Penerapan Algoritma A* dalam *Pathfinding* Permainan Total War: Shogun 2

Fitrah Ramadhani Nugroho - 13520030
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13520030@std.stei.itb.ac.id

***Abstract*—Total War: Shogun 2 adalah permainan strategi yang bertujuan untuk mengalahkan musuh dan mempertahankan wilayah dari serangan musuh. Total War: Shogun 2 terdiri dari dua bagian yaitu mempertahankan musuh dari serangan dan menyerang kota musuh. Kedua bagian ini dilakukan dengan menghancurkan pasukan musuh dalam pertempuran. Pasukan pemain perlu digerakkan menuju kota musuh. Terdapat berbagai macam jalan bagi pasukan agar bisa sampai ke kota musuh. Pasukan pemain perlu untuk sampai di kota musuh sebelum kota tersebut dipertahankan oleh pasukan mush. Untuk itu, dalam makalah ini akan membahas penggunaan algoritma A* untuk mencapai ke kota musuh secepat mungkin. (*Abstract*)**

Keywords—a-star;Total War; pathfinding;

I. PENDAHULUAN

Permainan seri Total War adalah permainan strategi yang dikembangkan oleh pengembang Creative Assembly. Total War mengkombinasikan permainan berbasis giliran (*turn based*) dengan manajemen sumber daya beserta permainan taktikal waktu nyata (*Real Time Strategy*). Seri permainan Total War yang pertama kali dirilis adalah Shogun: Total War yang dirilis pada tahun 2000.

Total War: Shogun 2 merupakan salah satu seri Total War yang berlatarkan di Jepang pada masa *Sengoku Jidai* pada abad ke 16. Di permainan Total War: Shogun 2 pemain memainkan sebuah klan atau faksi. Pemain ditugaskan untuk memenangkan permainan dengan menguasai kota di jepang sesuai dengan jumlah yang ditentukan. Pemain dapat menguasai kota ini dengan cara mengalahkan klan lainnya yang menguasai kota tersebut dan mengalahkan pasukan musuh.

Terdapat berbagai macam cara bagi pemain untuk memenangkan permainan. Pemain dapat menggunakan diplomasi dengan klan lain seperti membuat persekutuan, mendeklarasikan perang, mensahkan perjanjian damai dan masih banyak lagi. Pemain juga bisa menguasai sumber daya alam yang penting untuk membuat pasukan atau menambah kekayaan bagi klan. Pemain juga bisa mengirim agen mata-mata untuk mensabotase pasukan musuh, menyuap pasukan musuh atau pemimpin pasukan, Namun, cara utama bagi pemain untuk bisa memenangkan permainan adalah dengan mengalahkan pasukan musuh dan menguasai kota musuh.

Penting bagi pemain untuk secepat mungkin menguasai kota musuh sebelum pasukan musuh mendudukinya. Karena kota yang diduduki oleh pasukan musuh lebih susah untuk dikuasai dibandingkan dengan kota yang tidak diduduki pasukan musuh. Selain itu, lebih mudah mempertahankan kota yang kita rebut dengan menduduki kota tersebut dengan pasukan kita daripada melawan musuh langsung di tempat terbuka. Persoalan pencarian rute pasukan untuk sampai ke kota musuh dapat dimodelkan dengan beberapa algoritma pencarian rute. Salah satu algoritma pencarian rute tersebut adalah algoritma A*. Algoritma A* dipilih karana salah satu algoritma dengan pencarian heuristik yang optimal. Dengan memilih perkiraaan sisa jarak dari posisi pasukan dengan kota musuh sebagai pertimbangan fungsi evaluasi heuristik, algoritma ini dapat digunakan untuk mencari rute yang paling cepat.

II. DASAR TEORI

A. *Permainan Total War: Shogun 2*

Permainan Total War: Shogun 2 merupakan seri permainan Total War yang dikembangkan oleh pengembang Creative Assembly dan diterbitkan oleh Sega. Total War: Shogun 2 diterbitkan pada tahun 2011 untuk komputer. Seri Total War merupakan permainan yang mengkombinasikan permainan berbasis giliran (*turn based*) dengan manajemen sumber daya beserta permainan taktikal waktu nyata (*Real Time Strategy*).

