Perbandingan Algoritma Dijkstra Dan Algoritma A* Pada Video Game Berbasis Grid Dua Dimensi

Muhammad Habibi Husni - 13521169
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13521169@itb.ac.id

***Abstrak*—*Video game* merupakan salah satu sektor yang banyak diminati. Terdapat banyak macam *video game*, salah satunya adalah *grid-based video game* atau gim berbasis *grid*. Untuk dapat menggerakkan sebuah entitas pada gim jenis ini, diperlukan mencarian rute yang paling efisien. Terdapat beberapa algoritma penyelesaian masalah ini, beberapanya adalah algoritma dijkstra dan algoritma A*. Pada makalah ini dilakukan perbandingan keefisienan kedua algoritma pada graf berbasis *grid* dengan melihat banyaknya sel yang dikunjungi. Hasil eksperimen menunjukkan algoritma A* memberikan kunjungan sel yang lebih sedikit dari algoritma Dijkstra. Dari hasil eksperimen dapat disimpulkan bahwa untuk kasus permainan berbasis *grid*, algoritma A* dapat bekerja lebih efisien daripada algoritma dijkstra.**

***Keywords*—About four key words or phrases in alphabetical order, separated by commas.**

I. PENDAHULUAN

Pada abad 21 ini, video game merupakan salah satu hal yang paling diminati oleh banyak orang, khususnya orang-orang yang berada pada rentang 10-30 tahun. Kesenangan dan tantangan yang diberikan oleh permainan video menjadi daya tarik utama dari sektor ini. Video game sendiri terdiri dari banyak genre, seperti action, sandbox, horror, dan sebagainya.

Video game dapat direpresentasikan dalam berbagai model, beberapanya adalah model tiga dimensi, dua dimensi, teks, dan lain-lain. Salah satunya adalah model *grid*. Model ini membatasi letak objeknya pada sebuah sel. Karena kesimpelannya, model ini sering digunakan oleh game bergenre RPG dan strategi. Salah satu game itu adalah Crypt of the Necrodancer

Gambar 1. Crypt of the Necrodancer.
(Sumber: https://www.nme.com/news/gaming-news/crypt-of-the-necrodancer-massive-update-3260549)

Pada video game jenis ini, terdapat entitas yang tidak dikontrol pemain. Salah satunya adalah musuh atau biasa disebuh sebagai monster. Musuh adalah entitas yang bertujuan untuk menghalangi pemain untuk mencapai tujuan akhir permainan, yaitu dengan cara menyerang pemain. Namuns sebelum dapat menyerang pemain, maka entitas musuh harus dapat menentukan rute terbaik yang akan membawanya kepada targetnya. Permasalah ini disebut sebagai *shortest path problem*.

Shortest path problem merupakan permasalahan klasik dalam dunia graf. Permsalahan ini adalah menentukan rute dengan jarak paling minimal dari satu titik ke titik lainnya. Beberapa algoritma yang dapat menyelesaikan permasalahan ini adalah algoritma dijkstra, A*, Bellman-Ford, dan sebagainya. Beberapa video game harus dapat dijalankan secara realtime sehingga algoritma yang digunakan untuk permasalahan shortest path haruslah yang paling efisien. Pada makalah ini akan dilakukan perbandingan antara algoritma dijkstra dan algoritma A*.

II. TEORI DASAR

A. Graf

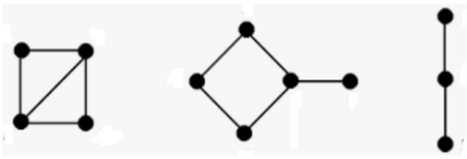
Graf dapat didefinisikan sebagai struktur diskrit yang merepresentasikan objek-objek pada struktur tersebut serta hubungan antara objek-objek tersebut. Secara formal, graf dapat direpresentasikan sebagai 2-tupel:

$$G = (V, E)$$

dengan V adalah himpunan tidak-kosong dari simpul (vertices) dan E adalah himpunan sisi (edges) yang menghubungkan sepasang simpul [1].

