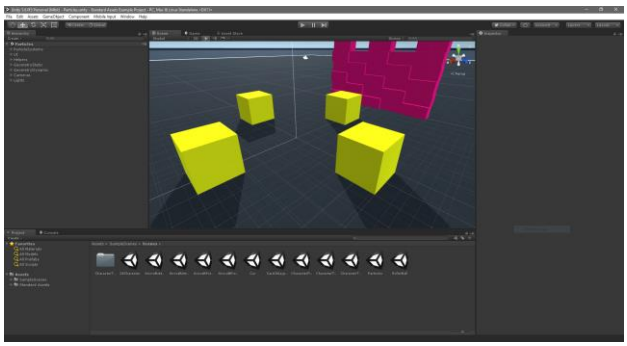
Implementasi Branch and Bound untuk Pathfinding pada Game 2D di Game Engine Unity

Winarto (13515061)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515061@std.stei.itb.ac.id

***Abstract*—Video game pada zaman sekarang mengalami perkembangan yang luar biasa jika dibandingkan dengan video game pada 20 tahun yang lalu. Perkembangan dari segi visual, grafis, *physic*, dll sangat pesat dan video game juga mencoba untuk mensimulasikannya sesuai dengan kehidupan nyata. Di satu sisi video game semakin mendekati kehidupan nyata, di sisi lain orang-orang haus akan tipe video game di 20 tahun silam di mana video game masih dalam bentuk 2D. Baik video game 3D maupun 2D memiliki faktor-faktor yang menentukan kesuksesannya dalam pasar global, salah satu faktornya adalah *Artificial Intelligence*. Makalah ini akan membahas tentang implementasi algoritma *branch and bound* untuk menentukan rute dari suatu titik ke titik tujuan pada tipe game 2D di dalam Game Engine Unity.**

Keywords—pathfinding, branch and bound, game engine, unity, game 2D

I. PENDAHULUAN



Gambar 1 Antarmuka Game Engine Unity
Sumber: Dokumen pribadi

Video game merupakan proses pengembangan sebuah *software* yang terdiri dari tampilan atau gambar, musik, dan tentunya mekanik permainan itu sendiri. Pengembangan video game memerlukan perencanaan yang matang untuk mendapatkan hasil yang diharapkan dengan pengeluaran yang optimal dan dalam waktu yang ditentukan. Membangun sebuah video game dari nol berarti harus membangun *physics engine*, *rendering engine*, *audio engine*, dan hal-hal lain yang dibutuhkan oleh game tersebut atau dengan kata lain harus membangun dasar atau *engine* terlebih dulu. Membangun *engine* dari awal akan memakan biaya dan waktu yang lama yang seharusnya dapat dialokasi untuk hal lain. Oleh karena itu, digunakanlah *game engine* untuk mempercepat waktu dan menghemat biaya.

Salah satu *game engine* komersial di pasaran dan yang populer digunakan adalah *game engine* Unity. Unity merupakan alat yang dapat digunakan untuk membangun video game baik 3D maupun 2D sehingga *game engine* ini sangat *powerful* jika digunakan dengan benar. Unity banyak digunakan oleh perusahaan-perusahaan independent karena biayanya yang terjangkau dan proses *prototyping* yang cepat di Unity. Terkadang solusi yang ditawarkan oleh Unity tidak sesuai dengan kebutuhan yang diinginkan oleh pengguna. Oleh karena itu, solusi umum yang diambil adalah membangun solusi sendiri untuk memenuhi kebutuhan tersebut. Dalam kasus ini adalah membangun pathfinding untuk game 2D di Unity.

II. TEORI DASAR

A. Algoritma Branch and Bound

Branch and Bound merupakan suatu algoritma yang mirip dengan algoritma A* dan algoritma ini umum digunakan untuk masalah optimisasi yaitu meminimalkan atau memaksimalkan sebuah fungsi objektif tanpa melebihi batas yang ditentukan. Metode ini merupakan metode pencarian di dalam ruang solusi secara sistematis di mana ruang solusi dibagi ke dalam pohon ruang status. Ruang solusi pada *branch and bound* dibangun dengan skema BFS dan untuk optimisasinya ditambahkan cost untuk setiap simpul dan simpul yang akan diekspansi berdasarkan cost yang paling kecil.

