OPEN ACCESS
Indo-JC Indonesia Journal of Computing
ISSN 2460-9056
socj.telkomuniversity.ac.id/indojc

Ind. Journal on Computing
Vol. 7, Issue. 3, December 2022. pp. 81-94
doi:10.34818/indojc.2022.7.3.677

Implementasi Metode Pathfinding dengan Algoritma A* pada Game Rogue-like menggunakan Unity

Elvitro Gumelar Agung [#1], Dania Eridani [#2], Adnan Fauzi [#3]

#Departemen Teknik Komputer, Fakultas Teknik, Universitas Diponegoro
Jalan Prof. Soedarto, S.H., Kampus Undip Tembalang, Semarang, Indonesia 50275

[1] elvitrogumelar@students.undip.ac.id
[2] dania@live.undip.ac.id
[3] adnan@ce.undip.ac.id

Abstract

In the current era, digital media is very well known by the public, digital world at this time has a fairly rapid development, one of the digital media known by the public is games. Along with the times, game media has also developed, one of the applications for developing games is Unity, the program can develop games with the support of the C# programming language, from games that were originally simple, until now artificial intelligence is also taking part in making games. The application of artificial intelligence in games is expected to make games more interactive and can improve the playing experience for users, one method of implementing artificial intelligence in games is the pathfinding system, this system is an automatic path finding for a unit in a game, algorithm A * will be used in the search for the path it takes the shortest time and with the shortest path. This A* algorithm is an algorithm that can be used to find the shortest path with the shortest time. Pathfinding or path planning is one of the problems in making computer games, it usually includes the movement of the AI system of a unit in the game, the A* algorithm is one method that can solve this problem, the algorithm is a path-finding system from the starting point and end point by avoiding the given obstacle

Keywords: Game, Unity, A* Alogarithm, Pathfinding

Abstrak

Pada era sekarang, media digital sudah sangat dikenal oleh masyarakat, dunia digital pada masa ini memiliki perkembangan yang cukup pesat, salah satu media digital yang dikenal oleh masyarakat adalah game. Seiring bertambahnya zaman, media game juga mengalami perkembangan, salah satu aplikasi untuk mengembangkan game adalah Unity, program tersebut dapat mengembangkan game dengan dukungan bahasa pemrograman C#, dari game yang pada awalnya sederhana, hingga sekarang kecerdasan buatan juga ikut ambil bagian dalam pembuatan game. Penerapan kecerdasan buatan dalam game diharapkan dapat membuat game menjadi lebih interaktif dan dapat meningkatkan pengalaman bermain bagi pengguna, salah satu metode dalam implementasi kecerdasan buatan dalam game adalah pada sistem pathfinding, sistem ini merupakan pencarian jalur otomatis bagi suatu unit di dalam sebuah game, algoritma A* akan digunakan dalam pencarian jalur tersebut dibutuhkan waktu yang singkat dan dengan jalur terpendek. Algoritma A* ini adalah suatu algoritma yang dapat digunakan untuk mencari jalur terpendek dengan waktu yang paling singkat. Pathfinding atau perencanaan jalur adalah salah satu masalah dalam pembuatan game komputer, hal tersebut biasanya mencangkup pergerakan dari sistem AI suatu unit di dalam game, algoritma A* merupakan salah satu metode yang dapat mengatasi masalah tersebut, algoritma tersebut merupakan sistem pencarian jalur dari titik awal dan titik akhir dengan menghindari halangan yang diberikan

Kata Kunci: Game, Unity, Algoritma A*, Pathfinding

Received on September, 2022. Accepted on December, 2022

I. PENDAHULUAN

Pathfinding telah menjadi masalah yang cukup umum pada industri game. Pergerakan dari suatu agen adalah salah satu tantangan terbesar dalam desain AI dalam *game* komputer. Strategi pencarian jalur pada umumnya digunakan sebagai inti dari setiap sistem pergerakan AI. Tantangan paling umum dari pencarian jalan dalam game adalah cara menghindari rintangan dan mencari jalur paling cepat dalam berbagai tempat [1]. Sistem pencarian jalur secara *real-time* pada *game* memiliki beberapa faktor yang harus dimaksimalkan seperti durasi untuk menemukan jalur, keakuratan jalur dan dibuat dan efisiensi waktu untuk bergerak menuju titik tujuan. Algoritma A* adalah metode yang dapat digunakan sebagai solusi untuk permasalahan sistem pencarian jalur [2]. Algoritma A* adalah akan menghitung arah terbaik untuk bergerak dan kemudian dapat menelusuri kembali jalur tersebut, tidak hanya menemukan jalur antara titik awal dan titik tujuan tetapi algoritma ini juga akan menemukan jalur terpendek dengan cepat. Algoritma A* merupakan salah satu algoritma yang paling sering digunakan dalam pencarian jalur. Pada game yang dibuat, algoritma pencarian A* diimplementasikan untuk menemukan jalur terpendek antara titik awal dan titik tujuan [1]. Pada permainan rogue-like ini, algoritma ini menghitung jarak berdasarkan petak yang akan diberikan nilai untuk membuat jalur ke titik tujuan, peta yang diberikan oleh permainan ini memiliki objek yang merupakan halangan, *unit* yang dibuat harus dapat berjalan menuju titik akhir dengan melewati halangan yang diberikan, jika pencarian jalur tidak dirancang dengan baik, maka unit tidak dapat melewati halangan yang diberikan, peta yang dibuat dalam permainan ini terus berubah berdasarkan tingkatan level yang diberikan sehingga dibutuhkan algoritma yang dapat bekerja secara real-time. *Rogue-like* adalah permainan komputer yang dimainkan dengan cara eksplorasi pada peta yang dibuat secara acak. Algoritma A* dipilih karena algoritma ini dapat berjalan secara real-time dan dapat digunakan pada tempat atau area yang bersifat dinamis atau terus berubah-ubah seperti dalam *game rogue-like* dengan peta yang diatur secara acak, karena peta yang diberikan terus berubah, maka metode *Machine Learning* kurang efektif jika diterapkan karena metode tersebut harus melalui proses training pada unit, proses tersebut memakan waktu yang tidak singkat sehingga dapat mempengaruhi proses pengembangan pada permainan ini. Algoritma A* adalah pengembangan dari Algoritma Dijkstra, yaitu algoritma yang bertujuan untuk memproses perencanaan jalur yang efisien antara beberapa titik dengan menggunakan fungsi heuristic. Algoritma A* menemukan jalur dengan nilai terendah dari titik awal yang diberikan ke titik tujuan. A* [3]..

II. TINJAUAN PUSTAKA

A. Kajian Penelitian Terdahulu

Kajian penelitian terdahulu adalah penelusuran terhadap penelitian yang berhubungan dengan permasalahan yang sudah dibahas sebelumnya dan dapat dijadikan sebagai bahan kajian penelitian dengan permasalahan yang menyerupai dengan penelitian yang akan dilakukan sehingga dapat dijadikan referensi.

