Aplikasi Graf dalam *Pathfinding* di *Video Games*

Eka Sunandika, 13517130
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517130@std.stei.itb.ac.id

***Abstract*—*Video games* menjadi hiburan yang digemari saat ini. Semakin canggih teknologi yang ada membuat perkembangannya menjadi sangat pesat dan memiliki banyak macamnya. Dalam pembuatannya, berbagai algoritma berdasarkan sebuah teori digunakan oleh *developer video games* dalam pendekatan dibeberapa aspeknya. Salah satunya adalah teori graf berbobot yang digunakan untuk menentukan jalan yang ditempuh dalam permainan tersebut.**

***Keywords*— *AI*, Algoritma, *Games*, Graf**

I. PENDAHULUAN

Menurut Clark C. Abt, *Game* adalah kegiatan yang melibatkan keputusan pemain, berupaya mencapai tujuan dengan "dibatasi oleh konteks tertentu" (misalnya, dibatasi oleh peraturan). (Serious Games.New York,:Viking Press, 1970). *Video games* saat ini sangat beragam macamnya. Banyak orang yang senang bermain *video games* sebagai rekreasi. Hiburan ini diminati oleh berbagai kalangan mulai dari yang muda hingga tua. Terdapat banyak *developer* yang selalu berusaha untuk menciptakan *games* sebaik mungkin untuk menghasilkan pengalaman bermain yang lebih menyenangkan.

Teknologi yang digunakan *video games* selalu berkembang dengan seiring zaman. Perangkat yang diperlukan dalam bermain *video games* semakin cepat, sehingga bisa menciptakan tampilan dan permainannya yang semakin realistis. Dalam suatu *video games*, pasti terdapat *player* atau *AI* yang dapat bergerak. Semua pergerakannya dilakukan berdasarkan beberapa hal.

Pathfinding sangat berguna dalam perkembangan suatu *video games.* Sesuai namanya, teori ini digunakan untuk menemukan jalan terpendek dari dua titik. Semua komponen yang bergerak di *video game* harus menentukan jalan yang akan dilaluinya dengan efisien. Terdapat berbagai kemungkinan jalan yang bisa dilewati pastinya, tetapi program harus bisa menentukan jalan yang memiliki jarak lebih dekat. Jalan yang diambil harus sesuai dengan ketentuan cara jalan yang benar, sehingga harus bisa menghindari objek yang mengganggu apabila ditemukan dalam menentukan jalan.

Teori permasalahan lintasan terpendek (*shortest path problem*) dalam graf memiliki kemiripan dengan *pathfinding*. Dengan menentukan jarak terpendek antara dua titik yang menggunakan biaya paling dikit. Terdapat berbagai algoritma yang digunakan dalam menyelesaikan permasalahan tersebut. Algoritma yang baik terus dicari dan dikembangkan untuk memperoleh hasil yang lebih baik pula.

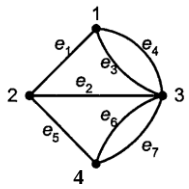
II. LANDASAN TEORI

A. Graf

a. Definisi Graf

Secara matematis, dapat didefinisikan bahwa suatu graf G merupakan pasangan himpunan (*V, E)*, dapat ditulis dengan notasi G = (V,E). *V* adalah simbol dari himpunan simpul (*vertices* atau *node*) yang tidak boleh kosong dan E adalah simbol dari himpunan sisi (*edgesI atau* arcs) yang boleh kosong dan menghubungkan sepasang simpul.

Simpul pada graf dapat diberi nama dengan huruf (a, b, c, d, …), angka (1, 2, 3, …), atau gabungan keduanya. Simpul *u* dan simpul *v* yang dihubungkan dengan suatu sisi (*u, v*) dapat dinyatakan dengan lambang e_1, e_2, ….