Total War: Shogun 2 merupakan salah satu seri Total War yang berlatarkan di Jepang pada masa *Sengoku Jidai* pada abad ke 16. Di permainan Total War: Shogun 2 pemain memainkan sebuah klan atau faksi. Terdapat sembilan klan ditambah tiga klan sebagai DLC (*Downloadable Content)* yang bisa dimainkan oleh pemain. Pemain ditugaskan untuk memenangkan permainan dengan menguasai kota di jepang sesuai dengan jumlah yang ditentukan. Terdapat tiga mode permainan yang mengatur jumlah kota yang harus dikuasai. Pemain dapat menguasai kota ini dengan cara mengalahkan klan lainnya yang menguasai kota tersebut dan mengalahkan pasukan musuh.

Terdapat berbagai macam fitur di permainan Total War: Shogun 2 yang bisa digunakan oleh pemain. Fitur pertama adalah diplomasi. Pemain bisa membuat persekutuan dengan

klan lain, menikahkan salah satu anaknya dengan anak dari klan lain.

Fitur kedua adalah spionase. Pemain bisa mengirimkan ninja untuk mensabotase pasukan atau benteng musuh. Pemain juga bisa mengirimkan pendeta untuk mengubah agama dari klan musuh. Pemain juga bisa mengirimkan "Metsuke" atau agen rahasia untuk menyuap pasukan atau pemimpin pasukan. Masih ada agen lainnya yang bisa digunakan oleh pemain seperti "Geisha". "Geisha" bisa digunakan untuk mendikstraksi pasukan musuh atau membunuh pemimpin pasukan musuh.

Fitur ketiga dan merupakan fitur utama adalah pertarungan. Ketika pasukan musuh bertemu dengan pasukan pemain maka kedua pasukan akan saling bertarung. Disini pemain bisa mengendalikan pasukan yang dimilikinya secara langsung. Ada banyak jenis pasukan yang bisa dimiliki pemain seperti pemanah, samurai, kavaleri, dan lainnya. Pemain juga bisa menggunakan berbagai macam taktik dalam mengalahkan pasukan musuh. Pertarungan berakhir ketika semua pasukan musuh atau pemain sudah menyerah atau waktu pertarungan sudah berakhir. Pasukan yang kalah biasanya akan mundur dari pertarungan. Jumlah pasukan yang hilang bisa diisi lagi ketika pasukan berada di kota yang dimiliki klan tersebut. Masih banyak fitur-fitur permainan yang dimiliki oleh Total War: Shogun 2 seperti fitur untuk membuat armada laut, fitur pohon keluarga klan, fitur peningkatan unit pasukan dan lainnya.

Total War: Shogun 2 juga memiliki mode pertempuran bersejarah. Dalam mode ini pemain tidak memainkan satu klan secara keseluruhan namun pemain hanya memainkan salah satu pasukan di salah satu pertempuran bersejarah ini.

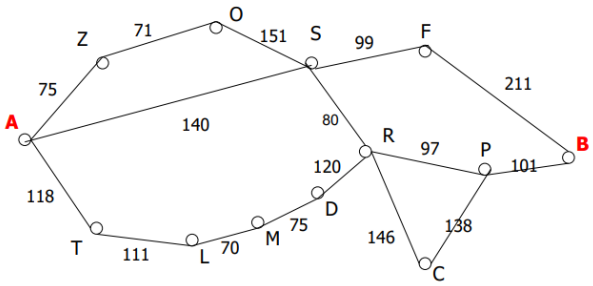
Total War: Shogun 2 menawarkan dua DLC (*Downloadable Content*) yang berada di era yang berbeda dengan permainan utamanya. DLC pertama adalah "Rise of the Samurai" merupakan DLC yang berlatar belakang di jaman perang Genpai pada tahun 1175. DLC kedua adalah "Fall of the Samurai" merupakan DLC yang berlatar belakang di jaman perang Boshin pada tahun 1864. Kedua DLC ini selain berada di latar yang berbeda juga menawarkan unit-unit pasukan baru, agen-agen baru serta fitur-fitur baru yang belum ada sebelumnya.

B. Algoritma Penentuan Rute

Algoritma penentuan rute merupakan algoritma yang digunakan untuk menentukan rute dengan biaya paling sedikit atau jarak paling dekat. Terdapat dua jenis pencarian dengan menggunakan algoritma tersebut.