Setiap simpul pada pohon diberikan nilai ongkos:

c(i) = nilai taksiran lintasan termurah dari simpul status i ke status tujuan

c(i) merupakan batas bawah (*lower bound*) dari onkos pencarian solusi di simpul i. Ongkos ini merupakan fungsi pembatas, dengan kata lain jika ada simpul yang tidak mengarah ke solusi, maka simpul tersebut tidak akan diekspansi.

Fungsi heuristik untuk menghitung nilai ongkos taksiran secara umum dinyatakan sebagai berikut:

c(i) = $f(i) + g(i)$

dalam hal ini,

c(i) = ongkos simpul ke i
f(i) = ongkos mencapai simpul i dari akar
g(i) = ongkos mencapai simpul tujuan dari simpul i

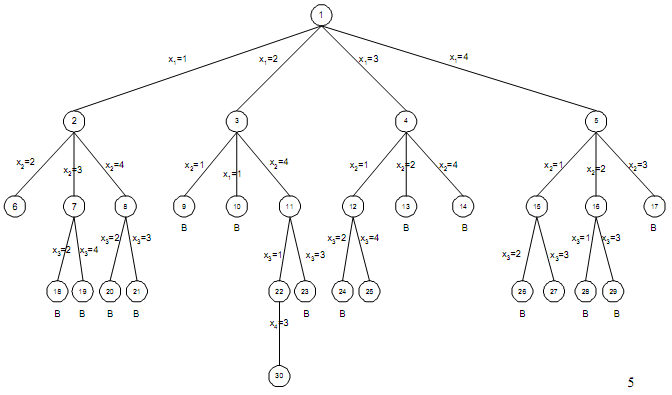
Langkah algoritma *Branch and Bound* secara umum adalah sebagai berikut:

1. Memasukkan simpul akar ke dalam antrian Q. Jika simpul akar adalah simpul solusi, maka solusi telah ditemukan dan hentikan pencarian.
2. Jika Q kosong, maka tidak terdapat solusi dan hentikan pencarian.
3. Jika Q tidak kosong, maka pilih simpul i dari antrian Q yang mempunyai nilai c(i) paling kecil. Jika terdapat beberapa simpul i yang memenuhi, maka dipilih satu secara sembarang.
4. Jika simpul i adalah simpul solusi, berarti simpul solusi telah ditemukan, hentikan pencarian. Jika simpul i bukan sebuah simpul solusi, maka ekspansi semua simpul anak dari simpul i. Jika simpul i tidak memiliki simpul anak, maka kembali ke langkah 2.
5. Untuk setiap anak j pada simpul i, dihitung c(j), dan masukkan semua anak-anak tersebut ke dalam antrian Q terurut dari nilai terkecil.
6. Kembali ke langkah 2.

Kriteria pemangkasan sebuah simpul secara umum pada algoritma *Branch and Bound*:

1. Nilai ongkos sebuah simpul tidak lebih baik dari nilai ongkos sampai ke simpul i.
2. Simpul tidak merepresentasikan solusi yang layak karena terdapat batasan yang dilanggar.
3. Solusi yang layak pada suatu simpul hanya terdiri dari satu titik di mana tidak terdapat pilihan lain. Mengambil satu nilai dari fungsi objektif yang tersedia sebagai solusi terbaik.

Salah satu contoh penerapan algoritma *branch and bround* adalah pada persoalan 4-ratu.



Gambar 2 Pohon ruang status dengan metode BFS
Sumber: Slide Kuliah IF2211 Branch and Bound

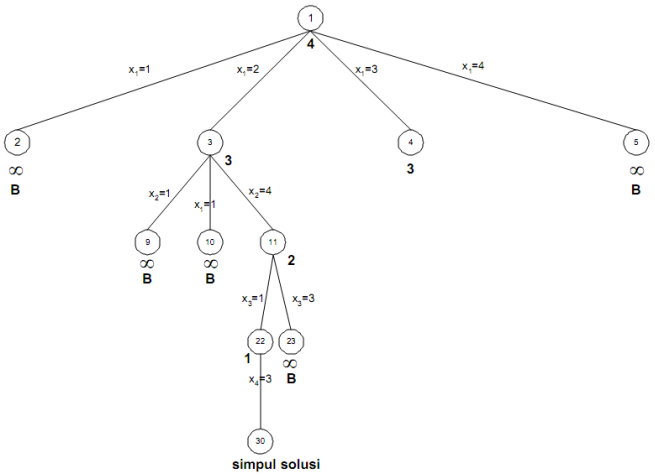
Pencarian solusi pada metode BFS di atas dapat dipercepat dengan menggunakan algoritma B&B. Caranya adalah dengan memilih simpul hidup berdasarkan nilai ongkos. Setiap simpul diberikan sebuah ongkos yang menyatakan nilai batas. Untuk setiap simpul i, nilai batas berupa:

1. Jumlah simpul dalam suatu upapohon i yang perlu dibangkitkan sebelum simpul solusi ditemukan.
2. Panjang lintasan dari suatu simpul i ke simpul solusi.

