Seminar Nasional Ilmu Komputer (SNIK 2024) - Semarang, 17 Oktober 2024
ISSN: 2614-1205

Perbandingan Non-Playable Character Pathfinding menggunakan Algoritma A* dan D* pada Simulasi Video Game

Muhammad Alif Samudra[1], Kholiq Budiman[2]

[1,2]Jurusan Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Semarang
Email: [1]alif_samudra74@students.unnes.ac.id, [2]kholiq.budiman@mail.unnes.ac.id

Abstrak

Sebuah *video game* adalah sistem di mana pemain terlibat dalam konflik buatan yang didefinisikan oleh aturan yang menghasilkan hasil yang terukur. *Video game* memerlukan banyak sistem kompleks dan terhubung untuk mendukung simulasi dunia lain sebagai nilai hiburan utamanya. Salah satu sistem tersebut adalah sistem pencarian jalur karakter yang tidak dapat dimainkan (*Non-Playable Character*/NPC). Algoritma pencarian jalur A* dan variasi perkembangannya merupakan metode yang paling banyak digunakan dalam pengembangan *video game*. Namun, hasil pencarian jalur menggunakan algoritma A* hanya efektif untuk situasi dengan hambatan statis. Algoritma pencarian jalur D* memiliki solusi untuk situasi dengan hambatan dinamis seperti dalam *video game*, yaitu melalui proses *Path Replanning*. Oleh karena itu, dengan fungsi pencarian heuristik dan proses *Path Replanning* algoritma D*, algoritma ini cocok digunakan di area dengan kondisi hambatan dinamis. Penelitian ini bertujuan untuk menentukan algoritma mana yang lebih baik antara algoritma A* dan D* dalam lingkungan dengan kendala dinamis. Hasil penelitian menunjukkan bahwa algoritma A* masih memiliki kecepatan pencarian jalur yang lebih cepat dibandingkan dengan algoritma D* di lingkungan tanpa hambatan dinamis. Ditemukan bahwa perbedaan waktu perjalanan mencapai 6-8% antara algoritma A* dan D*. Namun, dalam lingkungan dengan hambatan dinamis, algoritma A* tidak dapat mengatasi hal ini di semua kasus simulasi lingkungan dan terjadi kesalahan, sedangkan algoritma D* dapat mengatasi hambatan dinamis tersebut dengan perbedaan waktu perjalanan mencapai 8-30% jika dibandingkan antara lingkungan tanpa hambatan dinamis dan dengan hambatan dinamis.
Kata Kunci: *Pathfinding*, A*, D*, *Video Game*, *Environments with Dynamic Obstacles*

Abstract

A video game is a system in which players engage in an artificial conflict defined by rules that produce measurable results. Video games require many complex and connected systems to support other world simulations as the main entertainment value. One of the systems is the Non-Playable Character (NPC) pathfinding system. The A pathfinding algorithm and its developmental variations are the most widely used methods in video game developments. However, the pathfinding results using the A* algorithm are only effective for situations with static obstacles. The D* pathfinding algorithm has a solution for situations with dynamic obstacles such as in a video game, namely the Path Replanning process. Therefore, with the D* algorithm's heuristic search function and Path Replanning process, the algorithm is suiTabel for use in areas with dynamic obstacle conditions. This research aims to determine which algorithm is better between the A* and D* algorithms in an environment with dynamic constraints. The results showed that the A* algorithm still has a faster pathfinding speed than the D* algorithm in an environment without dynamic obstacles. It is found that the difference in*

travel time reaches 6-8% between the A and D* algorithms. However, in an environment with dynamic obstacles, the A* algorithm cannot overcome this in all environmental simulation cases and an error occurs, while the D* algorithm can overcome these dynamic obstacles with a difference in travel time reaching 8-30% when compared between environments without dynamic obstacles and with dynamic obstacles.*
***Keyword**: Pathfinding, A*, D*, Video Game, Environments with Dynamic Obstacles*

1. PENDAHULUAN (10pt huruf besar, rata kiri/bold)

Video game adalah sistem di mana satu atau lebih pemain terlibat dalam konflik buatan yang didefinisikan oleh aturan untuk menghasilkan hasil yang terukur [1]. Konflik buatan ini dapat memberikan pemain rasa kesenangan, kepuasan, atau frustrasi, tergantung pada pengalaman yang ingin disampaikan oleh pengembang *video game*. Kualitas produk *video game* dapat diukur dari nilai hiburan yang terkandung di dalamnya. Salah satu nilai hiburan yang didapatkan dari bermain video game adalah pengalaman menjelajahi dunia lain, di mana pemain dapat merasakan pengalaman baru yang tidak dapat dilakukan di dunia nyata. Pemahaman sebelumnya terhadap teknologi digital dapat membantu pengguna merasa lebih nyaman saat mengoperasikan teknologi tersebut [2] .

Untuk memberikan pengalaman menjelajahi dunia lain, *video game* memerlukan banyak sistem kompleks dan saling terhubung. Salah satu sistem yang membantu memberikan pengalaman dunia lain kepada pemain adalah sistem *Non-Playable Character* (NPC). NPC berperan membantu pemain mencapai tujuan atau bahkan menghambat atau mendahului pemain dalam mencapai tujuan di *video game* [3]. NPC dapat berupa karakter yang dapat berinteraksi dengan pemain atau hanya berdiri di dunia tempat *game* berlangsung. Sistem NPC terdiri dari beberapa subsistem pendukung yang membuat NPC terasa hidup, salah satunya adalah sistem pencarian jalur (pathfinding) untuk pergerakan NPC. Pada *video game* yang memiliki nilai hiburan dalam simulasi dunia lain, akurasi sistem sangatlah penting. Satu kesalahan sistem dapat merusak seluruh nilai hiburan [4], terutama dalam sistem pergerakan dan pencarian jalur NPC yang harus dirancang secepat dan sehumanis mungkin dari titik awal hingga tujuan.

Akurasi sistem dapat dihitung dari rasio keberhasilan dan kesalahan suatu sistem [5], termasuk dalam *video game*. Kesalahan dalam *video game* merupakan hasil dari kesalahan perangkat lunak yang terlihat namun tidak teridentifikasi, atau baru terlihat ketika dipicu oleh masukan pemain atau interaksi sistem yang tidak terduga [6]. Ada berbagai jenis kesalahan pada *video game* yang memiliki banyak variasi sistem. Salah satu jenis kesalahan adalah kesalahan navigasi, seperti kesalahan pada sistem pencarian jalur NPC yang menyebabkan NPC tidak dapat mencapai target pergerakannya atau terhambat dalam mencapai target sehingga mengganggu sistem pergerakan NPC lain. Sistem pencarian jalur NPC mengalami kesalahan ketika NPC membutuhkan waktu lama untuk menemukan rute atau terdapat hambatan dinamis yang menghalangi rute setelah rute ditemukan sehingga NPC tidak dapat mencapai tujuan. Diperlukan algoritma pencarian jalur yang dapat meminimalkan kesalahan tersebut.

Kecepatan juga menjadi faktor penting dalam memilih algoritma pencarian jalur [7]. Kecepatan adalah salah satu ukuran yang dapat dijadikan referensi untuk kinerja algoritma pencarian jalur [8]. Jika rute rata-rata yang dihasilkan algoritma dalam beberapa simulasi lingkungan berbeda atau dalam simulasi lingkungan yang sama membuat subjek melalui banyak titik pengurangan kecepatan seperti tikungan tajam dan *U-turn*, algoritma dianggap memiliki kinerja buruk karena dapat menyebabkan keterlambatan sistem subjek untuk mencapai tujuan. Kecepatan tidak hanya dihitung dari saat subjek bergerak, tetapi juga sejak perhitungan jalur dimulai. Hal ini membuat kecepatan perhitungan jalur menjadi salah satu penentu kecepatan keseluruhan algoritma. Oleh karena itu, kecepatan menjadi salah satu faktor penentu kinerja algoritma.

Penelitian tentang algoritma pencarian jalur untuk diterapkan dalam *video game* telah banyak dilakukan. Algoritma pencarian jalur A* dan variasi perkembangannya merupakan metode yang paling banyak digunakan dalam *video game* [9]. Dengan menghitung langkah yang harus diambil menggunakan fungsi perhitungan heuristik, algoritma A* mengeliminasi langkah yang tidak diperlukan atau langkah yang tidak menuju titik tujuan [10]. Oleh karena itu, pencarian jalur menggunakan algoritma A* lebih mudah menemukan rute dan membutuhkan waktu yang relatif singkat. Namun, menurut Wang & Lin [11], hasil pencarian jalur menggunakan algoritma A* hanya efektif untuk situasi dengan hambatan statis, dimana hambatan di lingkungan subjek tidak dapat bergerak.
Untuk penggunaannya dalam *video game*, ketidakmampuan algoritma A* untuk mengatasi hambatan dinamis menjadi kekurangan terbesar dalam akurasi sistem dan nilai hiburan *video game*. Oleh sebab itu, algoritma pencarian jalur D* menawarkan solusi untuk situasi dengan hambatan dinamis seperti dalam *video game*, melalui proses *Path Replanning*. Sama seperti algoritma A*, algoritma D* juga menggunakan fungsi perhitungan heuristik yang dilakukan berulang pada semua titik di sekitar subjek dari titik awal ke tujuan untuk menyederhanakan dan mempercepat pencarian rute [12]. Perbedaan utama antara algoritma D* dan A* terletak pada proses *Path Replanning*. Algoritma D* menggunakan proses *Path Replanning* yang dikenal sebagai *Incompletion Replanning* [13], di mana titik awal perencanaan ulang berbeda dari titik awal pencarian jalur. Dengan fungsi pencarian heuristik dan proses *Path Replanning*, algoritma D* cocok digunakan dalam lingkungan dengan kondisi hambatan dinamis.
Berdasarkan hal tersebut, penelitian ini berfokus pada membandingkan akurasi dan kecepatan pencarian jalur antara algoritma A* dan D* dalam lingkungan dengan hambatan dinamis.