Pertama adalah *uninformed search* atau pencarian tanpa informasi tambahan. Dalam algoritma *uninformed search* atau sering juga disebut *blind search* penentuan rute paling cepat hanya menggunakan informasi yang tersedia dari algoritma seperti biaya setiap jalur yang ada. Algoritma *uninformed search* terdiri dari *Bread-First Search* (BFS), *Depth First Search* (DFS), *Iterative Deepening Search* (IDS) dan *Uniform Cost Search* (UCS).

Jenis kedua adalah *informed search*. Berbeda dengan *uninformed search,* algoritma *informed search* menggunakan informasi tambahan dalam menentukan rute dengan biaya paling sedikit. Informasi tambahan ini dicari dengan menggunakan pendekatan heuristik sehingga algoritma *informed search* juga bisa disebut sebagai *heuristic search*. Ada banyak sekali pendekatan heuristik dalam memperkirakan biaya setiap simpul. Salah satunya adalah dengan menggunakan jarak garis lurus dari simpul A ke simpul C walaupun kedua simpul tersebut tidak memiliki hubungan garis secara langsung. Algoritma *informed search* terdiri dari Algoritma *Greedy Best First Search* dan Algoritma A*.

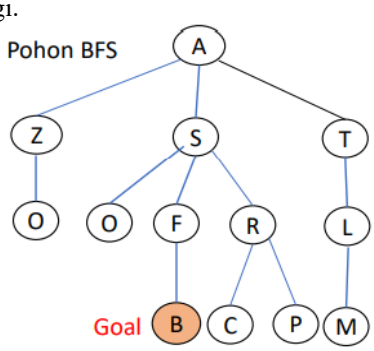


Gambar 1. Contoh Peta untuk Persoalan Pencarian Rute

C. Algoritma Breadth First Search (BFS)

Algoritma *Breadth First Search* (BFS) adalah salah satu algoritma traversal graf yaitu algoritmma yang mengunjungi simpul dengan cara yang sistematik. Graf adalah representasi persoalan yang akan diselesaikan sehingga traversal graf adalah representasi dari pencarian solusi. Algoritma BFS adalah algoritma traversal graf yang menggunakan pendekatan graf statis. Graf statis adalah graf yang telah terbentuk sebelum proses pencarian dilakukan.

Algoritma BFS adalah algoritma pencarian melebar, pencarian dilakukan secara *First In First Out* (FIFO), dimana simpul-simpul pada graf dimasukkan ke dalam sebuah antrian. Algoritma BFS dilakukan dengan mengunjungi simpul v kemudian mengunjungi semua simpul tetangga dari simpul v terlebih dahulu. Setelah semua simpul tetangga dari simpul v telah dikunjungi, algoritma baru mengunjungi simpul tetangga dari simpul tetanga tersebut demikian seterusnya hingga ditemukan simpul yang dicari. Pengujungan simpul tetangga diperlakukan seperti antrian karena pada saat tingkatan pohon atau graf di ekspansi setiap simpul akan dimasukkan ke antrian. Simpul yang sudah dikunjungi sebelumnya tidak akan dikunjungi lagi.



Gambar 2. Contoh Pohon pada Penelusuran Secara BFS

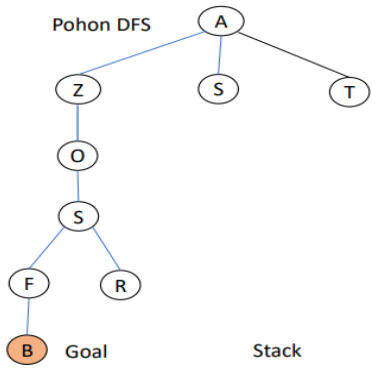
Simpul-E	Simpul Hidup
A	Z_A,S_A,T_A
Z_A	S_A,T_A,O_{AZ}
S_A	T_A,O_{AZ},O_{AS},F_{AS},R_{AS}
T_A	O_{AZ},O_{AS},F_{AS},R_{AS},L_{AT}
O_{AZ}	O_{AS},F_{AS},R_{AS},L_{AT}
O_{AS}	F_{AS},R_{AS},L_{AT}
F_{AS}	R_{AS},L_{AT},B_{ASF}
R_{AS}	L_{AT},B_{ASF},D_{ASR},C_{ASR},P_{ASR}
L_{AT}	B_{ASF},D_{ASR},C_{ASR},P_{ASR},M_{ATL}
B_{ASF}	Solusi ketemu