Pada penelitian berjudul "Application of A-Star Algorithm on Pathfinding Game" oleh Ade Candra, dan Mohammad Andri Budiman, algoritma A* dapat dikembangkan pada sebuah *game* 2D pada *platform* Android. Permainan pada penelitian ini memiliki beberapa objek seperti musuh, penghalang, dan objek pohon yang harus dilindungi. Algoritma A* pada penelitian pertama digunakan untuk menentukan jalur terdekat yang dapat ditempuh musuh menuju pohon. Selanjutnya, pemain harus berusaha melindungi pohon dari musuh dengan menyentuh musuh atau menggunakan penghalang untuk menjaga pohon hidup. Permainan akan berakhir ketika musuh berhasil menebang pohon. Penelitian ini akan menunjukkan jalan yang akan dilewati musuh untuk sampai ke pohon dan waktu yang dibutuhkan untuk menempuh jalan tersebut [3], hasil yang didapatkan dari penelitian tersebut adalah jumlah node berbanding lurus dengan panjang rute dan waktu memproses jalur. Selanjutnya jumlah hambatan juga berbanding lurus dengan node. Semakin banyak objek yang menjadi, semakin lama waktu pemrosesan yang diperlukan untuk menemukan rute dari setiap node.

Penelitian dengan judul "Comparative Analysis of Pathfinding Algorithms A*, Dijkstra, and BFS on Maze Runner Game" oleh Silvester Dian Handy Permana, dan Ketut Bayu Yogha Bintoro menguji algoritma A* dengan beberapa metode pencarian jalur seperti Algoritma Dijkstra dan Algoritma *Breadth-First Search* (BFS)

menggunakan *game* "Maze Runner", yaitu permainan di mana satu unit bergerak ke titik akhir dengan melewati halangan berbentuk labirin yang diberikan, permainan tersebut digunakan sebagai uji perbandingan. proses perbandingan pada algoritma dilakukan dengan mengukur 3 variabel komputasi pencarian jalan. Ketiga variabel tersebut adalah pengukuran waktu proses, panjang jalur, dan jumlah blok yang dimainkan dalam proses komputasi yang ada. Hasil dari penelitian kedua tersebut digunakan untuk membantu pengembang game untuk mengimplementasikan algoritma terbaik untuk membuat permainan berbasis labirin [4], kesimpulan yang didapat dari penelitian tersebut adalah algoritma A*, Dijkstra, dan *Breadth-First Search* dapat digunakan untuk mencari jalur terpendek di "Maze Runner Game". A* adalah algoritma terbaik dalam pathfinding terutama di *game/grid* Maze. Hal ini didukung dengan proses komputasi minimal dan pencarian yang singkat.

Penelitian oleh Ghani Mutaqin, dan Juniardi Nur Fadilah dengan judul "Implementasi Metode Path Finding dengan Penerapan Algoritma A-Star untuk Mencari Jalur Terpendek pada *Game* "Jumrah Launch Story" menggunakan algoritma A* untuk mengatur pergerakan NPC. NPC tersebut ialah musuh sebagai NPC tanpa AI dan Bos yaitu "Jumrah" sebagai NPC dengan AI. Pengujian ini dilakukan terhadap algoritma tersebut dengan acuan perbedaan perilaku karakter yang menggunakan AI dan tidak menggunakan AI ketika game dimainkan [5] teknologi augmented reality dapat mencapai nilai 89,5% disebut efektif sebagai media pembelajaran [6], dari penelitian tersebut NPC yang menggunakan algoritma ini dapat mengejar player dan membunuh saat player kurang berhati-hati. NPC juga bisa mengejar walaupun player bersembunyi dan terdapat penghalang.

Pada penelitian selanjutnya berjudul "Improved A-Star Algorithm for Long-Distance Off-Road Path Planning Using Terrain Data Map" oleh Zhonghua Hong, dan Pengfei Sun, algoritma A* digunakan untuk mengatasi masalah terkait dengan tugas perencanaan jalur jarak jauh pada transportasi *off-road,* algoritma A* ditingkatkan untuk mengurangi waktu eksekusi dan meningkatkan efisiensi dari algoritma tanpa mengubah proses pencarian jalur dari Algoritma A* konvensional [6], Algoritme A* pada penelitian terbukti meningkatkan efisiensi peningkatan minimal 4,6 kali, dan akselerasi maksimum yang mencapai 550 kali pada perencanaan jalur *off-road* jarak jauh.

Berdasarkan penelitian yang sudah dilakukan pada 4 penelitian terdahulu, persamaan dari 4 penelitian tersebut adalah semua penelitian menggunakan algoritma A* dan metode tersebut digunakan untuk pencarian jalur atau rute terpendek, sementara perbedaan dari 4 penelitian tersebut adalah penerapan dari metode algoritma A* digunakan bukan hanya pada game namun dapat juga digunakan di lingkungan sekitar seperti pencarian jalur off-road. Dalam pembuatan *game*, algoritma A* digunakan pada *game* yang melibatkan pembuatan jalur pada suatu unit.

*B. Algoritma A**

Algoritma A* adalah salah satu algoritma perencanaan jalur yang dapat diterapkan pada metrik atau konfigurasi pada topologi ruang. Algoritma ini menggunakan kombinasi pencarian heuristik dan pencarian berdasarkan jalur yang terpendek jalur. Setiap node dalam ruang konfigurasi dievaluasi nilainya [8]. Algoritma A* pertama kali diusulkan di Hart et al. (1968), algoritma ini digunakan untuk mencari nilai minimum dari titik awal ke titik akhir. Algoritma A* adalah algoritma berbasis grid. Titik awal dan titik akhir dengan informasi jalur dalam peta yang dihasilkan dipilih untuk menjalankan algoritma A* [7].

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

H(v) adalah jarak heuristik dari sel ke titik akhir dan G(v) adalah panjang jalur dari titik awal ke titik akhir melalui urutan sel yang dipilih. Setiap node atau sel yang berdekatan dievaluasi oleh nilai F(v). Sel dengan nilai F(v) terendah dipilih sebagai urutan dari jalur. Keuntungan dari algoritma ini adalah bahwa jarak yang digunakan dapat dimodifikasi atau jarak lain dapat ditambahkan kembali [8]

C. Perencanaan Jalur (Pathfinding)

Pathfinding atau pencarian jalur dalam *game* komputer telah dilakukan riset selama bertahun-tahun. Pathfinding dalam game komputer komersial harus diselesaikan secara real-time, pencarian jalur tersebut dapat dilakukan dengan algoritma [1].

Algoritma dalam pencarian jalur berkaitan dengan masalah menemukan jalur terpendek dari titik awal ke titik tujuan dan dengan menghindari rintangan. Beberapa algoritma pencarian termasuk algoritma pencarian A*,

algoritma pencarian *Breadth-First* dan algoritma pencarian *Depth-First*, diciptakan untuk memecahkan masalah pencarian jalur terpendek [1]

Algoritma dalam perencanaan jalur menggunakan komputer mencakup beberapa metode klasifikasi, yang dibedakan berdasarkan pengetahuan dari area yang tersedia. Ada beberapa metode klasifikasi untuk algoritma perencanaan jalur. Misalnya, algoritma berbasis pencarian grafik adalah algoritma Dijkstra, algoritma *State Lattice*, dll. Algoritma berbasis sampling adalah RRT, dll. Selain itu, berbagai penelitian mengusulkan bahwa algoritma perencanaan jalur dapat digambarkan sebagai algoritma pencarian klasik dan algoritma pencarian heuristik. Algoritma klasik meliputi *depth-first search* (DFS), *breadth-first search* (BFS), dan algoritma Dijkstra. Algoritma tersebut merupakan algoritma perencanaan jalur berdasarkan pencarian grafik. Algoritma heuristik meliputi algoritma A*, algoritma D*, algoritma GA, algoritma ACO, algoritma *Artificial Neural Network* (ANN), dan algoritma *Simulated Annealing* (SA) [12].

