Pencarian rute terdekat musuh di multi player dengan A star Dan Mencari pintu ke next level pada single player dengan Best First Search

Taufik Adriansyah
0102516901
Teknik Informatika
Universitas Al Azhar Indonesia

***Abstract* —** Game sebagai media hiburan telah berkembang dengan pesat, seiring dengan berkembangnya juga teknologi. Salah satu unsur yang berperan penting dalam sebuah game adalah kecerdasan buatan. Dengan kecerdasan buatan, diharapkan elemen-elemen dalam game dapat berperilaku sealami mungkin layaknya manusia, sehingga bias lebih memberikan tantangan yang lebih bagi para penggemar game tersebut. Dalam permainan Bomberman klasik ini yang tentunya telah melegenda dikalangan anak 90an penulis ingin menerapkan game dengan metode Best First Search dan A star.

Dari Permainan yang sudah ada pada mode multi player dimana saat kita mengincar lawan dan menaruh bom didekatnya, maka dengan mudah lawan tersebut akan kabur untuk mencari posisi aman. Sedangkan pada mode single player, pemain diharuskan mencari pintu yang tersembunyi dibalik batu abu abu ke level selanjutnya dan harus memecahkan rintangan berupa blok abu abu dan sejumlah monster atau alien yang membutuhkan waktu yang lama. sejauh ini belum ada solusi agar player yang berada 3blok terdekat agar terkunci posisinya dan mencari jalur tercepat mencari pintu ke level selanjutnya.

Dari game Bomberman tersebut masalah yang akan dibahas pada mode multi player yaitu mencari rute terdekat dengan musuh dan menaruh bom dengan range 3 blok abu abu sehingga dapat mengunci posisi musuh agar tidak kabur dengan metode A star. Sedangkan pada metode single player di utamakan mencari jalur tercepat ke pintu next level pada multi dengan metode Best First Search.

Keywords—Game, **Bomberman, Kecerdasan buatan.**

I Pendahuluan

Bomberman adalah sebuah game bergenre strategi yang mengharuskan pemainnya memiliki tingkat konsentrasi tinggi untuk memenangkannya dan tidak sedikit pemain akan merasa penasaran jika belum berhasil memenangkan permainan. Selain itu ditambah lagi pada mode single player ada beberapa level yang telah disiapkan didalamnya yang membuat pemain penasaran rintangan, bonus serta kesulitan apa saja yang ada disetiap tingkatan level tersebut.

Bomberman, juga dikenal sebagai Dynablaster, adalah franchise permainan video berbasis labirin yang strategis, awalnya dikembangkan oleh Hudson Soft dengan nama itu di tahun 1985.game tersebut telah diterbitkan untuk berbagai platform game, termasuk official 2016 versi mobile yang merayakan 30 tahun franchise Bomberman. Sejak berdirinya permainan telah menyimpan sejumlah besar pengikut, termasuk komunitas yang berdedikasi pengguna yang membuat Bomberman Wiki dengan lebih dari 1.400 [6].

Gambar 1.Tampilan game bomberman [4]

Pada game ini di mode multi player kita sebagai pemain akan bermain dan bertarung melawan komputer ataupun player lain. masing-masing pemain harus saling mengalahkan dengan menggunakan bom, siapa yang paling terakhir bertahan, maka dia yang akan menang. pada saat permainan terjadi, ada beberapa dinding pembatas yang harus kita hancurkan agar kita bisa bertemu langsung dengan lawan yang berada di sisi lain, jika kita beruntung kita bisa menghancurkan dinding yang ada bonusnya. Bonus tersebut bisa saja menguntungkan, bisa juga merugikan Bomberman dapat dikatakan sederhana namun memerlukan tingkat konsentrasi dan pemilihan strategi yang baik untuk memenangkannya, pemain cukup meletakkan bom pada tempat yang ingin diledakkan, bom ini digunakan untuk menghancurkan rintangan, musuh dan untuk meledakkan bom lain[4].

Bomberman memiliki mode multi player dan single player, pada mode multi player ini terdiri dari beberapa

pemain yang berwujud bomberman lain yang bertindak sebagai lawan yang saling menghancurkan satu sama lain, pada mode ini pemain yang berdiri terakhir / pemain yang berhasil mengalahkan pemain yang lain adalah pemenangnya.