Gambar 3. Contoh Antrian Simpul pada Penelusuran Secara BFS

Dalam kasus penentuan rute, BFS menggunakan prinsip yang sama dengan traversal graf. Namun, karena algoritma BFS akan berhenti setelah menemukan rute yang dicari, hasil pencarian rute ini hanya berdasar dengan jumlah simpul yang paling sedikit. Hasil pencarian rute ini juga belum pasti pencarian rute dengan biaya paling sedikit karena walaupun rute yang dilewati dengan simpul paling sedikit bisa juga memiliki total biaya rute yang lebih besar daripada rute dengan simpul lebih banyak. Hal ini dicontohkan pada gambar 2, rute solusi adalah A-S-F-B dengan total biaya adalah 450. Padahal ada rute solusi yang lebih efisien yaitu A-S-R-P-B dengan total biaya 418. Sehingga BFS bukanlah algoritma yang tepat untuk penentuan rute dengan biaya minimum.

D. Depth-First Search (DFS)

Sama seperti BFS, *Depth-First Search* (DFS) adalah salah satu algoritma pencarian dengan traversal graf. Perbedaanya adalah algoritma pencarian dilakukan dengan menggunakan *Last In First Out*), dimana simpul-simpul pada graf dimasukkan ke dalam sebuah tumpukan. Pencarian dilakukan dengan mengunjungi simpul v kemudian mengunjungi simpul w yang merupakan tetangga dari simpul v. Algoritma akan mengunjungi tetangga dari simpul w dan mengunjungi tetangga dari tetangga simpul w begitu terus hingga tidak ada lagi simpul yang ada dikunjungi. Kemudian algoritma akan mengunjungi simpul x yang merupakan tetangga lainnya dari simpul v dan begitu terus hingga ditemukan simpul yang sedang dicari.



Gambar 4. Contoh Pohon pada Penelusuran Secara DFS

Simpul-E	Simpul Hidup
A	Z_A,S_A,T_A
Z_A	O_{AZ}, S_A,T_A
O_{AZ}	S_{AZO},S_A,T_A
S_{AZO}	F_{AZOS}, R_{AZOS},S_A,T_A
F_{AZOS}	B_{AZOSF}, R_{AZOS},S_A,T_A
B_{AZOSF}	Solusi ketemu

Gambar 5. Contoh Antrian Simpul pada Penelusuran Secara DFS

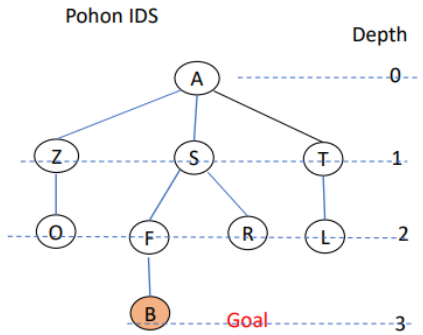
Dalam kasus penentuaan rute, DFS menggunakan prinsip yang sama dengan BFS. Sama juga seperti BFS, karena algoritma akan berhenti setelah menemukan rute yang dicari, hasil pencarian rute ini tidak berdasar dengan jumlah simpul ataupun berdasarkan. Hal ini dicontohkan pada gambar 4, rute solusi adalah A-Z-O-S-F-B dengan total biaya adalah 607. Padahal ada rute yang lebih efisien dengan total biaya 418. Perbedaan biaya yang signifikan menunjukkan DFS bukanlah algoritma yang tepat untuk penentuan rute dengan biaya minimum.

E. Depth Limited Search (DLS)

Depth Limited Search (DLS) merupakan salah satu pengembangan dari DFS. DFS memiliki kelemahan yaitu jika memilih langkah yang salah lintasan yang dibentuk akan panjang bahkan tak terhingga, sehingga pemilihan langkah sangat penting. Oleh karena itu, DLS menawarkan solusinya yaitu dengan membatasi kedalaman sampai tingkat tertentu. Pembatasan ini bertujuan agar DFS tidak terus mencari di lintasan yang sama walaupun sudah jelas tidak akan ketemu. Walaupun begitu permasalahan tidak sepenuhnya selesai karena bagaimana cara menentukan batas tingkat atau *level* yang tepat. Bisa saja DLS membatasi hanya sampai *level* 8 padahal simpul yang dicari berada di *level* 9. Pembatasan tingkat bisa menggunakan pendekatan heuristik walau tidak menjamin kefektifannya.