Algoritma perencanaan jalur dibagi menjadi perencanaan jalur global dan perencanaan jalur lokal berdasarkan data area yang tersedia. Perencanaan jalur global mencari jalur optimal yang diberikan oleh informasi area yang lengkap dan paling baik dilakukan pada lingkungan statis dan diketahui dengan sempurna oleh robot. Oleh karena itu, perencanaan jalur global disebut juga dengan perencanaan jalur statis. Sebaliknya, perencanaan jalur lokal paling sering dilakukan di lingkungan yang tidak diketahui atau dinamis, dan perencanaan jalur lokal juga disebut perencanaan jalur dinamis [12].

D. Unity

Unity adalah game engine dan *Integrated Development Environment* (IDE) untuk membuat media interaktif contohnya video game. Sebagai CEO, David Helgason mengatakan bahwa Unity adalah perangkat yang digunakan untuk membangun game, dan teknologi untuk mengeksekusi grafis, audio, fisika, interaksi, [dan] jaringan. Unity dikenal karena kemampuan prototyping cepat dengan *target publishing* yang cukup besar [9].

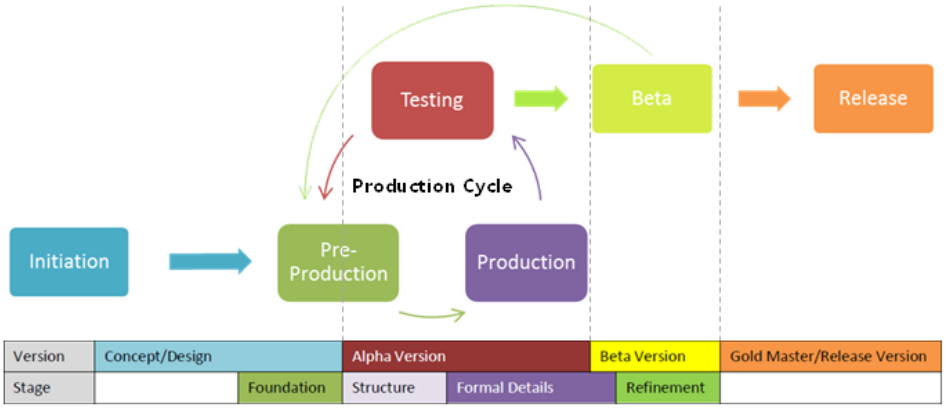
Versi pertama Unity (1.0.0) dibuat oleh: David Helgason, Joachim Ante dan Nicholas Francis di Denmark. Produk awal diluncurkan pada 6 Juni 2005. Tujuannya adalah untuk membuat game engine terjangkau dengan tools yang lengkap untuk pengembangan game. Unity memiliki alur kerja yang mudah, pengaturan aset sederhana, dan antarmuka drag-and-drop dari Final Cut buatan Apple. Saat pertama kali dirilis, Unity hanya tersedia untuk Mac OS X, tetapi pada versi sekarang ini Unity dapat digunakan pada Windows dan Mac OS X [9]..

III. METODE PENELITIAN

Metode yang menjadi dasar dalam proses pembuatan game supaya dalam pembuatan *game* ini dapat dilakukan secara terstruktur. Metode yang akan dijelaskan sebagai berikut akan diimplementasi dalam pembuatan *game*.

A. Game Development Life Cycle

Metode penelitian pada pembuatan “Implementasi Metode Pathfinding dengan Algoritma A* pada Game Rogue-like menggunakan Unity” dilakukan menggunakan metode penelitian *Game Development Life Cycle* seperti pada Gambar 3. 1 [11], Dibandingkan metode pengembangan seperti *Software Development Life Cycle* (SDLC), dengan metode ini *game* dibuat secara terstruktur dan dapat melakukan pengulangan dalam beberapa proses supaya *game* dapat dibuat lebih berkualitas.



Gambar 1 *Game Development Life Cycle*

Langkah awal dalam pembuatan game menggunakan *Game Development Life Cycle* adalah Inisiasi (*Initiation*), yaitu proses perancangan konsep awal dari *game* yang dibuat, setelah proses tersebut dilakukan, langkah selanjutnya adalah proses Pra-produksi (*Pre-production*) yang merupakan proses untuk merancang fitur-fitur dasar dari *game*, selanjutnya adalah proses Produksi (*Production*) yaitu proses untuk meningkatkan performa dari fitur dasar yang dibuat dan membuat fitur-fitur tambahan dari *game*, tahap selanjutnya adalah tahap *Testing*, yaitu tahap untuk melihat apakah terdapat masalah pada *game*, jika terdapat masalah maka proses tersebut kembali ke tahap Pra-produksi (*Pre-production*), setelah *game* sudah tidak terdapat masalah, maka langkah selanjutnya adalah proses Beta, yaitu proses pengecekan oleh pihak ketiga, setelah tidak terdapat masalah apa pun dari *game*, langkah terakhir adalah Rilis(*Release*) yaitu langkah di mana *game* sudah dapat digunakan.

1. Inisiasi (*Initiation*)

Proses inisiasi merupakan pembuatan konsep awal dari *game*, *gameplay* ditentukan dan serta menentukan target pengguna dari *game*. Hasil dari tahap ini adalah sebuah konsep serta deskripsi *game* yang sederhana [10]. Pada tahap ini ditentukan konsep awal dari *game* yang menentukan bagaimana *game* dimainkan, selain itu dilakukan juga riset untuk metode dan algoritma yang digunakan, proses ini juga memilih *game engine* yang digunakan yaitu Unity, *game engine* Unity dipilih karena Unity merupakan aplikasi gratis dan tidak membutuhkan spesifikasi yang cukup tinggi pada komputer yang digunakan untuk membuat *game*.

2. Pra-produksi (*Pre-production*)

Tahap ini merupakan pembuatan dan revisi dari *game* serta pembuatan prototipe *game*. *Game* dibuat berdasarkan genre permainan, *gameplay*, mekanik *game*, karakter, tingkat kesulitan, aspek teknis, dan dokumentasi dari *game* tersebut. Tahap ini berakhir saat revisi atau perubahan desain *game* telah disetujui [10]. Prototipe dari *game* dibuat dan masih berupa aset sederhana, tahap ini dibuat mekanik dan *gameplay*. Pada tahap ini, dibuat peta *grid* sebagai media yang digunakan untuk pathfinding pada *game*, peta *grid* adalah susunan yang terdiri dari kotak atau *node* yang membentuk suatu peta. Langkah pertama dalam pembuatan *grid* adalah dengan membuat *node* atau kotak yang disusun pada *grid* menggunakan *script*.

