Algoritma A* Memanfaatkan *Navigation Meshes* dalam *3 Dimensional Pathfinding*

Edwin Kumara Tandiono, 13515039
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515039@std.stei.itb.ac.id – tandiono98@gmail.com

***Abstract*—Berkembangnya teknologi grafika komputer turut diikuti dengan pesatnya kemunculan berbagai jenis perangkat lunak yang memanfaatkan lingkungan 3 dimensi dalam penggunaannya. Salah satu kategori yang ramai di pasaran adalah video game, khususnya yang bertema *open-world, first person shooter,* ataupun *role-playing game*. Ketiganya memiliki sebuah fitur yang mirip—kecerdasan buatan—yang memungkinkan karakter selain pemain seolah tampak hidup dan bergerak secara logis. Makalah ini akan menelusuri lebih jauh bagaimana sebuah entitas-bukan-pemain dapat bergerak ke sebuah titik dalam peta dengan efektif tanpa melalui jalur yang tidak diperlukan dengan menggunakan algoritma A* beserta bagaimana representasinya dalam lingkungan 3 dimensi.**

Keywords—navmesh*, algoritma A*, jalur, *pathfinding

I. PENDAHULUAN

Pathfinding atau Pencarian Jalur adalah suatu strategi untuk mendapatkan pilihan jalur terpendek antara titik awal dengan titik tujuan dari sekumpulan jalur yang tersedia. Biasanya, teknik pencarian ini banyak digunakan dalam pemecahan masalah yang meliputi graf dan struktur sejenisnya, seperti perancangan jaringan komputer, jalur transportasi, dan lain – lain. Tersedia banyak algoritma dalam perancangan *pathfinding* yang bersifat mengurangi kompleksitas data yang harus ditelusuri, sehingga perancang dapat menghindari pencarian secara *brute-force*.

Dalam dunia video game yang didesain dalam lingkungan 3 dimensi, diperlukan adanya abstraksi untuk merepresentasikan sebuah dunia 3 dimensi dalam bentuk sedemikian rupa sehingga algoritma dapat diterapkan layaknya pada sebuah graf. Abstraksi ini berguna untuk memenuhi tuntutan salah satu fitur game yang kini hampir selalu ada, yakni kecerdasan buatan. Untuk game – game bertema *open-world*, *role-playing game*, ataupun *first person shooter*, salah satu kemampuan yang wajib dimiliki oleh sebuah kecerdasan buatan adalah kemampuan untuk bergerak dari suatu titik ke titik tujuan secara efektif dan masuk akal—artinya, seluruh jalur yang ditempuh memang benar menuju ke titik tujuan dengan memperhitungkan adanya halangan atau jalan buntu. Untuk mempermudah penjelasan, selanjutnya entitas yang memiliki kemampuan *pathfinding* ini kita sebut saja sebagai mob.

Terdapat banyak cara untuk melakukan *pathfinding*, yakni dengan memanfaatkan algoritma – algoritma yang dapat menyeleksi pilihan rute terpendek, seperti BFS, DFS, Greedy, A*, Algoritma Dijkstra, dan lain – lain. Pada awalnya, seluruh algoritma ini populer digunakan untuk membantu menentukan jalur mob. Tetapi seiring bertambahnya kemunculan game – game yang menggunakan sistem *pathfinding*, Algoritma A* terlihat lebih banyak membantu dibandingkan dengan algoritma lainnya. Salah satu alasan yang mungkin, Algoritma A* mengikutsertakan perhitungan heuristik yang mengarah ke tujuan, sehingga pemilihan jalur lebih akurat daripada algoritma lainnya.

II. LANDASAN TEORI

1. Fungsi Heuristik

   Fungsi Heuristik memiliki peranan penting dalam proses pencarian jalur terpendek, yakni sebagai fungsi yang memberikan nilai estimasi harga dari suatu titik dalam peta ke titik tujuan. Adanya fungsi heuristik membantu kita untuk menentukan nilai kepercayaan sebuah titik akan membawa kita ke tujuan dengna harga paling sedikit.

