Aplikasi Algoritma A-Star dalam Mencari Rute Terpendek pada Permainan Tower Defense

Nicholas Chen 13519029
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13519029@std.stei.itb.ac.id

***Abstrak*—Dalam beberapa permainan seperti *Tower Defense* dibutuhkan algoritma yang efektif untuk menentukan rute terpendek secara otomatis. Ada berbagai algoritma yang dapat digunakan untuk menghitung rute terpendek ini, salah satu alternatifnya dengan memanfaatkan teori graf. Algoritma Djikstra dan A-Star adalah contoh dari algoritma umum yang memanfaatkan teori graf. Pada makalah ini akan dibahas mengenai kedua algoritma tersebut lalu dibandingkan keefiktifannya saat diimplementasi kedalam game seperti *Tower Defense*.**

***Keywords*—Graf, Algoritma Djisktra, Algoritma A-Star, *Tower Defense*.**

I. PENDAHULUAN

Tower Defense adalah salah satu permainan bergenre strategi yang mulai berkembang pada tahun 1980. Pada permainan ini, pemain akan bertindak sebagai kubu bertahan yang bertugas menjaga markas dengan membangun menara yang menembak musuh agar tidak ada yang dapat menerobos kedalam markas. Sementara itu, kubu penyerang secara otomatis akan mengeluarkan pasukan yang bertujuan untuk menerobos menara pertahanan pemain dan menginvasi markasnya. Pasukan musuh akan selalu memilih rute terdekat dan terefisien untuk masuk kedalam markas pemain. Pemain dapat memperkuat pertahanan mereka dengan membangun menara untuk menghalangi musuh sehingga rute musuh lebih panjang, namun menara yang dibangun tidak boleh memblokir penuh jalan untuk ke markas.

Gambar 1. Robo Defense, diambil dari [1]

Peta dalam permainan merupakan suatu graf yang saling berhubungan. Setiap petak yang merupakan titik dari graf akan terhubung dengan petak yang berada di atas, kanan, kiri, dan bawahnya. Pasukan musuh dapat bergerak secara vertikal dan horizontal. Banyak algoritma yang dapat diimplementasikan untuk menentukan rute terpendek yang harus diambil pasukan. Algoritma yang umumnya diimplementasi dalam permainan *Tower Defense* yang beredar adalah Djikstra dan A-Star. Oleh karena itu, pada artikel akan dibahas lebih mendalam mengenai kedua algoritma tersebut beserta perbandingan kinerjanya dalam mencari rute terpendek untuk pasukan musuh.

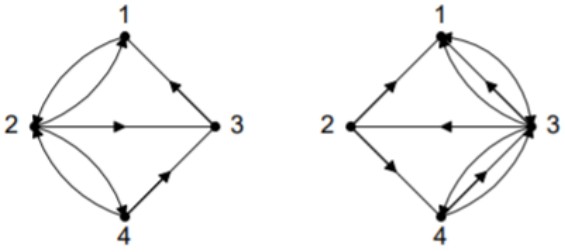
II. LANDASAN TEORI

A. Graf

Graf secara umum digunakan untuk mempresentasikan hubungan antara objek-objek diskrit. Graf disimbolkan sebagai sebuah *tuple* yang berisi kumpulan simpul (*vertices*) dan sisi (*edges*) yang membentuknya.

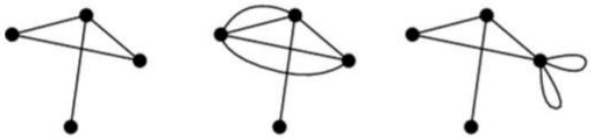
$$G = (V, E)$$

Berdasarkan jenis sisinya, Graf dibagi menjadi graf berarah dan graf tak-berarah. Graf sederhana adalah graf yang memiliki orientasi arah yang spesifik pada setiap sisinya, sedangkan graf yang tak-berarah tidak memiliki oritentasi arah pada sisinya.