Berbeda halnya dengan mode single player, mode single player ini dibuat seperti misi yang harus dituntaskan oleh pemain, pemain diharuskan mencari pintu yang harus dimasuki pemain ketika musuh-musuh yang ada telah berhasil dikalahkan menggunakan bom, musuh-musuh yang disediakan berjumlah sedikit pada awalnya namun dapat bertambah apabila pemain meletakkan bom pada pintu yang disediakan pada akhir permainan . Pemain yang berhasil memasuki pintu akan lanjut ke stage berikutnya dengan tantangan yang semakin meningkat .

Yang akan dibahas pada makalah ini adalah bagaimana cara cepat memenangkan game bomberman dengan cepat karena terbatasnya waktu , yaitu pada mode multi player adalah mencari rute tercepat untuk menaruh bom dengan range 3 blok agar posisi lawan terkunci dan tidak kabur, sedangkan pada mode single player yaitu bagaimana cara mencari pintu yang tersembunyi di balik batu penghalang menuju ke level selanjutnya dengan cepat, sehingga bisa menghemat waktu bermain dan menambah poin tentunya setelah membunuh semua monster yang ada.

II Dasar Teori

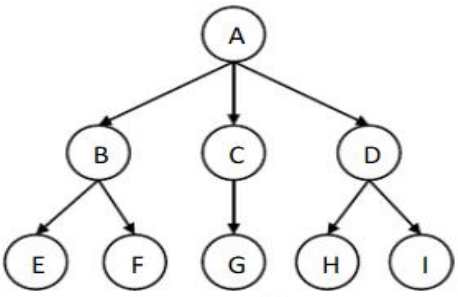
Terdapat beberapa metode yang dapat dilakukan untuk bermain permainan bomberman ini. Di sini metode yang akan dibahas adalah penggunaan algoritma dasar diantaranya adalah algoritma Best First Search dan algoritma A star untuk menyelesaikannya.

Pada pembahasan ini tidak digunakan algoritma yang kompleks tapi hanya algoritma sederhana dengan memakai Best First Search dan algoritma A star. pada game ini waktu juga menjadi permasalahan mendasar, dimana kita harus menyelesaikan game dalam waktu 2 menit di multiplayer dan 200 detik di singel player. Kalau waktu habis maka player akan kalah dan gagal mencapai tujuannya, tujuanya yaitu mengalahkan musuh sampai naik kelevel selanjutnya di mode single player dan tujuan pada mode multi player yaitu mengalahkan player lain dan menjadi yang terakhir di arena[7].

a. Algoritma Best First Search

Best First Search adalah metode pencarian secara melebar. BFS mengunjungi semua simpul yang belum dikunjungi dari akar. Algoritma BFS menggunakan sebuah queue Q untuk menyimpan antrian [5]. Algoritma BFS ditunjukkan sebagai berikut:

1. Memasukkan simpul akar misal simpul A ke Q.
2. Mengambil simpul dari antrian pertama pada Q.
3. Jika simpul tersebut merupakan solusi, maka pencarian dihentikan dan solusi telah ditemukan.
4. Jika simpul tersebut bukan solusi, maka memasukkan simpul yang bertetangga dengan simpul tersebut ke antrian.
5. Jika antrian kosong, maka solusi tidak ditemukan.
6. Jika antrian tidak kosong, maka ulangi pencarian dari langkah 2.



Gambar 2 Pencarian secara BFS [5].

Jika melakukan pencarian secara BFS pada graf di gambar 2, maka penelusurannya dimulai dari simpul A. Setelah itu, akan dibangkitkan semua simpul anak dari A yaitu B, C dan D. Lalu dimulai lagi dengan membangkitkan anak simpul B yaitu E dan F. Setelah itu, G dibangkitkan karena anak dari simpul C serta H dan I dibangkitkan oleh D. E dan F tidak memiliki anak lagi sehingga tidak ada simpul yang bisa dibangitkan. Begitu pula G, H dan I. Secara keseluruhan, penelusurannya akan menjadi A-B-C-D-E-F-G-H-I. [2].

Pseudocode BFS ditunjukkan sebagai berikut. Misal terdapat graf G, simpul akar v dan tabel Boolean yang bernama dikunjungi berukuran sebesar jumlah simpul.