   Meski berperan penting dalam menentukan pilihan jalur terpendek, kita tidak dapat memungkiri bahwa fungsi heuristik hanya memberikan estimasi dari nilai yang kita inginkan. Heuristik sendiri berarti sebuah pendekatan, dimana kita memanfaatkan fungsi ini untuk membantu mengarahkan kita ke solusi yang sebenarnya. Hal ini berarti terdapat kemungkinan dalam beberapa kasus dimana pemanfaatan fungsi heuristik justru akan memperlambat proses perhitungan algoritma, walaupun jarang.

2. Algoritma A*

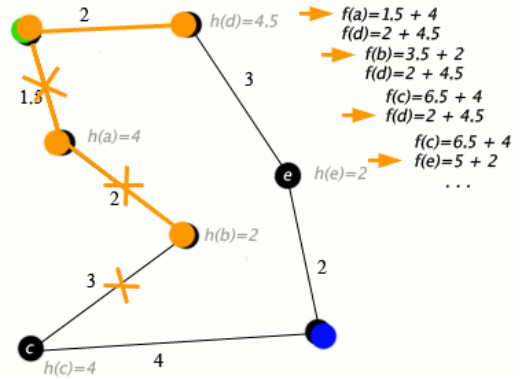
   Seperti halnya algoritma *pathfinding* yang lain, Algoritma A* akan melakukan pengecekan *cost* untuk setiap *node* yang dijadikan kandidat jalur terpendek. Perhitungan harga dari *node* dilakukan dengan fungsi berikut.

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

dimana
$f(n)$ = harga total perjalanan melalui n ke tujuan
$g(n)$ = harga dari titik awal ke n
$h(n)$ = perkiraan harga dari n ke tujuan

Algoritma A* adalah satu dari beberapa algoritma yang menerapkan sifat heuristik. Hal ini berarti Algoritma A* akan memperhitungkan tidak hanya sudah seberapa jauh jalur yang ditempuh, tetapi juga seberapa menjanjikan sebuah *node* dalam graf untuk dipilih sehingga jalur terpendek didapat.

Salah satu sifat dari Algoritma A* adalah algoritma ini akan selalu menghasilkan jalur terpendek yang ada jika memang terdapat sebuah jalur antara titik awal dengan tujuan dalam sebuah peta dan jumlah jalur keseluruhan tidak tak terbatas. Selain itu, algoritma ini juga bersifat optimal jika heuristiknya bersifat *admissible*, dimana perkiraan harga yang digunakan [$h(n)$] lebih kecil atau sama dengan harga sebenarnya yang terdapat pada peta [$h^*(n)$]. Sifat ini membuat Algoritma A* lebih unggul dibandingkan dengan algoritma lainya yang juga menerapkan sifat heuristik.



Gambar 2.1: Contoh penerapan Algoritma A*
https://upload.wikimedia.org – Akses 16 mei 2017

Dalam Algoritma A*, fungsi heuristik berperan sebagai salah satu penentu harga sebuah *node* bersama fungsi $g(n)$, sehingga :

- Jika $h(n)$ = 0, Algoritma A* tidak berbeda dengan Algoritma Dijkstra, menyebabkan jalur terpendek pasti didapatkan meskipun waktu komputasinya relatif lama.
- Jika $h(n) \leq h^*(n)$, Algoritma A* akan bersifat optimal.
- Jika $h(n) = h^*(n)$, Algoritma A* akan menghasilkan jalur terbaik dengan waktu komputasi relatif cepat.
- Jika $h(n) > h^*(n)$, Algoritma A* bisa saja menghasilkan jalur yang bukan terpendek, tetapi waktu komputasinya lebih cepat.
- Jika $h(n)$ relatif lebih besar dibandingkan $g(n)$, Algoritma A* tidak jauh berbeda dengan Algoritma *greedy-Best First Search*.

