Penerapan *Pathfinding* dengan A* untuk Permainan *Tower Defense*

Steven Nataniel / 13519002
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519002@std.stei.itb.ac.id

***Abstract*—*Tower Defense* adalah permainan strategi yang bertujuan untuk mempertahankan wilayah atau sesuatu yang dimiliki pemain dari serangan musuh yang umumnya dicapai dengan meletakkan menara-menara yang akan menghambat, manahan, menyerang, atau menghancurkan musuh. Terdapat beberapa permainan *tower defense* yang membebaskan pemainnya menyusun menara secara bebas sehingga musuh atau monster akan mencari jalan menuju tujuan akhir.**

Keywords—a-star; tower defense; pathfinding;

I. Pendahuluan

Permainan *Tower Defense* selalu dihubungkan dengan strategi pemain dalam mempertahankan wilayah atau sesuatu yang dimiliki pemainnya dari serangan musuh. Umumnya, dilakukan dengan membangun menara-menara yang dapat menghambat, menahan, menyerang, serta menghancurkan musuh. Umumnya, *tower defense* selalu berbasis waktu nyata, namun ada juga yang berbasis giliran.

Salah satu permainan *tower defense* ialah GemTD atau Gem Tower Defense yang dirilis pada 18 Februari 2007 oleh Bryan K. Permainan ini merupakan *custom games* yang berada pada permainan DOTA 2 walaupun awalnya diciptakan pada World of Warcraft 3. Saat ini, permainan ini sudah dikembangkan sehingga banyak *developer-developer* lain memakai konsep dan aturan game ini dan dibuat ke platform lain seperti *browser game*, *mobile game*, dan *game clients* lainnya. Permainan ini berdasarkan atas strategi, membuat maze, *decision-making*, dan *chance*. Permainan ini disukai oleh 760 ribu orang dan telah diunduh sebanyak 1,85 akun pada platform DOTA 2.

Gambar 1. Gem Tower Defense

Pada permainan ini, pemain dibebaskan untuk meletakkan menara secara bebas selama ada minimum satu jalan dari pintu masuk sampai pintu keluar untuk musuh. Sehingga, *developer game* merlukan *pathfinding* yang optimal dalam menentukan jalan yang terbaik yang harus dilalui oleh musuh untuk membuat game menjadi menantang bagi pemain.

II. Landasan Teori

A. *Uninformed Search*

Tentu dalam melakukan *route planning* akan dilakukan pencarian menuju jalan akhir. Dalam melakukan pencarian tersebut, terdapat beberapa algoritma umum yang dipakai untuk melakukan pencarian dengan tujuan umum. Algoritma ini beroperasi secara *brute force* karena sebenarnya mencoba ke semua simpul dalam sebuah pohon atau graf. Algoritma dengan karakteristik seperti ini disebut sebagai *uninformed search* atau pencarian buta.

Pertama, Breadth First Search (BFS), algoritma yang mengunjungi sebuah simpul dan mengambil semua tetangganya kedalam sebuah antrian untuk dikunjungi berikutnya. BFS akan memberikan solusi apabila terdapat sebuah solusi dan solusi yang dioberikan akan selalu solusi minimal yang memerlukan langkah paling sedikit. Namun, algoritma ini memerlukan banyak memori oleh karena penyimpanan setiap simpul yang sudah didatangi dan yang akan didatangi. Tidak hanya itu, BFS memerlukan banyak waktu apabila graf sangat besar serta luas ditambah solusi jauh dari simpul mulai.

Kedua, Depth First Search (DFS), algoritma yang mirip dengan BFS hanya saja menggunakan struktur data *stack*. DFS akan membutuhkan lebih sedikit memori daripada BFS. Namun, terdapat kemungkinan simpul yang dikunjungi diproses ulang sehingga tidak ada jaminan menemukan solusi dan bisa terjadi *infinite loop* apabila graf memiliki sirkuit. DFS akan selalu menghasilkan solusi yang lebih panjang dibanding BFS namun performa DFS akan lebih bagus apabila simpul solusi jauh dari simpul awal.