2. METODE

2.1. Permainan Video

Permainan video (*Video Game)* adalah bentuk permainan yang melibatkan interaksi, baik dengan pemain lain, dengan sistem permainan itu sendiri, maupun dengan keberuntungan atau nasib [14]. Semua interaksi yang terjadi bersifat dua arah, baik interaksi antara pemain dan sistem, pemain dengan pemain, maupun sistem dengan sistem lain dalam permainan video. Ada berbagai jenis permainan video, mulai dari permainan teka-teki, permainan aksi, permainan petualangan, hingga permainan peran

(*role-playing game*/RPG). Pada dasarnya, tujuan pembuatan permainan video adalah untuk menghibur pemain dengan dunia interaktif di mana pemain dapat mengekspresikan diri mereka secara bebas sesuai dengan aturan sistem.

Aspek lain yang menonjol dalam permainan video adalah adanya konflik atau pertarungan antara pemain dengan pemain lain, pemain dengan sistem, atau sistem dengan sistem lain dalam permainan video itu sendiri. Berdasarkan pendapat yang diungkapkan [15], permainan video adalah aktivitas yang memanfaatkan layar video digital dengan cara tertentu, yang dibatasi oleh sistem aturan di mana seorang pemain dapat berinteraksi dengan pemain lain, serta dengan sistem permainan itu sendiri, untuk mencapai tujuan dan hasil yang diinginkan.

2.2. *A* Pathfinding*

Algoritma *pathfinding* A* pertama kali dikembangkan oleh Hart et al. [16] dan merupakan algoritma pencarian jalur yang dikembangkan dari algoritma Dijkstra. Berbeda dengan Dijkstra, A* menggunakan fungsi heuristik yang mempertimbangkan biaya dari keadaan saat ini dan biaya yang telah diambil dari titik awal hingga keadaan saat ini. Perhitungan dilakukan secara berulang untuk setiap titik di sekitar subjek algoritma A* dari titik awal hingga titik tujuan. Dari perhitungan tersebut, diperoleh jalur optimal dari titik awal ke titik tujuan. Dengan perhitungan ini, algoritma A* dapat menemukan jalur optimal dari titik awal ke titik akhir [17]. Algoritma A* dapat dinyatakan dalam notasi matematis seperti yang dituliskan pada Persamaan 1 (Wang & Lin, 2012).

$$f(n) = g(n) + h(n) \quad (1)$$

Dimana:
$f(n)$ = Estimasi biaya termurah dari titik awal ke titik tujuan.
$g(n)$ = Biaya dari titik awal ke titik nnn.
$h(n)$ = Estimasi biaya untuk mencapai titik tujuan dari titik nnn menggunakan fungsi heuristik.

Perhitungan persamaan ini diulang untuk semua titik di sekitar objek yang sudah berada dalam daftar terbuka (*open list*). Setelah itu, setiap titik mendapatkan biayanya masing-masing. Setelah diperoleh titik di sekitar dengan biaya terendah, titik tersebut diambil dan dimasukkan ke dalam daftar tertutup (*closed list*), dan titik-titik di sekitar titik tersebut dihitung ulang menggunakan persamaan di atas. Proses ini diulang hingga jalur optimal ditemukan.

2.3. *D* Pathfinding*

Algoritma D* atau *Dynamic A** adalah metode pencarian heuristik inkremental. Dikembangkan oleh Stentz [18], algoritma D* digunakan untuk melakukan pencarian jalur di area yang tidak diketahui atau hanya sebagian diketahui. Karena D* merupakan pengembangan spesifik dari algoritma A*, algoritma D* juga menggunakan fungsi heuristik yang sama dengan A*. Perhitungan heuristik dilakukan secara berulang untuk setiap titik di sekitar titik awal hingga titik tujuan. Selain fungsi heuristik yang sama, D* juga sama seperti algoritma pencarian jalur pada umumnya,

yang membuat asumsi bahwa tidak ada hambatan di area pencarian dan mencari jalur terpendek dari titik awal ke titik tujuan. Setelah menemukan jalur terpendek namun tiba-tiba ada informasi baru (seperti hambatan yang sebelumnya tidak diketahui), algoritma D* menambahkan informasi hambatan baru tersebut ke dalam memori dan, jika diperlukan, mengganti jalur yang telah ditemukan sebelumnya sehingga jalur terpendek baru dapat diperoleh dengan informasi baru tersebut [19]. Perubahan jalur juga melalui proses perhitungan heuristik iteratif pada titik di mana hambatan tiba-tiba muncul hingga ke titik tujuan.

Algoritma D* bekerja dengan memilih satu titik dari daftar terbuka (*open list*), mengevaluasi titik tersebut, dan memberikan biaya pada titik tersebut. Kemudian algoritma bergerak ke semua titik di sekitarnya (atas, kanan, bawah, dan kiri) dan memasukkan titik-titik tersebut ke dalam daftar terbuka. Proses pergerakan ini disebut proses ekspansi. Algoritma D* memulai pencarian jalur dari titik tujuan. Setiap titik yang terkena ekspansi memiliki penunjuk balik (*backpointer*) yang mengarah ke titik berikutnya dan kemudian mengarah ke titik awal. Ketika ekspansi telah mencapai titik awal, perhitungan algoritma selesai, dan jalur tercepat dapat ditemukan dengan mengikuti penunjuk balik yang ada.

Ketika ada hambatan tiba-tiba pada jalur yang telah ditentukan, semua titik dari keadaan saat ini hingga ke titik tujuan dimasukkan kembali ke dalam daftar terbuka dan diberi status *raise*. Sebelum biaya titik-titik tersebut dinaikkan, algoritma D* memeriksa titik-titik di sekitarnya dan menentukan apakah titik-titik di sekitarnya memiliki biaya yang lebih kecil. Jika tidak, status *raise* diberikan kepada semua titik yang ada di jalur yang telah ditentukan. Titik-titik ini kemudian dievaluasi, dan status *raise* diteruskan melalui ekspansi. Namun, jika titik di sekitarnya memiliki biaya yang lebih rendah, penunjuk balik (*backpointer*) yang ada diperbarui untuk titik tersebut, dan status lebih rendah diberikan ke titik di sekitar yang juga melalui ekspansi.

2.4. Akurasi dari Algoritma *Video Game*

Akurasi adalah salah satu metrik perhitungan yang digunakan untuk menganalisis kinerja sebuah algoritma. Akurasi dapat dihitung dengan cara yang dituliskan pada Persamaan 2 [20].

$$Accuracy = \frac{Jumlah\ data\ terprediksi\ benar}{Jumlah\ semua\ prediksi} \quad (2)$$

Akurasi juga dapat dihitung dengan membandingkan jumlah kesalahan dalam perhitungan algoritma dengan total perhitungan. Kesalahan dalam *video game* merupakan produk dari kesalahan perangkat lunak yang terlihat tetapi tidak teridentifikasi, atau yang mungkin baru terlihat ketika dipicu oleh input pemain yang tidak terduga atau interaksi sistem [6].

Salah satu jenis kesalahan dalam *video game* adalah kesalahan navigasi, seperti kesalahan dalam sistem *pathfinding* NPC. Kesalahan dengan jenis navigasi dapat diidentifikasi jika algoritma tidak dapat menemukan jalur optimal, waktu perhitungan jalur terlalu lama, atau jika subjek tidak dapat mencapai titik tujuan meskipun jalur telah ditemukan karena adanya hambatan dinamis.

Akurasi dari dua atau lebih algoritma dapat dibandingkan dengan melihat persentase akhir dari kedua algoritma tersebut. Jika perbedaan antara rata-rata akurasi dua atau lebih algoritma tidak signifikan, maka dapat dikatakan bahwa akurasi algoritma tersebut tidak dapat dibedakan. Namun, jika perbedaannya signifikan, maka satu algoritma memiliki akurasi yang lebih baik dibandingkan algoritma lainnya [21].

