperiksa, dipilihlah simpul *node* (1,1) dengan

biaya terkecil yaitu 3,4. Dipilih *node* (1,1) dengan nilai f(n) terkecil sebagai *Path List*.

(2) Langkah kedua:

Setelah mendapat 1(satu) *path* sebagai *list* rute, maka posisi simpul A berpindah ke simpul (1,1). Selanjutnya *node* yang menjadi *neighbor* dari posisi A yang baru, dihitung untuk mencari nilai f(n) yang terkecil. Terdapat 3 *node* yang menjadi *BestNode* dan masuk kedalam *open list*, yaitu *node* (2,1), *node* (1,2) dan *node* (2,2)

- **Node (2,1)**

g(2,1) = 1

h(2,1) = (abs(A(x) - goal(x)) + abs(A(y) - goal(y)))

= (abs (2 – 2) + abs (1 – 2))

= 1

f(2,1) = g(2,1) + h(2,1)

= 1 + 1

= 2

- **Node (1,2)**

g(1,2) = 1

h(1,2) = (abs(A(x) - goal(x)) + abs(A(y) - goal(y)))

= (abs (1 – 2) + abs (2– 2))

= 1

f(1,2) = g(1,2) + h(1,2)

= 1 + 1

= 2

- **Node (2,2)**

g(2,2) = 1,4

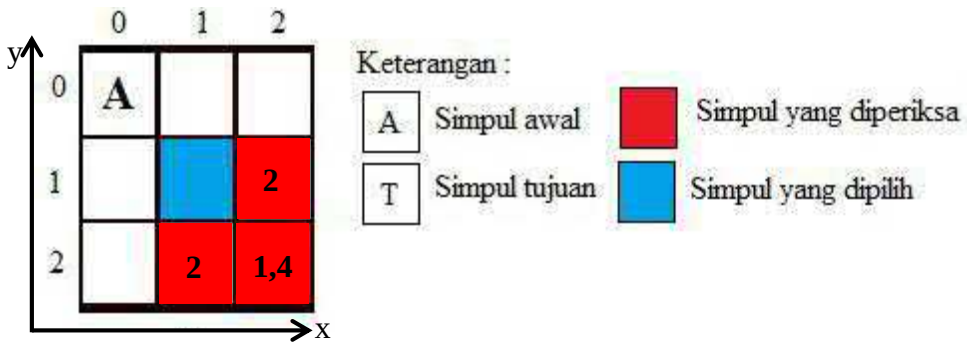
h(2,2) = (abs(A(x) - goal(x)) + abs(A(y) - goal(y)))

= (abs (2 – 2) + abs (2 – 2)) = 0

f(2,2) = g(2,2) + h(2,2)

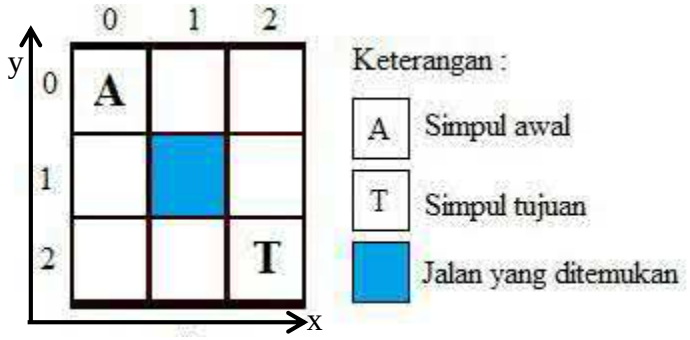
= 1,4 + 0

= 1,4




**Gambar 2.13** Langkah Kedua Pencarian *BestNode*

Pada gambar 2.13 , *BestNode* yang diperiksa memiliki nilai masing-masing yaitu *node* (2,1) dengan f(n) = 2, *node* (1,2) dengan f(n) =2 dan *node* (2,2) dengan f(n)= 1,4. Dari ketiga *node* yang diperiksa maka dipilihlah *node* (2,2) dengan biaya terkecil yaitu 1,4. Dipilih *node* (1,1) dengan nilai f(n) terkecil sebagai *Path List*.




**Gambar 2.14** Hasil Pencarian Rute dengan Algoritma A* pada Kondisi Tanpa *Obstacle*

Dari semua perhitungan yang telah dilakukan dipilihlah biaya/*cost* terkecil pada setiap langkahnya sehingga akan menghasilkan jalur terpendek yang terlihat pada gambar 2.14.

