Makalah Strategi Algoritma IF2211

Penerapan algoritma pencarian rute dalam pergerakan otomatis entitas *tile-based game* terhadap target dinamis.

Leonardus James Wang 13519189
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13519189@std.stei.itb.ac.id

***Abstract*—Makalah ini membahas penggunaan algoritma pencarian rute A*, UCS dan Greedy Best First Search untuk melakukan pergerakan otomatis pada entitas game. Entitas hanya akan diberu tahu suatu tujuan yang dapat bergerak, dan harus bergerak mencarinya. Konklusi yang dicapai adalah algoritma A* adalah algoritma yang paling efisien untuk semua kasus, algoritma UCS tidak efisien dan algoritma Greedy Best First Search dapat gagal dalam mencari tujuan.**

Keywords— tile; simpul; A*; Greedy Best First Search; UCS

I. PENDAHULUAN

Video game adalah satu jenis hiburan yang banyak digunakan orang-orang pada masa kini. Bahkan, industri game merupakan salah satu industri hiburan yang paling banyak menghasilkan uang[1]. Terdapat banyak jenis dan genre game yang berada dalam pasar sekarang. Dari genre atau jenis game yang ada, terdapat juga banyak cara memindahkan suatu entitas.

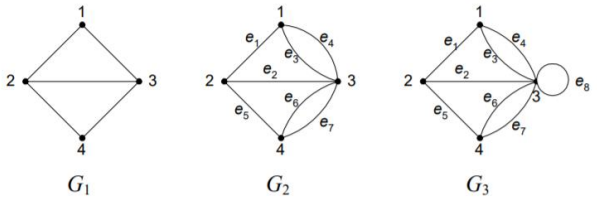
Pada banyak genre RTS (real-time strategy) dan beberapa game dalam genre RPG (role-playing game), biasanya yang memiliki view top down atau isometric, pergerakan entitas dilakukan secara otomatis. Kita cukup memberi tahu entitas untuk bergerak pada suatu tujuan dan entitas akan bergerak sendiri . Pada game RTS, entitas yang dimaksud merupakan unit yang dibuat atau dimiliki oleh pemain. Tujuan unit dapat berupa map, sehingga tujuan unit statis, atau dapat berupa unit lawan, yang dinamis karena unit lawan dapat bergerak-gerak.Pada game RPG, entitas yang dimaksud merupakan karakter kita sendiri, dan tujuannya dapat berupa map (statik) atau monster, yang dapat bergerak.

Pada makalah ini, akan dibahas implementasi pergerakan entitas otomatis yang dinamis, artinya tujuan dapat bergerak-gerak, pada tile based game, artinya game berdasarkan tile dan setiap unit hanya dapat menempati satu tile. Akan juga dilihat apabila ada perbedaan bila algoritma pencarian rute yang digunakan menghasilkan hasil yang berbeda. Algoritma pencarian rute yang digunakan merupakan algoritma A*, Greedy Best First Search dan Uniform Cost Search (disingkat UCS).

II. LANDASAN TEORI

A. *Graf*

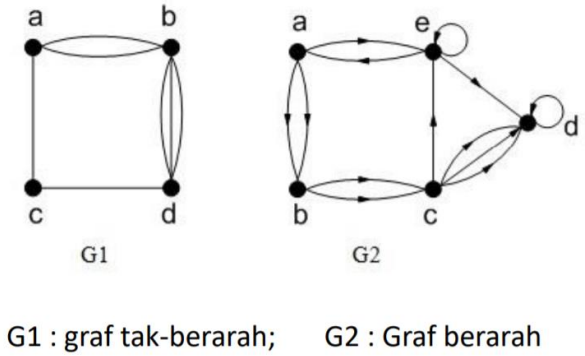
Graf adalah himpunan tidak kosong dari simpul-simpul dan himpunan sisi yang menghubungkan sepasang simpul[2]. Graf dapat diklasifikasian dengan beberapa cara. Yang pertama adalah berdasarkan ada tidaknya gelang atau sisi ganda. Bila tidak ada keduanya, dinamakan graf sederhana. Bila terdapat gelang pada graf, yaitu sebuah sisi yang berasal dan menuju simpul yang sama, graf tersebut dinamakan graf semu. Bila terdapat sisi lebih dari satu sisi yang sama pada graf, graf tersebut dinamakan graf ganda.