2.5. Kecepatan *Pathfinding*

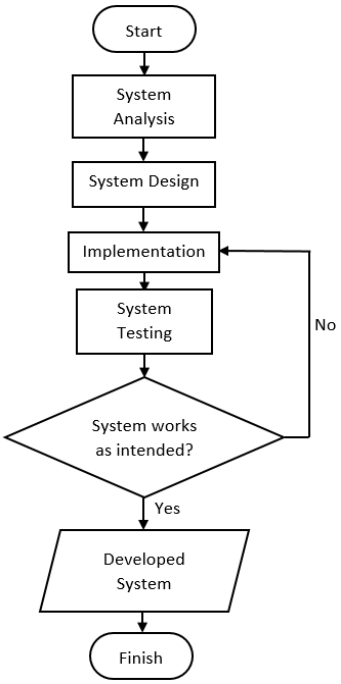
Kecepatan atau waktu tempuh adalah waktu yang dibutuhkan oleh subjek *pathfinding* untuk menghitung jalur dan bergerak dari titik awal ke titik tujuan. Faktor-faktor yang mempengaruhi kecepatan algoritma *pathfinding* meliputi kecepatan perhitungan jalur, jalur yang diambil, dan jarak [8]. Jika jalur yang diambil melibatkan banyak tikungan tajam, hambatan, atau lebih memilih jalur lurus daripada jalur diagonal jika diperlukan, maka kecepatan akan terpengaruh dan waktu tempuh akan menjadi lebih lama.

Sama seperti menghitung akurasi algoritma *video game*, kecepatan rata-rata pencarian jalur juga dapat dihitung dengan membagi total waktu yang diperlukan subjek untuk mencapai tujuan dengan jumlah simulasi yang dilakukan. Perhitungan ini dapat dilihat pada Persamaan 3.

$$Average\ Speed = \frac{Total\ time\ needed}{Total\ simulation\ amount} \quad (3)$$

2.6. Metode Penelitian

Penelitian dilakukan dengan membuat simulasi *video game* sederhana dengan tujuan mensimulasikan pergerakan NPC dari satu titik ke titik lainnya dalam suatu lingkungan. Lingkungan tersebut dibuat menjadi dua jenis, yaitu lingkungan tanpa hambatan dinamis dan lingkungan dengan hambatan dinamis. Simulasi *video game* dalam penelitian ini dibuat menggunakan *Unity Game Engine* sebagai *game engine*. Setelah simulasi *video game* dikembangkan, dilakukan pengujian untuk memastikan simulasi berjalan dengan baik. Setelah itu, data penelitian dapat dikumpulkan. Ada beberapa langkah dalam pembuatan simulasi *video game*. Penjelasan lebih rinci dapat dilihat pada Gambar 1.



Gambar 1. Alur Penelitian

Pengembangan simulasi dimulai dengan menganalisis kebutuhan sistem yang akan dikembangkan. Kebutuhan sistem dibagi menjadi dua, yaitu tujuan dan alur tugas. Tujuan dari sistem ini adalah untuk mensimulasikan algoritma *pathfinding* A* dan D* yang umum digunakan dalam *video game* dengan hambatan dinamis. Sedangkan alur tugas dari sistem ini dimulai dari peneliti yang menginisialisasi simulasi dan dua subjek NPC yang bergerak menuju titik tujuan. Selama NPC belum mencapai tujuan dan masih ada jalur yang tersedia untuk NPC mencapai tujuan, beberapa hambatan ditempatkan di depan NPC. Jika NPC telah mencapai tujuan, atau NPC tidak dapat mencapai tujuan, simulasi dihentikan dan waktu perjalanan NPC dihitung.

Desain sistem simulasi ini menentukan aspek *gameplay*, aset yang digunakan, dan implementasi algoritma A* dan D* itu sendiri. Karena tujuan penelitian ini adalah untuk mengetahui apakah terdapat perbedaan akurasi dan kesalahan pada sistem dengan dua algoritma yang berbeda, *gameplay* dari sistem yang dikembangkan hanya memiliki sistem *gameplay* pendukung yang minimalis. Beberapa sistem pendukung tersebut meliputi sistem untuk menempatkan tujuan *pathfinding* NPC, sistem untuk menempatkan hambatan, dan sistem *pathfinding* NPC itu sendiri yang mensimulasikan keadaan *video game* yang memiliki banyak hambatan dinamis dalam dunianya. Begitu pula dengan penggunaan aset dalam sistem. Aset yang digunakan hanya berupa aset primitif yang diperoleh dari *game engine* yang digunakan. Aset-aset ini dapat dilihat pada Gambar 2 dan 3.

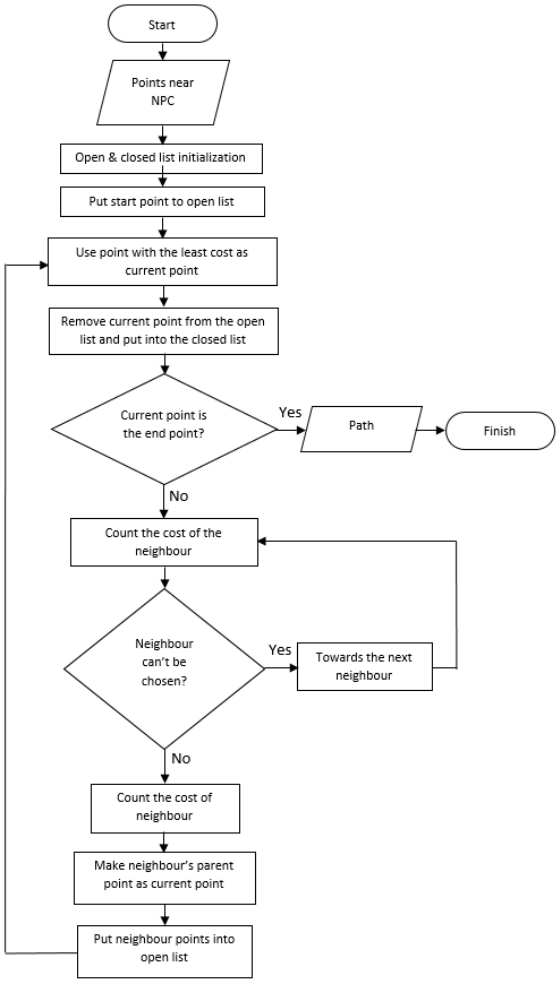
ISSN: 2614-1205

Gambar 2. Objek NPC

Gambar 3. Objek *Obstacle*

Implementasi algoritma A* dilakukan dengan menghitung *cost* dari titik-titik di sekitar subjek yang dihitung menggunakan fungsi heuristik *Euclidean Distance* dengan persamaan $f(n) = g(n) + h(n)$ Titik dengan *cost* terkecil akan dimasukkan ke dalam *closed list* dan diambil terlebih dahulu, sehingga proses pemberian *cost* pada titik-titik sangat penting dalam menentukan hasil pencarian. Proses pencarian titik dengan *cost* terkecil dari titik-titik di sekitarnya diulang untuk setiap titik yang dimasukkan ke dalam *closed list* hingga menemukan titik tujuan. Jika titik tujuan telah ditemukan, maka titik-titik dalam *closed list* dimasukkan ke dalam *path list* untuk digunakan sebagai jalur yang akan dilalui oleh subjek melalui proses *backtrack*.

Hasil *pathfinding* berupa daftar jalur (*path list*) adalah jalur yang harus dilalui oleh NPC. NPC hanya perlu mengambil titik-titik yang tersedia dalam *path list* yang disediakan oleh fungsi *backtrack*. Proses *pathfinding* menggunakan algoritma A* dapat dilihat pada Gambar 4.

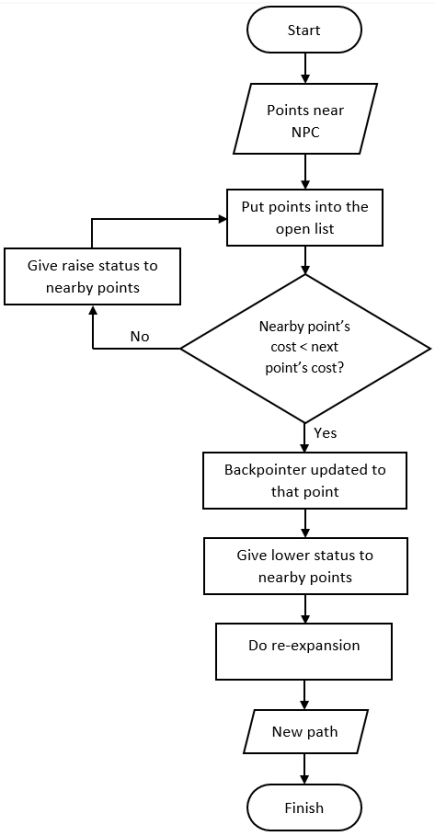


Gambar 4. Alur Algoritma A*

Sementara itu, implementasi algoritma D* pada dasarnya sama dengan implementasi A*, dimulai dari awal dengan mencari titik dengan *cost* terendah secara berulang dari titik awal hingga titik tujuan menggunakan fungsi heuristik *Euclidean Distance*, memasukkan titik-titik tersebut ke dalam *open list*, hingga mencapai titik tujuan, jalur ditemukan, dan NPC mulai bergerak. Perbedaan antara implementasi D* dan A* terletak pada metode penanganan rintangan. Setelah terjadi rintangan, titik-titik di sekitarnya dimasukkan kembali ke dalam *open list* dan dievaluasi.