Gambar 2. Graf Sederhana dan Graf Tak-Sederhana, diambil dari [2]

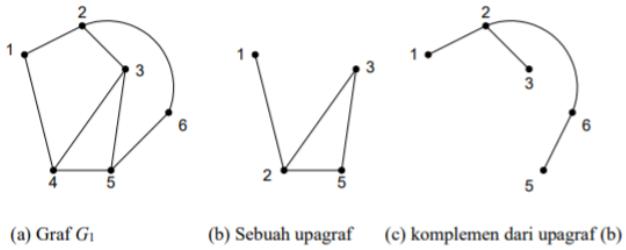
Berdasarkan eksistensi gelang atau sisi ganda pada graf, graf dibagi menjadi graf sederhana dan graf tak-sederhana. Graf sederhana adalah graf yang tidak memiliki sisi yang membuat gelang ataupun sisi ganda. Gelang adalah sisi yang menghubungkan satu simpul yang sama. Kemudian graf tak-sederhana dibagi lagi menjadi graf ganda dan graf semu (*pseudo-graph*). Graf ganda memiliki sisi yang sama lebih dari satu, sedangkan graf semu adalah graf yang dapat memiliki gelang maupun sisi berganda.



Gambar 3. Graf Sederhana, Graf Ganda, dan Graf Semu, diambil dari [2]

Terdapat beberapa terminologi yang digunakan untuk mendeskripsikan suatu graf. Ketetanggaan (Adjacent) adalah kondisi dimana kedua simpul saling terhubung langsung oleh sebuah sisi. Derajat simpul adalah jumlah ujung sisi yang terhubung pada suatu simpul. Lintasan merupakan jalan dari simpul awal ke simpul tujuan didalam graf melalui simpul dan sisi perantara. Panjang dari suatu lintasan dapat dihitung dari jumlah sisi perantaranya.

Sebuah graf dapat dibagi menjadi graf-graf yang lebih kecil. Graf yang lebih kecil ini disebut juga Upagraf atau *Subgraph*. Secara formal, pada graf G = (V, E), H = (V1, E1) merupakan upagraf dari G jika dan hanya jika V1 ⊆ V dan E1 ⊆ E. Komplemen dari upagraf H merupakan upagraf K = (V2, E2) dari G sedemikian sehingga E2 = E - E1 dan V2 adalah simpul yang bersisian pada E2. Graf Teratur adalah graf yang setiap simpulnya memiliki drejat simpul yang seragam, sedangkan graf berbobot adalah graf yang memiliki harga pada setiap sisiya.

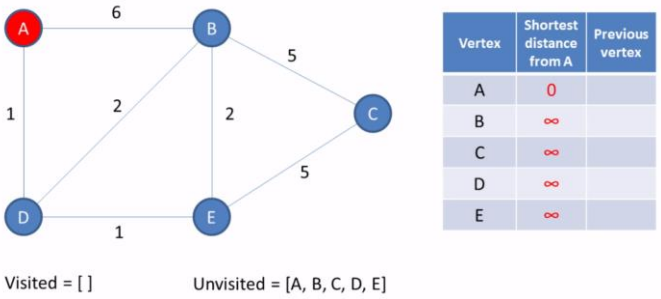


Gambar 4. Upagraf dan Komplemennya, diambil dari [2]

B. Algoritma Djikstra

Algoritma Djikstra adalah sebuah algoritma untuk mencari jarak terpendek dari simpul awal ke simpul tujuan pada suatu graf berbobot. Algoritma ini ditemukan oleh seorang ilmuan komputer bernama Edsger W. Djikstra pada tahun 1956.

Pengerjaan algoritma ini dimulai dari simpul awal yang ditentukan sebelumnya. Sebagai contoh perhatikan graf berikut dibawah ini.



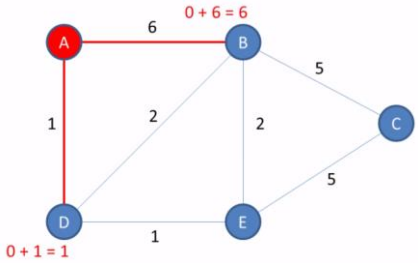
Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Gambar 5. Inisialisasi Algoritma Djikstra, diambil dari [3]

Pada contoh ini, simpul A ditetapkan sebagai simpul awal sekaligus simpul acuan dan simpul C ditetapkan sebagi simpul tujuan. Algoritma Djikstra membutuhkan matriks yang mencatat jarak terpendek setiap simpul terhadap simpul A serta simpul sebelumnya (*Previous Vertex*) untuk pencatatan jejak rute. Selain itu algoritma ini juga membutuhkan dua buah list yang berupa list dari simpul yang sudah "dikunjungi" (*Visited*) dan list dari simpul yang belum "dikunjungi" (*Unvisited*) untuk melacak kondisi dari setiap simpul.