3. Produksi (*Production*)

Proses ini merupakan pembuatan aset dan *source code* dari *game*. Tahap ini terkait dengan penciptaan dan penyempurnaan detail, penambahan fitur baru, meningkatkan performa secara keseluruhan, dan memperbaiki bug yang masih terdapat di dalam *game*. Kegiatan selama penyempurnaan dilakukan untuk membuat permainan lebih interaktif, menantang, dan lebih mudah dipahami [10]. Tahap ini, *game* kemudian diberikan aset, serta peningkatan performa dan memperbaiki *bug* yang masih terdapat pada *game*. Pada proses ini, *script* dari

algoritma A* dihubungkan pada prefab yang sudah disediakan dengan cara membuat *script* pada prefab untuk menghubungkan variabel pada script algoritma A* dan script di dalam prefab

4. Pengujian (*Testing*)

Tahap ini merupakan pengujian internal yang dilakukan untuk menguji keseluruhan *game*. Pada proses ini, jika ditemukan *bug*, atau kegagalan selama pengujian, penyebab dan skenario tersebut dianalisis dan didokumentasi.. Pengujian terkait dilakukan juga untuk meningkatkan kualitas aksesibilitas yang dapat diuji melalui pengamatan perilaku penguji. Jika *tester* merasa sulit untuk bermain dan memahami permainan, artinya *game* tersebut tidak cukup dapat diakses. Hasil dari tahap pengujian menentukan apakah *game* dapat dilanjutkan ke tahap berikutnya atau mengulangi siklus produksi [10]

5. Beta

Tahap ini dilakukan pengujian oleh pihak ketiga atau. Pengujian beta masih menggunakan metode pengujian yang sama dengan metode pengujian sebelumnya. Dalam tahap ini, penguji diminta untuk menemukan bug supaya game dapat lebih disempurnakan, penguji diberi kebebasan untuk menggunakan game. Keluaran dari pengujian beta adalah laporan bug dan masukan dari penguji. Tahap beta akan dilanjutkan ke tahap selanjutnya jika jangka waktu pengujian beta berakhir atau jumlah penguji beta yang ditentukan telah memenuhi kriteria [10]

6. Rilis (*Release*)

Pada tahap ini, *game* telah mencapai tahap akhir dan siap untuk dirilis. Rilis melibatkan dokumentasi proyek, perencanaan untuk pemeliharaan game dan ekspansi dari *game* [10]. Setelah *game* selesai dibuat, langkah terakhir adalah pemeliharaan dari game dan menambahkan fitur tambahan dari *game* yang dibuat.

IV. HASIL PENELITIAN

A. Game Development Cycle

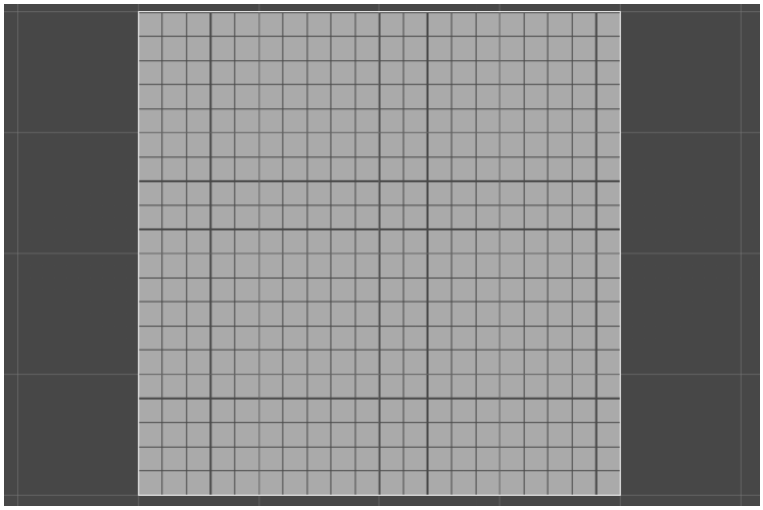
Pada implementasi algoritma A* untuk game menggunakan Unity, dapat digunakan 4 tahap dari *game development life cycle* yaitu, inisiasi yang merupakan pembuatan konsep awal dari game, pra-produksi yaitu pembuatan prototipe atau purwarupa untuk game, tahap produksi yaitu pembuatan game secara keseluruhan, dan tahap pengujian untuk memantau apakah terdapat masalah atau fitur yang tidak diinginkan pada game.

1. Inisiasi (*Initiation*)

Pada tahap ini, dibuat konsep bagaimana sistem yang dibutuhkan supaya *game* dapat bekerja, *game* yang dibuat merupakan *game single-player* yaitu hanya dapat dimainkan oleh 1 pengguna dengan perspektif 2 dimensi, *game* yang dibuat memiliki 2 objek utama yaitu objek yang dimainkan oleh pengguna, serta objek yang menjadi musuh bagi pengguna, objek musuh tersebut berfungsi sebagai halangan bagi pengguna untuk dapat masuk menuju level selanjutnya serta jika objek musuh tersebut berhasil dikalahkan oleh pengguna skor dari *game* akan meningkat, game yang dibuat merupakan *game* dengan *genre rouge-like* yang memiliki tujuan untuk mengumpulkan skor sebanyak mungkin dan mencapai tingkat level setinggi mungkin, *game* dengan jenis ini memiliki salah satu ciri yaitu peta yang digenerasi secara acak pada setiap level, sehingga dibutuhkan sistem pencarian jalur bagi objek musuh supaya dapat bergerak menuju objek pemain dengan melewati halangan pada *game* oleh peta yang dibuat secara acak dan berbeda setiap level. Pada tahap ini dilakukan riset untuk sistem pergerakan musuh, yaitu bagaimana supaya suatu musuh dapat bergerak sendiri mendekati pemain tanpa mengalami perhentian dan dapat melewati setiap halangan yang diberikan, metode yang dipilih untuk pencarian jalur pada objek musuh adalah menggunakan algoritma A*, objek musuh juga diberikan sistem serangan untuk mengurangi nilai health pada pemain untuk meningkatkan kesulitan pada saat bermain.

2. Pra-produksi (*Pre-production*)

Tahap ini merupakan pembuatan purwarupa dari *game*, metode pergerakan pada musuh sudah ditentukan pada tahap ini, metode yang digunakan adalah metode A* Pathfinding, metode ini dipilih karena *game* yang dibuat memiliki peta yang terus berubah pada tiap level. Proses pertama adalah pembuatan peta *grid* yang ditunjukkan pada Gambar 2, peta *grid* tersebut menjadi media dari algoritma A*, grid tersebut terdiri dari susunan *node* yang dibentuk menjadi *array*. *Node* tersebut juga menjadi media untuk memasukkan nilai yang digunakan pada algoritma A*.