Perhitungan heuristik dimulai lagi untuk setiap titik di sekitar titik saat ini atau titik di mana rintangan dinamis terjadi. Jika ada titik di sekitarnya yang memiliki *cost* lebih rendah dibandingkan dengan *cost* titik yang sudah ada dalam daftar jalur rute sebelumnya, algoritma akan memilih titik tersebut sebagai titik berikutnya dalam proses *backtrack*. Perhitungan heuristik dilakukan kembali untuk setiap titik di sekitar

titik tempat rintangan dinamis terjadi hingga mencapai titik tujuan. Proses perencanaan ulang jalur menggunakan algoritma D* dapat dilihat pada Gambar 5.



Gambar 5. Alur Algoritma D*

Implementasi dilakukan menggunakan *game engine* Unity dengan menggunakan bahasa pemrograman C#. Unity dipilih sebagai *game engine* yang digunakan dalam penelitian ini karena Unity umum digunakan dalam industri pengembangan video game. Oleh karena itu, diharapkan hasil penelitian ini dapat membantu banyak praktisi industri dalam pekerjaan mereka. Sementara itu, C# dipilih sebagai bahasa pemrograman untuk pengembangan sistem yang digunakan dalam penelitian ini karena Unity menggunakan C# sebagai bahasa pemrograman utamanya.

Untuk pengujian, peneliti menggunakan metode pengujian *Black Box*, di mana peneliti menggunakan semua fungsi yang ada dalam sistem dan membandingkan hasil aktual yang diperoleh dengan hasil yang diharapkan. Jika hasil aktual yang diperoleh berbeda dari ekspektasi awal hasil pengembang, maka sistem dianggap gagal.

2.7. Pengambilan dan Analisis Data

Parameter yang diuji dalam penelitian ini adalah kinerja NPC berdasarkan waktu tempuh dan akurasi atau kesalahan, apakah NPC berhasil mencapai titik tujuan atau tidak. Pengujian waktu tempuh diperoleh dengan mengukur waktu yang dibutuhkan oleh NPC sejak perhitungan jalur dilakukan hingga NPC mencapai titik tujuan. Pengujian waktu tempuh digunakan untuk menghitung perbedaan antara algoritma A* dan D*. Pengujian waktu tempuh dihitung dalam satuan milidetik agar keakuratan perhitungan tetap terjaga dan tidak berbeda antara satu simulasi dengan simulasi lainnya. Sementara itu, pengujian akurasi diperoleh dengan mencatat apakah NPC dapat mencapai titik tujuan atau tidak. Akurasi dihitung berdasarkan berhasil atau tidaknya suatu fungsi dieksekusi dengan benar setiap kali percobaan dilakukan. Jika NPC berhasil mencapai titik tujuan, berarti akurasi algoritma dinyatakan benar. Namun, jika NPC tidak mencapai titik tujuan, akurasi algoritma dianggap gagal. Pengujian akurasi digunakan untuk membandingkan akurasi kedua algoritma tersebut.

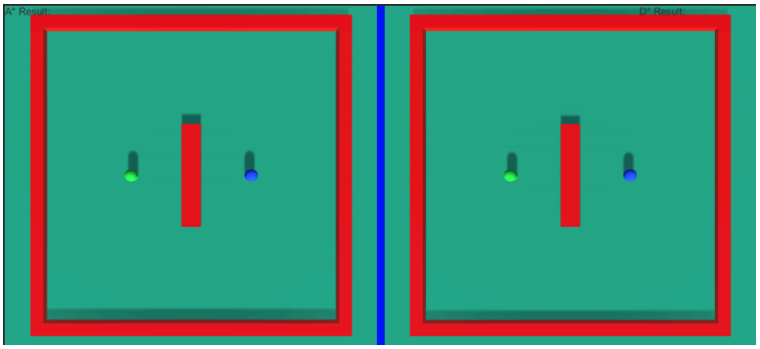
Dalam penelitian ini, data yang diperoleh adalah hasil simulasi yang dilakukan pada implementasi sistem pencarian rute menggunakan algoritma A* dan D* sebanyak 10 kali. Setiap simulasi dilakukan pada dua kasus lingkungan, yaitu lingkungan dengan rintangan statis dan lingkungan dengan rintangan dinamis. Data tersebut akan dibandingkan satu sama lain untuk mengetahui sejauh mana perbedaan akurasi dan kecepatan atau waktu tempuh antara algoritma A* dan D* dalam sistem pencarian rute NPC. Jika terdapat perbedaan yang signifikan, maka salah satu algoritma dinyatakan lebih baik. Jika tidak terdapat perbedaan yang signifikan, maka kedua algoritma dinyatakan sama. Data yang diperoleh adalah hasil simulasi yang dilakukan pada implementasi sistem pencarian rute menggunakan algoritma A* dan D* sebanyak 10 kali. Setiap simulasi dilakukan pada dua kasus lingkungan, yaitu lingkungan dengan rintangan statis dan lingkungan dengan rintangan dinamis.

3. HASIL DAN PEMBAHASAN

3.1. Hasil

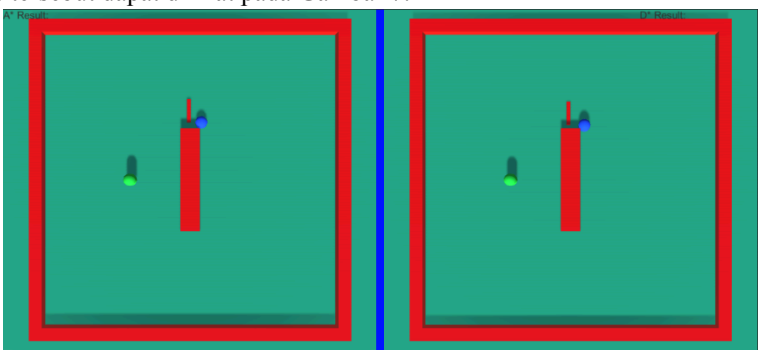
3.1.1. Deskripsi Simulasi

Perbandingan algoritma A* dan D* dilakukan dalam lingkungan simulasi yang memiliki dua subjek penelitian berupa NPC, dua target untuk masing-masing NPC yang menggunakan algoritma pencarian jalur berbeda (A* dan D*), rintangan statis, serta rintangan dinamis. Area penelitian yang dikembangkan dibagi menjadi dua area, yaitu area penelitian A* dan area penelitian D*. Tampilan simulasi yang dikembangkan menggabungkan area simulasi algoritma A* dan D* pada satu layar. Hal ini dimaksudkan untuk mempermudah pengamatan NPC dan penempatan rintangan dinamis selama simulasi. Tampilan simulasi dapat dilihat pada Gambar 6.



Gambar 6. *Simulation Display*

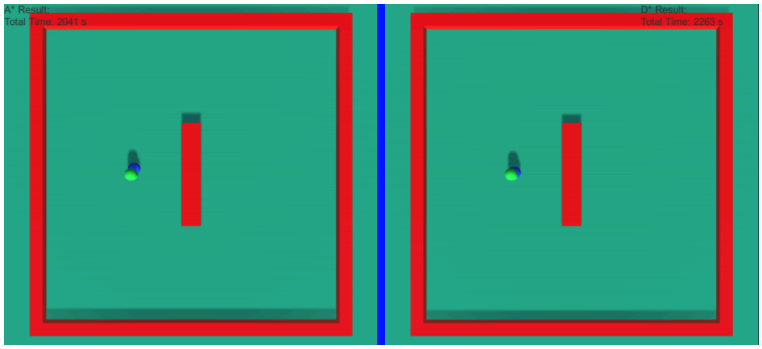
Jika sebuah rintangan dinamis ditempatkan di area simulasi D*, maka rintangan tersebut akan muncul di tempat yang sama di area simulasi A* sehingga subjek NPC selalu mendapatkan rintangan yang sesuai dan adil. Contoh penempatan rintangan dinamis tersebut dapat dilihat pada Gambar 7.



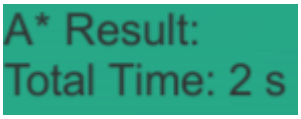
Gambar 7. Penempatan *Dynamic Obstacle*

Ketika simulasi dimulai, algoritma akan mencari jalur terpendek menuju titik tujuan. Karena kedua algoritma menggunakan fungsi heuristik yang sama, yaitu fungsi heuristik Jarak *Euclidean*, pencarian jalur dilakukan melalui arah diagonal jika memungkinkan untuk menemukan jalur menuju titik tujuan. Ketika jalur telah ditemukan, subjek NPC akan mulai bergerak menuju titik tujuan sesuai jalur yang telah diperoleh. Jika subjek NPC telah mencapai titik tujuan, simulasi berhenti, dan waktu tempuh subjek saat bergerak ke titik tujuan akan muncul pada tampilan simulasi. Keadaan simulasi ketika NPC telah sampai di titik tujuan dan waktu tempuh NPC dapat dilihat pada Gambar 8 dan Gambar 9.