Gambar 1: Ilustrasi graf (Sumber: Munir, Rinaldi. 2006. Diktat Kuliah IF2120 Matematika Diskrit. Bandung : Institut Teknolgi Bandung)

Gambar diatas merupakan contoh graf yang miliki himpunan simpul sebagai berikut :

$V = \{1, 2, 3, 4\}$

$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$

$= \{(1, 2), (2, 3), (1, 3), (1, 3), (2, 4), (3, 4), (3, 4)\}$

b. Jenis Graf

Berdasarkan ada tidaknya sisi ganda atau kalang :

1. Graf Sederhana (*simple graph*)
   Merupakan graf yang tidak mengandung gelang maupun sisi-ganda. Sisi (*u, v*) dapat dulis juga dengan (*v, u*).
2. Graf tak-sederhana (*unsimple-graph*)
   Merupakan graf yang mengandung sisi ganda atau gelang. Memiliki dua macam, yaitu graf ganda (*multigraph*) yang mengandung sisi ganda dan graf semu (*pseudograph*) yang mengandung gelang (*loop*).

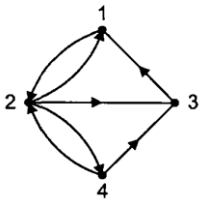
Berdasarkan orientasi arah :

1. Graf tak-berarah (*undirected graph*)
   Merupakan graf yang sisinya tidak memiliki orientasi arah. Pasangan simpul yang dihubungkan sisi tidak

diperhatikan urutannya, maka $(u, v) = (v, u)$.

2. Graf berarah (*directed graph* atau *digraph*)

   Merupakan graf yang setiap sisinya diberikan orientasi arah. Pada busur (u , v), simpul u adalah simpul asal dan simpul v adalah simpul terminal.



Gambar 2: Ilustrasi graf berarah (Sumber: Munir, Rinaldi. 2006. Diktat Kuliah IF2120 Matematika Diskrit. Bandung: Institut Teknolgi Bandung)

c. Terminologi Dasar

1. Bertetangga (*Adjacent*)

   Dua buah simpul pada graf tak-berarah yang terhubung oleh sebuah sisi dikatakan bertetangga. Simpul u dan simpul v dikatakan bertetangga jika terdapat sisi (u, v) pada suatu graf.

2. Bersisian (*Incident*)

   Sisi e dikatakan bersisian dengan simpul u dan simpul v jika terdapat sisi $e = (u, v)$ pada suatu graf.

3. Simpul Terpencil (*Isolated Vertex*)

   Simpul yang tidak mempunyai sisi yang bersisian dengannya dan tidak bertetangga dengan simpul lainnya disebut dengan simpul terpencil.

4. Graf Kosong (*Null Graph*)

   Graf kosong merupakan graf yang himpunan sisinya adalah himpunan kosong.

5. Degree (*Degree*)

   Jumlah sisi yang bersisian dengan suatu simpul disebut sebagai derajat simpul tersebut. Memiliki notasi $d(v)$. Pada graf berarah, derajat simpul v dinyatakan dengan $d_{in}(v)$ yang merupakan derajat masuk ke simpul v dan $d_{out}(v)$ yang merupakan derajat keluar dari simpul v.

6. Lintasan

   Lintasan memiliki definisi apabila sebuah lintasan yang panjangnya n dari simpul awal v_0 ke simpul tujuan v_n di dalam graf G adalah barisan berselang – seling simpul – simpul dan sisi – sisi yang memiliki bentuk v_0, e_1, v_1, e_2, v_3, … , v_{n-1}, e_m, v_n sedemikian sehingga $e_1 = (v_0, v_1)$, $e_2 = (v_1, v_2)$,… , $e_n = (v_{n-1}, v_n)$ adalah sisi – sisi dari graf G. Apabila graf tersebut adalah graf sederhana, maka dalam menyatakan lintasannya cukup dengan menuliskan lintasan sebagai barisan simpul-simpulnya. Pada graf ganda, penulisan lintasan tersebut berupa simpul dan sisi dengan berselang – seling. Lintasan yang berawal dan berakhiran di simpul yang sama disebut lintasan tertutup (*closed path*), sedangkan yang tidak seperti itu disebut lintasan terbuka (*open path*).

7. Siklus atau sirkuit (*Cycle* atau *Cirkuit*)

   Lintasan tertutup atau yang memiliki simpul awal dan akhir yang sama disebut sebagai siklus atau sirkuit. Sirkuit sederhana (*simple circuit*) adalah sirkuit dengan setiap sisi yang dilaluinya berbeda.