Gambar 2. (a) graf sederhana, (b) graf ganda, dan (c) graf semu

Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

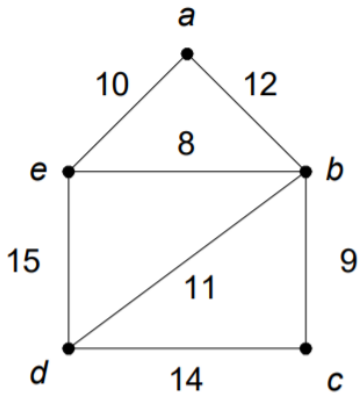
Graf dapat dipisah lagi berdasarkan orientasi arah. Graf yang memiliki arah dinamakan graf berarah, dan graf yang tidak memiliki arah dinamakan graf tak-berarah.



G1 : graf tak-berarah; G2 : Graf berarah

Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

Terdapat beberapa terminologi graf yang diketahui pada makalah ini. Dua simpul dikatakan bertetangga jika keduanya terhubung dengan langsung. Sebuah sisi dikatakan dengan simpul x bila salah satu akhir dari sisi tersebut merupakan simpul x. Graf berbobot berarti sisi pada graf memilki harga atau bobot.



Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

B. *Algoritma A**

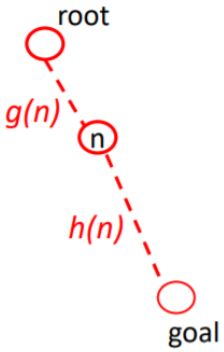
Algoritma A* merupakan algoritma untuk mencari rute dengan harga bobot paling kecil antara 2 simpul. Algoritma menerima satu graf dan 2 buah simpul. Graf haruslah berbobot dan biasanya tidak berarah.

Cara kerja algoritma A* adalaha dengan memasukkan simpul pertama ke dalam sebuah queue, lalu mengekspansi simpul dengan nilai evaluasi terendah sampai menemukan simpul tujuan dengan nilai evaluasi terendah. Ekspansi di sini artinya adalah melakukan pemrosesan semua simpul yang bertetanggan dengan simpul yang diekspansi, kecuali simpul yang sudah dilewatinya. Simpul yang terdapat pada queue dinamakan simpul hidup.

Pemrosesan yang dilakukan oleh algoritma A* adalah penghitungan nilai evaluasi dan dimasukkan ke dalam queue. Pada pemrosesan, simpul harus mengetahui jalur atau simpul mana saja yang dilewatinya, sehingga data tersebut juga dimasukkan kedalam queue. Jika simpul yang diekspansi merupakan simpul goal, maka semua simpul yang nilai evaluasi nya lebih besar dari nilai evaluasi simpul tersebut dihilangkan.

Nilai evaluasi merupakan nilai yang didapatkan dari fungsi evaluasi (disingkat f(n)). Fungsi evaluasi adalah fungsi yang dipakai untuk menerka bobot jalur yang digunakan untuk mencapai simpul goal. Nilai evaluasi akan selalu lebih kecil atau sama dengan nilau atau bobot jalur sesungguhnya. Pada algoritma A*, nilai evaluasi dihitung berdasarkan besarnya bobot jalur yang sudah dilewatinya sampai saat ini (disingkat g(n)) dan perkiraan jarak simpul ini sampai simpul goal (disingkat h(n)). Perkiraan h(n) merupakan fungsi heuristik, yang h(n) berarti pasti selalu lebih kecil atau sama dengan jalur sesungguhnya dari simpul ini sampai simpul goal. Dapat ditulis juga fungsi heuristik algoritma A* adalah f(n) = g(n) + h(n)

Untuk lebih jelasnya, bisa lihat gambar berikut.



Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf

C. *Algoritma UCS*

Algoritma UCS, singkatan dari *Uniform Cost Search*, merupakan algoritma pencarian rute yang mirip dengan A*. Bedanya algoritma UCS dengan algoritma A* adalah bahwa algoritma UCS hanya memperhatikan harga jarak yang ditempuh pada evaluasi nya, atau dengan kata lain f(n) = g(n). Algoritma UCS tetap akan memproses dan menyimpan semua simpul hidupnya.