ISSN: 2614-1205



Gambar 8. *State of Finished Simulation*



Gambar 9. Waktu tempuh NPC

3.1.2. Pengujian Simulasi

Dalam proses pengujian simulasi *video game* ini, penulis menggunakan jenis pengujian Black Box. Pengujian dilakukan dengan menjalankan sistem dan melihat *output*nya, apakah sesuai dengan yang diharapkan atau tidak. Pengujian yang dilakukan oleh penulis disajikan dalam Tabel 1.

Tabel 1. Hasil pengujian sistem menggunakan metode Black Box

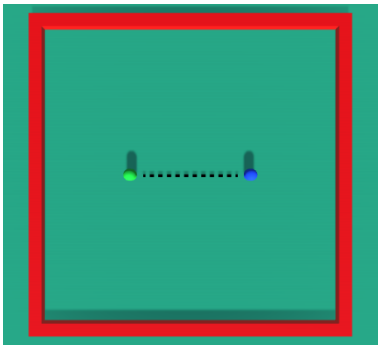
	Skenario Pengujian	Ekspektasi Hasil	Hasil Pengujian
1	Memulai Simulasi	NPC mulai mencari jalur kemudian bergerak sesuai dengan jalur	Sesuai
2	Menempatkan rintangan dinamis secara horizontal	Rintangan dinamis muncul sesuai lokasi yang ditempatkan secara horizontal	Sesuai
3	Menempatkan rintangan dinamis secara vertikal	Rintangan dinamis muncul sesuai lokasi yang ditempatkan secara vertikal	Sesuai
4	NPC tiba di titik tujuan	NPC berhenti dan kemudian waktu tempuh dihitung	Sesuai
5	NPC terhalang oleh rintangan dinamis	NPC berhenti tanpa menghitung waktu tempuh	Sesuai

6	Mereset simulasi	Lingkungan simulasi kembali normal dan waktu tempuh sebelumnya dihapus	Sesuai

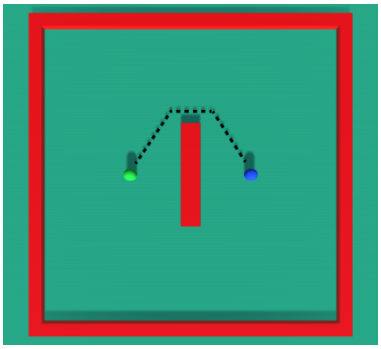
3.1.3. Inisialisasi Simulasi

Perbandingan algoritma A* dan D* dilakukan dalam 10 skenario (*scene*) yang dibagi menjadi dua tahap, yaitu tahap tanpa rintangan dinamis dan tahap dengan rintangan dinamis. Setiap *scene* yang diuji memiliki titik awal, titik akhir, dan bentuk rintangan yang berbeda. Tujuan dari perbandingan pada 10 *scene* ini adalah untuk mendapatkan variasi jalur yang berbeda dan waktu eksekusi algoritma. Berikut adalah *scene* yang digunakan dalam penelitian ini, di mana objek berwarna biru sebagai NPC, hijau sebagai titik tujuan/target, dan merah sebagai rintangan.

Scene 1 diuji tanpa rintangan statis dengan jalur lurus, dengan koordinat NPC (x,z) pada 15.0, 0.0 dan koordinat target pada -15.0, 0.0. *Scene* 2 dilakukan dengan 1 rintangan statis. Koordinat NPC tetap pada 15.0, 0.0 dan koordinat target pada -15.0, 0.0. Masing-masing *scene* ini dapat dilihat pada Gambar 10 dan Gambar 11.

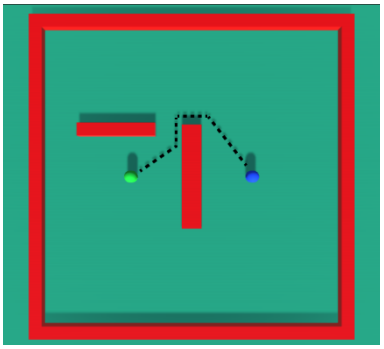


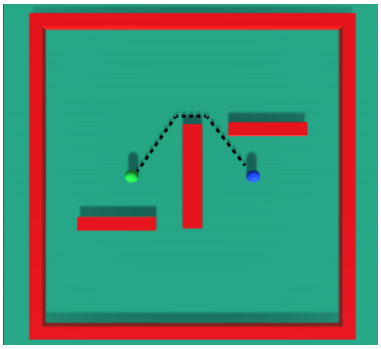
Gambar 10. Scene 1



Gambar 11. Scene 2

Scene 3 dilakukan dengan 2 rintangan. Koordinat NPC tetap pada 15.0, 0.0 dan koordinat target pada -15.0, 0.0. *Scene* 4 dilakukan dengan 3 rintangan. Koordinat NPC tetap pada 15.0, 0.0 dan koordinat target pada -15.0, 0.0. Masing-masing *scene* ini dapat dilihat pada Gambar 12 dan Gambar 13.

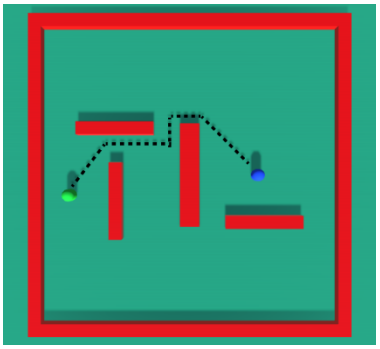




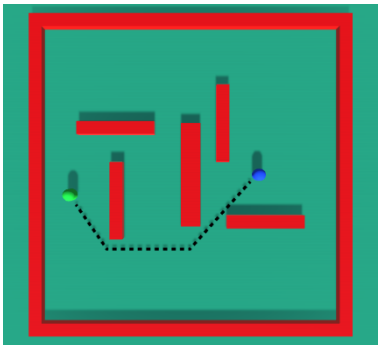
Gambar 12. Scene 3

Gambar 13. Scene 4

Scene 5 dilakukan dengan 4 rintangan. Koordinat NPC diubah menjadi 17.0, 0.0 dan koordinat target menjadi -30.0, -5.0. *Scene* 6 dilakukan dengan 5 rintangan. Koordinat NPC tetap pada 17.0, 0.0 dan koordinat target pada -30.0, -5.0. Masing-masing *scene* ini dapat dilihat pada Gambar 13 dan Gambar 14.

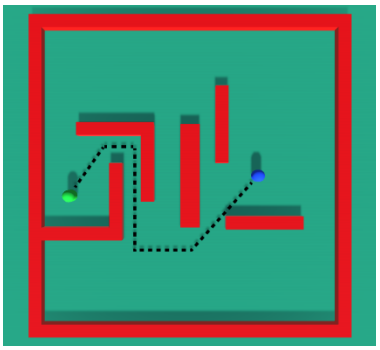


Gambar 13. Scene 5

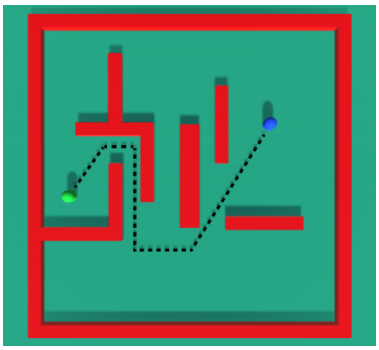


Gambar 14. Scene 6

Scene 7 dilakukan dengan 6 rintangan. Koordinat NPC tetap pada 17.0, 0.0 dan koordinat target pada -30.0, -5.0. *Scene* 8 dilakukan dengan 7 rintangan. Koordinat NPC diubah menjadi 20.0, 13.0 dan koordinat target menjadi -30.0, -5.0. Masing-masing *scene* ini dapat dilihat pada Gambar 15 dan Gambar 16.

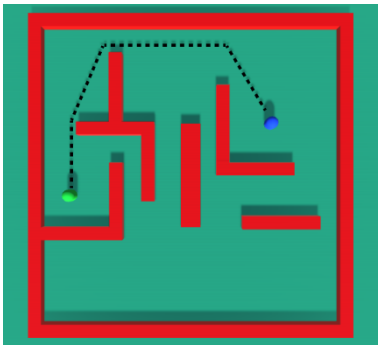


Gambar 15. Scene 7

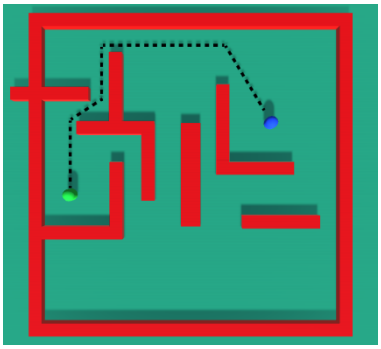


Gambar 16. Scene 8

Scene 9 dilakukan dengan 8 rintangan. Koordinat NPC tetap pada 20.0, 13.0 dan koordinat target pada -30.0, -5.0. *Scene* 10 dilakukan dengan 9 rintangan. Koordinat NPC tetap pada 20.0, 13.0 dan koordinat target pada -30.0, -5.0. Masing-masing *scene* ini dapat dilihat pada Gambar 17 dan Gambar 18.