8. Terhubung (*Connected*)

   Jika terdapat lintasari dari simpul u ke simpul v, maka kedua simpul tersebut dikatakan terhubung. Dua simpul pada graf berarah dikatakan terhubung kuat (*strongly connected*) apabila terdapat lintasan dari u ke v dan sebaliknya. Dua buah simpul yang tidak terhubung kuat tetapi terhubung pada graf tak-berarahnya dikatakan terhubung lemah (*weakly connected*).

9. Upagraf (*Subgraph*) dan Komplemen Upagraf

   Sebuah graf G_1 disebut upagraf dari G jika terdapat V_1 yang merupakan himpunan bagian dari V dan terdapat E_1 yang merupakan himpunan bagian dari E. Komplemen dari upagrag G_1 terhadap graf G adalah graf $G_2 = (V_2, E_2)$ sedemikian sehingga $E_2 = E – E_1$ dan V_2 adalah himpunan simpul yang anggota-anggota E_2 bersisian dengannya.

10. Upagraf Merentang (*Spanning Subgraph*)

    Upagraf G_1 dikatakan upagraf merentang jika pada G_1 terdapat semua simpul yang ada di graf G.

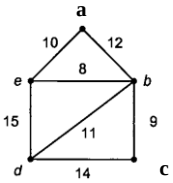
11. *Cut-set*

    Cut-set dari graf terhubung G adalah himpunan sisi yang bila dibuang dari G menyebabkan G tidak terhubung.

12. Graf berbobot (*weighted graph*)

    Apabila terdapat graf yang setiap sisinya diberi harga (bobot), maka graf tersebut disebut graf berbobot. Bobot yang terdapat pada sisi graf memiliki makna yang berbeda – beda bergantung pada masalah yang ada. Contoh – contoh nya adalah bobot yang menyatakan jarak antara dua kota, biaya perjalanan antara dua kota, ongkos produksi, dan jarak yang ditempuh player atau *AI* dalam suatu permainan,

    Pada *pathfinding*, graf yang digunakan adalah graf berbobot. Graf ini yang akan memodelkan jarak yang ingin ditempuh antara dua titik. Bobot pada sisi tersebut menyatakan jarak di peta *game* tersebut.



Gambar 3: Ilustrasi graf berbobot (Sumber: Munir, Rinaldi. 2006. Diktat Kuliah IF2120 Matematika Diskrit. Bandung: Institut Teknolgi Bandung)

B. Permasalahan Lintasan Terpendek (*Shortest Path Problem*)

Graf berbobot digunakan dalam permasalahan optimasi dalam mencari lintasan terpendek ini. Asumsi bahwa semua bobot bernilai positif digunakan dipermasalahan ini. Kata “terpendek” memiliki arti meminimisasi bobot pada suatu lintasan. Contoh penerapan pencarian lintasan terpendek ini bisa digunakan untuk mencari lintasan terpendek atau waktu tersingkat yang harus dilalui dari kota A ke kota B dan menentukan jarak antara dua buah terminal dalam suatu jaringan untuk menentukan jalur komunikasi terpendek agar waktu pengiriman pesan bisa lebih cepat dan menghemat biaya.

Terdapat banyak algoritma untuk mencari lintasan terpendek. Salah satu yang terkenal adalah algoritma Djikstra. Algoritma tersebut bisa digunakan untuk mencari lintasan terpendek graf berarah ataupun graf tak ber-arah.Algoritma Djikstra dapat digunakan dalam *pathfinding* di sebuah *game*

III. ALGORITMA UNTUK PATHFINDING

A. Algoritma Dijkstra

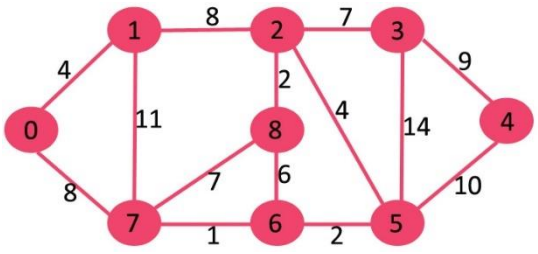
Edsger Dijkstra (1930-2002) adalah penemu dari algoritma ini. Dia lahir di belanda dan merupakan seorang peneliti sekaligus profesor University of Texas di Austin. Banyak kontribusi yang dia lakukan dalam perkembangan ilmu komputer didunia. Salah satunya adalah algoritma Djikstra yang dia buat dan sangat berguna di berbagai bidang. Algoritma Dijkstra diciptakan pada tahun 1959 oleh Dutch Computer Scientist Edsger Dijkstra, ketika dia masih bekerja pada Mathematical Center di Amsterdam.