Sebagai inisiasi, jarak terpendek setiap simpul pada matriks diberi nilai yang cukup besar, misalnya tak-hingga (∞).Khusus jarak simpul awal diberi jarak 0. simpul sebelum pada matriks dibiarkan kosong. Setelah itu, semua simpul pada graf didaftarkankan kedalam list dari simpul yang belum "dikunjungi", sedangkan list dari simpul yang sudah "dikunjungi" dibiarkan kosong.

Langkah pertama, hitung jarak dari simpul acuan ke simpul tetangga yang masih terdaftar pada list "belum dikunjungi". Jarak merupakan bobot sisi ditambah jarak simpul terhadap simpul awal yang terdapat dalam matriks. Pada contoh dapat dilihat jarak A ke B adalah 6 serta simpul A bernilai 0 dimatriks, sehingga jarak A ke B adalah 6 + 0 = 6. Jika jarak yang didapat lebih kecil nilainya dari yang ada di matriks, maka nilai yang matriks diganti. Pada contoh ini, nilai simpul B menjadi 6 karena tak-hingga lebih besar dari 6. Simpul sebelum dari simpul B juga diisi titik acuan. Setelah semua simpul tetangga sudah dihitung, simpul acuan dipindahkan dari list *Unvisited* kedalam list *Visited*.

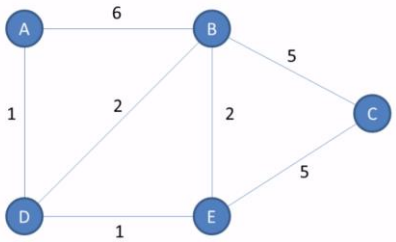


Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Gambar 6. Menghitung jarak simpul-simpul tetangga, diambil dari [3]

Visited = [A] Unvisited = [B, C, D, E]

Setelah itu, ganti simpul acuan ke simpul dengan nilai terkecil pada matriks yang belum pernah "dikunjungi". Pada contoh ini, simpul acuan berubah menjadi simpul D. Pada tahap ini, algoritma mengulang dari langkah pertama lagi sampai semua simpul "dikunjungi" (list *Unvisited* sudah kosong). Setelah itu, jarak terdekat dari simpul awal ke simpul akhir dapat diperoleh dari matriks serta lintasannya dapat dilacak melalui simpul sebelum (*Previous Vertex*). Pada contoh tersebut, jarak terpendek dari A ke C adalah 7. Simpul sebelum dari C adalah E, simpul sebelum dari E adalah D, serta simpul sebelum dari D adalah A. Maka dapat diketahui lintasan terpendek dari simpul A ke C adalah A → D → E → C.



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Gambar 7. Hasil dari Algoritma Djikstra, diambil dari [3]

C. Algoritma A-Star

Algoritma pencarian A* atau disebut juga A-Star merupakan salah satu algoritma yang digunakan untuk mencari lintasan terpendek antara dua simpul pada suatu graf. Algoritma ini ditemukan oleh Peter Hart, Nils Nilsson, dan Bertram Raphael dari *Stanford Research Institute* pada tahun 1968. Algoritma ini merupakan optimalisasi lanjut dari algoritma Djikstra yang ditemukan oleh Edsger W. Djikstra dengan memanfaatkan fungsi heuristik.

Didalam lingkup ilmu komputer dan intelegensi buatan, fungsi heuristik digunakan untuk memecahkan masalah lebih cepat dengan cara memperkecil lingkup pencarian dan mengaprokmasi hasil pencarian. Dengan memanfaatkan fungsi ini, algoritma dapat dibuat lebih optimal, tepat sasaran, dan memakan waktu lebih sedikit.