F. Iterative Deepening Search (IDS)

Selain DLS, *Iterative Deepening Search* (IDS) merupakan salah satu pengembangan dari DFS. IDS juga menerapkan pembatasan tingkat. Berbeda dengan DLS yang sama-sama menerapkan pembatasan tingkat. IDS tidak langsung membatasi pencarian hingga tingkat tertentu namun pembatasan dilakukan satu-persatu. Jika pencarian pada semua simpul di tingkat satu tidak ditemukan. Maka IDS akan membatasi pencarian di tingkat dua dan mencari semua simpul di tingkat tersebut. Begitu seterusnya hingga ditemukan simpul yang diinginkan. Penerapan pembatasan tingkat satu persatu membuat algoritma IDS mirip dengan algoritma BFS yang cenderung mencari di tingkat yang sama terlebih dahulu sebelum lanjut ke tingkat sebelumnya. Hal ini karena tetangga dari simpul v pasti berbeda satu tingkat dengan simpul v itu sendiri.



Gambar 6. Contoh Pohon pada Penelusuran Secara IDS

Seperti yang ditunjukkan pada gambar ke 6 rute yang dihasilkan sama dengan rute yang dihasilkan dengan algoritma BFS pada gambar ke 2 menunjukkan kemiripan algoritma BFS dengan algoritma IDS.

G. *Uniform Cost Search* (UCS)

Uniform Cost Search (UCS) merupakan pengembangan dari algoritma BFS. Seperti yang sudah dijelaskan algoritma BFS hanya mencari rute dengan jumlah simpul terkecil dan bukan dengan biaya terkecil. UCS berusaha mengubah hal itu sehingga rute yang dihasilkan tidak berdasarkan dengan jumlah simpul terkecil namun dengan biaya terkecil. Hal ini dilakukan dengan membuat antrian bukan berdasar urutan simpul dibangkitkan namun berdasar biaya rute. Sehingga simpul hidup seakan-akan ditempatkan pada sebuah *priority queue.*

H. *Greedy Best-First Search*

Greedy Best-First Search merupakan algoritma yang menggunakan informasi tambahan atau *informed search* dalam menentukan rute minimum. *Greedy Best-First Search* menggunakan fungsi evaluasi *f(n)* di setiap simpul. Fungsi evaluasi ini adalah perkiraan estimasi biaya dari simpul n menuju simpul tujuan. Dalam gambar 1 fungsi heuristik yang digunakan adalah fungsi jarak garis lurus dari simpul ke simpul B.

Pencarian dilakukan dengan mebangkitkan simpul yang bertetangga kemudian dengan fungsi evaluasi memilih mana simpul dengan jarak garis lurus paling kecil ke simpul B. Setelah dipilih salah satu simpul, kemudian algoritma dengan cara yang sama akan memilih simpul dengan estimasi jarak garis lurus paling kecil ke simpul B. Hal itu dilakukan terus hingga ditemukan simpul yang dicari.

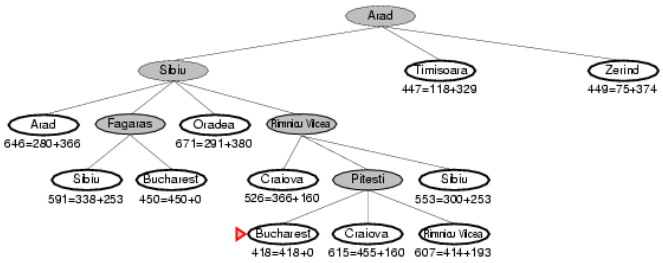
Algoritma pencarian disebut algoritma *greedy* karena algoritma akan selalu mencari jarak dengan fungsi evaluasi terkecil tanpa menghitung total jarak dari simpul n ke simpul tujuan. Sayangnya cara tersebut tidak selalu menemukan rute dengan biaya minimum. Kelemahan lainnya karena algoritma membolehkan kembali ke simpul yang sudah dievaluasi akan ada kemungkinan algoritma akan terjebak di kedua simpul minimum lokal sehingga algoritma hanya akan berputar-putar di simpul tersebut. Algoritma *Greedy Best-First Search* tidak bisa putar balik atau diubah kembali ke simpul sebelumnya.