Kelebihan metode BFS adalah tidak akan menemui jalan buntu, menjamin menemukan solusi jika memang solusinya ada dan solusi yang ditemukan adalah solusi yang optimal. Kelemahan metode ini adalah membutuhkan memori yang cukup banyak karena harus menyimpan semua simpul yang pernah dibangkitkan dan membutuhkan waktu yang lama.

b. Algoritma A Star

A* diperkenalkan oleh Peter Hart, Nils Nilsson dan Bertram Raphael pertama kali pada tahun 1968 dengan menggunakan *heuristic* [3]. Algoritma A* merupakan format pencarian heuristik untuk menghitung efisiensi solusi optimal.

Algoritma A* adalah algoritma *best-first search* dimana *cost* yang terkait dengan *node* adalah f (n) = g(n) + h(n), dan g(n) adalah *cost of the path* dari keadaan awal ke *node* n dan h(n) adalah perkiraan heuristik atau *cost* atau *path* dari *node* n ke tujuan. Jadi, f(n) adalah perkiraan total *cost* terendah dari setiap *path* yang akan dilalui *node* n ke *node* tujuan[13]. Dengan kata lain, *cost* adalah jarak yang telah ditempuh, dan panjang garis lurus antara *node* n dengan *node* akhir adalah perkiraan heuristiknya. Semakin rendah nilai f(n), semakin tinggi prioritasnya.

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

dimana,
h(n) = Nilai heuristik antar Koordinat
g(n) = Jarak Koordinat ke titik tujuan

Algoritma A* menggunakan estimasi jarak terdekat untuk mencapai tujuan (*goal*) dan memiliki nilai heuristik yang digunakan sebagai dasar pertimbangan. Algoritma A* merupakan algoritma yang membantu menemukan solusi pencarian ruang keadaan dengan mempertimbangkan total biaya lintasan yang akan dilacak sesuai dengan node yang akan dilewati. Masalah ruang keadaan disebut juga dengan *state space problem*. Ruang keadaan (*state space*) merupakan suatu ruang yang berisi semua keadaan yang mungkin dalam suatu kasus AI. *State* adalah representasi suatu keadaan pada suatu saat ataupun dekripsi konfigurasi sistem. *State space* adalah semua *state* (keadaan) yang mungkin, dan biasanya digambarkan sebagai jaringan dengan *verteks* merupakan *state* dan *edge* merupakan perubahan yang mungkin[3].

Kondisi dalam ruang keadaan meliputi:
a. Keadaan awal (*initial state* / *start node*)
b. Keadaan tujuan, merupakan solusi yang dijangkau dan perlu diperiksa apakah telah mencapai sasaran.
c. Kaidah atau aturan yang memberikan bagaimana mengubah keadaan.

Selain itu metode A* merupakan metode hasil pengembangan dari metode dasar Best First Search . caranya mengevaluasi semua titik dan mengkombinasikan g(n) nilai titik dari titik awal dengan h(n) nilai perkiraan atau heuristic untuk mencapai titik akhir atau tujuan.

Urutan pencarian nodenya menggunakan fungsi distance – plus – cost (f(n)). yaitu berasal dari penjumlahan antara g(n) dan h(n). Notasi metode tersebut adalah sebagai berikut: f(n)=g(n) + h(n)

f(n) = distance – plus – cost (jarak – plus- Biaya)
g(n)= nilai jumlah jalur yang dilewati dari titik awal ke tujuan
h(n)= perkiraan heuristik

terminologi dasar yang terdapat pada algoritma A* (star) meliputi starting point, simpul (nodes), A, open list, closed.
Prinsip algoritma ini adalah mencari jalur terpendek dari sebuah simpul awal (starting point) menuju simpul tujuan dengan memperhatikan harga (F) terkecil.

Formula yang akan dipakai dalam metode A star disini ada dua yaitu modified Manhattan distance dan Euclidean distance.

1. Manhanttan distance

   menghitung jarak yang akan ditempuh untuk mendapatkan dari satu titik data ke jalur lain jika jalan seperti grid diikuti. Jarak Manhattan antara dua item adalah jumlah perbedaan komponen yang sesuai[6].

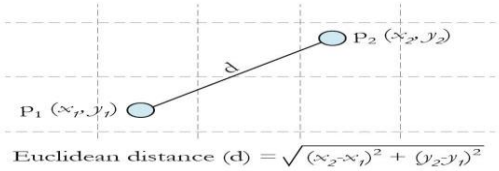
   Rumus untuk jarak antara titik X = (X1, X2, dll) dan titik Y = (Y1, Y2, dll) adalah:

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

Gambar 1.1 Formula Manhattan Distance [8]

Dimana n adalah jumlah variabel, dan Xi dan Yi adalah nilai dari variabel ke-i, pada titik X dan Y masing-masing.