Graf dapat dibedakan menjadi beberapa jenis. Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf, graf dapat dibagi menjadi,

1. Graf sederhana (*simple graph)*
   Graf yang tidak mengandung gelang maupun sisi ganda

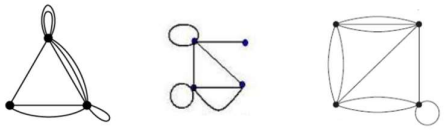


***Gambar 2**. Contoh Graf Sederhana[1]*

2. Graf tak-sederhana (unsimple graph)
   Graf yang mengandung gelang dan/atau sisi ganda. Graf tak-sederhana dapat dibedakan lagi menjadi,
   a. Graf ganda (multi-graph)
      Graf yang memiliki sisi ganda

***Gambar 3**. Contoh Graf Ganda[1]*

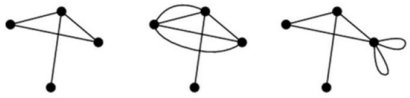
   b. Graf semu (pseudo-grap)
      Graf yang memiliki sisi gelang



***Gambar 4**. Contoh Graf Semu[1]*

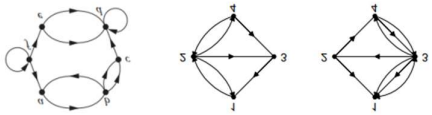
Graf juga dapat dibedakan berdasarkan orientasi arah pada sisi. Pembagian tersebut adalah,

1. Graf tak-berarah (undirected graph)
   Graf yang setiap sisinya tidak memiliki orientasi arah



***Gambar 5**. Contoh Graf Tak-berarah[1]*

2. Graf berarah (directed graph atau digraph)
   Graf yang setiap sisinya diberikan orientasi arah



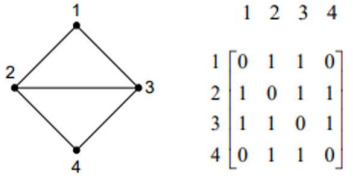
***Gambar 6**. Contoh Graf Berarah[1]*

Dalam graf terdapat beberapa terminologi sebagai berikut:

1. Ketetanggaan (Adjacent)
   Bertetangga merupakan sebuah istilah yang diberikan pada simpul yang terhubung secara langsung dengan simpul lain.
2. Bersisian (Incidency)
   Bersisian merupakan istilah yang diberikan pada sebuah simpul yang terhubung secara langsung dengan sebuah sisi.
3. Simpul Terpencil (Isolated vertex)
   Simpul terpencil adalah simpul yang tidak bertetangga dengan simpul apapun atau tidak bersisian dengan sisi manapun.
4. Graf Kosong (Null graph atau Empty graph)
   Graf kosong adalah graf yang himpunan E nya merupakan himpunan kosong.
5. Derajat (Degree)
   Derajat dari sebuah simpul adalah banyaknya sisi yang bersisian dengan simpul tersebut. Dinotasikan sebagai d(v). Pada graf berarah, derajat dibedakan lagi menjadi derajat masuk (in-degree) dan derajat keluar (out-degree).
6. Lintasan (Path)
   Lintasan adalah barisan dari sisi-sisi yang menghubungkan barisan dari simpul-simpul. Panjang sebuah lintasan didefinisikan sebagai banyak sisi yang dilalui lintasan tersebut..
7. Siklus (Cycle) atau Sirkuit (Circuit)
   Lintasan yang berawal dan berakhir pada simpul yang sama.
8. Keterhubungan (Connected)
   Dua buah simpul dikatakan terhubung jika terdapat lintasan yang menghubungkan dua simpul tersebut. Sebuah graf G disebut graf terhubung (connected graph) jika setiap pasang simpul pada G saling terhubung, dan dikatakan graf tak-terhubung (disconnected graph) jika tidak memenuhi syarat tersebut.
9. Graf Berbobot (Weighted Graph)
   Graf yang setiap sisinya diberi sebuah harga (bobot).