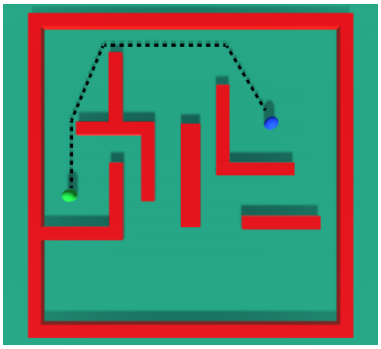
Gambar 17. Scene 9



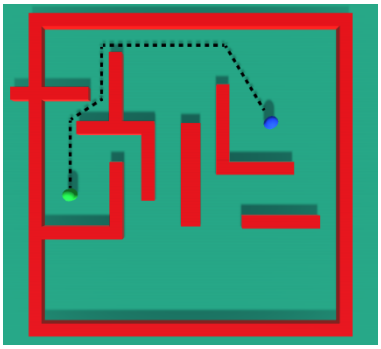
Gambar 18. Scene 10

Simulasi dilakukan dalam dua tahap, yaitu tanpa rintangan dinamis dan dengan rintangan dinamis. Pada tahap dengan rintangan dinamis, sebuah rintangan dinamis akan ditempatkan di tengah jalur antara NPC yang sedang berjalan menuju titik tujuan dan titik tujuan tersebut.

Pada *Scene* 1, rintangan dinamis ditempatkan di tengah lingkungan simulasi. Hal ini membuat NPC harus sedikit berbelok mengelilingi rintangan. Pada *Scene* 2, rintangan dinamis ditempatkan tepat di samping rintangan statis. Hal ini membuat NPC harus berjalan lebih jauh. Masing-masing *scene* ini dapat dilihat pada Gambar 19 dan Gambar 20.

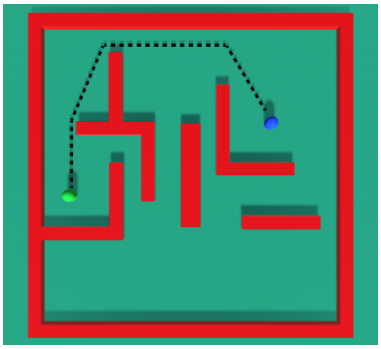


Gambar 19. Scene 1

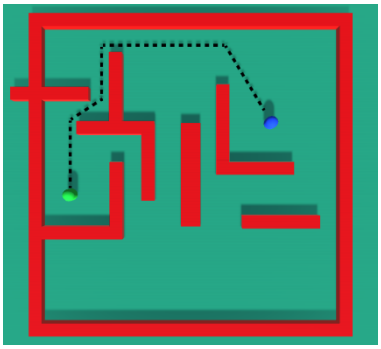


Gambar 20. Scene 2

Pada *Scene* 3, rintangan dinamis ditempatkan tepat di pintu menuju titik tujuan. Hal ini membuat NPC harus memutar mengelilingi rintangan statis. Pada *Scene* 4, rintangan dinamis ditempatkan tepat di pintu menuju titik tujuan. Hal ini membuat NPC harus mencari jalur lain menuju titik tujuan. Masing-masing *scene* ini dapat dilihat pada Gambar 21 dan Gambar 22.

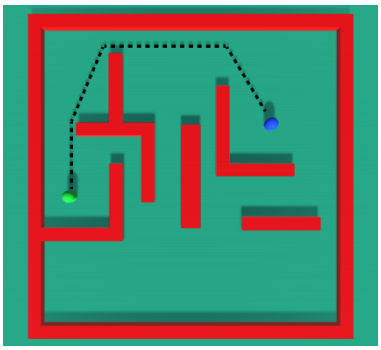


Gambar 21. Scene 3

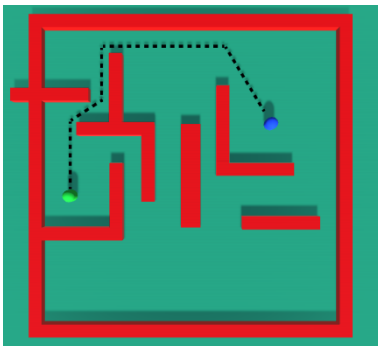


Gambar 22. Scene 4

Pada *Scene* 5, rintangan dinamis ditempatkan tepat di pintu menuju titik tujuan. Hal ini membuat NPC harus mencari jalur lain menuju titik tujuan. Pada *Scene* 6, rintangan dinamis ditempatkan tepat di pintu menuju titik tujuan. Hal ini membuat NPC harus mencari jalur lain menuju titik tujuan. Masing-masing *scene* ini dapat dilihat pada Gambar 23 dan Gambar 24.

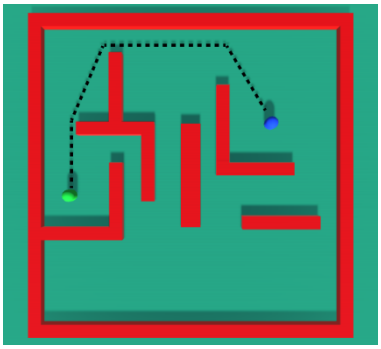


Gambar 23. Scene 5

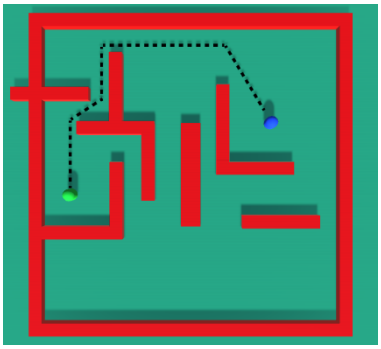


Gambar 24. Scene 6

Pada *Scene* 7, rintangan dinamis ditempatkan tepat di pintu menuju titik tujuan. Hal ini membuat NPC harus mencari jalur lain menuju titik tujuan. Pada *Scene* 8, rintangan dinamis ditempatkan tepat di pintu menuju titik tujuan. Hal ini membuat NPC harus mencari jalur lain menuju titik tujuan. Masing-masing *scene* ini dapat dilihat pada Gambar 25 dan Gambar 26.

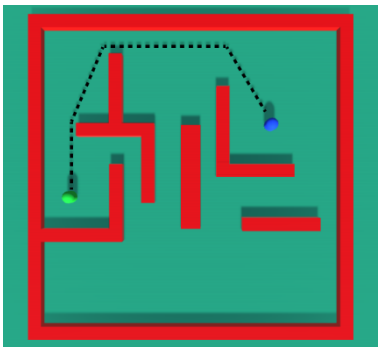


Gambar 25. Scene 7

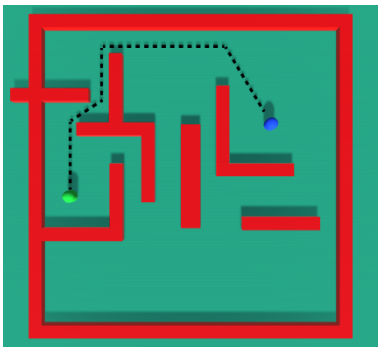


Gambar 26. Scene 8

Pada *Scene* 9, rintangan dinamis ditempatkan tepat di depan NPC. Hal ini membuat NPC harus sedikit berbelok melewati rintangan dinamis. Pada *Scene* 10, rintangan dinamis ditempatkan tepat di pintu menuju titik tujuan. Hal ini membuat NPC harus mencari jalur lain menuju titik tujuan. Masing-masing *scene* ini dapat dilihat pada Gambar 27 dan Gambar 28.



Gambar 27. Scene 9



Gambar 28. Scene 10

3.1.4. Waktu Tempuh

Waktu tempuh subjek NPC tanpa rintangan dinamis dihitung mulai dari waktu algoritma mulai mencari jalur, hingga NPC bergerak dan berhenti ketika mencapai titik tujuan. Waktu tempuh NPC dihitung pada setiap kasus simulasi dalam milidetik sebanyak 3 kali, kemudian rata-rata dihitung untuk dimasukkan ke dalam tabel. Perhitungan waktu tempuh yang dilakukan sebanyak 3 kali bertujuan untuk mencegah kesalahan dalam pengumpulan data atau kesalahan dalam menjalankan sistem, seperti objek yang tiba-tiba melambat, perangkat keras yang digunakan selama penelitian mengalami masalah kinerja, dan sebagainya. Rata-rata waktu tempuh untuk setiap NPC dengan algoritma yang berbeda dapat dilihat pada Tabel 2.