Algoritma Greedy menyelesaikan masalah berdasarkan tindakan yang dilakukannya aoakah merupakan strategi yang baik untuk jangka panjang. Algoritma Dijkstra menggunakan pendekatan algoritma ini dalam menyelesaikan permasalahannya. Dengan memilih simpul yang belum dipilih dan terdekat ke simpul sumbernya. Kemudian jarak dari simpul itu ke simpul tetangganya diperbarui.

Algoritma Dijkstra digunakan untuk mencari lintasan terpendek dari suatu simpul ke simpul lain yang ada. Ide utama dalam algoritma ini yaitu terdapat simpul utama yang nantinya diperluas, hingga ditemukannya lintasan terpendek ke simpul tujuan. Bobot pada graf harus memiliki nilai positif.

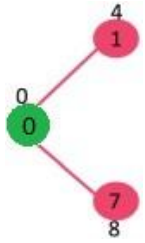
Berikut adalah contoh implementasi dari algoritma Dijkstra untuk menentukan jarak terpendek dari suatu simpul ke semua simpul yang ada pada graf.

1. Pertama buat himpunan sptSet (*shortest path tree set)* yang menyimpan jalur dari simpul yang terdapat pada shortest path tree yang ingin dibuat.
2. Jarak dari simpul awal ke simpul lainnya ditetapkan dengan nilai tak-hingga. Simpul utama diberi nilai jarak 0.
3. Ambil simpul yang tidak terdapat pada himpunan sptSet dan memiliki jarak yang minimum. Kemudian perbarui jaraknya dengan simpul tetangganya



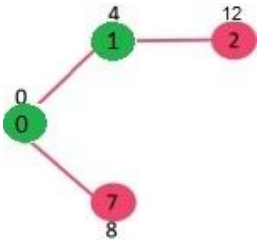
Gambar 4 (Sumber: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/)

Pada himpunan sptSet di inisialisasi masih kosong dan jarak simpul yang ada ditetapkan dengan {0, INF, INF, INF, INF, INF, INF, INF} dengan INF merupakan tak-hingga. Selanjutnya ambil simpul yang memiliki jarak minimum. Simpul 0 diambil dan dimasukan ke himpunan sptSet. Simpul 1 dan 7 yang merupakan simpul tetangganya diperbarui jaraknya menjadi 4 dan 8. Gambar dibawah adalah upagraf yang menampilkan simpul dan jaraknya yang tidak tak-hingga. Simpul yang terdapat pada SPT berwarna hijau.



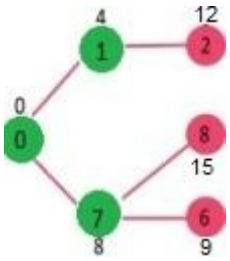
Gambar 5 (Sumber: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/)

Selanjutnya diambil simpul lain yang memiliki jarak minimum dan tidak terdapat pada himpunan sptSet. Simpul 1 diambil dan himpunan sptSet sekarang menjadi {0, 1}. Perbarui jarak simpul tetangga 1 sehingga jarak simpul 1 ke 2 menjadi 12.



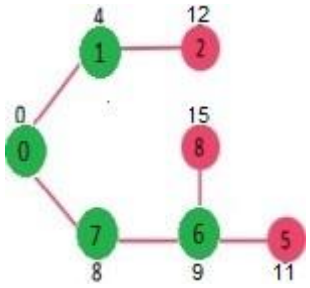
Gambar 6 (Sumber: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/)

Langkah selanjutnya adalah simpul 7 diambil karena memiliki jarak minimum dan belum masuk ke himpunan sptSet. Sekarang himpunan sptSet menjadi {0, 1, 7} dan jarak antara simpul 7 ke simpul 8 dan 9 menjadi 15 dan 9.