Ketiga, Depth Limited Search (DLS), algoritma DFS yang di*enchance* dengan memberikan limit agar tidak terjadi *infinite loop*. Meskipun algoritma ini sangat efisien dalam menggunakan memori. Namun, algoritma ini tidak menjamin

ada solusi meskipun terdapat sebuah solusi. Algoritma ini juga tidak optimal apabila terdapat banyak solusi.

Selanjutnya, Uniform-Cost Search (UCS), algoritma yang bisa melakukan search dengan faktor bobot disetiap sisinya. Algoritma ini memakai struktur data priority queue. Solusi akan optimal karena selalu diambil biaya terrendah. Namun, algoritma ini tidak peduli terhadap berapa langkah yang diambil dan hanya memikirkan biaya sisi. Dan berkemungkinan untuk terjebak dalam *infinite loop*.

Melakukan pencarian rute sudah pasti akan berhubungan dengan graf yang memiliki simpul bersisi sirkuit. Sehinga, tidak sembarang algoritma *search* dapat diterapkan untuk melakukan pencarian rute terbaik. Sehingga, diperlukan algoritma *search* yang memakai fungsi heuristik.

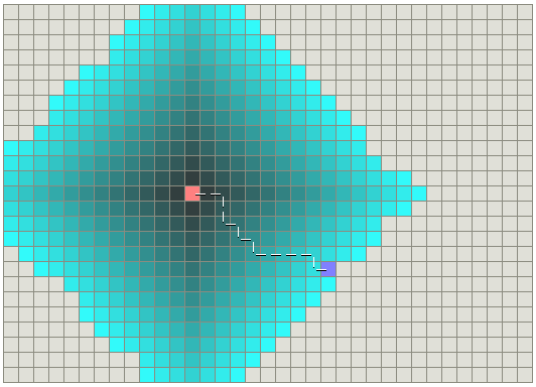
B. Fungsi Heuristik

Fungsi heuristik merupakan pengetahuan tambahan untuk program mencarikan jalur yang lebih menjanjikan. Kata ini berasal dari bahasa yunani yang berasal dari kata *eureka*, yang berarti untuk menemukan. Heuristik memperkirakan seberapa baik pergerakan dari simpul ke simpul. Heuristik biasanya terdiri dari suatu fungsi yang disusun dengan beban yang ditentukan dari seberapa besar peluang suatu simpul mengarah kepada simpul solusi. Sehingga, pada konteks pencarian rute, fungsi heuristik biasanya dikaitkan dengan jarak suatu simpul ke simpul tujuan.

*C. Algoritma A**

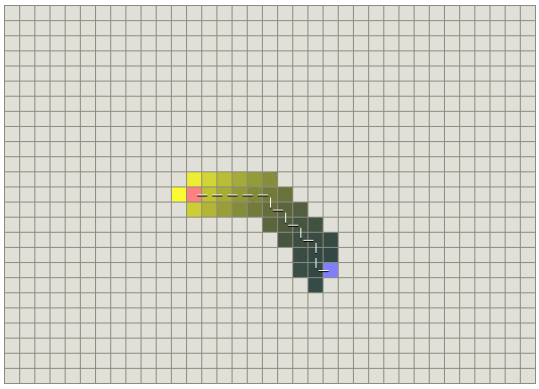
Sebelum membahas tentang algoritma A*, perlu dijelaskan dua algoritma dasar pembentuk A* yaitu Dijkstra dan Greedy Best-First-Search yaitu sebagai berikut.