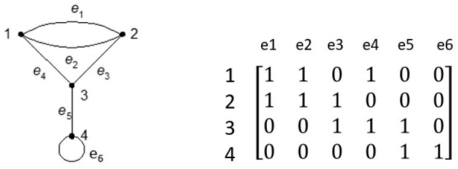
Untuk merepresentasikan graf, terdapat beberapa model yang dapat digunakan, yaitu:

1. Matriks Ketetanggaan (adjacency matrix)
   Elemen pada matriks yaitu a_{ij} menyatakan banyaknya sisi yang menghubungkan simpul i dan j.



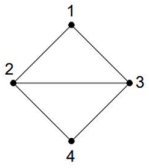
***Gambar 7**. Contoh Matriks Ketetanggaan[2]*

2. Matriks Bersisian (incidency matrix)
   Elemen padSa matriks yaitu a_{ij} menyatakan simpul i bersisisan dengan sisi j.



***Gambar 8**. Contoh Matriks Bersisian[2]*

3. Senarai Ketetanggaan (adjacency list)
   Tetangga dari sebuah simpul di representasikan sebagai sebuah list.



Simpul	Simpul Tetangga
1	2, 3
2	1, 3, 4
3	1, 2, 4
4	2, 3

***Gambar 9**. Contoh Senarai Ketetanggaan[2]*

B. Shortest path

Shortest path merupakan salah satu permasalahan umum yang ada pada graf. Permasalahan shortest path dapat didefinisikan sebagai pencarisan lintasan terpendek diantara dua simpul tertentu pada sebuah graf. Jika graf merupakan graf berbobot, maka lintasan yang dicari haruslah lintasan yang jumlah bobotnya terkecil.

Shortest path sendiri dapat dibedakan menajadi dua jenis, yaitu:

1. Single-source Shortest Path
   Yaitu pencarian lintasan terpendek dari sebuah simpul ke seluruh simpul yang ada pada graf.
2. Single-destination Shortest Path
   Yaitu pencarian lintasan terpendek dari seluruh simpul yang ada pada graf ke sebuah simpul tertentu.
3. All-pairs Shortest Path
   Yaitu pencarian lintasan terpendek dari setiap pasang simpul pada sebuah graf.

Untuk mencari shortest path, terdapat beberapa algoritma yang dapat digunakan. Beberapa algoritma tersebut adalah:

1. Algoritma Dijkstra
   Algoritma Dijkstra merupakan salah satu algoritma pencarian single-source shortest path. Algoritma Dijkstra dimulai pada sebuah simpul sumber, dan lalu dilanjutkan dengan menentukan jarak terpendek menuju simpul yang terhubung secara langsung dengan simpul yang sudah ditentukan shortest pathnya. Kasus awal ialah shortest path dari simpul sumber bernilai 0.

   Berikut adalah algoritma Dijkstra:
   1) Pilih sebuah simpul *source* pada graf G yang akan dijadikan simpul sumber dan tetapkan $dist_{source}$ = 0. Untuk simpul selain *Source,* tetapkan nilai $dist = \infty$. Masukkan seluruh simpul pada sebuah himpunan Q yang menunjukkan simpul-simpul yang masih belum ditentukan nilai shortest pathnya.
   2) Pilih sebuah simpul v pada Q dengan nilai *dist* terkecil. Hapus v dari dari Q.
   3) Cek setiap simpul u yang merupakan tetangga dari v yang belum ada pada Q. Nilai $dist_u$ diperbaharui apabila $dist_u < dist_v + W_{vu}$, dengan W_{vu} adalah besar bobot dari sisi yang menghubungkan v dan u.
   4) Ulangi langkah 2 hingga 4 hingga himpunan Q kosong.