Misalnya digunakan ukuran (2), maka:

- Memberikan nilai ongkos 4 pada simpul akar (panjang lintasan dari simpul 1 ke 30 adalah 4).
- Memberikan nilai ongkos 3 pada simpul 3 dan 4 (simpul solusi juga terdapat pada upa-pohon 4)
- Simpul 2 dan 5 diberi nilai ongkos ∞ (karena upa-pohon tidak mengandung simpul solusi)

Menggunakan strategi pencarian berdasarkan nilai terkecil, di mana simpul-simpul hidup diurutkan dari biaya kecil ke biaya besar. Dari simpul-simpul yang telah diurutkan akan dipilih simpul-E yaitu simpul dengan biaya terkecil untuk di ekspansi.



Gambar 3 Pohon ruang status dengan metode B&B
Sumber: Slide Kuliah IF2211 Branch and Bound

Dengan demikian,

1. Ekspansi dari simpul 1 menghasilkan simpul hidup dengan urutan: 3,4,2,5. Dalam kasus ini Simpul-E adalah 3.
2. Ekspansi simpul 3 membangkitkan simpul 9, 10, 11. Nilai ongkos simpul 9 dan 10 diberi ∞ karena upa-pohon tidak menuju simpul solusi. Simpul 11 diberi nilai ongkos 2. Urutan simpul hidup menjadi 11,4,2,5,9,10. Simpul-E pada kasus ini adalah simpul 11.
3. Ekspansi dari simpul 11 membangkitkan simpul 22 dan 23. Simpul 23 diberikan nilai ongkos ∞ karena upa-pohonnya tidak menuju simpul solusi dan simpul 22 diberi nilai ongkos 1. Urutan simpul hidup menjadi 22,4,2,5,9,10,23. Simpul-E pada kasus ini adalah simpul 22.
4. Simpul 22 diekspansi dan membangkitkan simpul 30. Simpul 30 merupakan solusi sehingga solusi pertama pada persoalan 4-Ratu ditemukan.

Proses pembentukan pohon ruang status dengan algoritma B&B diperlihatkan pada Gambar 3.

B. Unity

Unity merupakan salah satu dari sekian banyak *game engine* yang tersedia kepada publik pada saat ini. Proses prototyping dapat dilakukan dengan cepat dengan memanfaatkan *game engine*. *Game engine* unity menyediakan banyak fitur untuk pengguna mulai dari animasi, graphics, 2D, 3D Physics, optimisasi, scripting, audio, dll. Bahasa pemrograman yang digunakan untuk membangun unity adalah C++, untuk pengguna sendiri terdapat 3 pilihan bahasa pemrograman yang terdiri dari C#, Javascript, dan Boo. Contoh fitur yang terdapat pada animasi, grafis dan 2D adalah sebagai berikut

Fitur animasi di unity terbagi menjadi:

- **Retargeting**: fitur ini berguna untuk *retargeting* animasi pada model karakter *humanoid*, artinya animasi set untuk suatu model dapat dengan mudah dipindahkan ke model lain.
- **Blend Trees**: untuk memperlembut transisi antar animasi.
- **State machines**: untuk menetapkan state dari animasi.
- **Integrated animation editor**: membuat animasi langsung di dalam game engine unity.
- **Inverse Kinematics**: Mengunci posisi dari suatu bagian model di suatu koordinat tertentu.
- **Sync Layers and Additional Curves**: Mengalokasikan suatu kurva animasi pada klip animasi tertentu.