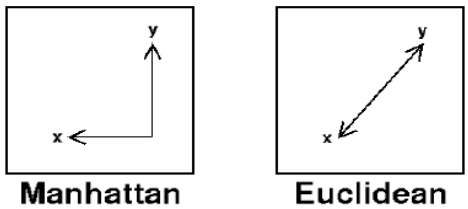
2. Euclidean distance
   - Euclidean distanc atau Euclidean metricis adalah garis "biasa" (yaitu garis lurus) di antara dua titik di ruang Euclidean[1].
   - Euclidean distance antara titik p dan q adalah panjang segmen garis yang menghubungkannya (pq)



Gambar 2.1 Formula Euclidean distace[8].

3. Manhattan distance vs Euclidean distance

   Gambar berikut menggambarkan perbedaan antara jarak Manhattan dan jarak Euclidean:



Gambar 3.1 Jalur Mahnhatan dan Euclidean[8].

Perbedaan Manhattan distance dan Euclidean distance tergantung pada data anda. Untuk vektor berdimensi tinggi Anda mungkin menemukan bahwa Manhattan bekerja lebih baik daripada jarak Euclidean[8].

Alasannya cukup sederhana untuk dijelaskan. Pertimbangkan kasus di mana kita menggunakan norma yang merupakan jarak Minkowski dengan eksponen = tak terhingga. Maka jarak adalah perbedaan tertinggi antara dua dimensi vektor Anda. Kita dapat melihat ini tidak masuk akal dalam banyak dimensi karena kita akan mengabaikan sebagian besar dimensi dan jarak pengukuran berdasarkan atribut tunggal.

Dengan demikian mengurangi eksponen membuat fitur lain memainkan peran lebih besar dalam perhitungan jarak. Semakin rendah eksponennya, semakin rendah perbedaan yang cukup tinggi dalam beberapa dimensi tertentu

III Implementasi

Pada Pembahasan kali ini yang di fokuskan yaitu pada cara memenangkan game dengan waktu yang singkat. Pada kasus ini yang ingin dimplementasikan yaitu mencari rute terdekat dengan musuh dengan A star sehingga kita dapat menaruh bom 3 range dengan akurat dan menyebabkan lawan tidak bisa kabur. Lalu dengan metode Best first search masalah yang akan dibahas yaitu mencari pintu ke level selanjutnya dengan cepat. Dimana kita harus mencari ke semua jalur yang ada sampai pintu yang tertutup tembok akan terlihat.

Berikut pembahasanya secara singkat:

a. Mencari Semua jalur ke pintu next level dengan BFS

Disini masalah yang akan dibahas adalah peluang pemain untuk pemain mencari kemungkinan semua jalan kearah pintu pada multi player untuk ke level selanjutnya. Dimana tujuan nya agar lebih menghemat waktu dan mempercepat pemain menemukan pintu yang tertutup dinding yang harus dipecahkan dengan bomb.

Jalur setapak itu terhalang oleh dinding abu-abu yang dapat dihancurkan dengan bom. Karena itu, Player disin harus menjelajahi peta, mereka perlu membuka blokir jalurnya satu persatu dahulu. Untuk mencapai tujuan ini, kita harus menempatkan bom di dekat dinding setelah itu saat bom meledak maka dinding tersebut akan hancur dan jika kita menemukan pintu kelevel selanjutnya maka kita telah menemukan pintu ke level selanjutnya, tapi kita harus terlebih dahulu membunuh monster di map sampai habis.

Gambar 3. Ilustrasi player mencari pintu menjelajah semua rute [6].

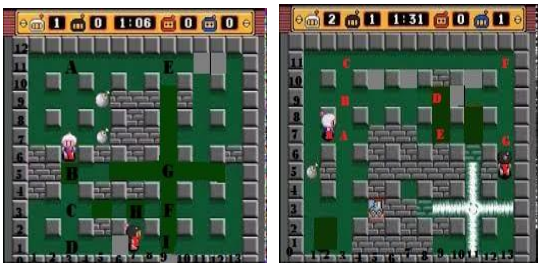
Pemain disini harus mencari ke semua jalur dengan menghancurkan tembok abu abu yang menutupi pintu ke level selanjutnya, selain itu pemain di single player diharuskan menghindari monster yang ada di map, jika pemain terkena monster tersebut maka akan mati. Berikut adalah algoritma pemanfaatan BFS untuk menentukan nilai minimum agar menemukan pintu ke level selanjutnya.