Algoritma Dijkstra, algoritma yang mengunjungi simpul-simpul pada graf dengan mencari titik awal lalu memeriksa simpul terdekat yang belum pernah dikunjungi dan menambahkan simpul-simpul yang terkedat ke dalam himpunan simpul yang akan diperiksa dan mengembang dari awal hingga titik tujuan. Algoritma ini dapat menjamin ditemukannya sebuah jalur terpendek dari titik awal ke tujuan selama tidak ada biaya negatif. Pada gambar berikut, terdapat suatu titik berwarna merah mudah yang merupakan titik awal dan titik berwarna ungu sebagai tujuannya. Dan daerah *teal* merupakan titik-titik yang telah diperiksa. Keragaman warna *teal* tersebut ditujukan untuk menandakan jauh-dekatnya, gelap menandakan dekat dan semakin terang menandakan jauh.



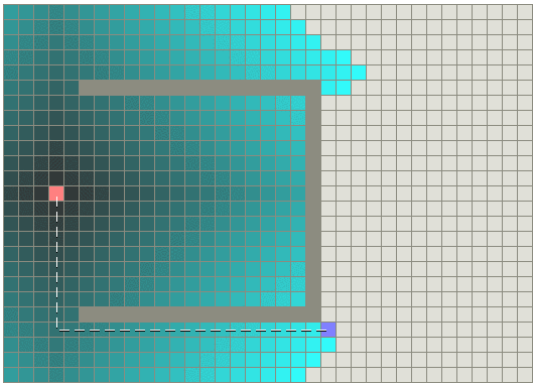
Gambar 2. Algoritma Dijkstra

Algoritma Greedy Best-First-Search, algoritma ini mirip dengan algoritma dijkstra diatas namun diberi fungsi *heuristik* berupa seberapa jauh titik yang akan dipilih dengan tujuan dan akan diambil yang paling dekat. Namun, algoritma ini tidak bisa menjamin jalur terpendek meski algoritma ini lebih cepat dibanding dijkstra karena *heuristik*nya. Pada gambar berikut, terdapat kotak kuning yang menandakan *herustik* dengan nilai tinggi dan hitam dengan nilai *heuristik* kecil. Dengan kotak berwarna yang sedikit, dapat disimpulkan algoritma ini lebih cepat dibandingkan algoritma dijkstra karena pemrosesannya tidak perlu mendatangi simpul yang sangat banyak.

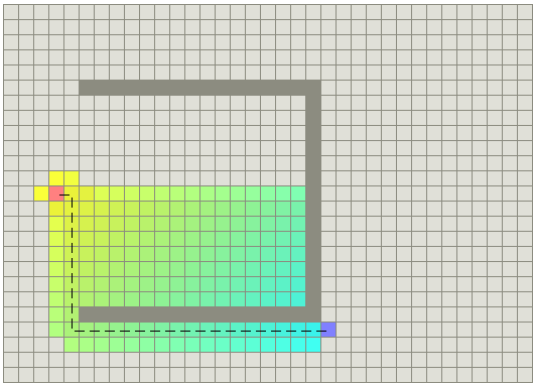


Gambar 3. Algoritma Greedy Best-First-Search

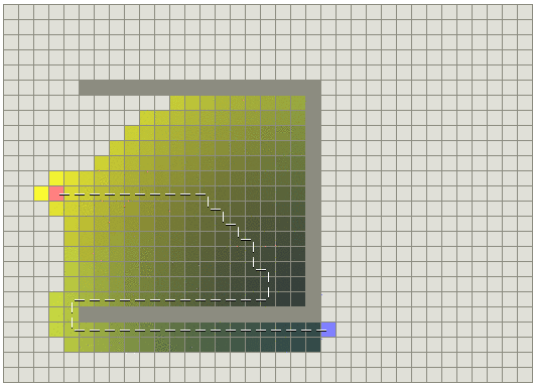
Kedua algoritma tersebut terlihat baik-baik saja dan terlihat seperti layak untuk dipertimbangkan. Hal tersebut akan berubah ketika diberi hambatan. Terlihat seperti pada gambar 4 dan 5. Meskipun begitu, *statement* tentang algoritma dijkstra pasti menemukan jalur terpendek tidak terbantahkan. Sedangkan, Algoritma Greedy Best-First-Search sudah terlihat kacau karena keserakahannya menuju ke tujuan meskipun itu bukan jalan yang benar membuat jalur menjadi panjang.