Pada Algoritma A-Star, fungsi heuristic dimanfaatkan untuk menentukan seberapa dekat setiap simpul pada graf ke simpul tujuan. Rumus untuk penentuan fungsi heuristik bermacam-macam. Sebagai contoh perhatikan contoh graf dibawah ini. Misalkan pada graf tersebut ditetapkan sebuah simpul tujuan pada koordinat (5,3). Untuk titik (5,3) diberi nilai heuristik H(5,3) = 0 karena jarak sebuah simpul yang sama adalah 0. Selanjutnya dibuat rumusan aturan heuristik bahwa setiap simpul saling bersisian dengan simpul lainnya yang berada diatas, kiri, bawah, dan kanan serta sisi-sisi tersebut diberi bobot 10. Setelah itu, nilai heuristik semua simpul dapat dihitung dari jarak simpul dengan titik tujuan sebagai acuannya seperti pada gambar.

	1	2	3	4	5	6	7	8
1	60	50	40	30	20	30	40	50
2	50	40	30	20	10	20	30	40
3	40	30	20	10	0	10	20	30
4	50	40	30	20	10	20	30	40
5	60	50	40	30	20	30	40	50
6	70	60	50	40	30	40	50	60

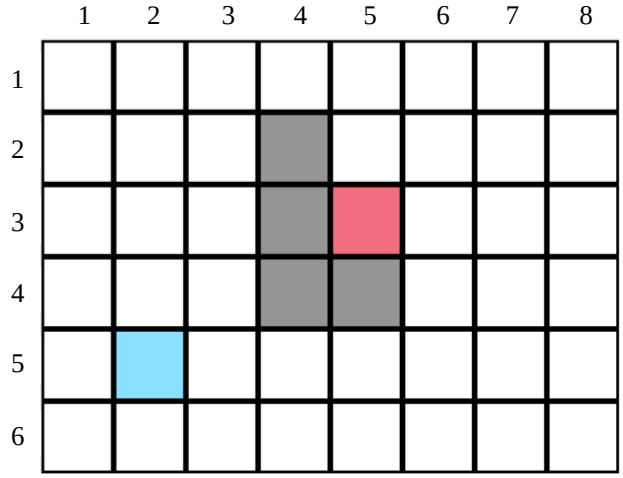
Gambar 8. Contoh Perhitungan Fungsi Heuristik Setiap Simpul pada Peta

Selain nilai heuristik, terdapat nilai G yang menyimpan besar total langkah dari simpul awal dan nilai F sebagai jumlah dari nilai H dan G.

$$F(x,y) = H(x,y) + G(x,y)$$

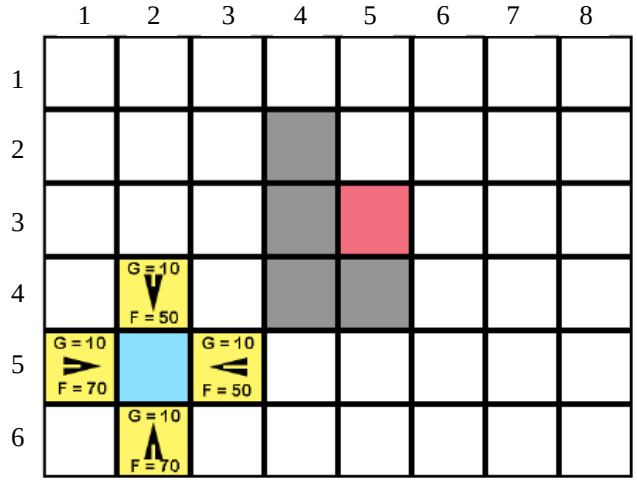
.

Sebagai inisialisasi awal, algoritma ini membutuhkan sebuah *open list* dan *closed list*. *Open list* berisi simpul-simpul yang pernah bertetangga dengan simpul acuan, sedangkan *closed list* berisi simpul-simpul yang berpeluang menjadi kandidat rute ke simpul tujuan. Pada contoh ini, kita tetapkan simpul (2,5) sebagai simpul awal sekaligus simpul acuan pertama. Selain itu kita tetapkan simpul (4,2), (4,3), (4,4), dan (4,5) sebagai pengahalang yang berarti simpul tidak dapat dilewati.