D. *Algoritma Greedy Best First Search*

Algoritma *Greedy Best First Search*, merupakan algoritma pencarian rute terakhir yang akan dibahas. Algoritma ini menggunakakan fungsi evaluasi f(n) = h(n), yaitu hanya fungsi heuristiknya perkiraan simpul sekarang sampai simpul goal. Perbedaan yang mencolok algoritma ini dengan algoritma A* dan algoritma UCS adalah algoritma ini tidak menyimpan simpul-simpul hidupnya.

Proses *Greedy Best First Search* adalah melakukan ekspansi simpul pertama, lalu setelah memproses simpul-simpul yang bertetanggan dengan simpul yang diekspan, pilih simpul yang memiliki nilai evaluasi terendah, dan lupakan semua simpul hidup lainnya.

Akibat dari proses yang unik ini, simpul yang diproses akan lebih sedikit, tetapi ada kemungkinan simpul goal tidak tercapai jika ternyata tidak ada simpul yang membuat nilai heuristiknya lebih rendah. Algoritma *Greedy Best First Search* tidak dapat melakukan backtracking, sehingga jika simpul tidak dapat diekspan, maka simpul akan mati.

E. *Tile Based Game*

Seperti dinamanya, *tile based game* menggunakan *tile* untuk penempatan entitas-entitas pada game. Semua entitas hanya dapat menempati tile untik menentukan posisinya, dan hanya bisa bergerak berdasarkan *tile*.

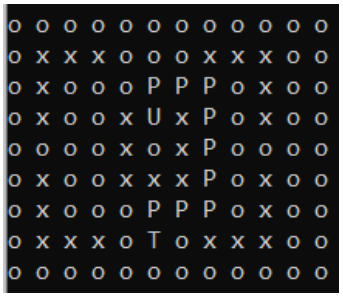
Source : https://www.pcgamesn.com/advance-wars-pc

Gambar di atas merupakan game dari advanced wars, salah satu game yang menggunakan *tile*. Bisa dilihat bahwa setiap unit dan setiap gedung hanya dapat menempati satu tile. Pada game ini, sistem pergerakannya beda dengan sistem yang akan dibahas, tetapi yang ingin diperlihatkan adalah penggunaaan tile untuk menentukan posisi dan pergerakan entitas.

III. IMPLEMENTASI

A. *Game*

*Game (*atau simulasi *game*) yang akan digunakan memiliki tampilan seperti berikut.



Gambar 1: Contoh tampilan

Terlihat ada beberapa simbol, berikut penjelasannya:

- x merupakan sebuah tembok, tidak dapat ditempuh
- o merupakan sebuah jalan, dapat ditempuh oleh unit dan target.
- U adalah unit, yang akan bergerak ke posisi target. Unit akan bergerak sekali ke arah kiri, kanan, atas, atau bawah setiap *turn* berdasarkan jalur yang di dapat dari algoritma pencarian.
- T adalah unit, yang akan bergerak ke secara random ke kiri, kanan, atas, atau bawah setiap 2 *turn.*
- P menyatakan jalur yang ditemukan unit menggunakan algoritma pencarian rute.

B. *Reduksi game menjadi soal pencarian rute.*

Soal pencarian rute memerlukan beberapa komponen. Pertama adalah graf berbobot. Untuk membuat *game* menjadi graf berbobot, setiap *tile* pada *map* dianggap menjadi satu simpul dengan nama sesuai koordinatnya. Koordinat yang dipakai pada *game* ini adalah (x,y) dengan x = 0 di kolom paling kiri dan y = 0 di baris paling atas. Lalu untuk setiap simpul, terdapat sisi ke simpul di kiri, kanan, atas dan bawahnya jika *tile* tersebut merupakan jalan yang dapat ditempuh (bukan tembok).

Pada contoh di Gambar 1, *tile* (1,0) memiliki sisi ke *tile* (0,0) dan *tile* (2,0). Setiap sisi yang dibuat memiliki bobot 1.

Komponen kedua adalah fungsi heuristik yang dipakai oleh A* dan *Greedy Best First Search* untuk menerka jarak sebuah simpul lain. Untuk ini digunakan fungsi SLD (*Straight Line Distance*), sehingga untuk fungsi heuristik dapat dihitung dengan jarak euclidean antara 2 titik koordinat.