   Berikut adalah pseudocode dari algoritma Dijkstra,

```
function dijkstra(G, source)
    dist = map()
    Q = set()
    for v in G:
        if (v = source):
            dist[v] := 0
        else:
            dist[v] := infinity
            add v to Q
    while Q is not empty:
        v := v in Q with minimum dist[v]
        remove v from Q
        for each neighbor u of v and u in Q:
            newDist := dist[v] + weight(v,u)
            if (newDist < dist[u]):
                dist[u] := newDist
    return dist[]
```

***Gambar 10**. Pseudocode Algoritma Dijkstra*
(Sumber: Dokumentasi penulis)

2. Algoritma A*
   Algoritma A* merupakan algoritma path finding yang juga dapat digunakan sebagai algoritma pencarian shortest path. Dibandingkan dengan Dijkstra, algoritma A* hanya berfokus mencari shortest path dari satu pasang simpul saja. A* memiliki cara kerja yang sama dengan algoritma Dijkstra. Dibandingkan dengan algoritma dijkstra, A* tidak hanya menggunakan jarak terdekat simpul dari sumber sebagai patokan pencarian shortest path, melainkan juga menambahkan biaya perkiraan dari simpul saat ini hingga menuju simpul tujuan. A* menentukan prioritas pemilihan simpul melalui fungsi berikut :

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

Dengan *f(n)* menunjukkan fungsi evaluasi, *g(n)* biaya yang telah dikeluarkan dari simpul awal, dan *h'(n)* estimasi biaya menuju simpul akhir untuk n adalah sebuah simpul. Nilai kedua, yaitu *h'(n)*, disebut sebagai fungsi heuristic. Nilai heuristic merupakan nilai perkiraan sehingga penentuan fungsinya perlu ditentukan terlebih dahulu agar hasil pencarian shortest path tidak keliru. Pada kasus umum, h'(n) ditetapkan sebagai jarak euclidean dari simpul n dan simpul tujuan.

Keberadaan fungsi heuristic membuat traversal graf menjadi unik tergantung nilainya. Hal ini membuat nilai heuristic dapat diganti sesuai kebutuhan. Berdasarkan nilah h'(n), algoritma A* akan memiliki perilaku seberti berikut[2] :

a) Jika $h'(n) = 0$, f(n) hanya dipengaruhi oleh g(n) dan algoritma A* akan bekerja seperti algoritma dijkstra biasa.
b) Jika $h'(n) <$ cost menuju tujuan, membuat algoritma A* pasti akan menemukan shortest path. Semakin dekat fungsi heuristic mendekati cost menuju tujuan, semakin sedikit lingkup pencarian

dari algoritma.

c) Jika $h'(n) = cost\ menuju\ tujuan$, membuat algoritma A* selalu mencari pada jalur terbaik dan membuat algoritma menjadi sangat efisien. Namun, karena mencari fungsi heuristic yang tepat adalah tugas yang sulit, maka kasus ini jarang diigunakan.
d) Jika ada kemungkinan $h'(n) > cost\ menuju\ tujuan$, maka algoritma A* tidak dapat dipastikan menghasilkan jalur terbaik.
e) Jika $h'(n) \gg g(n)$, f(n) hanya akan dipengaruhi h'(n) dan algoritma A* berubah menjadi algoritma greedy best-fist search.

Berikut adalah algoritma dari algoritma A*,

1) Pilih sebuah simpul *source* pada graf G yang akan dijadikan simpul sumber dan tetapkan $g(source) = 0$. Untuk simpul selain *Source,* tetapkan nilai $g(n) = \infty$. Masukkan seluruh simpul pada sebuah himpunan Q yang menunjukkan simpul-simpul yang masih belum ditentukan nilai shortest pathnya.
2) Pilih sebuah simpul *v* pada Q dengan nilai $f(v) = g(v) + h'(v)$ terkecil. Hapus *v* dari dari Q.
3) Cek setiap simpul *u* yang merupakan tetangga dari *v* yang belum ada pada Q. Nilai $g(u)$ diperbaharui apabila $g(u) < g(v) + W_{vu}$, dengan W_{vu} adalah besar bobot dari sisi yang menghubungkan *v* dan *u*.
4) Ulangi langkah 2 hingga 4 hingga simpul tujuan tidak ada pada Q atau sudah dikunjungi.