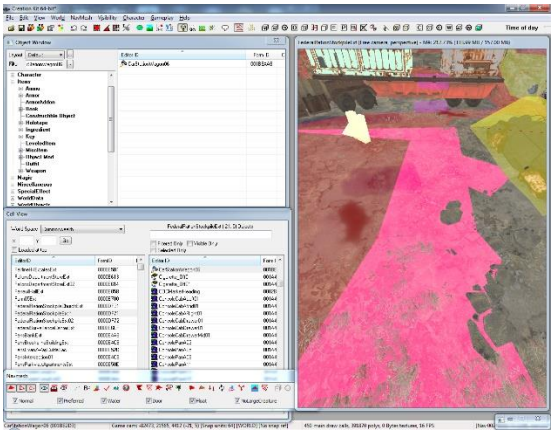
3. Navigation Meshes

Navigation Meshes (untuk seterusnya disingkat navmesh saja) adalah representasi lingkungan yang akan kita gunakan untuk penerapan algoritma *pathfinding* kita. Navmesh adalah sebuah bidang yang bisa berupa beragam bentuk, seperi segitiga, segiempat, ataupun poligon, yang memiliki sudut (*vertices*) dan sisi (*sides*). Bidang ini secara sederhana adalah area dimana mob dapat bergerak secara bebas tanpa terhalang oleh sesuatu.

Gambar 2.2: Navmesh segiempat
http://steamcommunity.com – akses 16 Mei 2017

Umumnya, navmesh memiliki ukuran lebih kecil dari peta dan berada tidak diluar peta agar mob tetap berada dalam lingkungan permainan.

Navmesh dapat diciptakan secara manual oleh perancang atau dengan pembangkitan navmesh secara dinamis, yakni dengan memanfaatkan *mesh editor*, contohnya Creation Kit. Pembangkitan secara manual tentunya akan mudah dan praktis, tetapi untuk alasan akurasi dan kesesuaian, pembuatan secara manual lebih banyak digemari oleh sebagian besar perancang.



Gambar 2.3: Creation Kit
https://www.creationkit.com – akses 16 Mei 2017

Salah satu sifat dari sebuah navmesh adalah dalam seluruh poligon, setiap penghubung antara suatu titik dalam poligon dengan titik lain dalam poligon yang sama harus membentuk sebuah garis lurus. Hal ini terkait dengan keterbatasan algoritma *pathfinding* kita dalam menyeleksi jalur yang

berupa kurva. *Workaround* untuk permasalahan ini, antara lain menggabungkan beberapa poligon sehingga seolah – olah jalur yang terbentuk membentuk sebuah kurva.

Dengan memanfaatkan navmesh, kita dapat menggantikan representasi peta dalam graf menjadi sebuah bangun 3 dimensi yang terdiri atas beberapa buah poligon dalam sebuah lingkungan 3 dimensi. Representasi peta yang ditunjukkan navmesh dapat dilihat dalam ilustrasi berikut.

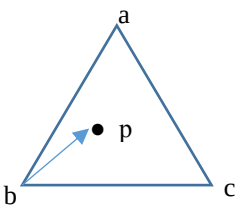
Gambar 2.4: Representasi navmesh (merah)
https://dietervanhooren.files.wordpress.com – akses 16 mei 2017

Pada gambar diatas, terlihat area *overlay* yang berwarna merah pada peta dikelilingi oleh area yang tidak diberi warna *overlay*. Area yang berwarna merah adalah bidang navmesh dari peta tersebut yang merupakan daerah dimana mob bisa bergerak bebas. Terlihat juga bagian dinding, batu, dan properti lainnya dalam peta merupakan bagian yang berada diluar navmesh.

III. *PATHFINDING* DENGAN MEMANFAATKAN ALGORITMA A* DAN NAVMESH

Penerapan Algoritma A* dalam sebuah navmesh tidaklah berbeda jauh dengan penerapannya dengan memanfaatkan graf. Seperti halnya sebuah graf, pada navmesh juga terdapat *node* – *node* yang berfungsi sebagai kandidat jalur terpendek yang akan diambil. Dalam hal ini, node yang dimaksud direpresentasikan sebagai *vertices* dari masing – masing poligon, dimana setiap *vertices* memiliki harganya masing – masing.