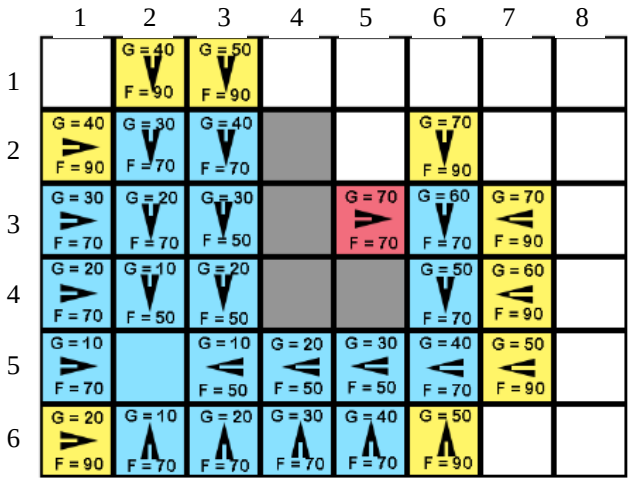
Gambar 9. Contoh sampel peta, simpul biru adalah simpul awal, simpul merah adalah simpul tujuan, sedangkan simpul abu-abu adalah simpul halangan

Langkah pertama, masukkan simpul acuan kedalam *closed list* dan semua simpul yang bertetangga dengannya kedalam *open list*. Untuk setiap simpul yang bertetangga, dihuting nilai G dan F. Setelah diperoleh nilai F pada setiap simpul tersebut, pilih simpul dengan nilai F terkecil untuk dijadikan simpul acuan berikutnya. Jika simpul dengan F terkecil ada lebih dari satu, maka semua simpul terkecil akan diproses sebagai simpul acuan berikutnya. Simpul-simpul yang berwarna kuning adalah simpul pada *open list*¸ sedangkan simpul-simpul berwarna biru merupakan simpul pada *closed list*.



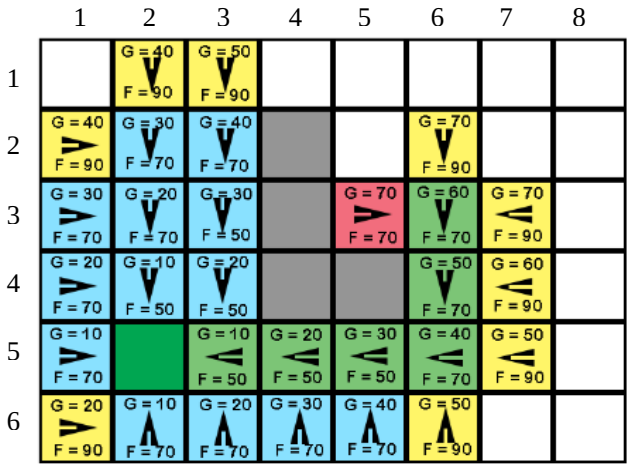
Gambar 10. Perhitungan nilai G dan F beserta arahnya pada setiap simpul di *open list*

Selanjutnya ulangi proses dari langkah pertama sampai simpul tujuan ada didalam *closed list* atau sampai *open list* kosong. Saat menghitung nilai G pada simpul tetangga, jika nilai G yang baru lebih kecil dari nilai G sebelumnya, maka nilai G diganti dengan nilai yang baru serta anak panah berganti arah menunjuk ke simpul acuan sekarang.



Gambar 11. Hasil dari Algoritma A-Star.

Jika simpul tujuan sudah terdaftar didalam *closed list*, maka dapat dilakukan *back-tracking* pada *closed list* untuk menentukan rute terpendeknya. Perlu dicatat tidak semua anggota *closed list* merupakan anggota dari rute terpendek. Pada contoh ini, anggota dari hasil rute diwarnai warna hijau.



Gambar 12. Penjejakan kembali untuk menentukan rute hasil perhitungan Algoritma A-Star

Jika anggota pada *open list* sudah kosong, padahal simpul tujuan belum masuk kedalam *closed list*, maka dapat disimpulkan bahwa simpul tujuan tidak dapat diakses dari simpul awal, atau dengan kata lain graf simpul awal dan simpul tujuan tidak terhubung oleh sisi apapun.