I. *A* Search*

Algoritma A* *Search* merupakan pengembangan dari algoritma *Greedy Best-First Search*. Berdasarkan kelemahan dari algoritma *Greedy Best-First Search* diketahui bahwa algoritma tersebut tidak mendekatan pendektan heuristik. Algoritma A* dikembangkan dengan ide menghindari memilih jalur yang sudah mahal dengan pendekatan heuristik. Sama seperti *Greedy Best-First Search*, Algoritma A* juga menggunakan fungsi evaluasi dalam penentuan jalur minimum. Fungsi evaluasi dari algoritma A* adalah

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

Fungsi *g(n)* berasal dari algoritma UCS yaitu biaya dari simpul awal hingga simpul N. Sedangkan fungsi *h(n)* berasal dari algoritma *Greedy Best-First Search* yaitu fungsi jarak garis lurus dari simpul N hingga simpul tujuan. Sehingga fungsi evaluasi *f(n)* adalah total biaya dari simpul awal hingga simpul N ditambah dengan jarak garis lurus dari simpul N hingga simpul tujuan. Penggunaan fungsi evaluasi ini menyebabkan algoritma tidak serta-merta berhenti ketika menemukan simpul yang dicari, namun tetap mencari kemungkinan simpul dengan total biaya *f(n)* yang lebih kecil daripada total biaya rute sementara.



Gambar 7. Contoh Pohon pada Penelusuran Secara A*

Pada gambar 7 terlihat bahwa walaupun pada simpul kota Fagaras sudah ada simpul kota Bucharest yang merupakan kota tujuan, penentuan rute tidak langsung memutuskan bahwa rute Arad-Sibiu-Fagaras-Bucharest merupakan rute dengan biaya minimum. Melainkan algoritma A* mengecek apakah masih ada total biaya yang lebih kecil dari rute tersebut. Ternyata rute Arad-Sibiu-Pitesti-Bucharest memiliki total biaya yang lebih kecil daripada rute sebelumnya. Hal ini menjadi bukti bahwa algoritma A* akan selalu menghasil penentuan rute dengan biaya minimum.

III. Implementasi

Dalam penggunaan algoritma A* untuk penentuan rute perlu diketahui terlebih dahulu fungsi evaluasi heuristik. Fungsi evaluasi heuristik pada algoritma A* terdiri dari dua bagian, yaitu *g(n)* yang menyatakan biaya dari simpul awal hingga simpul ke n, dan fungsi *h(n)* yang merupakan estimasi cost dari simpul n ke simpul tujuan.

A. *Nilai g(n)*

Nilai *g(n)* yaitu biaya dari simpul awal ke simpul N. Pada makalah ini, biaya yang dimaksud adalah jumlah ubin yang dibutuhkan pasukan untuk jalan dari kota awal ke

kota N. Berikut ini jalur yang tersedia dalam peta permainan Total War: Shogun 2

Gambar 8. Peta Jalur Antarkota di Jepang pada Permainan Total War: Shogun 2
(Sumber:http://forums.totalwar.org/vb/images/guides/tws2accessmap.jpg)

Berdasarkan gambar ke 8 terlihat ada 3 macam akses yang bisa dilewati. Jalan yang pertama adalah akses jalan langsung yaitu jalan yang sudah ditentukan sebelumnya. Jalan yang kedua adalah akses jalan tidak langsung yaitu jalan yang harus melewati kota lain terlebih dahulu. Jalan yang ketiga adalah jalan *off road* yaitu jalur yang tidak ada jalan pastinya. Di permainan Total War yang berbasis giliran, pasukan yang melewati jalan yang tersedia akan lebih cepat daripada pasukan yang melewati jalan *off road.* Jumlah ubin pada akses jalan langsung inilah yang akan menjadi fungsi *g(n)* pada makalah ini.

B. *Nilai h(n)*

Nilai *h(n)* merupakan fungsi heuristik yaitu perkiraan jarak garis lurus dari kota ke N ke kota tujuan. Pada penerapan di makalah ini, jarak garis lurus ditentukan oleh perkiraan jumlah ubin yang dibutuhkan jika ditarik garis lurus dari kota ke N ke kota tujuan. Penggunaan jumlah ubin saat ditarik garis lurus karena jarak garis lurus adalah jarak paling dekat yang harus dilewati oleh sebuah pasukan dari kota ke N ke kota tujuan.