Berikut adalah pseudocode dari algoritma A*,

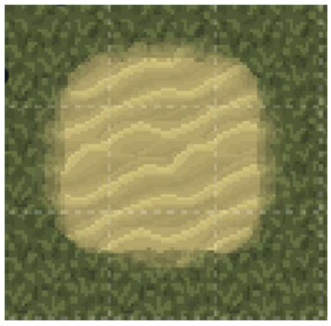
```
function aStar(G, source, goal)
    g = map()
    Q = set()
    for v in G:
        if (v = source):
            g[v] := 0
        else:
            g[v] := infinity
            add v to Q
    while Q is not empty:
        v := v in Q with minimum f(v) = g[v] + h'(v)
        remove v from Q
        if (v = goal):
            break
        for each neighbor u of v and u in Q:
            newDist := g[v] + weight(v,u)
            if (newDist < g[u]):
                g[u] := newDist
    return g[goal]
```

Gambar 11. *Pseudocode Algoritma A**
(Sumber: Dokumentasi penulis)

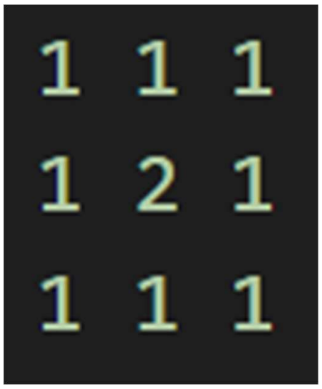
III. Perbandingan Algoritma Dijkstra dan A* pada Video Game Grid Dua Dimensi

A. Deskripsi Singkat

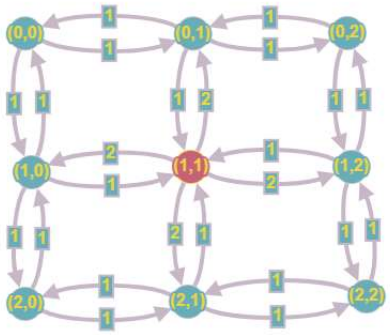
Dalam sebuah permainan video grid dua dimensi, peta permainan direpresentasikan dalam bentuk kisi atau grid. Grid peta akan terdiri dari sel yang bisa dilalui, dan sel yang tidak bisa dilalui berupa dinding atau lubang. Setiap sel yang dapat dilalui pada grid peta tersebut akan memiliki waktu tempuh ataupun jarak tempuh. Selain itu, pada beberapa permainan video terdapat beberapa sel khusus yang memiliki nilai cost yang berbeda. Pada contohnya, sel rerumputan membutuhkan waktu tempuh 1 satuan waktu, sedangkan sel lumpur membutuhkan waktu tempuh 2 satuan waktu. Oleh karena itu, sebuah grid peta dapat direprentasikan sebagai sebuah graf.



Gambar 12. *Contoh Grid Peta*
(Sumber: Dokumentasi penulis menggunakan https://deepnight.net/tools/rpg-map)



Gambar 13. *Matriks biaya dari gambar 11*
(Sumber: Dokumetasi penulis)



Gambar 14. *Bentuk graf dari gambar 11*
(Sumber: Dokumentasi penulis menggunakan https://graphonline.ru/en/)

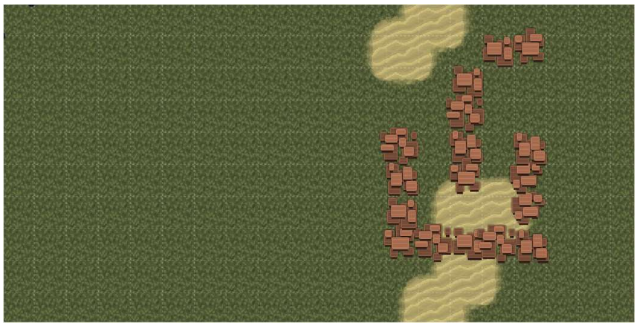
B. Spesifikasi Eksperimen

Untuk eksperimen pada makalah ini, grid peta yang akan

digunakan berukuran 10 x 20 dengan variasi grid peta adalah sel rumput, sel lumpur, dan sel dinding. Untuk sel dinding akan direpresentasikan dengan angka -1. Pergerakan player dibatasi dengan sel yang terletak tepat disamping selnya saat ini.