Gambar 4. Algoritma Dijkstra dengan halangan



*Gambar 6. Algoritma A**



Gambar 5. Algoritma Greedy Best-First-Search dengan halangan

Dari permasalahan tersebut, muncul ide untuk menggabungkan kedua algoritma tersebut agar pemrosesan tidak terlalu banyak namun jalan dipastikan terpendek. Sehingga, Algoritma A* diciptakan dengan menggabungkan hal-hal terbaik dari dua algoritma sebelumnya.

Algoritma A* sangat populer dalam pencarian rute karena fleksibilitas yang tinggi dan dapat digunakan dalam berbagai konteks. A* menurunkan dua sifat dari Dijkstra dan Greedy Best-First-Search. Pertama, A* menggunakan heuristik. Kedua, A* menemukan jalur sebaik Algoritma Dijkstra. Pendekatan yang digunakan algoritma ini adalah menggabungkan kedua algoritma tersebut yang dapat dilambangkan sebagai rumus berikut.

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

Dengan g(n) adalah biaya jalur yang tepat dari titik awal ke titik manapun yang disebut sebagai n dan h(n) mewakili perkiraan biaya *heuristik* dari titik yang disebut sebagai n ke tujuan. Sehingga saat mengunjungi simpul-simpul, A* dapat menyeimbangkan keduanya saat bergerak dari titik awal ke tujuan. Setiap kali berpindah, A* selalu mengunjungi simpul dengan f(n) terrendah. Bisa dilihat pada algoritma A*.

D. Gem Tower Defense

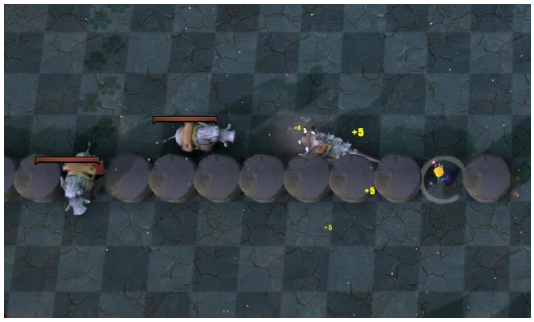
Permainan *tower defense* pada umumnya memiliki jalan-jalan yang sudah ditentukan oleh pengembangnya. Namun, Berbeda dengan *tower defense* pada GemTD, pada permainan ini, pemain dibebaskan untuk menentukan jalur yang akan dilewati oleh musuh karena pemain dapat menaruh menara dimanapun. Awalnya, akan terdapat petak bebas dan beberapa batu untuk pemain seperti gambar berikut.

Gambar 7. Peta

Permainan dimulai dengan giliran pemain, pemain harus meletakkan 5 buah menara dan memilih salah satu dari lima tersebut dan membuang sisanya menjadi batu biasa yang akan berguna sebagai menghalangi jalan musuh. Setelah memilih, giliran akan langsung berakhir dan musuh akan langsung berjalan. Terlihat pada gambar 8, pemain meletakkan 5 buah menara dan pada gambar 9, setelah dipilih, sisanya akan menjadi batu biasa. Terdapat juga papan penanda dengan nomor satu hingga lima yang menandakan bahwa musuh akan pergi ke papan nomor satu lalu papan nomor dua dan

seterusnya hingga papan nomor lima lalu pergi ke pintu keluar.