III. KINERJA ALGORITMA DJIKSTRA DAN ALGORITMA A-STAR BESERTA PERBANDINGANNYA

A. Kinerja Algoritma Djikstra

Dalam mencari rute terpendek antara kedua simpul yang sudah ditentukan sebelumnya, algoritma djikstra mengecek semua kemungkinan rute dengan cara "mengunjungi" setiap simpul yang ada pada graf. Algoritma ini seakan membanjiri semua simpul sampai . Hal inilah yang menyebabkan algoritma djikstra masuk kedalam kategori algoritma rakus (*greedy*).

Algoritma rakus (*Greedy Algorithm*) adalah algoritma yang sederhana dan intuitif yang menggunakan metode mirip dengan *brute-force* dalam memecahkan suatu permasalahan. Algoritma semacam ini dapat menyelesaikan masalah-masalah dalam skala kecil dengan baik, namun seiring bertambahnya data yang harus dikelola, waktu penyelesaian yang dibutuhkan bertambah secara signifikan.

Dalam implementasinya ke *video game*, algoritma ini membutuhkan ruang memori yang relatif besar. Selain itu, beberapa *video game* seperti *Tower Defense* membutuhkan perhitungan rute terpendek berulang kali membuat algoritma ini kurang efektif untuk diterapkan ke pasaran.

B. Kinerja Algoritma A-Star

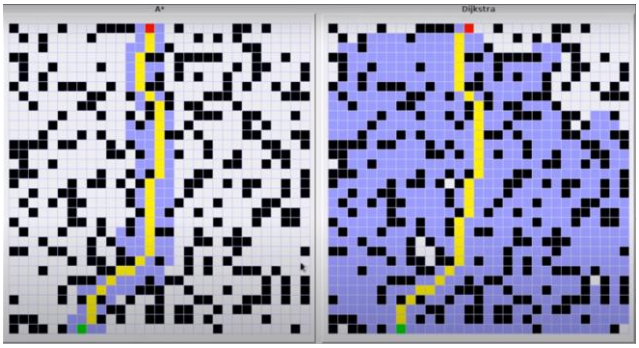
Sebagai ekstensi lanjut dari algoritma djikstra, algoritma A-Star memanfaatkan intelegensi buatan untuk mempersempit lingkup perhitungan pada algoritma. Fungsi heuristik pada algoritma ini membantu perhitungan agar memprioritaskan simpul-simpul yang letaknya lebih dekat dengan simpul tujuan.

Tingkat keefektifan dari algoritma ini sangat berpengaruh dengan fungsi heuristik yang ditetapkan. Semakin dekat aproksimasi nilai heuristik terhadap jarak asli simpul, semakin baik kinerja program dalam mencari rute terpendek.

Algoritma A-Star sudah banyak diimplementasikan dalam pembuatan *video game*. Beberapa *genre game* seperti *Tower Defense* sangat bergantung pada algoritma ini dalam menentukan pergerakan pasukan bot yang paling efektif untuk menerobos masuk ke markas pemain..

C. Perbandingan Algoritma Djikstra dan Algortima A-Star

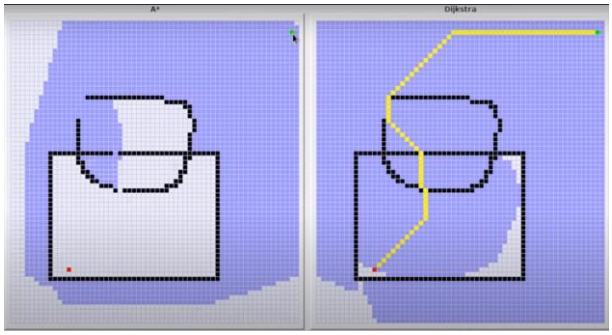
Pada gambar dibawah, ditunjukkan perbandingan kinerja antara algoritma Djikstra dan algoritma A-Star. Algoritma Djikstra memproses semua simpul yang bertetanggaan mulai dari simpul acuan sampai simpul tujuan ditemukan., sedangkan A-Star memproses simpul-simpul yang lebih dekat dengan simpul tujuan.