Tabel 2. Waktu tempuh NPC tanpa *Dynamic Obstacle*

	Jumlah *Static Obstacle*	Waktu tempuh A*	Waktu tempuh D*
1	0 *obstacles*	1252 milidetik	1356 milidetik
2	1 *obstacles*	2021 milidetik	2156 milidetik
3	2 *obstacles*	2016 milidetik	2177 milidetik
4	3 *obstacles*	2020 milidetik	2160 milidetik
5	4 *obstacles*	2892 milidetik	3098 milidetik
6	5 *obstacles*	2857 milidetik	3095 milidetik
7	6 *obstacles*	3346 milidetik	3630 milidetik
8	7 *obstacles*	4020 milidetik	4330 milidetik
9	8 *obstacles*	4338 milidetik	4670 milidetik
10	9 *obstacles*	4381 milidetik	4731 milidetik

Seperti sebelumnya, waktu perjalanan untuk subjek NPC dengan hambatan dinamis dihitung mulai dari saat algoritma mulai mencari jalur, hingga NPC mencapai titik tujuan. Namun, pada tahap ini, hambatan dinamis ditempatkan di tengah jalur yang telah dihitung oleh algoritma. Jadi, subjek NPC akan terhambat oleh hambatan dinamis sepanjang perjalanannya menuju titik tujuan. Waktu perjalanan NPC tetap dihitung dalam setiap kasus simulasi sebanyak 3 kali, dengan penempatan hambatan dinamis pada titik yang sama di setiap iterasi perhitungan. Penempatan hambatan dinamis pada titik yang sama dilakukan untuk mencegah perbedaan jalur yang diambil antara 3 kali perhitungan waktu perjalanan. Waktu perjalanan rata-rata untuk setiap NPC dengan algoritma yang berbeda dapat dilihat pada Tabel 3.

Tabel 3. Waktu tempuh NPC dengan *dynamic obstacles*

Jumlah *obstacle*	Waktu tempuh A*	Waktu tempuh D*
0 *obstacle* + 1 *dynamic obstacle*	- (*error*)	1470 milidetik
1 *obstacles* + 1 *dynamic obstacle*	- (*error*)	2413 milidetik
2 *obstacles* + 1 *dynamic obstacle*	- (*error*)	3848 milidetik
3 *obstacles* + 1 *dynamic obstacle*	- (*error*)	3173 milidetik
4 *obstacles* + 1 *dynamic obstacle*	- (*error*)	4384 milidetik
5 *obstacles* + 1 *dynamic obstacle*	- (*error*)	3325 milidetik
6 *obstacles* + 1 *dynamic obstacle*	- (*error*)	4658 milidetik
7 *obstacles* + 1 *dynamic obstacle*	- (*error*)	6275 milidetik

8 *obstacles* + 1 *dynamic obstacle*	- (*error*)	4670 milidetik
9 *obstacles* + 1 *dynamic obstacle*	- (*error*)	5339 milidetik

3.2. Pembahasan

3.2.1. Perbedaan Simulasi tanpa *Dynamic Obstacle*

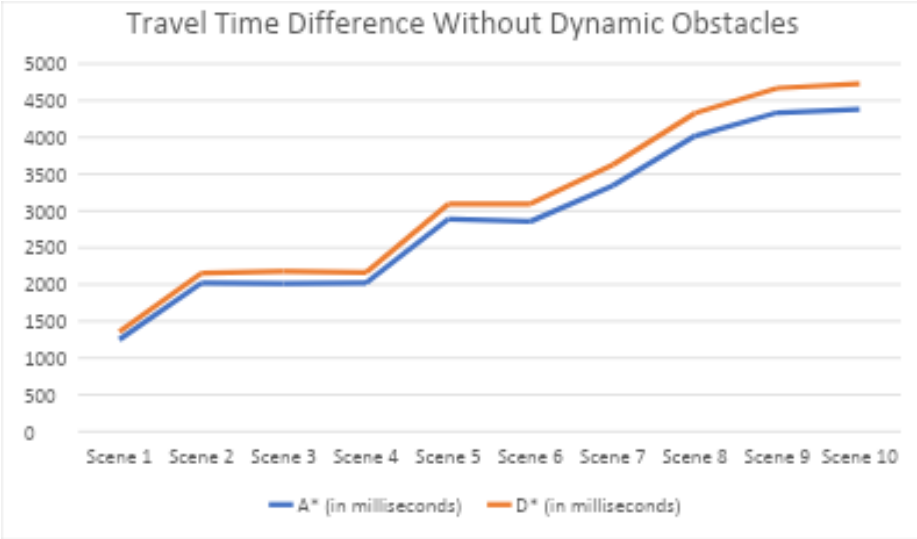
Waktu perjalanan subjek NPC tanpa hambatan dinamis dihitung mulai dari saat algoritma mulai mencari jalur, hingga NPC bergerak dan berhenti ketika mencapai titik tujuan. Waktu perjalanan NPC dihitung dalam setiap kasus simulasi dalam milidetik sebanyak 3 kali, kemudian rata-rata dihitung untuk dimasukkan ke dalam tabel. Perhitungan waktu perjalanan yang dilakukan 3 kali ini dilakukan untuk mencegah kesalahan dalam pengumpulan data atau kesalahan dalam menjalankan sistem, seperti objek yang tiba-tiba memperlambat, perangkat keras yang digunakan selama penelitian mengalami masalah kinerja, dan sebagainya. Waktu perjalanan rata-rata untuk setiap NPC dengan algoritma yang berbeda dapat dilihat pada Tabel 4.

Tabel 4. Perbandingan waktu tempuh NPC tanpa *Dynamic Obstacle*

Scene	Waktu tempuh A*	Waktu tempuh D*	Perbedaan waktu tempuh	Perbedaan dalam persen
1	1252 milidetik	1356 milidetik	104 milidetik	7,67%
2	2021 milidetik	2156 milidetik	135 milidetik	6,26%
3	2016 milidetik	2177 milidetik	161 milidetik	7,4%
4	2020 milidetik	2160 milidetik	140 milidetik	6,5%
5	2892 milidetik	3098 milidetik	206 milidetik	6,65%
6	2857 milidetik	3095 milidetik	238 milidetik	7,69%
7	3346 milidetik	3630 milidetik	284 milidetik	7,82%
8	4020 milidetik	4330 milidetik	310 milidetik	7,16%
9	4338 milidetik	4670 milidetik	332 milidetik	7,11%
10	4381 milidetik	4731 milidetik	350 milidetik	7,4%

Gambar 29 menunjukkan waktu perjalanan NPC dalam simulasi tanpa hambatan dinamis dengan sedikit lebih banyak konteks. Karena scene 1 belum memiliki hambatan, waktu yang dibutuhkan relatif pendek dan perbedaan antara NPC yang menggunakan A* dan D* hanya sedikit. Scene 2 mengalami peningkatan waktu perjalanan dan perbedaan yang signifikan antara kedua algoritma, karena adanya hambatan yang memaksa algoritma untuk menghitung jalur secara diagonal. Jalur diagonal merupakan tantangan bagi algoritma pencarian jalur, baik A* maupun D*.

Scene 2 hingga 4 tidak menunjukkan peningkatan waktu perjalanan dan perbedaan algoritma, karena perubahan simulasi yang relatif sedikit. Scene 5 memberikan peningkatan signifikan dalam waktu perjalanan dan perbedaan algoritma seperti scene 2, dan tetap stabil hingga scene 6. Hal ini disebabkan oleh belokan tajam yang disebabkan oleh salah satu hambatan dekat titik tujuan. Scene 7 hingga 10 menunjukkan peningkatan bertahap dalam waktu perjalanan dan perbedaan dalam algoritma pencarian jalur, karena simulasi dan perhitungan jalur semakin kompleks. Perbedaan waktu perjalanan dapat dilihat pada Gambar 29.



Gambar 29. Grafik perbandingan waktu tempuh NPC tanpa *Dynamic Obstacle*

Sementara itu, untuk akurasi NPC, tidak ada perbedaan antara algoritma A* dan D*. Karena kedua algoritma berhasil menghitung jalur sebelum NPC mulai bergerak dan tanpa hambatan tambahan setelahnya, kedua NPC dapat sampai di titik tujuan mereka.

3.2.2. Perbedaan Simulasi dengan *Dynamic Obstacle*

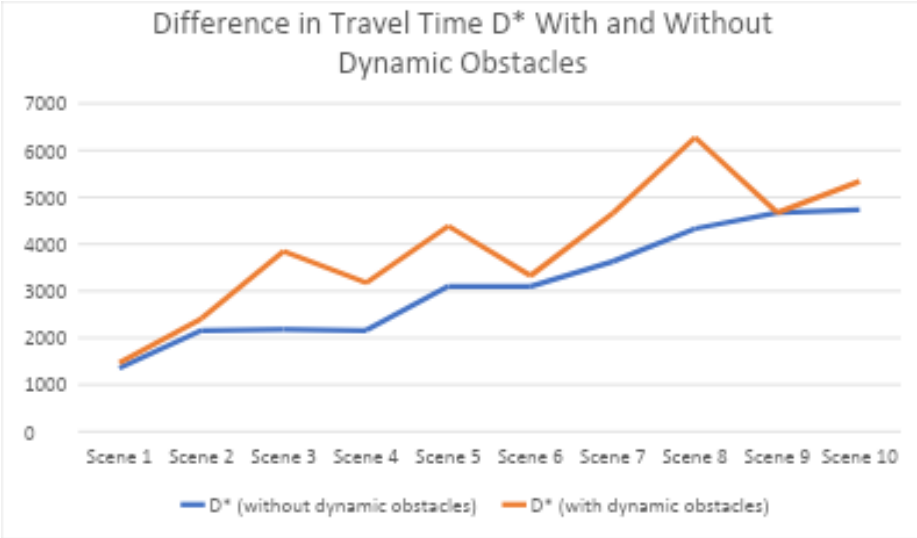
Seperti sebelumnya, waktu perjalanan untuk subjek NPC dengan hambatan dinamis dihitung mulai dari saat algoritma mulai mencari jalur, hingga NPC mencapai titik tujuan. Namun, pada tahap ini, hambatan dinamis ditempatkan di tengah jalur yang telah dihitung oleh algoritma. Jadi, subjek NPC akan terhambat oleh hambatan dinamis sepanjang perjalanannya menuju titik tujuan. Waktu perjalanan NPC tetap dihitung dalam setiap kasus simulasi sebanyak 3 kali, dengan penempatan hambatan dinamis pada titik yang sama di setiap iterasi perhitungan. Penempatan hambatan dinamis pada titik yang sama dilakukan untuk mencegah perbedaan jalur yang diambil antara 3 kali perhitungan waktu perjalanan. Waktu perjalanan rata-rata untuk setiap NPC dengan algoritma yang berbeda dapat dilihat pada Tabel 5