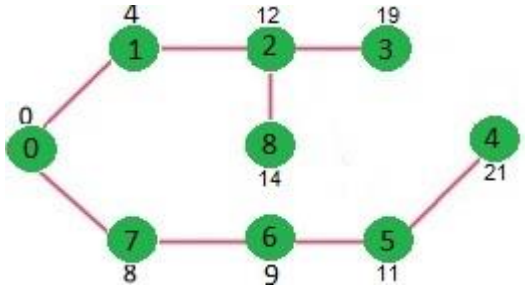
Gambar 7 (Sumber: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/)

Simpul 6 diambil dan dimasukan ke himpunan sptSet, sehingga sekarang menjadi {0, 1, 7 ,6}. Jarak antara simpul 6 ke simpul 5 dan 8 diperbarui menjadi 15 dan 11.



Gambar 8 (Sumber: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/)

Ulangi semua langkah tersebut sehingga didapat himpunan sptSet yang tidak mengandung semua graf sehingga diperoleh *shortest path tree* seperti dibawah ini.



Gambar 9 (Sumber: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/)

*B. Algoritma A**

Algoritma ini dapat digunakan untuk menyelesaikan berbagai permasalahan, *pathfinding* adalah salah satunya. Dalam *pathfinding*, algoritma A* akan memeriksa dari simpul awal yang belum diperiksa dan akan berhenti jika simpul yang dituju sudah diperiksa. Algoritma ini paling populer untuk digunakan dalam *AI* suatu *game*.

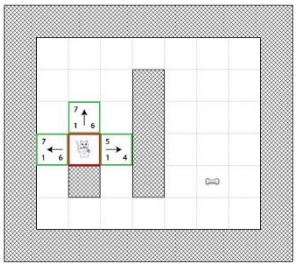
Terdapat terminologi dalam algoritma ini, $g(n)$ merepresentasikan biaya persis yang diperlukan suatu lintasan dari simpul n ke semua simpul, dan $h(n)$ yang merepresentasikan biaya heuristic dari simpul n ke simpul tujuan.. Heuristic dapat digunakan untuk mengontrol algoritma A*. Jika $h(n) = 0$, maka hanya $g(n)$ yang mempengaruhi algoritma. Semakin kecil nilai dari $h(n)$, semakin banyak simpul yang diperluas dan menyebabkan algoritma jadi melambat. Apabila $h(n)$ memiliki besar yang sama dengan biaya yang diperlukan simpul n ke tujuan, maka algoritma akan berjalan dengan cepat dan memiliki jalur terbaik. Jika $h(n)$ memiliki nilai lebih besar dari biaya yang diperlukan dari simpul n ke tujuan, ada kemungkinan jalur minimum tidak dapat ditemukan. Kemampuan algortima ini bisa bervariasi dan berguna dalam *game*.

Misalkan dalam suatu *game* terdapat 2 tipe daerah, dataran dan pegunungan. Biaya yang diperlukan jika melalui pegunungan lebih besar dibanding jika melalui dataran, Ada kemungkinan jalan yang dilalui melalui dataran lebih efektif tetapi harus memutar pegunungan. Maka kemungkinan itu akan dipilih walaupun mungkin memerlukan waktu dalam memproses lebih lama.

Algoritma A* akan memproses $f(n) = g(n) + h(n)$. Dua nilai tersebut harus memiliki skala yang sama untuk dihitung. Jika skalanya berbeda, maka algoritma ini akan berjalan lambat dan tidak akan memperoleh lintasan yang baik. Jika nilai heuristic memiliki nilai yang sama dengan jarak pada lintasan yang optimal, maka algoritma A* hanya memperluas beberapa simpul saja.

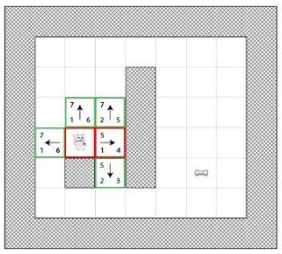
Berikut adalah contoh bagaimana algoritma A* bekerja dengan mencari jalan yang harus dilalui untuk mencapai tujuan. Tiap simpul direpresentasikan dengan kotak. Terdapat daftar yang berisi semua kotak kemungkinan untuk mendapat lintasan terpendek yang kita sebut open list dan daftar yang berisi kotak yang tidak perlu dipertimbangkan lagi yang kita sebut closed list. Dalam contoh dibawah ini, di pojok kiri atas berisi nilai suatu kotak (f), di pojok kiri bawah berisi biaya yang diperlukan dari A ke kotak (g), dan di kanan bawah berisi perkiraan biaya yang dibutuhkan dari kotak ke B (h).