Fitur grafis pada game engine unity terdiri dari:

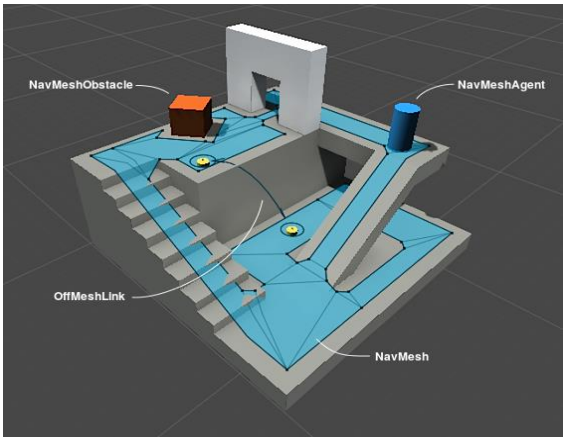
- **Physically-Based Shading**: adalah suatu shader yang memberikan ketelitian tinggi terhadap platform *mobile*, *desktop*, dan *console*.
- **Shuriken Particle System**: untuk menciptakan particle system dengan menggunakan sistem kurva.
- **Enlighten**: *Real-time Global Illumination* mengkalkulasikan bagaimana kelakuan cahaya yang jatuh pada permukaan dinamis.
- **Low-Level Rendering Access**: memberi akses kepada pengguna untuk mengimplementasi teknik *rendering* sendiri.
- **Dynamic Fonts with Markup**: menggunakan HTML markup untuk mengubah ukuran font, ketebalan, dll.
- **Static Batching**: Membuat *batch* geometri statik supaya CPU tidak menghabiskan waktu menciptakan *batch* yang sama.
- **Render-to-Texture Effects**: berguna untuk menjadikan *view* dari camera menjadi texture yang dapat digunakan pada sebuah permukaan secara *real-time*.
- **Full-Screen Post-Processing Effects**: seperti *depth of field*, *motion blur*, *bloom*, *color correction*, dan efek *post-processing* lainnya.

Fitur 2D pada game engine unity terdiri dari:

- **Automatic sprite slicing**: memotong spritesheet menjadi beberapa sprite yang terpisah.
- **Automatic sprite animation**: membentuk animasi dari beberapa sprite yang dipilih
- **Alpha cutout**: Mengoptimisasi *fillrate* dan meningkatkan jumlah piksel.
- **Sprite packer**: Menggabungkan beberapa sprite menjadi satu kumpulan sprite.
- **2D physics**: simulasi *physics* terdiri dari gaya, massa, dll.

Selain itu, unity merupakan *game engine* multiplatform jadi sangat mudah untuk mengimplementasi video game pada platform-platform terkenal.

C. Pathfinding di Unity



Gambar 4 Navigation pada unity
Sumber: https://docs.unity3d.com/540/Documentation/uploads/Main/NavMeshCover.png

Salah satu fitur yang menarik di unity adalah sistem *pathfinding* yang sudah terdapat di dalam unity. Dengan sistem navigasi dan *pathfinding* unity memungkinkan pengguna untuk menciptakan karakter yang dapat bergerak di dalam dunia game dengan cerdas. Hal ini dapat dicapai dengan menggunakan *navigation meshes* atau navmesh yang terbentuk secara otomatis pada geometri game.

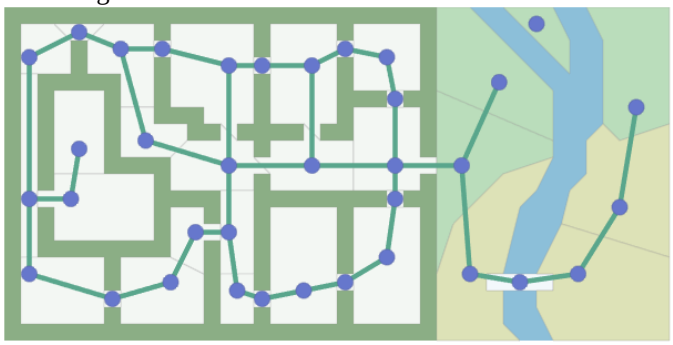
Sistem navigasi di unity terdiri dari 4 bagian:

1. **NavMesh** atau navigation mesh merupakan struktur data yang berisi daerah yang boleh dijalani pada permukaan bidang dan bertanggung jawab memberikan informasi lintasan dari suatu lokasi ke lokasi yang dituju. Struktur data ini terbentuk secara otomatis dari geometri *game*.
2. **NavMesh Agent** merupakan sebuah komponen yang membantu pengguna untuk menciptakan sebuah karakter yang dapat menghindar dari satu sama lain sambil berjalan menuju tujuan.
3. **Off-Mesh Link** merupakan komponen yang menggabungkan lintasan navigasi yang tidak dapat direpresentasikan sebagai permukaan yang dapat dilalui. Contoh membuka pintu sebelum berjalan melaluinya.
4. **NavMesh Obstacle** merupakan komponen yang menandakan daerah yang tidak dapat dilalui dan *agent* akan berusaha sebisa mungkin untuk menghindari *obstacle* tersebut.