Gambar 8. Pemain menaruh 5 menara

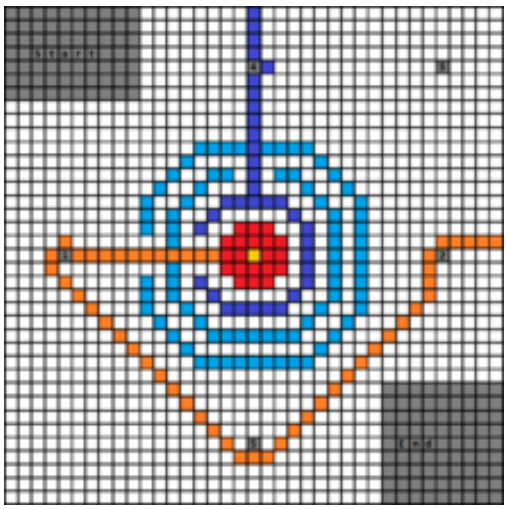


Gambar 9. Menara melawan musuh dan giliran musuh

Setelah musuh telah habis dibunuh oleh menara pemain atau berhasil menembus pertahanan pemain. Giliran akan dikembalikan ke pemain dan pemain harus membangun 5 menara lagi dan memilih salah satunya. Permainan akan terus berlangsung seperti itu hingga pemain kehabisan darah. Pemain akan dinyatakan menang apabila pemain berhasil melewati level 50 meskipun terdapat level diatas 50 yang akan dianggap sebagai bonus level.

Pathfinding digunakan oleh pengembang permainan dalam melarang pemain apabila tidak terdapat jalan menuju salah satu papan dari papan lainnya. Sehingga, pemain hanya memilih pilihan untuk membuat musuh melalui maze yang berisi menara-menara. Meskipun player sudah membuat maze, musuh dalam permainan ini bisa saja berputar melalui jalur lain apabila terdapat jalur lain yang lebih aman atau pendek untuk dilalui. Sehingga, diperlukan algoritma *pathfinding* yang optimal agar hal tersebut terjadi.

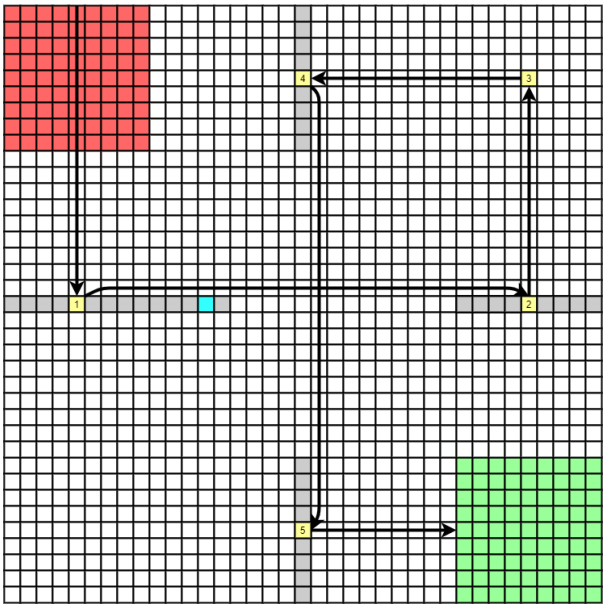
Sebenarnya, tidak hanya *pathfinding* yang digunakan dalam game ini. Teori-teori graf juga diperlukan dalam sisi pemain untuk membuat maze. Contohnya pada gambar 10, terdapat susunan maze yang paling sering dipakai oleh pemain. Namun, pada kali ini, hanya dibahas sisi *pathfinding* untuk *developer game*.



Gambar 10. Template Maze yang paling sering dipakai

III. PEMBAHASAN

Berikut ini adalah peta permainan GemTD yang memakai kondisi pada gambar 9. Musuh akan mengambil jalur seperti yang ditunjukan pada panah berikut. Kotak berwarna merah merupakan tempat musuh pertama kali muncul, dan kotak berwarna hijau merupakan tujuan akhir musuh. Kotak-kotak kuning angka merupakan papan tanda yang harus dicapai oleh musuh (papan tanda selalu sama tempatnya). Kotak biru merupakan menara yang dibuat pemain. Kotak abu-abu melambangkan batu. Ilustrasi-ilustrasi pada kali ini diasumsikan musuh bisa berjalan secara diagonal, karena pada permainan aslinya, musuh tidak bisa berjalan secara diagonal.