Dimana yang menjadi heuristic disini ialah jarak player dengan target(musuh) sebanyak maksimal 3 blok atau 3 dinding abu abu. Dimana saat kita menaruh bom yang mempunyai range sejauh maksimal 3 blok tersebut maka target kita tidak akan bergerak dan kabur menghindar dari bom lalu target mati.

b. Mencari Jalur Terdekat musuh dengan A star

Pada pembahasan mencari jarak terdekat dengan musuh, mana jalur yang efisien untuk menghemat waktu dalam permainan. Saat player sudah mengetahui jarak terdekat maka akan lebih mudah untuk menaruh bom dengan jarak 3blok dengan musuh sehingga musuh bisa terkunci.
Berikut adalah Algoritma (terlampir) yang sudah ada sebelumnya [2].

Berikut adalah dua comtoh variasi peta mencari jarak musuh terdekat pada bomberman.



Gambar 4.1 dan 4.2 Peta player dengan heuristic jarak terdekat dengan musuh.[4]

Pada gambar 4.1 titik awal dimulai dari x dimana player berada pada kordinat (3,6), A(3,11), B(3,5), C(3,3), D(3,1), E(9,11), F(9,3), G(9,5), H(7,3), I(9,1) dan goalnya musuh di titik y dengan kordinat (7,1). Disini rute yang bisa ditempuh player dengan rute tanpa penghalang tembok abu ada 4 jalur, yaitu:

- X-A-E-G-F-I-Y
- X-A-E-G-F-H-Y
- X-B-G-F-H-Y
- X-B-G-F-I-Y
- X-B-C-H-Y

Sedangkan Pada gambar 4.2 titik awal dimulai dari x dimana player berada pada kordinat (2,7), A(3,7), B(3,9), C(3,11), D(9,9), E(9,7), F(13,11), G(13,7) dan goalnya musuh dititk y dengan kordinat (13,5). Disini rute yang bisa ditempuh player dengan rute tanpa penghalang tembok abu ada 2 jalur, yaitu:

- X-A-B-D-E-G-Y
- X-A-B-C-F-G-Y

Pada kasus ini akan diambil gambar 4.2 untuk dicari nilai cost dengan mengunakan Manhattan distance vs Euclidean distance untuk mencari jarak terdekat. Untuk perhitungan pertama kita gunakan perhitungan dengan Euclidean distance, pertama tama kita harus mencari nilai heuristicnya terlebih

dahulu dari masing-masing jalur yang dapat ditempuh.

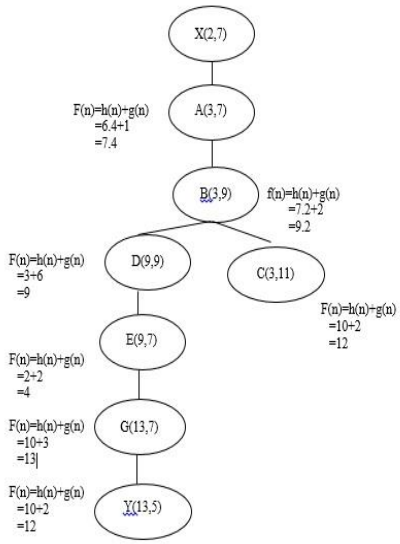
- Nilai heuristic X-A-B-D-E-G-Y
  Dimana untuk menghitung nilai heuristic digunakan rumus :

$$d(x,y)=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$

- Dari X(2,7) Ke A(3,7)
  d(x,y)= √((2-7)²+(3-7)²)=√41 =6.4
- Dari A(3,7) Ke B(3,9)
  d(x,y)= √((3-7)²+(3-9)²)=√52 =7.2
- Dari B (3,9) Ke D(9,9)
  d(x,y)= √((3-9)²+(9-9)²)=√36 =6
- Dari D(9,9) Ke E(9,7)
  d(x,y)= √((9-9)²+(9-7)²)=√4 = 2
- Dari E(9,7) Ke G(13,7)
  d(x,y)= √((9-9)²+(13-7)²)=√40 = 6.3
- Dari G(13,7) Ke Y(13,5)
  d(x,y)= √((13-7)²+(13-5)²)=√100 = 10