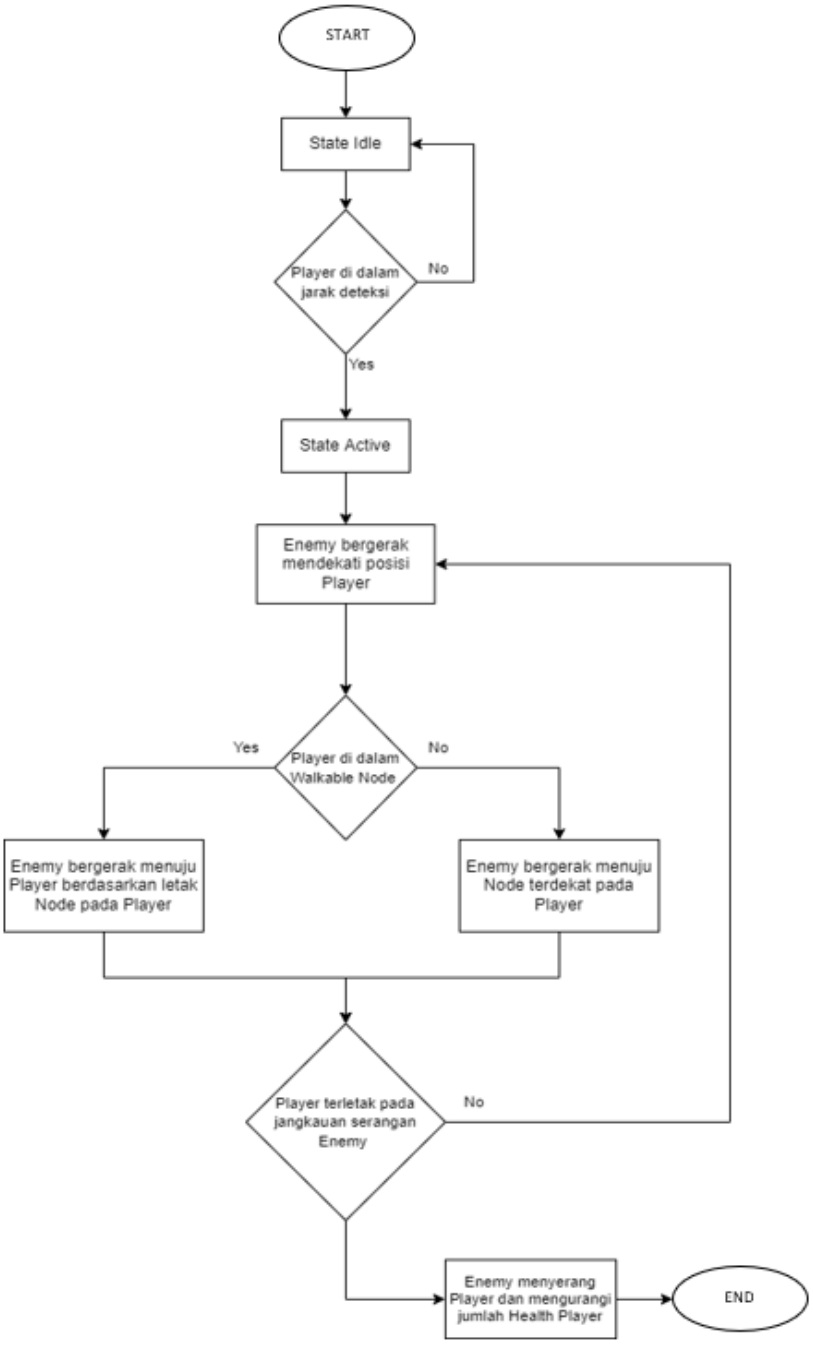
Gambar 2 Peta *grid* yang sudah dibuat pada Unity

Setelah itu dibuat unit atau musuh pada *game*, pada awalnya, musuh pada game ini merupakan prefab yang hanya berupa sprite, Rigidbody 2D dan Box Collider 2D dan belum memiliki *script*. Selanjutnya adalah pembuatan *grid* dan susunan *node* yang digunakan sebagai perhitungan algoritma A*. Setelah grid dan susunan *node* dibuat, langkah selanjutnya adalah membuat *script* yang mengatur implementasi dari A* Pathfinding. Selanjutnya adalah membuat sistem kontrol untuk pemain berupa prefab bernama Player, sistem kontrol tersebut berupa pergerakan, jenis serangan, dan sistem health pada pemain, jenis serangan yang diberikan pada Player adalah serangan jarak dekat dan jarak jauh, kedua serangan tersebut memiliki waktu penggunaan yang berbeda, waktu penggunaan serangan jarak jauh lebih lama dibandingkan waktu penggunaan serangan jarak dekat

3. Produksi (*Production*)

Tahap ini, merupakan tahap penyempurnaan dari purwarupa *game* yang dibuat, terdapat 2 hal utama yang dirancang pada tahap ini, pembuatan objek serta implementasi algoritma A* pada objek tersebut, objek akan dibuat supaya objek tersebut dapat bergerak ke titik tujuan dengan melewati halangan yang diberikan di dalam game.

Selain implementasi algoritma pada objek, serangan pada musuh juga ditambahkan pada tahap ini, beberapa fitur tambahan juga dilengkapi pada tahap ini, contohnya adalah suara dan musik untuk meningkatkan pengalaman bermain *game*, ditambahkan juga fitur skor dan level, fitur *level* pada *game* mempengaruhi serangan musuh dan *health* pada musuh dengan menambahkan nilai dari kedua variabel tersebut, semakin tinggi level maka semakin tinggi juga tingkat kesulitan dari *game*. Sistem skor dari *game* merupakan peningkatan nilai variabel skor yang meningkat jika musuh dikalahkan oleh pemain. *Game* yang dibuat memiliki objek bernama Enemy yang menjadi unit dengan implementasi algoritma A* yang sudah dibuat, Gambar 3 merupakan *flowchart* pada objek yang dibuat dengan implementasi algoritma A*.



Gambar 3 *Flowchart* pada Prefab Enemy

Enemy pada game memiliki 2 *state*, *idle* merupakan *state* dimana gerakan pada Enemy tidak diaktifkan, *state* ini merupakan *state* awal pada Enemy saat *game* pertama kali di jalankan. Pada *Idle state*, *script* untuk mengatur pergerakan pada Enemy berada dalam kondisi tidak aktif, untuk mengaktifkan *script* ini dapat dengan menggunakan fungsi Detect, fungsi bertujuan untuk mendeteksi apakah terdapat Player pada jarak yang sudah ditentukan di sekitar Enemy, jika Player memasuki jarak deteksi dari Enemy, maka Enemy memasuki *state* selanjutnya yaitu *Active*. Pada *State Active*, Enemy mulai bergerak mendekati Player sesuai dengan algoritma yang sudah di implementasi. Enemy pada *game* ini dibuat menggunakan Prefab.

Jenis Enemy yang pertama adalah “Melee Enemy” yang merupakan objek pada Gambar 4, objek tersebut menyerang pemain saat berada dalam jarak dekat, musuh ini bergerak hingga mencapai jarak berhenti yang sudah ditentukan, jika pemain bergerak atau berpindah *node* maka musuh ini bergerak kembali mendekati pemain. Di dalam *game*, ukuran dari objek ini sama dengan ukuran objek dari pemain.

Gambar 4 Melee Enemy

Jenis Enemy yang kedua adalah “Range Enemy” yang ditampilkan pada Gambar 5, jenis Enemy ini menyerang pemain dengan jarak yang lebih jauh dari “Melee Enemy”, gerakan dari jenis musuh ini berhenti jika sudah mencapai jarak yang sudah ditentukan, objek ini memiliki ukuran yang sama dengan “Melee Enemy”.