Gambar 11. Rute (1)

Berikut adalah langkah-langkah penyelesaian persoalan tersebut.

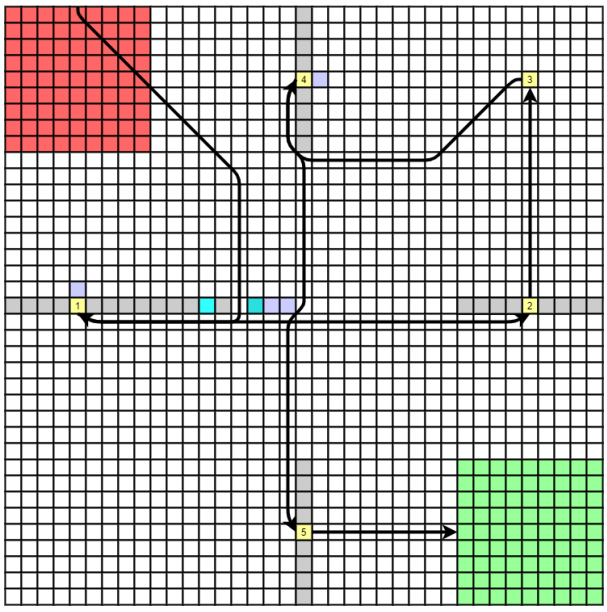
- Menuju papan tanda 1: Karena papan tanda 1 tidak terhalang oleh apapun, maka sudah pasti algoritma A* akan jalan lurus.
- Menuju papan tanda 2: Terdapat dua kandidat kotak untuk dilalui untuk menuju papan tanda 2 yang sama saja besarnya.
- Menuju papan tanda 3 dan 4: Tidak ada halangan, maka akan langsung diambil jalan lurus.
- Menuju papan tanda 5: Kasus yang sama seperti papan tanda 1 ke papan tanda 2.
- Menuju pintu keluar: Tidak ada halangan, jalan lurus.

Hal tersebut bisa tercapai oleh karena rumus algoritma A* yaitu sebagai berikut.

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

Dengan keterangan, bahwa f(n) adalah penjumlahan dari g(n) yang merupakan jarak titik awal ke titik manapun yang disebut sebagai n dan h(n) yang merupakan jarak titik n tersebut ke titik akhir. Setiap algoritma ini mengambil langkah baru, akan dicari kotak disekitar dengan f(n) paling minimum agar terjamin rute tersebut adalah rute terpendek.

Berikut ini adalah peta permainan GemTD yang melanjutkan pada gambar 10. Pada gambar berikut, terdapat kotak berwarna *teal* yang lebih tua yang menandakan menara yang baru dipilih, dan kotak ungu yang merupakan batu yang baru saja terbuat.



Gambar 12. Rute (2)

Terlihat rute berubah secara drastis oleh karena halangan baru yang dibuat oleh pemain. Berikut adalah langkah-langkah penyelesaian persoalan tersebut.

- Menuju papan tanda 1: Musuh akan langsung berjalan secara diagonal karena h(n) yang kecil.
- Menuju papan tanda 2 dan 3: Musuh akan langsung saja membuat jalan lurus karena tidak ada halangan.
- Menuju papan tanda 4: Diambil langkah diagonal karena h(n) yang kecil dan membeloki halangan.
- Menuju papan tanda 5: Diambil simpul-simpul yang g(n) dan h(n)-nya stabil, karena simpul yang berada disebelah kiri memiliki h(n) yang lebih tinggi jadi algoritma tidak akan mengambil jalur tersebut dan langsung mengambil jalur tengah dan melakukan jalan diagonal.
- Menuju pintu keluar: Tidak ada halangan, jalan lurus.