### 2.2.8 Gerak Lurus Berubah Beraturan

Menurut Muhammad Ishaq (2007, p.27) Gerak Lurus Berubah Beraturan (GLBB) adalah suatu gerak lurus yang memiliki kecepatan selalu berubah disetiap saat dan perubahan kecepatan tersebut di setiap saat selalu sama, tetap atau konstan. Gerak Lurus Berubah Beraturan (GLBB) juga dapat diartikan sebagai gerak benda dalam lintasan lurus dengan percepatan tetap. Sedangkan percepatan tetap adalah perubahan percepatan gerak benda yang berlangsung secara tetap dari waktu ke waktu. Mula-mula dari keadaan diam, benda mulai bergerak, semakin lama semakin cepat dan kecepatan gerak benda tersebut berubah secara teratur. Perubahan kecepatan bisa berarti terjadi pertambahan kecepatan atau pengurangan kecepatan.

Gerak lurus dibagi menjadi gerak lurus dipercepat dan gerak lurus diperlambat.

a. Gerak Lurus dipercepat beraturan adalah gerak yang lintasannya lurus dan kecepatannya setiap saat berubah secara beraturan (tetap). Karena perubahan kecepatan tiap satuan waktu disebut percepatan maka gerak lurus dipercepat beraturan dapat dinyatakan sebagai gerak yang lintasannya lurus dan percepatannya selalu tetap. Pada gerak lurus dipercepat beraturan, besarnya perpindahan benda sama dengan jarak yang ditempuh benda.
b. Gerak Lurus diperlambat beraturan yaitu gerak lurus yang kecepatannya berkurang secara beraturan (pengurangan kecepatan tiap selang waktu yang sama berharga tetap). Pengurangan kecepatan tiap selang waktu disebut perlambatan.

Percepatan atau perlambatan adalah perubahan kecepatan benda setiap satuan wakku. Secara sistematis percepatan atau perlambatan dirumuskan sebagai berikut:

$$\boldsymbol{a = \frac{Vt - Vo}{t}} \quad \text{..... (5)}$$

Dimana :

- Vo adalah kecepatan mula-mula (m/s)
- Vt adalah kecepatan setelah selang waktu t detik (m/s)
- t adalah selang waktu (s)
- a adalah pecepatan atau perlambatan ($m/s^2$).

Pada GLBB diperlambat Vt lebih kecil dari Vo, maka Vt – Vo = negatif, sehingga a bernilai negatif (-). Apabila kecepatan awal Vo dan kecepatan setelah t detik menjadi Vt, maka Vt dihitung dengan rumus sebagai berikut:

**Vt = Vo + a.t** ..... (6)

Sedangkan jarak tempuh untuk GLBB dapat dirumuskan sebagai berikut:

$\mathbf{S_t = Vo.t + ½ a.t^2}$ ..... (7)

Dimana:

- $S_t$ adalah jarak yang ditempuh selama t sekon (m)
- Vo adalah percepatan atau perlambatan ($m/s^2$)
- t adalah selang waktu tempuh (s).

Pada penelitian ini GLBB akan diterapkan pada NPC ketika mulai bergerak dari titik awal nya menuju ke *player* sesuai dengan jalur terpendek yang telah ditemukan. Rute yang ditemukan setelah proses *pathfinding* memiliki lintasan yang berbeda-beda. Ketika lintasan dideteksi sebagai lintasan yang lurus dan arah yang tetap dengan percepatan yang tetap, maka kecepatan NPC akan dipercepat, sedangkan pada lintasan yang tidak lurus maka NPC akan kembali ke kecepatan awal (Vo). Hal ini memungkinkan membuat *game* menjadi lebih menantang karena kecepatan NPC yang adatif ketika mengejar *player*.

# BAB V

# KESIMPULAN DAN SARAN

## 5.1 Kesimpulan

Berdasarkan hasil penelitian dan pembahasan, dapat ditarik kesimpulan yaitu:

1. Algoritma A* dan GLBB dapat diimplementasikan dengan baik sebagai dasar pergerakan NPC mengejar *player* pada lintasan dengan adanya *obstacle*.
2. NPC dapat bergerak lebih cepat dengan menggunakan GLBB dibandingan tanpa GLBB. Implementasi GLBB pada pergerakan NPC ini juga tidak menambah beban kinerja CPU, sehingga penggunaan *resource* lebih optimal.