Langkah pertama dalam *pathfinding* adalah dengan menentukan terlebih dahulu pada poligon mana kita berada, serta pada poligon mana titik tujuan yang ingin kita capai berada. Untuk itu, kita memerlukan 2 buah data: Koordinat posisi kita sekarang beserta struktur data yang menympan daftar *vertices* dari seluruh poligon. Jika kedua sumber tersebut sudah tersedia, kiita dapat mengecek posisi kita sekarang berada dalam poligon mana dengan memanfaatkan. Sebagai contoh, untuk navmesh berbentuk segitiga yang banyak digunakan, kita dapat melakukan pengecekan titiknya dengan memmanfaatkan fungsi *cross product*. Berikut adalah perhitungan yang dilakukan.



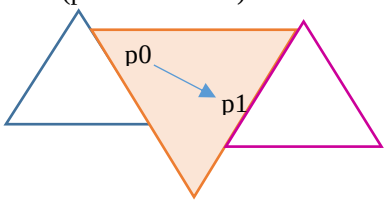
Pada gambar di samping, kita ingin mencari tahu apakah titik p berada dalam segitiga yang dibentuk oleh 3 *vertices* a, b, dan c. Ide dari perhitungan ini adalah, jika memang p berada pada bidang, maka ***ba*** dan ***bp*** akan sejajar, sehingga hasil *cross product* keduanya adalah sebuah vektor yang arahnya tegak lurus keluar bidang kertas ini. Tetapi, hal itu hanya berlaku untuk bidang 2 dimensi—bagaimana kita mengaplikasikannya pada bidang 3 dimensi? Dalam lingkungan 3 dimensi menggunakan persamaan di atas, kita akan menemukan bahwa jika *cross product* antara ***ba*** dan ***bp*** berarah yang sama dengan ***ba*** dan ***bc***, titik p berada di dalam bidang.

Maka, *pseudocode* dari perhitungan di atas:

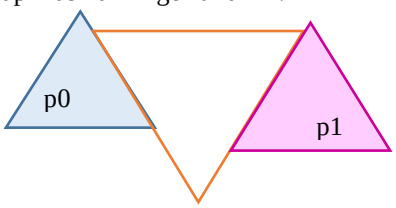
```
function pointInTriangle(p,a,b,c)-> bool

ALGORITMA
     if      (CrossProduct(a-b,p-b) =
              CrossProduct(a-b,c-b))
             then return true
     else
             return false
```

Dari perhitungan tersebut, kita mendapatkan 2 buah poligon yang berisi titik awal dan titik tujuan. Untuk mengetahui rute mana yang harus diambil, dilakukan perbandingan terhadap 2 poligon yang kita dapat tadi:

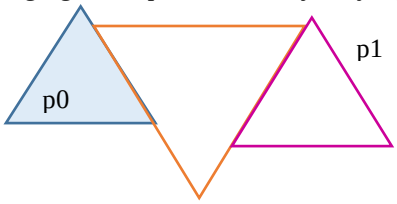
- Jika titik awal dan tujuan berada pada segitiga yang sama, jalur yang harus ditempuh dapat berupa garis lurus (problem selesai).



- Jika kedua titik berada pada segitiga yang berbeda, aplikasikan Algoritma A*.



- Jika titik awal dan/atau tujuan tidak berada pada segitiga manapun, tidak ada jalur yang bisa diambil.



Algoritma A* hanya akan kita gunakan apabila kedua titik berada pada segitiga yang berbeda.