C. *Program*

Contoh menyiapkan/mengkonfigurasi program adalah sebagai berikut. Program dan semua cuplikan kode dalam bahasa python.

```
g = gamemap("map.txt")
u1 = unit(5,4,g.map, "astar")
t = target(5,7,g.map)
u1.showMap(t.pos)
```

Bisa dilihat terdapat 3 komponen penting pada program, yaitu *gamemap*, *unit* dan *target*. *Gamemap* dapat di buat dengan memasukkan sebuah string yang merupakan nama file dari map atau menerima 2 angka dipisah dengan koma yang menyatakan panjang dan lebar dari *gamemap*. Jika menggunakan 2 angka, maka akan dibuat *gamemap* dengan dimensi yang sesuai, dan isi semua tile pada *gamemap* adalah jalan yang dapat ditelusuri (karakter 'o'). Jika input menggunakan file eksternal, maka file harus mengikuti format berikut.

```
<Panjang> <Lebar>
<[Konfigurasi map]>
```

Contoh format file eksternal ("map.txt"):

```
12 9
o o o o o o o o o o o o
o x x x o o o x x x o o
o x o o o o o o o x o o
o x o o x o x o o x o o
o o o o x o x o o o o o
o x o o x x x o o x o o
o x o o o o o o o x o o
o x x x o o o x x x o o
o o o o o o o o o o o o
```

Unit lalu dapat dibuat dengan parameter x, y, *gamemap* yang telah dibuat, dan pemilihan cara penentuan rute terdekat (opsional, jika tidak diinput maka akan didefault ke astar). *Unit* memiliki metode move, yang menerima parameter *tuple* dari 2 angka yang menyatakan posisi *target*, tetapi *unit* hanya dapat

bergerak satu *tile* ke kiri, atas, kanan, atau bawah tiap kali metode ini dipanggil.

Unit juga dapat "mati". Unit akan mati bila sudah menemukan target atau tidak menemukan jalur ke target. Unit juga menghitung banyak pemrosesan metode dan banyak move yang dilakukan untuk sampai ke target, dan ini yang akan digunakan untuk melakukan pengujian.

Target dibuat dengan parameter x dan y target yang menyatakan koordinatnya, lalu *gamemap*. Target memiliki metode move yang akan membuat target bergerak satu *tile* secara acak.

Bagian *main* dari program terlihat seperti berikut:

```
targetMove = 2
userInput = input()
while(userInput != 0 and u1.alive):
        path = u1.move(t.pos)

        targetMove = targetMove + 1
        if (targetMove >= 2):
                targetMove = 0
                t.move()
        print()
        sleep(0.5)

print(u1.processedNodesCount)
print(u1.moveCount)
print("Rata-rata node yang di process
per move:")
print(u1.processedNodesCount/u1.moveCoun
t)
```

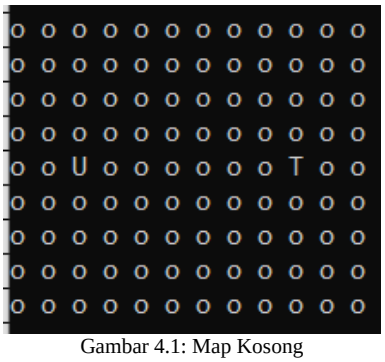
Bagian untuk mengatur seberapa sering target akan bergerak terletak pada targetMove, sehingga jika pembaca ingin mencoba dan mengatur dan menguji menggunakan program ini, dapat mengubah bagian main dan konfigurasinya. Program dapat diilihat di referensi [4]

IV. Pengujian dan Analisis

Pengujian dilakukan dengan cara menentukan suatu state map dan diperlihatkan mana yang lebih efisien berdasarkan rata-rata simpul yang diproses setiap gerakannya. Akan ada 2 jenis map yang akan diuji dan 2 map tersebut akan digunakan 3 metode pencarian yang berbeda, masing-masing akan diuji 10 kali agar dapat dilihat dari tabel.

A. Map 1: Map Kosong

Map yang digunakan adalah map 12 x 9, tanpa tembok, dengan unit pada koordinat (2,4) dan target pada koordinat (9,4).