Meskipun solusi yang ditawarkan unity merupakan solusi yang mangkus dan sangkil, akan tetapi solusi ini hanya dapat diterapkan pada objek 3D di unity sehingga untuk objek 2D yang ingin memakai sistem navigasi unity harus diakali dengan membuat peta dalam bentuk 3D. Pada kasus lebih kompleks, pendekatan ini tidak efektif dan pada kasus tertentu dapat menimbulkan *bug*. Oleh karena itu, pengguna diharuskan menciptakan solusi sendiri untuk *pathfinding*.

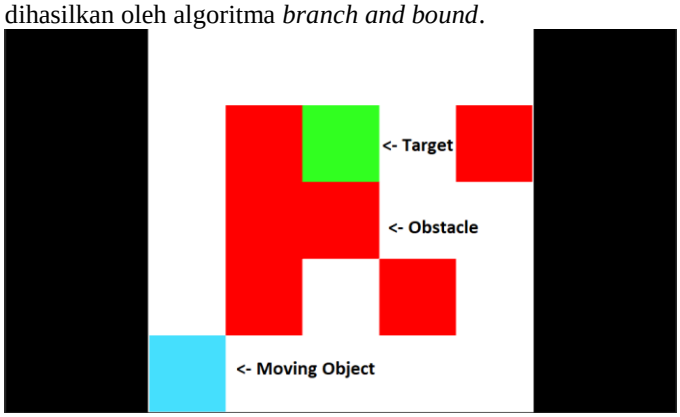
III. Persiapan Lingkungan Unity Sebelum Implementasi Algoritma Branch and Bound

Pemahaman tentang data dibutuhkan ketika berbicara tentang algoritma mulai dari input, output sampai pengorbanan yang harus dilakukan. Input dalam kasus ini adalah graf. Graf terdiri dari sekumpulan simpul yang terhubung oleh sisi-sisi diantara mereka.

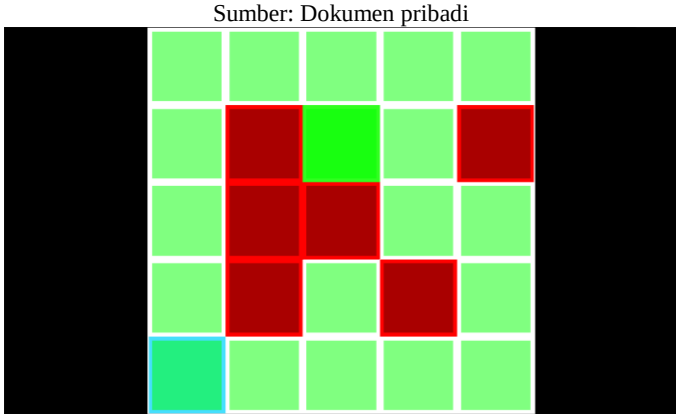


Gambar 5 Ilustrasi graf yang digunakan pada branch and bound
Sumber: http://www.redblobgames.com/pathfinding/

Dengan melihat contoh pada gambar 5, algoritma *branch and bound* hanya akan menyimpan data tentang simpul dan sisi dari graf. Informasi lain seperti objek dinding tidak akan disimpan. Output dalam kasus ini adalah lintasan yang dihasilkan oleh algoritma *branch and bound*.



Gambar 6 Contoh persoalan branch and bound
Sumber: Dokumen pribadi



Gambar 7 Pemotongan bidang menjadi matriks ukuran 5x5
Sumber: Dokumen pribadi

Diberikan contoh pada gambar 6, di mana objek biru (*moving object*) harus mencari lintasan untuk mencapai daerah berwarna hijau (*target*) sambil menghindari objek berwarna merah (*obstacle*). Langkah pertama yang perlu dilakukan adalah membagi bidang menjadi matriks berukuran 5 x 5.

Setiap lintasan yang dapat dilalui ditandai dengan warna hijau, dan yang tidak dapat dilalui ditandai dengan warna merah. Matriks ini akan merepresentasikan graf dengan setiap koordinat matriks merupakan simpul dan setiap koordinat yang bertetangga dianggap bersisian.