Berikut adalah bagaimana kita menentukan *vertices* mana dalam pohon solusi yang harus kita ambil dalam memanfaatkan Algoritma A*:

1. Hidupkan *node* akar sebagai titik awal.
2. Dengan memanfaatkan fungsi pointInTriangle(p,a,b,c), kita mendapatkan nilai a, b, dan c. Hidupkan ketiganya sebagai *node* anak dari titik awal.
3. Untuk setiap *node* bukan titik awal, hitung *cost* untuk masing – masing *node*. Masukkan setiap *node* beserta *cost*-nya dalam sebuah *priority queue* terurut menaik berdasarkan *cost*. Berikut adalah fungsi yang dapat kita gunakan dalam perhitungan *cost*:

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

   dimana
   $f(n)$ = harga total perjalanan melalui n ke tujuan
   $g(n)$ = $g(n-1)$ + jarak (n-1) ke n
   $h(n)$ = jarak lurus n ke tujuan

   sehingga *pseudocode*-nya:

```
function cost(n) -> float

KAMUS LOKAL
  f,g,h : float

ALGORITMA
  if (n == 1) then
    g = 0
  else
    g = distance(n,n-1)
  h = distance(n,target)
  f = g + h
  return f
```

4. Ambil indeks pertama dari *priority queue*. Lakukan kembali langkah 2 dengan p sebagai *node* pada indeks pertama *queue*, dan a, b, c sebagai *vertices* segitiga selanjutnya.

Jika diperhatikan, kita mendapatkan nilai $h(n)$ dengan cara menghitung panjang garis yang menghubungkan *vertices* aktif dengan titik tujuan. Perhitungan ini akan selalu menghasilkan nilai $h(n)$ yang lebih kecil atau sama dengan $h^*(n)$, sehingga Algoritma A* optimal. Tentunya, ada cara lain yang lebih akurat untuk mendapatkan $h(n)$ sehingga tidak ada *node* sembarang yang digunakan, akan tetapi untuk alasan kecepatan komputasi, ada baiknya kita menggunakan perhitungan jarak lurus ini.

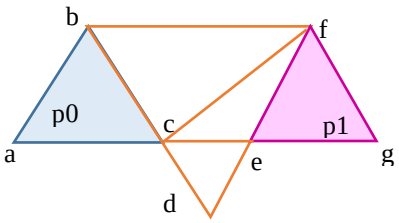
Sebagai tambahan, jarak lurus antara 2 buah titik dalam lingkungan 3 dimensi dapat dihitung dengan persamaan:

$$d = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$$

dimana
Δx = selisih antara komponen x kedua titik
Δy = selisih antara komponen y kedua titik
Δz = selisih antara komponen z kedua titik

```
function distance(a,b) -> float

ALGORITMA
  return sqrt((a.x-b.x)^2 +
              (a.y-b.y)^2 +
              (a.z-b.z)^2)
```

Untuk contoh, kita gunakan ilustrasi berikut.



Tabel titik

Titik	x	y	z
p0	1	1	0
p1	9	1	-1

Tabel *vetices*

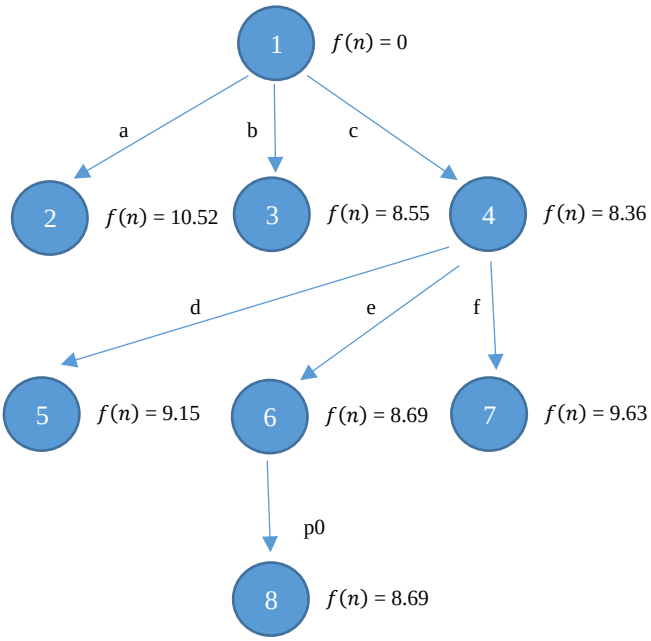
Titik	x	y	z
a	0	0	0
b	2	2	0
c	4	0	0
d	5	-1	0
e	8	0	-1
f	9	2	-1
g	12	0	-1