Gambar 4.1: Map Kosong

Kode konfigurasi yang digunakan sebagai berikut:

```
g = gamemap(12,9)
u1 = unit(2,4,g.map, "astar")
#astar dapat diganti dengan "greedy"
#atau "UCS"
t = target(9,4,g.map)
u1.showMap(t.pos)
```

Berikut merupakan tabel saat unit menggunakan algoritma pencarian rute A*, *Greedy Best First Search*, dan UCS.

No Percobaan	Jumlah simpul yang diproses	Move yang dilakukan	Rata-rata simpul yang diproses per move
1	529	7	75.57
2	431	8	53.88
3	459	11	41.73
4	646	9	71.78
5	264	9	29.33
6	248	8	31
7	62	5	12.4
8	354	9	39.33
9	312	6	52
10	267	9	29.6
Rata-rata	357.2	8.1	43.66

Tabel 4.1.1: A* Map Kosong

No Percobaan	Jumlah simpul yang diproses	Move yang dilakukan	Rata-rata simpul yang diproses per move
1	720	9	80
2	652	9	72.4
3	649	9	72.1
4	468	6	78
5	605	11	55
6	583	9	64.7
7	625	9	69.4
8	564	6	94
9	607	11	55.18
10	481	6	80.17
Rata-rata	595.4	8.5	72.1

Tabel 4.1.2: *Greedy Best First Search* Map Kosong

No Percobaan	Jumlah simpul yang diproses	Move yang dilakukan	Rata-rata simpul yang diproses per move
1	49245	9	5471.67
2	19717	7	2816.71
3	40276	11	3661.45
4	19901	9	2211.22
5	39250	6	6541.67
6	39142	6	6523.67
7	48857	7	6979.57
8	39183	6	6530.5
9	52632	9	5848
10	39250	6	6541.67
Rata-rata	38745.3	7.6	5312.61

Tabel 4.1.3: UCS Map Kosong

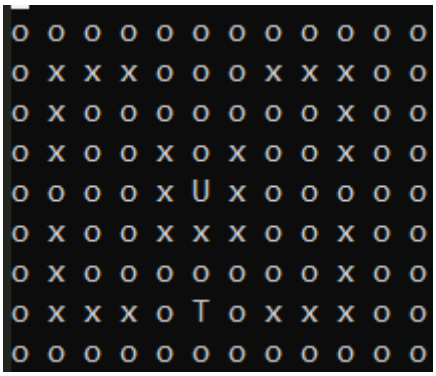
Pada konfigurasi map ini, tidak ada halangan atau tembok apapun, dan map sangat luas. Saat melakukan pengujian, semua algoritma sukses memberikan jalur ke unit sehingga unit tidak mati secara prematur (mati sebelum menemukan target).

Dari data bisa dilihat bahwa penggunaan algoritma A* merupakan algoritma yang paling efisien. Algoritma selanjutnya yang juga efisien adalah algoritma *Greedy Best First Search*, dan tidak begitu beda jauh efisiensinya dibandingkan dengan A*. Tetapi algorimta UCS sangat tidak efisien untuk melakukan pencarian rute, sampe 100x lipat lebih lambat dari A*. Saat pengujian, juga dapat dirasakan waktu yang lama untuk melakukan pemrosesan.

Alasan algoritma UCS sangat lama dibandingkan algoritma A* maupun *Greedy Best First Search* adalah karena algoritma A* dan *Greedy Best First Search* "mengetahui" kira-kira target dimana dan tile yang harus diekspansi agar unit jalur ekspansi menuju target. UCS tidak mengetahui target dimana, UCS hanya akan memproses simpul yang memiliki bobot terendah saat ini berdasarkan jarak yang sudah ditempuhnya. Akibatnya UCS akan mengkespansi ke kiri, atas, kanan, dan bawah dari unit, lalu mengkespansi semua simpul yang dihidupkan oleh 4 tile terdekat unit tadi sampai suatu saat secara tidak sengaja menemukan posisi target. Karena UCS tidak mengetahui target ada dimana, UCS termasuk algoritma *blind search*, yaitu melakukan search tanpa informasi tambahan dari luar.

B. Map 2: Map U

Map yang digunakan masih 12x9, tetapi memilki beberapa tembok dan tembok khusus ditengah yang berbentuk U. Akan ditaruh Unit ditengah U dan target dibawahnya, dengan tujuan agar Unit harus bergerak keluar ke atas dari U baru menuju tareget.