IV. KESIMPULAN

Dalam mencari rute, tidak bisa digunakan algoritma sembarang untuk melakukan *searching*. Algoritma yang sembarangan akan membuat program akan terjebak dalam *infinite loop* atau rute yang tidak optimal. Menggunakan *uninformed search* dalam melakukan pencarian rute merupakan hal yang naif.

Algoritma A* sangat bermanfaat dalam melakukan *route planning*. Algoritma ini merupakan gabungan dari Greedy Best-First-Search dan Dijkstra. *Route planning* sering muncul dalam permasalahan-permasalahan permainan-permainan zaman sekarang, contoh lainnya adalah, kecerdasan buatan untuk monster-monster atau *creep* pada DOTA 2 untuk mencari jalan menuju sesuatu. Tidak hanya itu, saat bermain game MOBA yang berplatform pada komputer, player dapat mengklik suatu tempat dan komputer akan mengkalkulasi rute terdekat yang akan dipakai untuk menempuh ke tempat tersebut. Pada makalah ini, algoritma ini sangat berguna untuk *developer game* dalam membuat rute untuk monster yang akan melewati maze buatan pemain.

V. SARAN

A. *Saran Pengembangan*

Saran untuk pengembangan makalah ini adalah dengan menambah informasi-informasi yang lebih mendetail seperti angka g(n) dan h(n) pada gambar rute. Dan visualisasi *game engine* untuk menampilkan kondisi f(n) dan tata letak map akan jauh lebih membuat penjelasan dan legenda map mudah dipahami.

B. *Saran Pemanfaatan*

Algoritma A* ini dapat dimodifikasi dengan menambahkan faktor berupa menara pemain supaya monster menjadi lebih pintar dan sulit untuk dikalahkan.

Menambahkan faktor tersebut tentu perlu percobaan dan pencarian konstanta untuk pengalian dengan faktor tersebut agar tidak merusak algoritma A* itu sendiri.

VI. Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa oleh karena berkat serta kemurahan hati-Nya makalah ini dapat diselesaikan dengan sebaik-baiknya.

Penulis juga ingin berterima kasih kepada kedua orang tua dan teman-teman saya yang telah mendukung penulisan makalah ini. Tidak lupa, penulis juga mengucapkan terima kasih kepada Bapak Ir. Rila Mandala, M.Eng., Ph. D. selaku dosen pengampu mata kuliah IF2211 Strategi Algoritma kelas K1. Oleh karena bimbingan beliau, saya dapat belajar dengan baik mengenai mata kuliah ini. Penulis juga ingin berterima kasih kepada Bapak Dr. Ir. Rinaldi Munir yang telah membuatkan website berisikan koleksi materi-materi serta video pembelajaran mengenai Strategi Algoritma.

Penulis juga ingin mengucapkan terima kasih kepada pencipta Gem Tower Defense atas pemikiran dan ide yang luar biasa dalam menciptakan permainan tersebut. Ucapan terima kasih juga diberikan kepada *developer* Gem Tower Defense yang membawakan permainan tersebut ke permainan DOTA 2 sebagai *custom games*.

References

[1] Munir, Rinaldi. Penentuan Rute Bagian 1. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf . Diakses pada 10 Mei 2021, pukul 09.56.

[2] Munir, Rinaldi. Penentuan Rute Bagian 2. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf. Diakses pada 10 Mei 2021, pukul 10.11.

[3] Stanford Theory Edu. Introduction to A*. http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html. Diakses pada 10 Mei 2021, pukul 11.21.

[4] Java T Point. Uninformed Search Algorithms. https://www.javatpoint.com/ai-uninformed-search-algorithms. Diakses pada 10 Mei 2021, pukul 18.07.

[5] Epic War. Gem TD. https://www.epicwar.com/maps/13261/. Diakses pada 10 Mei 2021, pukul 20.03.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 11 Mei 2021

Steven Nataniel / 13519002