IV. Penerapan Algoritma Branch and Bound

Setelah persialan selesai dilakukan selanjutnya akan dilakukan tinjau kasus. Diketahui posisi awal berada pada lokasi objek berwarna biru dan posisi dari lokasi yang ingin dituju berupa objek hijau juga diketahui. Oleh karena itu, dapat dipakai pendekatan heuristik dengan nilai ongkos sebagai berikut:

$$c(i) = f(i) + g(i)$$

dalam hal ini,

$c(i)$ = ongkos simpul ke i
$f(i)$ = ongkos mencapai simpul i dari akar
$g(i)$ = ongkos dihitung dengan *manhattan distance*.

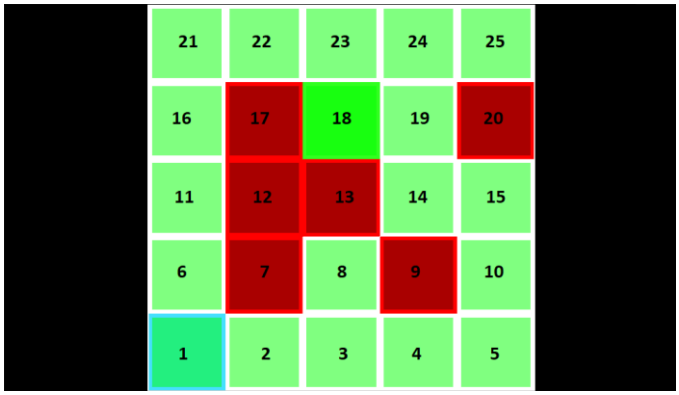
Dengan melakukan pencarian secara heuristik, maka ekspansi sebuah simpul cenderung mengarah ke tujuan. Berikut merupakan pseudocode untuk implementasi *branch and bound* dengan pendekatan heuristic:

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = {}
cost_so_far = {}
came_from[start] = None
cost_so_far[start] = 0

while not frontier.empty():
   current = frontier.get()