1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	-1	-1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	-1	1	-1	1	-1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	-1	1	-1	1	-1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	-1	1	2	2	-1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1

(a)



(b)

***Gambar 15**. (a) Kasus uji dalam bentuk matriks peta, (b) Peta yang ekivalen dengan 14(a) (Sumber: Dokumentasi Penulis menggunakan https://deepnight.net/tools/rpg-map)*

Percobaan akan dilakukan dengan bahasa pemrograman C++ dan keefektifan algoritma akan dibandingkan melalui banyaknya sel yang dikunjungi.

C. Implementasi Algoritma

Pada percobaan ini akan dibandingkan dua buah algoritma yaitu algoritma Dijkstra dan algoritma A*. Berikut adalah implementasi masing-masing algoritma pada bahasa C++. Pada C++ sudah ada pustaka priority_queue yang dapat dimanfaatkan untuk membuat pengambilan sel dengan prioritas tertinggi menjadi efisien. Berikut adalah implementasi dari masing-masing algoritma :

1. Algoritma Dijkstra

```cpp
int dijkstra(pair<int,int> start, pair<int,int> goal){
    for (int i = 0;i < n;i++){
        for (int j = 0;j < m;j++){
            g[i][j] = INF;
            visited[i][j] = false;
        }
    }
    g[start.first][start.second] = 0;
    priority_queue<pqElmtDjik, vector<pqElmtDjik>,
        greater<pqElmtDjik>> pque;
    // elemen priority queue berbentuk tupel (g(n), n)
    pque.push({0, {start.first,start.second}});
    while (!pque.empty()){
        int vy = pque.top().second.first;
        int vx = pque.top().second.second;
        pque.pop();
        visited[vy][vx] = true;
        if (vy == goal.first && vx == goal.second){
            break;
        }
        for (int i = 0;i < 4;i++){
            // iterasi tetangga dari v
            int uy = vy + relY[i];
            int ux = vx + relX[i];
            if (uy < 0 || uy >= n || ux < 0 || ux >= m
                || gameMap[uy][ux] == -1) continue;
            int newCost = g[vy][vx] + gameMap[vy][vx];
            if (!visited[uy][ux] && newCost < g[uy][ux]){
                g[uy][ux] = newCost;
                pque.push({newCost,{uy,ux}});
            }
        }
    }
    return g[goal.first][goal.second];
}
```

***Gambar 16**. Implementasi Algoritma Dijkstra (Sumber: Dokumentasi penulis)*

2. Algoritma A*

```cpp
int astar(pair<int,int> start, pair<int,int> goal){
    for (int i = 0;i < n;i++){
        for (int j = 0;j < m;j++){
            g[i][j] = INF;
            visited[i][j] = false;
        }
    }
    g[start.first][start.second] = 0;
    priority_queue<pqElmtAstar,
        vector<pqElmtAstar>, greater<pqElmtAstar>> pque;
    // elemen priority queue berbentuk tupel ((f(n),h(n)), n)
    int h0 = heuristic(start,goal);
    pque.push({{0+h0, h0}, {start.first,start.second}});
    while (!pque.empty()){
        int vy = pque.top().second.first;
        int vx = pque.top().second.second;
        pque.pop();
        visited[vy][vx] = true;
        if (vy == goal.first && vx == goal.second){
            break;
        }
        for (int i = 0;i < 4;i++){
            int uy = vy + relY[i];
            int ux = vx + relX[i];
            if (uy < 0 || uy >= n || ux < 0 ||
                ux >= m || gameMap[uy][ux] == -1) continue;
            int newCost = g[vy][vx] + gameMap[vy][vx];
            if (!visited[uy][ux] && newCost < g[uy][ux]){
                g[uy][ux] = newCost;
                int h = heuristic({uy,ux},goal);
                pque.push({{newCost+h,h},{uy,ux}});
            }
        }
    }
```

***Gambar 17**. Implementasi Algoritma A**

(Sumber: Dokumbentasi Penulis)

Pada Implementasi di atas, prioritas menggunakan tupel (f(n), h'(n)) agar dapat membedakan sel yang memiliki f(n) yang sama dan lebih memprioritaskan sel yang jarak menuju goalnya lebih kecil.