C. *Penghitungan Nilai f(n)*

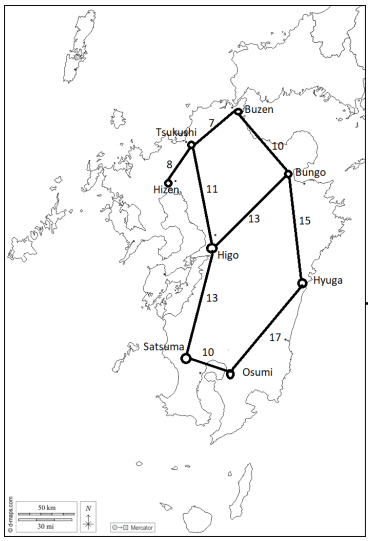
Dalam algoritma A*, nilai fungsi evaluasi total dihitung dengan menggunakan rumus berikut:

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

Dengan nilai *g(n)* dan *h(n)* seperti yang sudah dijelaskan di atas. Simpul selanjutnya akan ditelusuri berdasar pada simpul hidup dengan nilai *f(n)* terkecil, dengan anggapan simpul tersebut adalah jalur dengan jarak terpendek.

IV. PENGUJIAN

Pengujian akan dilakukan berdasarkan peta pulau Kyushu yang terdiri dari provinsi Satsuma, Osumi, Hyuga, Bungo Higo, Buzen, Tsukushi, dan Hizen. Selain itu, pengujian akan dilengkapi dengan tabel perkiraan jarak garis lurus dari provinsi satu ke provinsi lainnya di pulau Kyushu. Peta yang digunakan adalah sebagai berikut.



Gambar 9. Peta Jalur Antarkota di Pulau Kyushu pada Permainan Total War: Shogun 2

Pada pengujian ini akan menentukan rute tercepat dari Kota Osumi menuju Kota Buzen. Untuk itu diperlukan informasi tambahan yaitu tabel jarak garis lurus menuju kota Buzen.

TABLE I. JARAK GARIS LURUS KE KOTA BUZEN

Kota	Jarak
Osumi	24
Satsuma	22
Hyuga	17
Bungo	7
Tsukushi	5
Hizen	9
Higo	12
Buzen	0

Berdasarkan tabel diatas akan dicari rute tercepat dari Kota Osumi menuju Kota Buzen.

TABLE II. HASIL PENGUJIAN RUTE KOTA OSUMI KE KOTA BUZEN

Iterasi	Simpul-Ekspan	Simpul Hidup	g(n)+h(n)= f(n)
1	Osumi	Satsuma(Osumi)	22+10= 32
		Hyuga(Osumi)	17+17=34
2	Satsuma(Osumi)	Higo(Satsuma)	23+12=35
3	Hyuga(Osumi)	Bungo(Hyuga)	32+7=39
4	Higo(Satsuma)	Tsukushi(Higo)	34+5=39
		Bungo(Higo)	36+7=43
5	Tsukushi(Higo)	Buzen(Tsukushi)	41+0=41
6	Bungo(Hyuga)	Buzen(Bungo)	42+0=42

Dari tabel 2, dapat dilihat bahwa algoritma A* berhasil menemukan jalur tercepat dari Kota Osumi menuju ke Kota Buzen dengan biaya sebesar 41 dan dengan rute Osumi-Satsuma-Higo-Tsukushi-Buzen. Algoritma A* juga sempat mengevaluasi rute lainnya yang memiliki kemungkinan lebih cepat seperti rute Osumi-Hyuga-Bungo-Buzen dengan biaya

sebesar 42. Walaupun kedua rute hanya memiliki perbedaaan biaya sebesar 1 tetapi tetap saja rute pertama lebih cepat daripada rute kedua.

Pengujian kedua akan dilakukan dengan mencari rute tercepat dari Kota Higo menuju Kota Buzen

TABLE III. HASIL PENGUJIAN RUTE KOTA HIGO KE KOTA BUZEN

Iterasi	Simpul-Ekspan	Simpul Hidup	g(n)+h(n)= f(n)
1	Higo	Tsukushi(Higo)	11+5=16
		Bungo(Higo)	13+7=20
2	Tsukushi(Higo)	Buzen(Tsukushi)	18+0=18

Tabel ketiga meunjukkan bahwa rute tercepat dari Kota Higo ke Kota Buzen adalah melewati kota Tsukushi dan bukan Kota Bungo. Hal ini karena jarak garis lurus Kota Tsukushi ke Kota Buzen ditambah jarak dari Kota Higo ke Kota Tsukushi jauh lebih kecil daripada jarak garis lurus Kota Bungo ke Kota Buzen ditambah jarak dari Kota Higo ke Kota Bungo. Total biaya dari rute Higo-Tsukushi-Buzen adalah 18.