Gambar 4.2: Map U

Kode konfigurasi yang digunakan sebagai berikut:

```
g = gamemap("map.txt")
u1 = unit(5,4,g.map, "astar")
#astar dapat diganti dengan "greedy"
#atau "UCS"
t = target(5,7,g.map)
u1.showMap(t.pos)
```

Kode membaca file eksternal map.txt. map.txt sendiri memiliki isi sebagai berikut:

```
12 9
o o o o o o o o o o o o
o x x x o o o x x x o o
o x o o o o o o o x o o
o x o o x o x o o x o o
o o o o x o x o o o o o
o x o o x x x o o x o o
o x o o o o o o o x o o
o x x x o o o x x x o o
o o o o o o o o o o o o
```

Berikut merupakan tabel saat unit menggunakan algoritma pencarian rute A*, *Greedy Best First Search*, dan UCS.

No Percobaan	Jumlah simpul yang diproses	Move yang dilakukan	Rata-rata simpul yang diproses per move
1	633	13	48.69
2	366	9	40.67
3	416	11	37.82
4	406	9	45.11
5	458	15	30.53
6	467	11	42.45
7	841	13	64.69
8	1281	13	98.53
9	416	11	37.82
10	851	11	77.36
Rata-rata	613.5	11.6	52.37

Tabel 4.2.1: A* Map U

No Percobaan	Jumlah simpul yang diproses	Move yang dilakukan	Rata-rata simpul yang diproses per move
1*	35	1	35
2*	35	1	35
3*	35	1	35
4*	35	1	35
5*	35	1	35
6*	35	1	35
7*	35	1	35
8*	35	1	35
9*	35	1	35
10*	35	1	35
Rata-rata	35	1	35

Tabel 4.2.2: *Greedy Best First Search* Map U

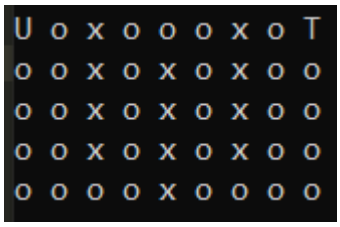
No Percobaan	Jumlah simpul yang diproses	Move yang dilakukan	Rata-rata simpul yang diproses per move
1	4967	13	382.08
2	7575	11	688.64
3	11722	13	901.69
4	11962	13	920.15
5	4834	9	537.11
6	4589	9	509.89
7	4789	11	435.36
8	2844	7	406.28
9	9948	11	904.36
10	7471	11	679.18
Rata-rata	7070.1	10.8	636.47

Tabel 4.2.3: UCS Map U

Dapat dilihat bahwa algoritma yang paling efisien adalah A*. Walaupun terlihat bahwa mungkin *Greedy Best First Search* lebih efisien, semua pencarian greedy best first search gagal pada map ini, disimbolkan dengan * di sebelah nomor. Algoritma UCS menemukan, tetapi tidak efisien, 10 kali lebih lambat jika dibanding A*.

C. Map 3: Map berbelok

Map yang digunakan berdimensi 9 x 5. Map memilki 3 tembok besar tetapi tidak ada yang benar-benar memaksa unit untuk balik.



Gambar 4.3.1: Map Berbelok

Kode konfigurasi yang digunakan sebagai berikut:

```
g = gamemap("map2.txt")
```

```
u1 = unit(0,0,g.map, "astar")
#astar dapat diganti dengan "greedy"
#atau "UCS"
t = target(8,0,g.map)
u1.showMap(t.pos)
```

Kode membaca file eksternal map2.txt. map2.txt sendiri memiliki isi sebagai berikut:

```
9 5
o o x o o o x o o
o o x o x o x o o
o o x o x o x o o
o o x o x o x o o
o o o o x o o o o
```

Berikut merupakan tabel saat unit menggunakan algoritma pencarian rute A*, *Greedy Best First Search*, dan UCS.