Pertama, tentukan kemungkinan jalan dari simpul tetangganya ke posisi awal, hitung nilai f dan tambahkan ke open list.



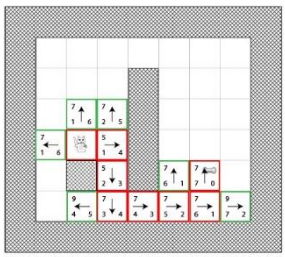
Gambar 10 (Sumber: https://www.raywenderlich.com/3016-introduction-to-a-pathfinding

Selanjutnya, pilih nilai f terkecil dan masukan ke closed list dan perbarui simpul tetangganya.



Gambar 11 (Sumber: https://www.raywenderlich.com/3016-introduction-to-a-pathfinding

Kotak yang memiliki biaya terkecil adalah yang *f* nya bernilai 4. Dua kotak yang ditambah ke open list, nilai *g* nya meningkat karena dua kotak tersebut berjarak 2 kotak dari titik awal. Ulangi langkah – langkah sebelumnya hingga titik tujuan terdapat di open list dan bisa dimasukan ke closed list.



Gambar 12 (Sumber: https://www.raywenderlich.com/3016-introduction-to-a-pathfinding

IV. PATHFINDING DALAM VIDEO GAMES

Algoritma yang populer digunakan dalam industri *game* adalah algoritma A*. Implementasi algoritma ini mudah dilakukan dan memiliki berbagai variasi yang selalu dikembangkan untuk efektivitas jalan yang diambil serta waktu pengolahan yang lebih cepat.

Dalam Age of Empires yang merupakan *classic real-time strategy game* menggunakan *grid* (kisi) dalam merepresentasikan peta lokasinya. Pada 256*256 *grid* memiliki 65.536 kemungkinan lokasi. Pergerakan yang dilakukan *military unit* seperti objek yang berjalan di labirin. Terkadang jalan yang diambil memiliki masalah. Contohnya ketika *unit* tersebut berjalan melawati hutan untuk mencapai posisi yang dituju beberapa dari mereka bisa terjebak dihutan. Tertutama apabila hutan tersebut memiliki kepadatan yang tinggi.

Gambar 10 (Sumber: Microsoft Games, “Microsoft Age of Empires”, http://www.microsoft.com/games/empires)

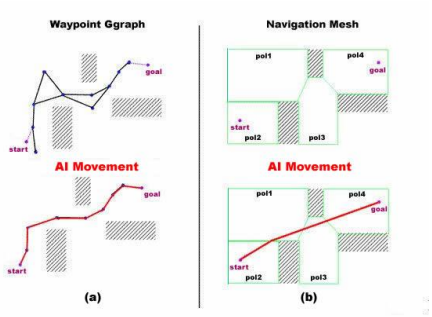
Pada *strategy game* Civilization V menggunakan hexagonal dalam merepresentasikan peta lokasinya. Algoritma untuk *pathfinding* digunakan untuk mengontrol jalan *military unit* ke lokasi yang diinginkan berdasarkan grup hexagonal yang bisa dilalui. *Pathfinding* dalam Civilization V memiliki kemiripan dengan Age of Empires. Walaupun dalam teori terlihat sempurna, ternyata masih memiliki kekurangan.

Gambar 10 (Sumber: Firaxis Games, “Sid Meier’s Civilization V”, http://www.civilization5.com)

Jika dibandingkan, algoritma A* yang digunakan di *strategy game* dengan di *fps game* memiliki perbedaan. Pada *fps game* algoritma A* bekerja lebih baik dalam *pathfinding*. Contohnya pada *game* Counter-Strike yang hanya menggerakan beberapa *unit* saja pada waktu yang sama. Penjelasan kenapa hal tersebut bisa terjadi mungkin dikarenakan ketika menggerakan *unit* yang banyak, maka diperlukan memori yang lebih besar dalam memprosesnya.