Gambar 5 Range Enemy

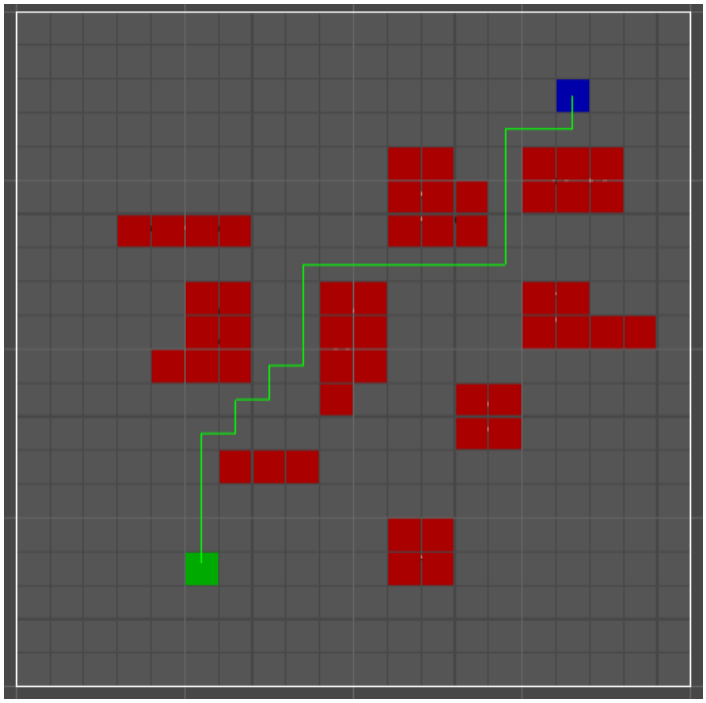
Jenis Enemy berikutnya adalah Boss Enemy, objek Boss Enemy memiliki ukuran yang lebih besar dibandingkan jenis Enemy sebelumnya, bukan hanya lebih besar tetapi jenis Enemy pada Gambar 6 ini juga memiliki nilai health yang lebih banyak juga. “Boss Enemy” harus dilawan oleh pemain jika pemain ingin melanjutkan game ke level selanjutnya, serangan pada jenis musuh ini mengurangi health pemain dalam jumlah yang cukup banyak, pada setiap level musuh ini hanya berjumlah 1. Gerakan pada jenis musuh ini sama dengan gerakan pada jenis musuh “Melee Enemy” tetapi kecepatan dari musuh ini lebih lambat dibandingkan “Melee Enemy”.

Gambar 6 Boss Enemy

Sistem pergerakan dan pencarian jalur dari objek-objek tersebut diatur menggunakan algoritma A* yang dibuat dengan *grid* atau peta yang terdiri dari blok-blok kotak, blok tersebut merupakan *node* yang digunakan sebagai tempat perhitungan nilai dari titik awal dan titik akhir, selanjutnya grid tersebut dapat digunakan sebagai jalan untuk Enemy, A* Pathfinding menghitung jalur tercepat yang dilewati berdasarkan 2 titik, titik pertama adalah titik awal yaitu pada Enemy, dan titik akhir adalah Player.

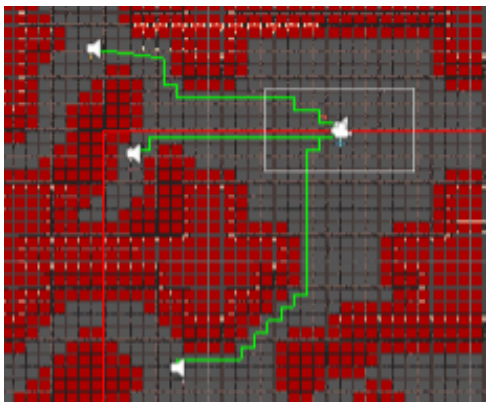
$$f(v) = h(v) + g(v) \quad (2)$$

Terdapat 2 jenis *node* yang digunakan, *node* pertama yaitu *unwalkable node* yaitu *node* yang menandai area yang tidak dapat dilewati Player maupun Enemy, *unwalkable node* dibuat menggunakan *Layer* bernama "Wall", *node* kedua adalah *walkable nod*e yang menandai area yang dapat dilewati oleh Player maupun Enemy, *walkable node* dibuat menggunakan *script* dan *node* ini akan digunakan sebagai perhitungan algoritma A*, di dalam setiap node terdapat 2 variabel yaitu G-cost dan H-cost, G-cost merupakan nilai letak *node* paling jauh dari titik awal, sedangkan H-cost merupakan nilai letak node paling jauh dari titik akhir, 2 variabel tersebut dijumlahkan menjadi F-cost, algoritma A* mengambil nilai F-cost yang paling kecil dimulai dari *node* di sekitar titik awal, *node* dengan nilai F-cost yang paling kecil dipilih untuk dilanjutkan perhitungan nilai F-cost kembali pada *node* sekitar titik yang dipilih, kemudian dari *node* dengan nilai F-cost terkecil diambil kembali, jika terdapat nilai F-cost yang sama di sekitar suatu *node*, maka *node* yang diambil nilainya adalah node yang memiliki nilai H-cost yang paling kecil. Operasi tersebut terus dilakukan hingga *node* mendapat titik akhir, *node* yang sudah dipilih menjadi titik yang digunakan sebagai posisi tujuan dari pergerakan dari Enemy. Jika terdapat *node* di sekitar titik merupakan *unwalkable node* maka titik tersebut tidak dihitung. Setelah jalur tersebut dibuat berdasarkan algoritma A*, objek Enemy dibuat untuk berjalan mengikuti jalur yang sudah dibuat hingga menuju titik akhir. Pembuatan jalur dibuat *real-time*, karena objek yang menjadi titik akhir jalur yaitu objek Player merupakan objek dinamis di mana objek tersebut terus bergerak, selain objek Player keadaan dari peta yang diberikan di dalam game juga terus berubah setiap peningkatan *level*.



Gambar 7 Implementasi algoritma A*

Pada Gambar 7, *node* berwarna hijau adalah titik awal, sementara *node* berwarna biru merupakan titik akhir dari jalur. Selain itu, rancangan *game* dilengkapi beberapa fitur utama dan tambahan dari *game*, pada prefab musuh ditambahkan sistem serangan dari jarak dekat hingga jarak jauh, sistem serangan dari jenis musuh "Boss" juga ditambahkan. Gambar 8 merupakan hasil saat algoritma A* diimplementasi ke dalam *game*.