Pertama, cari tahu segitiga tempat p1 dan p2 berada. Dengan mengecek table *vertices*, kita mendapatkan nilai *true* untuk pointInTriangle(p0,a,b,c) dan pointInTriangle(p1,g,h,i). Setelah *node* akar kita tentukan, kita hidupkan *node* a, b, dan c dengan cost :

node a:
$g(n) = d_{p0\text{-}a} = 1.41$ $h(n) = d_{a\text{-}p1} = 9.11$
$f(n) = g(n) + h(n) = 10.52$
node b:
$g(n) = d_{p0\text{-}b} = 1.41$ $h(n) = d_{b\text{-}p1} = 7.14$
$f(n) = g(n) + h(n) = 8.55$
node c:
$g(n) = d_{p0\text{-}c} = 3.16$ $h(n) = d_{c\text{-}p1} = 5.20$
$f(n) = g(n) + h(n) = 8.36$

Jika dimasukkan dalam *priority queue*, akan didapatkan *node* c sebagai *node* aktif. kita kembali ke langkah 2 dan mendapatkan nilai *true* untuk pointInTriangle(c,b,c,f), pointInTriangle(c,c,e,f), dan pointInTriangle(c,c,d,e). Maka, apabila diteruskan, kita akan terus menerus menghidupkan *node* – *node* anak dan memilih yang memiliki *cost* paling kecil. Dengan begitu, ketika sudah sampai pada solusi, akan didapatkan sebuha jalur dari *verticesi* ke *vertices* lainnya dengan harga yang paling murah.

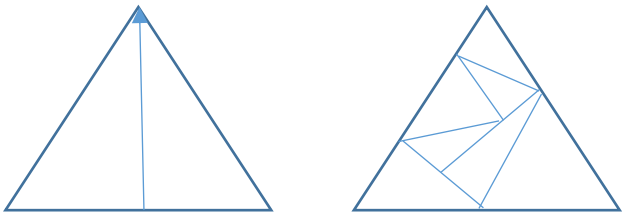
Terakhir, ketika Algoritma A* berhasil membawa kita ke salah satu *vertices* dari segitiga tempat tujuan berada, kita membangkitkan *node* ke tujuan kita dan menghentikan pencarian ke arah tersebut. Kita dapat memperlakukan titik tujuan kita layaknya sebuah *node* karena setiap navmesh yang kita buat memiliki permukaan yang datar—garis apapun yang menghubungkan kedua titik pada permukaan tersebut pasti menghasilkan garis lurus. Selanjutnya, apabila masih ada *node* lain yang harganya lebih murah, pencarian diteruskan pada *node* tersebut. Jika tidak ada, maka solusi optimal sudah ditemukan.




```
procedure A*(queue)

KAMUS LOKAL
  solved : boolean
  currentNode : Node

ALGORITMA
  while not solved do
    currentNode = queue[0]
    if target in currentTriangle
    then
       addToQueue(target)
    else
      for (a,b,c) in verticesList
        if pointInTriangle
              (currentNode,a,b,c)
        then
          addToQueue(a,b,c)
    if queue[0] = target then
      solved = true
  endwhile
```

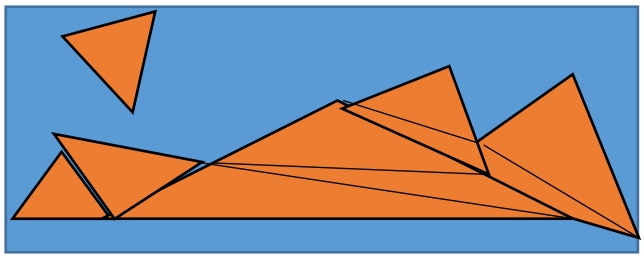
Dengan memanfaatkan algoritma di atas, kita dipastikan akan mendapatkan solusi dari permasalahan tersebut. Dalam pseudocode di atas, terlihat bahwa kita memanfaatkan pendekatan iteratif sebagai prosedur penyeleksian kandidat jalur. Akan tetapi, karena Algoritma A* juga memungkinkan pemanfaatan backtracking, algoritma ini tidak menutup kemungkinan penggunaan pendekatan rekursif dalam penerapannya.