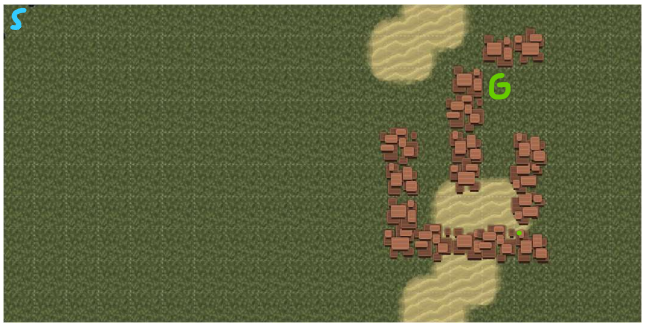
Karena peta yang berbentuk grid, dan langkah valid hanya sel yang tepat disamping, maka jarak manhattan dari simpul ke goal dapat dijadikan sebagai fungsi heuristic. Untuk meningkatkan efiensi, nilai jarak manhattan dapat dikalikan dengan bobot terkecil sel yang ada pada peta. Hal ini menyebabkan nilai h'(n) semakin mendekati jarak tujuan sebenarnya

```
int heuristic(pair<int,int> n, pair<int,int> goal){
    int minD = 1;
    int dx = abs(goal.second - n.second);
    int dy = abs(goal.first - n.first);
    return minD*(dx + dy);
}
```

***Gambar 18**. Algoritma Heuristik yang Digunakan*

D. Eksperimen

Eksperimen dilakukan dengan menetapkan sel awal pada koordinat (0,0) dari sudut kiri atas dan sel dengan koordinat (2,15) dari sudut kiri atas.



***Gambar 19**. Titik Awal dan Titik Akhir pada Kasus Uji (Sumber: Dokumentasi penulis menggunakan https://deepnight.net/tools/rpg-map)*

Berikut adalah hasil dari eksperimen:

```
Hasil algoritma Dijkstra:
0  1  2  3  4  5  6  7  8  9  10 11 12 13 15 16 17 18 19 20
1  2  3  4  5  6  7  8  9  10 11 12 13 15 16  #  # 19 20 21
2  3  4  5  6  7  8  9  10 11 12 13 14 15  #  22 21 20 21 22
3  4  5  6  7  8  9  10 11 12 13 14 15 16  #  .  22 21 22  .
4  5  6  7  8  9  10 11 12 13 14 15  # 17  #  .  # 22  .  .
5  6  7  8  9  10 11 12 13 14 15 16  # 18  #  .  #  .  .  .
6  7  8  9  10 11 12 13 14 15 16 17  # 19 20 22  #  .  .  .
7  8  9  10 11 12 13 14 15 16 17 18  #  #  #  #  #  .  .  .
8  9  10 11 12 13 14 15 16 17 18 19 20 21 22  .  .  .  .  .
9  10 11 12 13 14 15 16 17 18 19 20 21 22  .  .  .  .  .  .
Banyak sel yang dikunjungi: 157
```

(a)

```
Hasil algoritma A*:
0  1  2  3  4  5  6  7  8  9  10 11 12 13 15 16 17 18 19  .
1  2  3  4  5  6  7  8  9  10 11 12 13 15 16  #  # 19 20  .
2  3  4  5  6  7  8  9  10 11 12 13 14 15  #  22 21 20 21  .
3  4  5  6  7  8  9  10 11 12 13 14 15 16  #  .  22 21  .  .
4  5  6  7  8  9  10 11 12 13 14 15  # 17  #  .  #  .  .  .
5  6  7  8  9  10 11 12 13 14 15 16  # 18  #  .  #  .  .  .
.  .  .  .  .  .  .  .  .  .  .  .  # 19 20 22  #  .  .  .
.  .  .  .  .  .  .  .  .  .  .  .  #  #  #  #  #  .  .  .
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
Banyak sel yang dikunjungi: 96
```