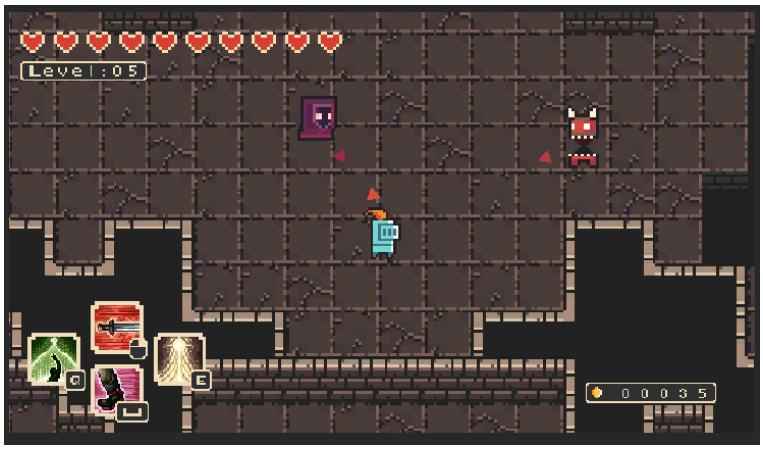


Gambar 8 *Pathfinding* pada *game*

Pada Gambar 8, kotak dengan warna abu-abu merupakan *node* "walkable" atau area yang dapat dilalui oleh objek, kotak dengan warna merah merupakan node "unWalkable" atau area yang tidak dapat dilewati oleh objek, garis hijau merupakan jalur yang dibuat berdasarkan node yang dipilih. Kotak tersebut ditampilkan menggunakan fitur Gizmo dari Unity, fitur tersebut berfungsi untuk menampilkan indikator atau visualisasi pada saat melakukan debug pada Unity

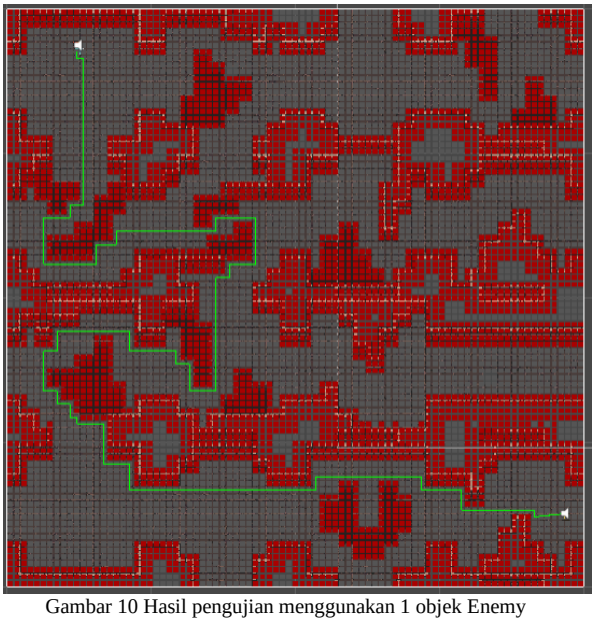
4. Pengujian (*Testing*)

Pada tahap ini, game dipantau kembali supaya jika terdapat bug atau hal yang tidak diharapkan pada *game* dapat diperbaiki, penyesuaian nilai variabel dalam *game* disesuaikan kembali, contohnya nilai pada variabel untuk mengatur kecepatan bergerak dari pemain, dan nilai pada variabel untuk mengatur serangan musuh. Prefab musuh yang sudah dipasang algoritma A* dites apakah jalur yang dibuat oleh algoritma A* sudah sesuai, apakah prefab mengikuti jalur dengan benar dan dapat sampai ke titik akhir tanpa melakukan perhentian, variabel pada setiap prefab disesuaikan berdasarkan jenis musuh. Gambar 9 merupakan tahap Playtesting dan juga tampilan saat *game* dimainkan.



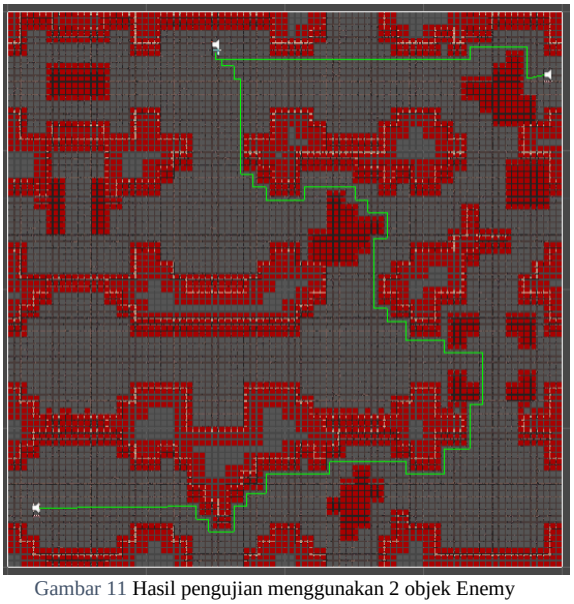
Gambar 9 Proses pengujian pada *game*

Playtesting merupakan pengujian *game* secara keseluruhan dengan memainkan *game* berkali-kali untuk melihat apakah *game* berfungsi sesuai yang diinginkan. Pada tahap ini dilakukan perubahan *gameplay* dengan mengubah berbagai nilai variabel dalam *game*. Algoritma A* digunakan pada peta yang digenerasi secara acak, deteksi halangan dan pembuatan jalur pada objek akan diuji dengan menempatkan objek Enemy pada suatu titik di peta dan pengujian deteksi halangan pada peta akan dilakukan dengan memainkan *game*, pengujian dilakukan menggunakan objek Player, objek yang menjadi halangan, dan objek Enemy, Gambar 10 merupakan contoh dari hasil pengujian menggunakan 1 objek.



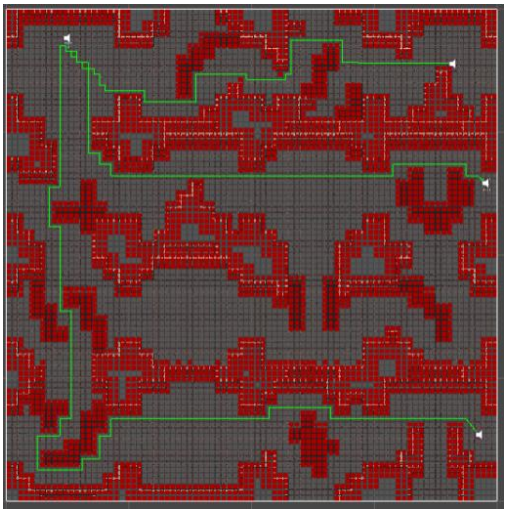
Gambar 10 Hasil pengujian menggunakan 1 objek Enemy

Gambar 10 merupakan hasil pengujian menggunakan 1 objek yang ditempatkan pada suatu titik pada peta, node merah merupakan deteksi halangan dari algoritma, sementara garis berwarna hijau merupakan jalur yang dibuat oleh algoritma yang diikuti oleh objek, Gambar 10 menunjukkan algoritma A* dapat berhasil bekerja. Gambar 11 merupakan hasil pengujian kedua menggunakan 2 objek, dan peta yang digenerasi secara acak.