- Nilai heuristic X-A-B-C-F-G-Y
  - Dari X(2,7) Ke A(3,7)
    d(x,y)= √((2-7)²+(3-7)²)=√41 =6.4
  - Dari A(3,7) Ke B(3,9)
    d(x,y)= √((3-7)²+(3-9)²)=√52 =7.2
  - Dari B (3,9) Ke C(3,11)
    d(x,y)= √((3-9)²+(3-11)²)=√100 = 10
  - Dari C(3,11) Ke F(13,11)
    d(x,y)= √((3-11)²+(13-11)²)=√68 = 8.2
  - Dari F(13,11) Ke G(13,7)
    d(x,y)= √((13-11)²+(13-7)²)=√40 = 6.3
  - Dari G(13,7) Ke Y(13,5)
    d(x,y)= √((13-7)²+(13-5)²)=√100 = 10

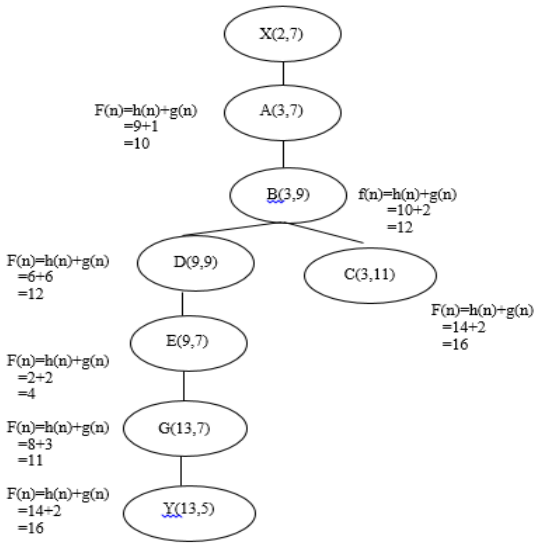
Berikut perhitungan Fn nya :



- Maka *f(n)* total yang didapat adalah = 7.4+9.2+9+4+13+12=54.6
- Jalur yang dilalui: X-A-B-D-E-G-Y

Selanjutnya kita gunakan perhitungan Heuristic dengan Manhattan distance :

- Dari X(2,7) Ke A(3,7)
  D(x,y)=(2-7)+(3-7) = 9
- Dari A(3,7) ke B(3,9)
  D(x,y)=(3-7)+(3-9) = 10
- Dari B(3,9) Ke D(9,9)
  D(x,y)=(3-9)+(9-9) = 6
- Dari D(9,9) Ke E(9,7)
  D(x,y)=(9-9)+(9-7) = 2
- Dari E(9,7) Ke G(13,7)
  D(x,y)=(9-7)+(13-7) = 8
- Dari G(13,7) Ke Y(13,5)
  D(x,y)=(13-7)+(13-5) = 14
- Dari B(3,9) Ke C(3,11)
  D(x,y)=(3-9)+(3-11) = 14
- Dari C(3,11) Ke F(13,11)
  D(x,y)=(3-11)+(13-11) = 10
- Dari F(13,11) Ke G(13,7)
  D(x,y)=(13-11)+(13-7) = 8



- Maka *f(n)* total yang didapat adalah = 10+12+12+4+11+16=65
- Jalur yang dilalui: X-A-B-D-E-G-Y

IV Kesimpulan

Pada pembahasan diatas dapat kita lihat hasilnya bahwa dengan metode BFS kita dapat lebih mudah mencari pintu ke level selanjutnya meskipun akan memerlukan waktu yang berbeda dikarena dalam metode ini kita harus mencari disemua jalur, karena banyaknya dinding yang ada di map dan menutupi pintu tersebut tapi diusahakan kita tidak kehabisan waktu dalam menemukan pintu tersebut.

Sedangkan dengan metode A star kita dapat mencarai jalur tercepat mendekati musuh agar kita bisa menaruh bom didekat musuh sejauh maksimal 3blok sehingga lawan akan terkunci posisinya dan tidak bisa kabur kemana mana. Selain itu pada metode A star digunakan dua formula perhitungan Manhattan distance vs Euclidean distance, Dimana pada kedua perhitungan diatas jalur yang ditempu sama yaitu X-A-B-D-E-G-Y namun dengan f(n) total yang berbeda. Pada Manhattan distance f(n) total 65 sedangkan pada Euclidean distance f(n) total 54.6.