Gambar 13. Perbandingan kinerja Algoritma A-Star (gambar kiri) dan Algoritma Djikstra (gambar kanan), diambil dari [4]

Dengan bantuan fungsi heuristik, algoritma A-Star membutuhkan waktu jauh lebih sedikit dibanding algoritma Djikstra karena algoritma A-Star tidak perlu mem-"banjiri" setiap simpul untuk mencari rute terpendek. Namun perlu diperhatikan jika fungsi heuristik yang dibangun tidak tepat sasaran atau kurang baik dalam mengaproksimasi jarak sesungguhnya, justru algoritma A-Star akan membutuhkan

waktu yang lebih lama dibandingkan algoritma Djikstra, seperti pada gambar dibawah ini.



Gambar 14. Perbandingan kinerja Algoritma A-Star (gambar kiri) dan Algoritma Djikstra (gambar kanan), diambil dari [4]

Oleh karena itu, berbagai faktor perlu dipertimbangkan dalam membangun fungsi heuristik yang baik pada algoritma A-Star agar fungsi yang dibangun tersebut tidak malah menghambat kinerja dari algoritma.

IV. Penerapan Algoritma A-Star dalam Mencari Rute Permainan Tower Defense

Seperti yang telah dibahas sebelumnya, algoritma A-Star dapat diimplementasi dengan baik untuk menentukan rute terpendek yang harus diambil oleh pasukan untuk dapat masuk ke markas pemain. Sebagai inisialisasi, tetapkan sebuah node sebagai simpul awal (simpul tempat pasukan muncul) dan simpul tujuan (simpul lokasi markas pemain). Kemudian buat 2 buah list (untuk *open list* dan *closed list*) berupa *array of tuple*. *Tuple* pada *array* menyimpan koordinat titik, arah panah rute, nilai G, dan nilai F. Selain itu, buat sebuah matriks yang menyimpan nilai Heuristik dari setiap simpul. Ukuran matriks yang dialokasi sesuai dengan jumlah simpul pada peta.

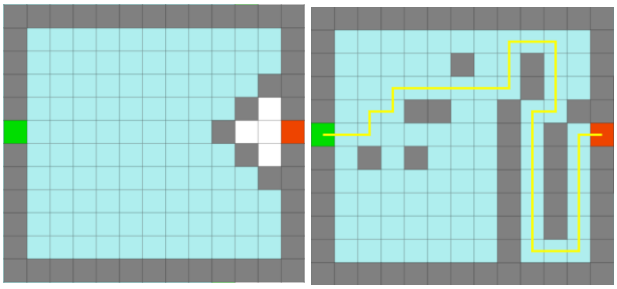
```
typedef struct
{
    Point Koordinat; // Berisi koordinat simpul
    Point Arah; // Berisi koordinat yang ditunjuk
    int G; // Nilai G(Simpul)
    int F; // Nilai H(Simpul) + G(Simpul)
} Simpul;

typedef struct {
    Simpul Mem[100];
    int NEff; // banyaknya/ukuran array yg terdefinisi
} ArraySimpul;

typedef struct {
    int Mem[BarisPeta][KolomPeta]; // Berisi Nilai Heuristik tiap simpul
} Heuristik;
```

Langkah pertama, rumuskan sebuah fungsi heuristik yang dapat mengaprokmasi jarak sebenarnya dengan baik. Contoh fungsi heuristik yang umum digunakan adalah simpul tujuan dijadikan simpul pusat dan diberi nilai heuristik 0. Kemudian dimulai dari simpul yang bertetangga dengan simpul pusat, nilai heuristik dari suatu simpul ditentukan dari nilai heuristik terkecil dari simpul yang bertetangga ditambah bobot sisi.

Langkah berikutnya dimulai dari simpul awal sebagai simpul acuan, masukkan simpul acuan kedalam *closed list* dan semua simpul yang bertetangga dengannya kedalam *open list*. Untuk setiap simpul yang bertetangga, dihuting nilai G dan F. Setelah diperoleh nilai F pada setiap simpul tersebut, pilih simpul dengan nilai F terkecil untuk dijadikan simpul acuan berikutnya. Algoritma ini diulang sampai simpul tujuan berada pada *closed list*.