Jika diperhatikan lebih dekat, Anda tentu menyadari bahwa penerapan Algoritma A* dalam navmesh akan menyebabkan gerakan mob yang bersifat *wall-hugging path*, dimana mob cenderung bergerak pada *side* dari navmesh yang menghubungkan 2 buah *vertices* pilihan. Hal ini dikarenakan gerakan yang paling efektif hanya akan menyentuh bagian ini saja, dan gerakan menjauhi bagian *side* justru akan memperlambat gerakan mob. Tetapi, apabila Anda ingin membuat gerakan mob yang lebih realistis, salah satu *workaround* yang dapat dilakukan adalah membuat bidang – bidang kecil lainnya di dalam sebuah bidang yang lebih besar, sehingga terdapat banyak pilihan jalur yang tidak terlihat aspek *wall-hugging*-nya, yang secara tidak langsung akan menambah waktu komputasi.



Gambar 3.1: Penambahan poligon dalam sebuah poligon induk

Penambahan poligon yang akan menambah waktu komputasi dapat diatasi dengan cara lain. Salah satunya adalah membuat navmesh yang tidak mencakup seluruh peta yang tersedia, tetapi merancangnya agar bentuk navmesh seolah – olah tidak bersifat *wall-hugging* seperti yang direpresentasikan pada ilustrasi berikut.



Gambar 3.2: Peta yang dapat dijelajahi mob (biru) dan navmesh mob (merah)

IV. SIMPULAN

Adanya sistem navmesh ini adalah salah satu bentuk abstraksi yang dapat kita gunakan untuk pengembangan teknologi komputasi yang lebih baik. Dari penggunaan algoritma perhitungan dalam graf yang ada pada video game yang beroperasi dalam lingkungan 2 dimensi, kini kita sudah dapat mengembangkan algoritma yang sama pada teknologi 3 dimensi. Algoritma komputasi ini, khususnya A*, dibantu dengan navmesh, terbukti membantu kita mengembangkan kecerdasan buatan yang

memiliki kemampuan untuk memilih jalur terpendek berdasarkan *pathfinding* yang realistis dan masuk akal.

V. UCAPAN TERIMA KASIH

Pertama-tama, puji syukur penulis ucapkan kepada Tuhan Yang Maha Esa karena berkat-Nya penulis dapat menyelesaikan makalah ini dengan baik. Penulis mengucapkan terima kasih kepada Bapak Rinaldi Munir sebagai pembimbing kuliah ini atas bimbingannya dalam pembuatan makalah ini. Tak lupa penulis juga mengucapkan terima kasih kepada keluarga dan teman – teman atas bantuannya selama pembuatan makalah ini.

DAFTAR PUSTAKA

[1] Matheus Cansian, *A* with Navigation Meshes*. Dipetik 16 Mei 2017 dari Medium: http://www.medium.com/@mscansian/a-with-navigation-meshes-246fd9e72424

[2] Xiao Cui and Hao Shi, *A*-based Pathfinding in Modern Computer Games*. Dipetik 16 Mei 2017 dari IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.1, January 2011: http://paper.ijcsns.org/07_book/201101/20110119.pdf

[3] Rinaldi Munir, *Route/Path Planning using A Star and UCS*. Dipetik 16 Mei 2017 dari IF2211-Strategi Algoritma – Semester II Tahun 2016/2017: http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2016-2017/stima16-17.htm

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, 17 Mei 2017

Edwin Kumara Tandiono
13515039