Gambar 13 (Sumber: https://www.ign.com/articles/2012/02/03/the-evolution-of-counter-strikes-dedust)

Kebanyakan *game engines* mendukung *pathfinding* dalam graf berarah. Dalam *2D games* mungkin penerapannya mudah, berbeda dengan *3D games* yang lebih sulit dan memiliki lingkungan yang kompleks. Teknik baru dalam *pathfinding* dengan nama Navigation Mesh (NavMesh) dibuat untuk mempermudahnya. NavMesh hanya membutuhkan beberapa poligon untuk merepresentasikan peta. Dengan data yang diperiksa lebih dikit, hasil yang diharapkan juga menjadi semakin cepat. Poligon yang ada merepresentasikan kemungkinan jalan dalam bidang 3D. Objek akan menentukan jalan ke poligon selanjutnya yang tidak sama dengan poligon tujuannya dengan jarak terpendek. Langkah tersebut diulang terus hingga objek dan titik tujuan memiliki poligon yang sama. Sehingga objek bisa berjalan dengan garis lurus. Metode ini sangat sederhana dan memiliki hasil yang lebih akurat.



Gambar 14 (Sumber: http://paper.ijcsns.org/07_book/201101/20110119.pdf)

V. KESIMPULAN

Pathfinding berguna dalam perkembangan *video games*. Teori graf berbobot digunakan dalam *video games* untuk menentukan jarak terpendek yang harus ditempuh dari suatu titik awal ke titik tujuan. Terdapat berbagai algoritma yang ada untuk menyelesaikan permasalahan *pathfinding*. Seperti algoritma Dijkstra dan algoritma A*. Walaupun dalam teori algoritma tersebut mungkin sudah sempurna, masih terdapat kemungkinan kesalahan dalam implementasinya di *video games*. Algoritma tersebut selalu dikembangkan untuk menghasilkan hasil yang lebih akurat dan efisien.

VI. UCAPAN TERIMA KASIH

Puji syukut penulis ucapkan kepada Tuhan Yang Maha Esa atas nikmat-Nya yang sangat melimpah dan atas diberikannya kesempatan kepada saya untuk menyelesaikan makalah ini. Juga ucapan terima kasih sebesar – besarnya kepada dosen – dosen mata kuliah IF2120 Matematika Diskrit, yaitu Bapak Dr. Ir. Rinaldi Munir, MT., Bapak Dr. Judhi Santoso M.Sc., dan Ibu Dra. Harlili S., M.Sc atas ilmu dan bimbingannya selama 1 semester ini sehingga makalah ini dapat terselesaikan. Saya juga berterima kasih kepada orang tua dan keluarga terdekat yang tiada hentinya selalu mendoakan agar saya mendapatkan segala sesuatu yang terbaik dan yang telah memberi dukungan atas segala hal yang saya lakukan. Dan terakhir, tidak lupa berterima kasih kepada teman – teman saya yang selalu membantu dalam menghadapi berbagai permasalahan dan selalu ada dikala senang maupun duka.

REFERENCES

[1] Munir, Rinaldi. 2006. Diktat Kuliah IF2120 Matematika Diskrit. Bandung: Institut Teknolgi Bandung

[2] Harika Redy, PATH FINDING - Dijkstra's and A* Algorithm. http://cs.indstate.edu/hgopireddy/algor.pdf, diakses pada tanggal 8 Desember 2018.

[3] Yan-Jiang SUN , Xiang-Qian DING, Lei-Na JIANG3,E, Heuristic Pathfinding Algorithm Based on Dijkstra. https://download.atlantis-press.com/article/25884610.pdf, diakses pada 8 Desember 2018.

[4] Xiao Cui and Hao Shi, A*-based Pathfinding in Modern Computer Games. http://paper.ijcsns.org/07_book/201101/20110119.pdf. Diakses pada tanggal 9 Desember 2018.

[5] ChitraNayal, estenger, Dijkstra's shortest path algorithm. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/, diakses pada tanggal 8 Desember 2018.

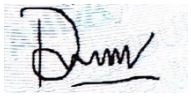
[6] Amit Patel, Introduction to A*, http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html, diakses pada tanggal 9 Desember 2018.

[7] Johann Fradj, Introduction to A* Pathfinding. https://www.raywenderlich.com/3016-introduction-to-a-pathfinding, diakses pada tanggal 9 Desember 2018.

PERYATAAN

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, 09 Desember 2018



Eka Sunandika, 13517130