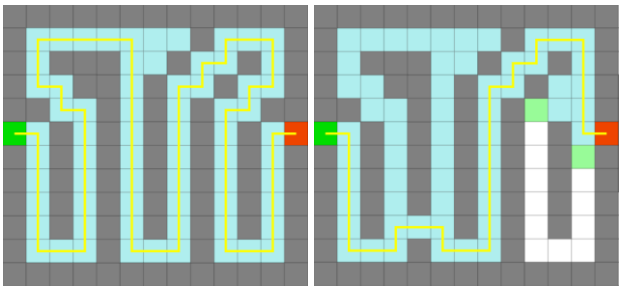
Algoritma ini akan digunakan saat awal permainan, saat pemain ingin meletakkan sebuah menara pada peta, saat pemain membangun sebuah menara, dan saat pemain membuang/menjual sebuah menara.

Ketika pemain ingin meletakkan sebuah menara, program harus mengetahui apakah lokasi yang ingin dibangun pemain menutup penuh jalan ke markas atau tidak. Hal ini dilakukan agar pasukan selalu memiliki rute untuk masuk kedalam markas pemain tidak peduli seberapa banyak menara yang diletakkan pemain. Jika lokasi yang dipilih pemain dapat menyebabkan rute terblokir, maka harus diberi penanganan error.



Gambar 15. Peta dengan simpul tujuan yang terblokir (gambar kiri) dan peta dengan simpul tujuan terbuka (gambar kanan)

Saat pemain membuang/menjual sebuah menara, program harus menghitung ulang rute terpendek pada peta karena dapat berpotensi untuk menemukan rute terpendek yang baru untuk pasukan. Hal inilah yang membuat optimalisasi dari Algoritma A-Star sangat krusial dibanding Algoritma Djikstra karena pasukan membutuhkan hasil perhitungan rute dengan cepat sesaat setelah menara dihilangkan.



Gambar 16. Rute sebelum menara dihapus (gambar kiri) dan rute setelah beberapa menara dihapus (gambar kanan)

Salah satu permainan *Tower Defense* yang mengimplementasikan algoritma ini dengan baik adalah *Robo Defense*. Permainan ini terdiri dari 100 level dan pada setiap levelnya akan muncul beberapa pasukan dari simpul awal untuk menginvasi markas pemain. Pemain dapat membangun, meng-upgrade dan menjual menara untuk menembak pasukan untuk mencegah mereka sampai dimarkas. Formasi lokasi menara yang dibangun pemain hanya dapat membuat rute pasukan semakin panjang, tidak dapat sampai memblokir penuh jalan menuju markas pemain. Jika pemain memilih lokasi yang berpotensi memblokir jalan menuju menara, permainan ini akan memberikan pesan error dan menara tidak dapat dibangun.

V. KESIMPULAN

Algoritma A-Star adalah salah satu algoritma pencarian rute terpendek yang efektif intuk diimplementasikan pada game seperti *Tower Defense*. Algoritma ini dapat mencari rute terpendek menggunakan bantuan fungsi heuristik tanpa perlu mengecek seluruh node pada peta. Semakin baik fungsi heuristik dalam memprediksi jarak setiap simpul ke simpul tujuan, semakin efektif pula algoritma ini dalam mencari hasil terpendeknya.

VI. UCAPAN TERIMA KASIH

Makalah ini dapat diselesaikan tepat waktu karena kasih dan karunia dari Tuhan Yang Maha Esa. Penulis sangat bersyukur karena mendapat kesempatan untuk mengikuti Kelas Matematika Diskrit dengan baik bersama Bapak Rinaldi. Tidak lupa penulis juga berterima kasih kepada orang tua penulis dan teman-teman yang selalu mendukung penulis secara langsung maupun tidak langsung. Penulis sebelumnya meminta maaf jika ada kesalahan kata dalam penulisan makalah ini dan berharap agar makalah ini dapat bermanfaat bagi para pembaca. Akhir kata, penulis mengucapkan selamat membaca.

REFERENSI

[1] https://www.youtube.com/watch?v=o9HWNI95d9A, diakses pada 9 Desember 2020.
[2] http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf, diakses pada 9 Desember 2020.
[3] https://www.youtube.com/watch?v=pVfj6mxhdMw&t=4s, diakses pada 9 Desember 2020.
[4] https://www.youtube.com/watch?v=g024lzsknDo, diakses pada 9 Desember 2020.

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, 9 Desember 2020

Nicholas Chen 13519029