Tabel 5. Waktu tempuh NPC travel dengan *dynamic obstacles*

Jumlah *obstacle*	A* Travel Time	D* Travel Time

0 *obstacle* + 1 *dynamic obstacle*	- (*error*)	1470 milidetik
1 *obstacle* + 1 *dynamic obstacle*	- (*error*)	2413 milidetik
2 *obstacles* + 1 *dynamic obstacle*	- (*error*)	3848 milidetik
3 *obstacles* + 1 *dynamic obstacle*	- (*error*)	3173 milidetik
4 *obstacles* + 1 *dynamic obstacle*	- (*error*)	4384 milidetik
5 *obstacles* + 1 *dynamic obstacle*	- (*error*)	3325 milidetik
6 *obstacles* + 1 *dynamic obstacle*	- (*error*)	4658 milidetik
7 *obstacles* + 1 *dynamic obstacle*	- (*error*)	6275 milidetik
8 *obstacles* + 1 *dynamic obstacle*	- (*error*)	4670 milidetik
9 *obstacles* + 1 *dynamic obstacle*	- (*error*)	5339 milidetik

Jika melihat perbedaan waktu perjalanan subjek NPC dengan algoritma D* dengan dan tanpa hambatan dinamis, perbedaan tersebut tampak meningkat secara signifikan pada scene 3, scene 5, dan scene 8. Hal ini disebabkan oleh hambatan dinamis yang membuat algoritma harus menghitung jalur memutar untuk mencapai titik tujuan. Begitu pula NPC harus mengambil jalur memutar menuju titik tujuan. Jika jalur yang dihasilkan dari perhitungan ulang tidak memerlukan belokan tajam atau melewati hampir arah yang sama, perbedaan waktu perjalanan tidak signifikan atau hampir tidak ada. Perbedaan waktu perjalanan D* dapat dilihat pada Gambar 30.



Gambar 30. Perbedaan waktu tempuh D* dengan dan tanpa *Dynamic Obstacle*

4. SIMPULAN

Makalah ini membahas peningkatan pencarian jalur di lingkungan dengan rintangan dinamis menggunakan algoritma D* yang menambahkan metode perencanaan ulang jalur. Meskipun memiliki waktu tempuh yang lebih tinggi dibandingkan algoritma A*, hasil yang diperoleh dari simulasi dengan rintangan dinamis menunjukkan bahwa algoritma D* memiliki akurasi yang jauh lebih tinggi. Peningkatan waktu tempuh antara algoritma A* dan D* berkisar antara 6,26% hingga 7,82%. Dengan demikian, dapat disimpulkan bahwa algoritma pencarian jalur D* dapat lebih baik dibandingkan algoritma pencarian jalur A* dalam beberapa skenario lingkungan dengan rintangan dinamis, khususnya skenario dengan jalur yang kompleks.

5. REFERENSI

[1] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals*. MIT Press, 2003.

[2] K. Budiman, N. Zaatsiyah, U. Niswah, F. Muhanna, and N. Faizi, "Analysis of Sexual Harassment Tweet Sentiment on Twitter in Indonesia using Naïve Bayes Method through National Institute of Standard and Technology Digital Forensic Acquisition Approach," *Journal of Advances in Information Systems and Technology*, vol. 2, no. 2, pp. 21–30, 2020, [Online]. Available: https://journal.unnes.ac.id/sju/index.php/jaist

[3] Y. Sazaki, H. Satria, A. Primanita, and M. Syahroyni, "Analisa Perbandingan Algoritma A* dan Dynamic Pathfinding Algorithm dengan Dynamic Pathfinding Algorithm untuk NPC pada Car Racing Game," *Jurnal Teknologi Informasi Dan Ilmu Komputer*, vol. 5, no. 1, p. 95, 2018, doi: https://doi.org/10.25126/jtiik.201851544.

[4] K. Budiman, S. Subhan, and D. A. Efrilianda, "Business process reengineering to support sustainability of the sales commodities in large transaction with quotation system," *Sci. J. Inform*, vol. 8, no. 1, pp. 84–91, 2021.

[5] R. Toshniwal, "How to select Performance Metrics for Classification Models," *Medium*, 2020.

[6] A. F. Meades, "The Well-Played Glitch: practice and meaning in Glitching communities," *Well Played Journal*, vol. 2, no. 2, pp. 79–98, 2013.

[7] K. Budiman and Y. N. Ifriza, "Analysis of earthquake forecasting using random forest," *Journal of Soft Computing Exploration*, vol. 2, no. 2, pp. 153–162, 2021.

[8] H. Wang, S. Lou, J. Jing, Y. Wang, W. Liu, and T. Liu, "The EBS-A* algorithm: An improved A* algorithm for path planning," *PLoS One*, vol. 17, no. 2, pp. 1–27, 2022, doi: https://doi.org/10.1371/journal.pone.0263841.

[9] P. Cowling *et al.*, "Search in Real-Time Video Games," *Artificial and Computational Intelligence in Games*, vol. 6, pp. 1–19, 2013, doi: https://doi.org/http://dx.doi.org/10.4230/DFU.Vol6.12191.1.

[10] A. Setiawan, P. Harsadi, and S. Siswanti, "Penerapan Pathfinding Menggunakan Algoritma A* Pada Non Player Character (NPC) Di Game," *Jurnal Ilmiah SINUS*, vol. 17, no. 2, p. 39, 2019, doi: https://doi.org/10.30646/sinus.v17i2.423.

[11] J. Y. Wang and Y. Bin Lin, "An effective method of pathfinding in a car racing game," in *2010 The 2nd International Conference on Computer and Automation Engineering, ICCAE 2010*, 2010, pp. 544–547. doi: https://doi.org/10.1109/ICCAE.2010.5452074.

[12] F. A. Raheem and U. I. Hameed, "Path Planning Algorithm using D* Heuristic Method Based on PSO in Dynamic Environment," *American Scientific Research Journal for Engineering, Technology, and Sciences*, vol. 49, no. 1, pp. 257–271, 2018.

[13] J. Guo, L. Liu, Q. Liu, and Y. Qu, "An improvement of D* algorithm for mobile robot path planning in partial unknown environment," in *2009 2nd International Conference on Intelligent Computing Technology and Automation, ICICTA 2009*, 2009, pp. 394–397. doi: https://doi.org/10.1109/ICICTA.2009.561.

[14] B. Brathwaite and I. Schreiber, *Challenges for Game Designers*, 1st ed. Course Technology, 2009.

[15] B. Purkiss and I. Khaliq, "A study of interaction in idle games & perceptions on the definition of a game," in *2015 IEEE Games Entertainment Media Conference, GEM 2015*, 2016. doi: https://doi.org/10.1109/GEM.2015.7377233.

[16] P. E. Hart, N. J. Nilsson, and R. Bertram, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968, doi: https://doi.org/10.1137/0109044.

[17] S. Kusumadewi and H. Purnomo, *Penyelesaian Masalah Optimasi dengan Teknik-Teknik Heuristik*. Graha Ilmu, 2005.

[18] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings - IEEE International Conference on Robotics and Automation, pt 4*, 1994, pp. 3310–3317. doi: https://doi.org/10.1007/978-1-4615-6325-9_11.

[19] S. Koenig, C. Tovey, and Y. Smirnov, "Performance Bounds for Planning in Unknown Terrain," *Artif Intell*, vol. 147, no. 1–2, pp. 253–279, 2003, doi: https://doi.org/10.1016/S0004-3702(03)00062-6.

[20] E. Prasetyo, *Data mining: Konsep dan Aplikasi Menggunakan MATLAB*, 1st ed. CV Andi Offset, 2012.

[21] M. Hofmann and R. Klinkenberg, *RapidMiner Data Mining Use Cases and Business Analytics Applications*, 1st ed. Chapman and Hall/CRC, 2014.