I. References

[1] D. C. T. Mr. Girish P Potdar1, *COMPARISON OF VARIOUS HEURISTICSEARCH TECHNIQUES FOR FINDING SHORTEST PATH,* vol. 5, July 2014 .

[2] C. E. L. Thomas H. Cormen, *Introduction to Algorithm(Cormen, Leiserson, and Rivest),* 1990.

[3] S. K. Shrawan Kumar Sharma, *COMPARATIVE ANALYSIS OF MANHATTAN AND EUCLIDEAN DISTANCE METRICS USING A* ALGORITHM.*

[4] p. diego, "AI for Bomberman-like game/http://www.dreamincode.net/forums/topic/273185-ai-for-bomberman-like-game/.," 2 April 2012. [Online].

[5] Pearl, *Artificial Intelligence/Search/Heuristicsearch/Best-firstsearch,* 1984..

[6] Manuel Antonio Da Cruz Lopes, *BOMBERMAN AS AN ARTIFICIAL INTELLIGENCE PLATFORM,* 2016.

[7] Manta Paul, *Giving a Bomberman AI intelligent bomb placement,* 11 maret 2012.

[8] R. B. Hart P. E Nilsson N. J, *A Formal Basis for the Heuristic Determination of Minimum.*

LAMPIRAN

Source code yang telah ada untuk metode bfs[4].

```
Int FindWall(Map $m,
Pair<int,int> pos) {
Bool visited[MAX] [MAX];
iniBool(visited);

int = post.first;
          int j = post.second;
          queue<pair<int,int>> q;
          queue<int> qn;
          int n = 0;
          bool found = false;

          q.push(pos);
          qn.push(n);
          visited[i][j] = true;

          while(!q.empty() && !found){
             pair<int,int> currentIdx
             = q.front();
             n = qn.front();

             i = currentIdx.first;
             j = currentIdx.second;
             q.pop();
             qn.pop();

             if ((i+1 < m.GetRow()) &&
             !visited[i+1][j] &&
             !danger[i+1][j] &&
             (m[i+1][j] != '#') &&
             !isalpha(m[i+1][j]))
             {
                if (m[i+1][j] == '+')
                   found = true;
                else{
                   q.push(pair<int,in
                   t>(i+1, j));
                   qn.push(n+1);
                   visited[i+1][j] =
                   true;
                }
             }
             if ((i-1 >= 0) &&
             !visited[i-1][j] &&
             !danger[i-1][j] && (m[i-
             1][j] != '#') &&
             !isalpha(m[i-1][j]))
             {
                if (m[i-1][j] == '+')
                   found = true;
                else{
                   q.push(pair<int,
                   qn.push(n+1);
                   visited[i+1][j] =
                   true;
                }
             }


             if ((i-1 >= 0) &&
             !visited[i-1][j] &&
             !danger[i-1][j] && (m[i-
             1][j] != '#') &&
             !isalpha(m[i-1][j]))
```

```
             {
                if (m[i-1][j] ==
                '+')
                   found = true;
                else{
                   q.push(pair<int,
                   int>(i-1, j));
                   qn.push(n+1);
                   visited[i+1][j]
                   = true;
                }
             }

             if ((j+1 < m.GetCol())
             && !visited[i][j+1] &&
             !danger[i][j+1] &&
             (m[i][j+1] != '#') &&
             !isalpha(m[i][j+1]))
             {
                if (m[i][j+1] ==
                '+')
                   found = true;
                else{
                   q.push(pair<int,
                   int>(i, j+1));
                   qn.push(n+1);
                   visited[i][j+1]
                   = true;
                }
             }

             if ((j-1 >= 0) &&
             !visited[i][j-1] &&
             !danger[i][j-1] &&
             (m[i][j-1] != '#') &&
             !isalpha(m[i][j-1]))
             {
                if (m[i][j-1] ==
                '+')
                   found = true;
                else{
                   q.push(pair<int,
                   int>(i, j-1));
                   qn.push(n+1);
                   visited[i][j-1]
                   = true;
                }
             }
          }

          if (found)
             return n;
          else
             return -1;
       }
```