Berdasarkan dua percobaan diatas membuktikan bahwa algoritma A* cukup efektif adalam menemukan rute dengan biaya minimum. Walaupun begitu menemukan rute dengan biaya minimum bukan merupakan satu-satunya faktor agar unit bisa segera sampai ke kota masuh.

V. KESIMPULAN

Ada banyak sekali algoritma yang bisa digunakan dalam pencarian rute seperti BFS, DFS, IDS, dan UCS. Ada juga algoritma yang fokus pada mencari rute dengan biaya minimum seperti *Greedy Best-First Search* dan *A* Search*. Walaupun begitu tidak semua algoritma efisien dalam mencari rute. Algoritma *Greeedy Best-First Search* contohnya memiliki permasalahan yaitu rute yang dihasilkan tidak selalu optimal dan algoritma bisa terjebak dalam *infinite loop.*

Algoritma A* merupakan algoritma yang paling cocok dalam penentuan rute atau *pathfinding* dengan biaya seminimum mungkin. Algoritma ini merupakan gabungan dari *Greedy Best-First Search* dan *Uniform Cost Search*. Penentuan rute merupakan permasalahan yang sering menucul dalam permainan-permainan berbasis strategi seperti seri Total War ataupun seri Civilization. Pada permainan Total War: Shogun 2 mencari rute tercepat diperlukan agar pasukan pemain lebih dulu menguasai kota musuh sebelum pasukan musuh tiba. Namun pencarian rute tercepat bukan satu-satunya faktor agar pasukan pemain bisa cepat tiba. Masih ada faktor lainnya seperti rute yang terhalangi oleh kota yang tidak bisa dilewati, pasukan pemain yang disabotase oleh agen musuh dan masih banyak faktor lainnya. Pada makalah ini, algoritma A* sangat berguna untuk pemain yang ingin menentukan rute yang harus dilewati oleh pasukan pemain.

VI. SARAN

Algoritma A* ini dapat dimofikasi dengan menghitung beberapa faktor lainnya seperti jalur mana yang bisa dilewati atau tidak, apakah di jalur tersebut ada agen musuh, dan apakah di jalur tersebut sudah terdapat pasukan musuh. Penambahan faktor-faktor diatas tentu perlu percobaan dan penggunaan fungsi heuristik yang sesuai agar algoritma A* bisa efektif.

VIDEO LINK AT YOUTUBE

https://youtu.be/fkB0vhXxcx4

UCAPAN TERIMA KASIH

Penulis mengucapkan puji syukur dan rasa terima kasih kepada Allah SWT atas rahmat dan pertolongan-Nya, penulis dapat menyelesaikan makalah ini.

Ucapan terima kasih penulis panjatkan kepada orang tua penulis karena telah mendukung penulisan makalah ini. Terakhir penulis ucapkan terima kasih kepada Bapak Rinaldi Munir sebagai dosen pengajar mata kuliah IF2211 Strategi Algirtma K3 yang telah menyediakan waktu dan tenaga beliau untuk mengajarkan ilmu Strategi Algoritma. Semoga ilmu yang diajarkan beliau berguna di masa depan. Penulis berharap makalah ini dapat menjadi pembelajaran bagi pembaca dan dimanfaatkan dengan baik.

Penulis juga ingin mengucapkan terima kasih kepada *developer* permainan Total War: Shogun 2 yang telah menginspirasi penulis dalam menulis makalah ini.

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 20 Mei 2022, pukul 20.20.

[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 20 Mei 2022, pukul 20.30.

[3] Munir, Rinaldi. Breadth First Search (BFS) dan Depth First Search (DFS) Bagian 1. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf. Diakses pada 20 Mei 2022, pukul 20.00.

[4] Munir, Rinaldi. Breadth First Search (BFS) dan Depth First Search (DFS) Bagian 2. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf. Diakses pada 20 Mei 2022, pukul 20.10.

[5] Creative Assembly. Total War: Shogun 2 Wiki. https://totalwar.fandom.com/wiki/Total_War:_Shogun_2. Diakses pada 20 Mei 2022, pukul 20.05

[6] Google. Google Map. https://www.google.co.id/maps/. Diakses pada 21 Mei 2022, pukul

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, 20 Mei 2022

Fitrah Ramadhani Nugroho / 13520030