(b)

***Gambar 20**. (a) Hasil Pencarian Algoritma Dijkstra. (b) Hasil Pencarian Algoritma A*. (Sumber: Dokumentasi penulis)*

Dari Hasil eksperimen tersebut, algoritma Dijkstra mendapatkan shortest path sebesar 22 satuan waktu setelah mengunjungi 157 sel. Sedangkan algoritma A* juga mendapatkan shortest path sebesar 22 satuan waktu dengan hanya mengunjungi 96 sel.

E. Pembahasan Hasil Eksperimen

Dari hasil eksperimen dapat dilihat bahwa kedua algoritma mampu menemukan rute terpendek dengan akurat. Namun, algoritma A* mengunjungi sel lebih sedikit daripada algoritma Dijkstra. Hal ini disebabkan karena algoritma A* tidak hanya menjadikan nilai jarak terpendek saat ini dari sel awal sebagai prioritas pengambilan sel, namun juga jarak terpendek dari sel saat ini menuju sel tujuan. Sedangkan pada algoritma dijkstra, penelusuran hanya berdasarkan jarak terpendek saat ini, sehingga algoritma akan melakukan pengecekan pada semua jarak terpendek yang sama. Selain itu, telah dilakukan optimisasi pada A* yaitu dengan membedakan prioritas dua buah sel yang nilai f(n) nya sama, dengan membandingkan nilai h'(n). Optimisasi ini dapat membuat algoritma A* hanya mengambil rute paling efisien meskipun banyaknya nilai evaluasi yang sama. Optimisasi dapat ditambah lagi dengan membedakan kasus dua sel yang f(n) serta h'(n) nya sama.

IV. KESIMPULAN

Video game merupakan hiburan yang sangat diminati orang. Terdapat berbagai macam permainan, salah satunya adalah permainan berbasis *grid*. Pada permainan ini terdapat permasalahan pencarian shortest path yang harus dilakukan seefisien mungkin. Beberapa algoritma yang dapat digunakan adalah algoritma dijkstra dan algoritma A*. Kedua algoritma memiliki cara kerja yang sama namun sedikit berbeda. Setelah dilakukannya eksperimen pada makalah ini, algoritma A* bekerja lebih efisien daripada algoritma dijkstra pada graf berbasis grid. Algoritma A* sendiri masih dapat dioptimisasi lebih jauh sehingga algoritma A* banyak digunakan dalam permainan video game.

V. UCAPAN TERIMA KASIH

Pertama sekali penulis mengucapkan puji syukur kepada Allah Swt.atas segala nikmat yang telah Diberikan-Nya sehingga penulis dapat menyelesaikan ini yang berjudul "Perbandingan Algoritma Dijkstra Dan Algoritma A* Pada Video Game Berbasis Grid Dua Dimensi" sebagai pemenuhan tugas makalah IF2120 Matematika Diskrit. Penulis juga ingin menyampaikan ucapan terima kasih kepada Bapak Dr. Rinaldi Munir, selaku Dosen Mata Kuliah IF2120 Matematika Diskrit beserta teman dan keluarga yang telah memberikan bantuan dan dorongan dalam penyelesaian makalah ini.

DAFTAR PUSTAKA

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf2020-Bagian1.pdf. Diakses pada tanggal 11 Desember 2022.

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf2020-Bagian2.pdf. Diakses pada tanggal 11 Desember 2022.
[3] https://www.redblobgames.com/pathfinding/a-star/introduction.html. Diakses pada tanggal 12 Desember 2022.
[4] https://brilliant.org/wiki/dijkstras-short-path-finder/. Diakses pada tanggal 12 Desember 2022
[5] http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html. Diakses pada tanggal 12 Desember 2022

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, 12 Desember 2022

Muhammad Habibi Husni
13521169