   if current == goal:
      break

   for next in graph.neighbors(current):
      new_cost = cost_so_far[current] +
graph.cost(current, next)
      if next not in cost_so_far or new_cost
< cost_so_far[next]:
         cost_so_far[next] = new_cost
         priority = new_cost +
heuristic(goal, next)
         frontier.put(next, priority)
         came_from[next] = current
```

Dikutip dari http://www.redblobgames.com

Langkah-langkah pencarian lintasan untuk mencapai tujuan di dalam unity dimulai dengan menentukan titik awal. Titik awal berada pada posisi objek berwarna biru yaitu titik(0,0). Semakin ke kanan maka nilai x semakin bertambah, demikian juga dengan nilai y, semakin ke atas maka nilai y

semakin bertambah. Penomoran setiap simpul adalah sebagai berikut:



Gambar 8 Penomoran untuk setiap simpul
Sumber: Dokumen pribadi

Dari gambar 8 dapat diperoleh tabel heuristik sebagai berikut:

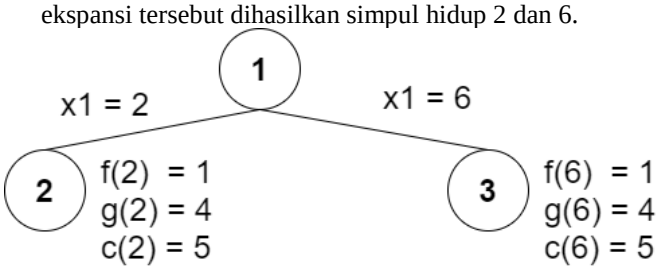
No. Simpul	g(x)
1	5
2	4
3	3
4	4
5	5
6	4
7	3
8	2
9	3
10	4
11	3
12	2
13	1

No. Simpul	g(x)
14	2
15	3
16	2
17	1
18	0
19	1
20	2
21	3
22	2
23	1
24	2
25	3

Tabel 1 Tabel heuristik dari gambar 9

Langkah-langkah pembangkitan simpul sebagai berikut:

1. Pencarian dimulai dengan mengekspansi simpul 1. Dari ekspansi tersebut dihasilkan simpul hidup 2 dan 6.

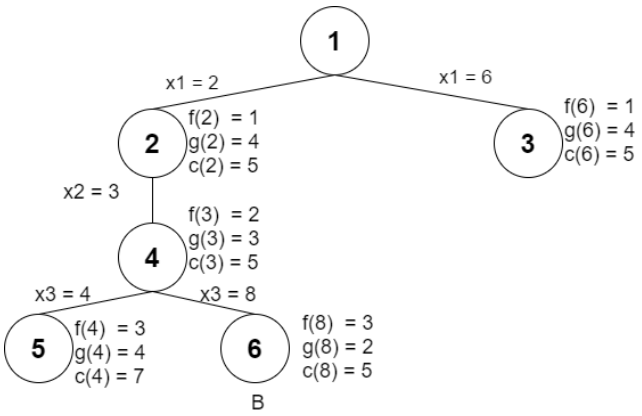


Gambar 9 Ekspansi simpul 1
Sumber: Dokumen pribadi

Nilai ongkos dari simpul 2 dan 6 adalah 5. Karena nilai ongkos sama maka akan diurutkan berdasarkan simpul paling kecil ke simpul paling besar. Urutan simpul hidup menjadi 2,6. Simpul-E pada kasus ini adalah simpul 2.

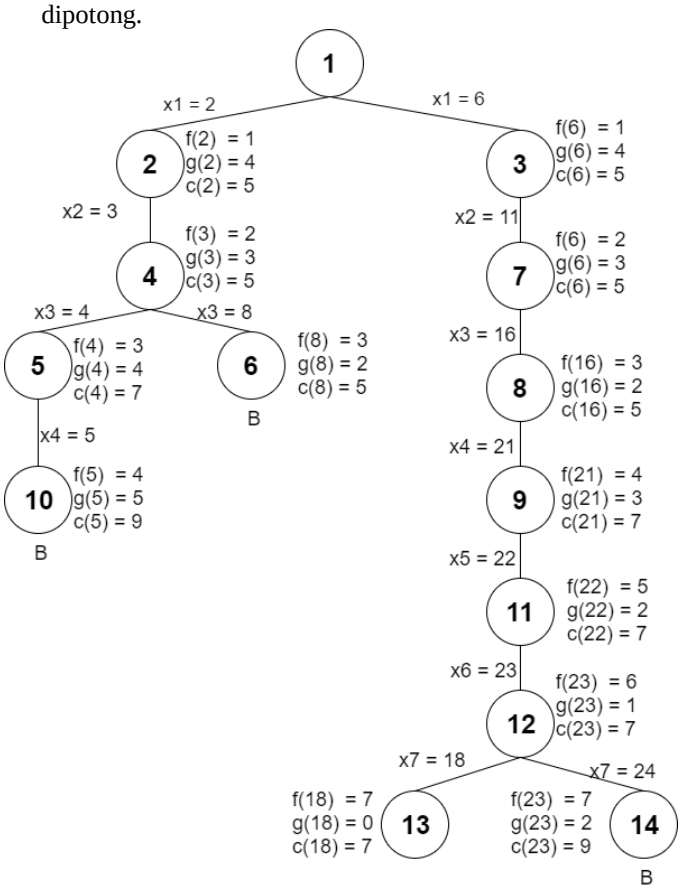
2. Selanjutnya simpul 2 diekspansi dan menghasilkan simpul hidup 3. Urutan simpul hidup sekarang adalah 3,6. Simpul-E pada kasus ini adalah simpul 3.

3. Simpul 3 diekspansi dan membangkitkan simpul 4 dan 8. Nilai ongkos dari simpul 4 dan 8 masing-masing adalah 7 dan 5. Simpul tersebut kemudian dimasukkan kedalam antrian. Urutan simpul hidup sekarang menjadi 8,6,4. Dari urutan simpul tersebut, didapatkan simpul-E adalah 8.



Gambar 10 Ekspansi simpul pohon cabang kiri
Sumber: Dokumen pribadi

4. Karena semua jalur menuju simpul 8 telah terpenuhi, maka simpul 8 tidak dapat diekspansi sehingga simpul ini dipotong.



Gambar 11 Pohon ruang status dari persoalan gambar 8
Sumber: Dokumen pribadi

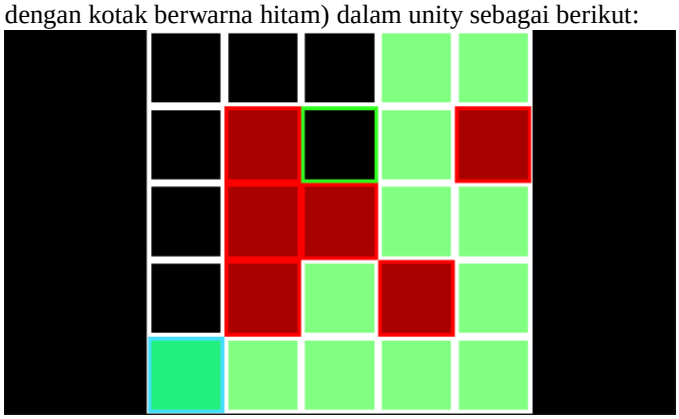
5. Simpul yang akan diekspansi berikutnya adalah simpul 6. Ekspansi simpul 6 membangkitkan simpul 11. Nilai

ongkos dari simpul 11 adalah 5. Sehingga urutan simpul adalah 11, 4.

6. Berikutnya simpul 11 diekspansi, karena simpul 12 tidak dapat dilalui objek, maka hanya simpul 16 yang dibangkitkan dan nilai ongkosnya adalah 5. Simpul 16 dimasukkan ke antrian, urutan simpul menjadi 16,4.