No Percobaan	Jumlah simpul yang diproses	Move yang dilakukan	Rata-rata simpul yang diproses per move
1	1932	20	96.6
2	2076	23	90.26
3	1986	21	94.57
4	1808	20	90.4
5	2062	23	89.65
6	2155	23	93.70
7	2023	23	87.96
8	1926	23	83.74
9	1905	21	90.71
10	1888	21	89.90
Rata-rata	1976.1	21.8	90.75

Tabel 4.3.1:A* Map Berbelok

No Percobaan	Jumlah simpul yang diproses	Move yang dilakukan	Rata-rata simpul yang diproses per move
1*	789	23	34.30
2*	861	25	34.44
3*	1843	52	35.44
4	688	27	25.48
5*	1009	29	34.79
6*	1049	30	34.97
7*	1121	32	35.03125
8*	3656	106	34.49
9*	1241	36	34.47
10*	701	23	30.48
Rata-rata	1295.8	38.3	33.38913

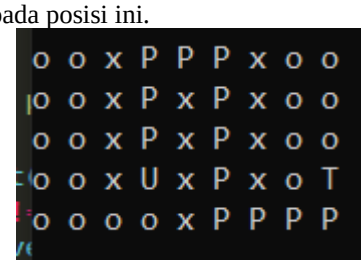
Tabel 4.3.2: *Greedy Best First Search* Map Berbelok

No Percobaan	Jumlah simpul yang diproses	Move yang dilakukan	Rata-rata simpul yang

			diproses per move
1	3454	19	181.80
2	3673	19	193.32
3	3750	21	178.57
4	3724	23	161.91
5	3400	21	161.90
6	3637	23	158.13
7	4030	23	175.21
8	4052	23	176.17
9	3678	20	183.9
10	4125	23	179.35
Rata-rata	3752.3	21.5	175.03

Tabel 4.3.3:UCS Map Berbelok

Dapat dilihat bahwa pada map ini, yang banyak halangan antara Unit dan Target, algoritma UCS sebenarnya tidak sejauh efisiensinya dnegan algoritma A*, hanya 2 kali lebih lambat. Algorimta A* tetaplah yang paling efisien, dan algoritma *Greedy Best First Search* memiliki banyak kegagalan. Untuk informasi lebih lanjut, algoritma *Greedy Best First Search* sering gagal pada posisi ini.



Gambar 4.3.2: Posisi terakhir sebelum unit mati pada algoritma greedy

Perlu diperhatikan juga bahwa move yang dilakukan algoritma *Greedy Best First Search* lebih banyak dibandingkan UCS atau algoirmta A*. Ini mungkin dikarenakan algoritma *Greedy Best First Search* membuat unit bolak balik pada tempatnya karena target dan unit pas ditengah, sehingga akan selalu ekspan node yang di sebelah unit, tetapi akan kembali lagi menurut heuristiknya.

V. KESIMPULAN

Dari pengujian yang telah dilakukan, dapat dikonklusikan bahwa algoritma A* merupakan algoritma yang paling baik dipakai untuk melakukan pergerakan otomatis entitas pada *tile based* game untuk target dinamis. Algoritma A* merupakan algoritma yang paling efisien karena jumlah pemrosesan per movenya sedikit dan jumlah move yang digunakan untuk mencapai target wajar. Algoritma A* juga tidak mengalami kegagalan seperti algoritma *greedy best first search* saat dilakukan pengujian.

VIDEO LINK AT YOUTUBE *(Heading 5)*

https://youtu.be/xx8-S_7AfxM

ACKNOWLEDGMENT *(Heading 5)*

Pertama-tama, syukur dipanjatkan kepada Tuhan YME karena tanpa Dia makalah ini tidak dapat selesai. Lalu saya ingin berterima kasih kepada orang tua untuk selalu mendukungku dan Pak Rinaldi Munir yang sudah memberi tugas, pengajaran dan arahan untuk mengerjakan tugas ini.

REFERENCES

[1] https://www.statista.com/chart/22392/global-revenue-of-selected-entertainment-industry-sectors/#:~:text=Gaming%3A%20The%20Most%20Lucrative%20Entertainment%20Industry%20By%20Far,-Entertainment%20Industry, diakses pada 10 Mei 2021, pukul 17:36

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf, diakses pada 10 Mei 2021, pukul 19:52

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf, diakses pada 10 Mei 2021, pukul 20:15

[4] https://github.com/jamesclaws/Makalah-Stima, diupload pada tanggal 11 Mei 2021, pukul 20:41

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

Ttd

Leonardus James Wang 13519189