## 5.2 Saran

Berdasarkan keterbatasan mengenai penelitian yang telah dilaksanakan, ada beberapa saran yang dapat dipertimbangkan pada penelitian selanjutnya yaitu:

1. Menambah jumlah *Non-Playable Character* (NPC) dalam satu *scene* pada *game* untuk mengetahui tingkat optimalisasi algoritma A* dan GLBB secara lebih mendetail.
2. Menambah modifikasi skenario pergerakan dengan Gerak Lurus Berubah Beraturan (GLBB) yang diperlambat ketika NPC mengejar *player* dengan kondisi tertentu.

# DAFTAR PUSTAKA


Amit.J, Patel. 2010. Introduction A* Algorithm. http://theory.stanford.edu/~amitp/GameProgramming/. 10 Juli 2019 (15.30).

Atmaja, P. W., Siahaan, D. O., & Kuswardayan, I.2016. Game Design Document Format For Video Games With Passive Dynamic Difficulty Adjustment, *2*, 86–97.

Attoyibi, M. M., Faradila, E. F., & Handayani, A. N.2019. The Implementation of A Star Algorithm ( A *) In the Game Education About Numbers Introduction, *242*(Icovet 2018), 234–238.

Cui, X., & Shi, H.2011. A * -based Pathfinding in Modern Computer Games, *11*(1), 125–130.

Dian, S., Permana, H., Bayu, K., Bintoro, Y., & Arifitama, B.2018. Comparative Analysis of Pathfinding Algorithms A *, Dijkstra , and BFS on Maze Runner Game, *1*(2), 1–8.

Febliama, A. B.2018. The Application of a Star ( A *) Algorithm on the Android-Based Pacman Adaptation Educational Game as a Learning Media for SMK, *242*(Icovet 2018), 200–206.

Ferguson, D., Likhachev, M., & Stentz, A.2005. A Guide to Heuristic-based Path Planning.

Harabor, D., & Grastien, A.2014. Online Graph Pruning for Pathfinding on Grid Maps, 1114–1119.

Ishaq, M. 2007.Fisika Dasar. Edisi Kedua. Yogyakarta: Graha Ilmu.

Kallem, S. R.2012. Artificial Intelligence Algorithms, *6*(3), 1–8.

Me-cse, G. E. M. I.-.2015. Direction Based Heuristic For Pathfinding In Video Games. *Procedia - Procedia Computer Science*, *47*, 262–271. https://doi.org/10.1016/j.procs.2015.03.206

Patrick, B., & Updated, L.2005. A * Pathfinding for Beginners, 1–11.

Putrady, E.2009. Penerapan Algoritma A * Sebagai Algoritma Pencari Jalan Dalam Game.

Sabri, A. N., Haizan, N., Radzi, M., Samah, A. A., & Games, A. P.2018. A study on Bee algorithm and A* algorithm for pathfinding in games. *2018 IEEE*

*Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, 224–229.

Sazaki, Y., Satria, H., Primanita, A., & Syahroyni, M.2018. Analisa Perbandingan Algoritma A * Dan Dynamic Pathfinding Algorithm Dengan Dynamic Pathfinding Algorithm Untuk Npc, *5*(1), 95–103. https://doi.org/10.25126/jtiik

Setiawan, K.2018. Menghitung Rute Terpendek Menggunakan Algoritma A * Dengan Fungsi Euclidean Distance, *2018*(Sentika), 23–24.

Sharma, S. K., & Pal, B. L.2015. Shortest Path Searching for Road Network using A * Algorithm, *4*(7), 513–522.

Shin, S., Jang, D., & Lee, H.2012. Telematics Specific Horizontal Distance Traveled by a Falling Car, *10*(2), 181–186.

Syukriyah, Y.2016. Penerapan algoritma a* (star) untuk mencari rute tercepat dengan hambatan, (Selisik).

Tayal, M. A., & Tayal, A. R.2012. Reconstruction of 3 Dimension Object from 2 Dimension Images of an Object using Method of Shocks, *9*(4), 413–417.

Wina Witanti, D. N. R.2013. Analisis Pengaruh Penggunaan Nilai Heuristik Terhadap Performansi Algoritma A * Pada Game, 2–4.

Zainulhayat, L.2017. Perbandingan rute optimum hasil perhitungan algoritma.