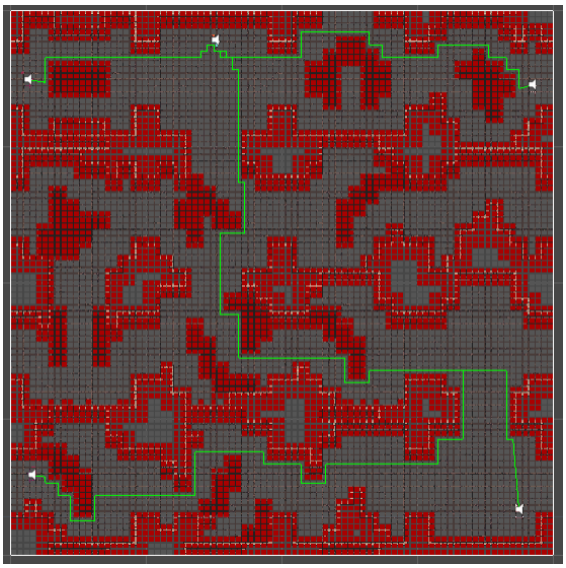
Gambar 11 Hasil pengujian menggunakan 2 objek Enemy

Pada Gambar 11, hasil pengujian dilakukan menggunakan 2 objek Enemy yang diletakkan pada tempat tertentu. Pada pengujian ini, algoritma berhasil mendeteksi objek yang menjadi halangan dan algoritma berhasil membuat jalur dari objek Enemy menuju objek Player, objek Enemy kemudian bergerak mengikuti jalur berwarna hijau yang dibuat oleh algoritma menuju objek Player yang menjadi titik tujuan, node berwarna merah merupakan halangan pada peta yang dideteksi oleh algoritma. Gambar 12 merupakan hasil pengujian menggunakan 3 objek dan peta yang digenerasi kembali secara acak.



Gambar 12 Hasil pengujian menggunakan 3 objek Enemy

Pengujian menggunakan 3 objek Enemy dilakukan dengan meletakkan 3 objek Enemy pada tempat tertentu pada peta yang digenerasi kembali secara acak, algoritma dapat mendeteksi halangan dan membuat jalur dari objek Enemy ke titik tujuan yaitu objek Player, Gambar 13 adalah hasil pengujian menggunakan 4 objek Enemy dan peta baru yang digenerasi secara acak.



Gambar 13 Hasil pengujian menggunakan 3 objek Enemy

Pengujian pada Gambar 13 dilakukan dengan menggunakan 4 objek Enemy. Pada pengujian tersebut, algoritma dapat mendeteksi halangan yang diberikan dan dapat membuat jalur menuju titik akhir yaitu objek Player. Objek Enemy kemudian bergerak mengikuti jalur yang dibuat algoritma menuju objek Player.

Dari pengujian yang dilakukan, hasil yang didapat adalah objek Enemy mampu bergerak menuju titik akhir yaitu objek Player dengan melewati rintangan yang diberikan di dalam game. Saat peta di dalam *game* mengalami perubahan, algoritma mampu membedakan objek yang menjadi halangan pada *game*, dan dapat mendeteksi objek tersebut secara real-time. Pada tahap ini, dilakukan juga penyesuaian nilai pada variabel yang diberikan, contohnya adalah mengatur jarak deteksi Enemy supaya tidak terlalu dekat atau terlalu jauh, dan menyesuaikan kecepatan pergerakan dari objek Enemy dan objek Player supaya pergerakan dari objek-objek

tersebut tidak terlalu pelan atau terlalu cepat, pada tahap ini juga ditemukan masalah yaitu serangan pada objek Range Enemy terkadang mencapai dua kali lipat dari nilai yang diberikan jika objek Player terkena serangan tersebut, masalah tersebut ditimbulkan karena objek Player pada awalnya memiliki 2 Collider, pada objek Player itu sendiri dan objek untuk mengatur arah serangan yang disebut Aim, Collider tersebut awalnya dibuat supaya objek Aim tidak dapat menembus objek Wall saat Player bergerak ke arah objek Wall. Collider tersebut kemudian dihilangkan dan dibuat skrip supaya Aim dapat mengikuti objek Player tanpa menembus objek Wall.

V. KESIMPULAN

Berdasarkan hasil dari "Implementasi Metode Pathfinding dengan Algoritma A* pada Game Rogue-like menggunakan Unity" yang sudah dilakukan, dapat ditarik kesimpulan sebagai berikut:

1. Unit pada *game rogue-like* dengan algoritma A* dapat bergerak melewati halangan yang diberikan menuju titik akhir.
2. Pada *game rogue-like* yang dibuat, unit dengan algoritma A* hanya dapat bergerak menuju 1 titik tujuan.
3. Cara kerja algoritma A* pada *game* adalah dengan menggunakan petak yang diberi nilai berdasarkan posisi dari titik awal dan titik akhir. Petak dengan nilai paling kecil akan digunakan sebagai jalur yang dilewati.
4. Metode A* Pathfinding dapat digunakan dalam *game* dengan kondisi dinamis atau berubah-ubah, karena data yang diambil dapat selalu melakukan update pada setiap *frame* saat *game* dimainkan.
5. Meskipun objek hasil implementasi algoritma A* pada *game* ini dapat bergerak menuju titik tujuan dan menghindari halangan yang diberikan, objek masih dapat menyentuh halangan tersebut.

References

[1] H. Nawaf, S. Sinan, dan A. Mustafa, "Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm", Baghdad : Journal of Computer and Communications, 2016.
[2] F. Daniel, dan G. Alifio, dll, "A Systematic Literature Review of A* Pathfinding", Jakarta : Computer Science Department, School of Computer Science, Bina Nusantara University, 2020.
[3] C. Ade, dan A. Mohammad, dkk, "Application of A-Star Algorithm on Pathfinding Game", Sumatra Utara : Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, 2020.
[4] D. Silvester, dan B. Ketut, dkk, "Comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS on Maze Runner Game", Medan : Journal of Physics: Conference Series, 2020.
[5] M. Ghani dan F. Juniardi, dkk, "Implementasi Metode Path Finding dengan Penerapan Algoritma A-Star untuk Mencari Jalur Terpendek pada Game "Jumrah Launch Story"", Walisongo Journal of Information Technology, 2021.
[6] H. Zhonghua, dan S. Pengfei , dkk, "Improved A-Star Algorithm for Long-Distance Off-Road Path Planning Using Terrain Data Map", ISPRS International Journal of Geo-Information, 2021.
[7] H. Zhibo, dan L. Chenguang , dkk, "Dynamic anti-collision A-star algorithm for multi-ship encounter situations", Wuhan : Intelligent Transport System Research Center, Wuhan University of Technology, 2021.
[8] D. František, dkk , "Path planning with modified A star algorithm for a mobile robot", Elsevier Ltd, 2014.
[9] J. K. Haas, "A History of the Unity Game Engine", Worcester Polytechnic Institute, 2014
[10] R. A. Krisdiawan, "IMPLEMENTASI PENGEMBANGAN SISTEM GDLS DAN ALGORITMA LINEAR CONGRUENTIAL GENERATOR PADA GAME PUZZLE", Kuningan : Fakultas Ilmu Komputer Universitas Kuningan, 2018
[11] R. Rido, dan W, Yani. "Game Development Life Cycle Guidelines", Bandung : School of Electrical Engineering and Informatics, Institut Teknologi Bandung, 2013.
[12] W. Huanwei, dan L. Shangjie, dkk, "The EBS-A* algorithm: An improved A* algorithm for path planning", Zhengzhou : State Key Laboratory of Mathematical Engineering and Advanced Computing, 2022.