7. Simpul-E adalah 16 dan ekspansinya membangkitkan simpul 21 dengan ongkos 7. Urutan simpul menjadi 4,21. Simpul-E adalah 4.
8. Simpul 4 diperluas, urutan simpul hidup adalah 21,5. Simpul-E sekarang adalah 21.
9. Simpul 21 diperluas dengan membangkitkan simpul 22 dan diberi ongkos 7. Setelah perluasan simpul 21, urutan simpul hidup adalah 22, 5. Simpul-E sekarang adalah 22.
10. Simpul 22 diperluas dengan membangkitkan simpul 23 dengan ongkos 7. Urutan simpul hidup adalah 23, 5. Simpul-E sekarang ialah simpul 23.
11. Simpul 23 diperluas membangkitkan simpul 18 dan 24. Karena simpul 18 merupakan simpul solusi, maka solusi pertama telah ditemukan.

Karena ada kemungkinan solusi lain dengan ongkos yang sama, maka simpul dalam antrian akan diperiksa. Simpul 5 dan simpul 24 memiliki ongkos yang lebih besar dari ongkos solusi, maka simpul dipotong dan tidak diperluas.

Dari langkah-langkah diatas diperoleh ruang solusi adalah 1 -> 6 -> 11 -> 16 -> 21 -> 22 -> 23 -> 18. Lintasan (ditandai dengan kotak berwarna hitam) dalam unity sebagai berikut:



Gambar 12 Lintasan terdekat dari objek ke target
Sumber: Dokumen pribadi

V. KESIMPULAN

Terkadang solusi *pathfinding* yang ditawarkan pada *game engine* unity tidak dapat diimplementasikan dengan mangkus dan sangkil. Salah satu contohnya ketika berhadapan dengan game 2D. Dalam kasus ini, algoritma *branch and bound* merupakan algoritma yang cocok untuk diterapkan pada *pathfinding* di unity. Selain itu, algoritma *branch and bound* juga dapat memberikan kontrol lebih kepada pengguna sehingga pengguna dapat menentukan perilaku dari AI sesuai dengan keinginan pengguna.

UCAPAN TERIMA KASIH

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa oleh karena anugerah-Nya penulis dapat menyelesaikan tulisan ini. Ucapan terima kasih juga penulis ucapkan kepada Dr. Ir. Rinaldi Munir, MT., Dr. Nur Ulfa Maulidevi, ST., M.Sc dan Dr. Masayu Leylia Khodra, ST., MT. selaku dosen pembimbing mata kuliah IF2211 Strategi Algoritma sehingga tulisan ini dapat terselesaikan.

REFERENSI

[1] Munir, Rinaldi, 2009. *Diktat Strategi Algoritma*. Bandung: Institut Teknologi Bandung.

[2] https://unity3d.com/unity/engine-features, diakses tanggal 17 Mei 2017

[3] https://docs.unity3d.com/540/Documentation/Manual/nav-NavigationSystem.html, diakses tanggal 17 Mei 2017

[4] http://www.redblobgames.com/pathfinding/, diakses tanggal 18 Mei 2017

[5] https://www.youtube.com/watch?v=gGQ-vAmdAOI, diakses tanggal 18 Mei 2017

[6] https://www.youtube.com/watch?v=nhiFx28e7JY&index=2&list=PLFt_AvWsXl0cq5Umv3pMC9SPnKjfp9eGW, diakses tanggal 18 Mei 2017

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, 18 Mei 2017

Winarto - 13515061