Penggunaan *Behavior Tree* untuk Menentukan Aksi NPC di dalam Gim

Josep Marcello/13519164[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*josep1403@students.itb.ac.id*

***Abstrak*—Gim atau permainan video adalah salah satu hiburan yang populer di zaman modern ini. Di dalam gim, dibutuhkan beberapa karakter tidak dapat dimainkan yang disebut *non-playable character* (NPC). Agar NPC terlihat nyata, untuk menentukan aksi NPC dibutuhkan algoritma kecerdasan buatan (*artificial intelligence*/AI). Salah satu algoritma AI ini adalah *behavior tree*, yaitu algoritma yang memanfaatkan struktur data *tree* atau pohon untuk menentukan aksi yang akan dilakukan oleh NPC. Solusi ini memiliki beberapa keuntungan jika dibandingkan dengan *finite state machine*—salah satu solusi untuk menentukan aksi NPC—seperti lebih mudah dibuat, di-debug, dan lebih modular.**

***Kata kunci*—gim, NPC, kecerdasan buatan, pohon, behavior tree**

I. Pendahuluan

Salah satu sumber hiburan di zaman modern ini adalah gim atau permainan video. Populer sejak tahun 1980-an, jenis hiburan ini telah berevolusi hingga menjadi industri hiburan yang sangat besar. Gim juga merupakan salah satu sarana untuk menyampaikan cerita yang dapat mengikutsertakan pemain dalam cerita itu.

Untuk membuat pengalaman pemain lebih nyata, dibutuhkan beberapa karakter yang tidak dapat dimainkan, disebut *non-playable character* (NPC), yang memanfaatkan kecerdasan buatan (*artificial intelligence*/AI) dalam menentukan aksinya. Penggunaan AI ini dibutuhkan agar keputusan yang diambil oleh NPC dapat terlihat nyata.

Ada beberapa alternatif solusi yang ditawarkan untuk memenuhi kebutuhan NPC dalam pengambilan keputusan, misalnya *finite state machine*/*finite state automaton* (FSM) yang memanfaatkan states untuk menentukan apa yang harus dilakukan oleh NPC. Lalu, *director* AI (AI sutradara), yaitu AI yang bertindak seperti sutradara suatu film dalam menentukan alur permainan.

Pada makalah ini akan dibahas salah satu bentuk alternatif solusi lain untuk pengambilan keputusan NPC, yaitu *behavior tree* (BT). *Behavior tree* bisa dikatakan sebagai salah satu kompetitor dari solusi *finite state machine* karena memiliki tujuan yang sama, yaitu menentukan apa yang harus dilakukan oleh NPC pada suatu waktu dalam permainan.

BT merupakan alternatif solusi yang populer digunakan pada gim-gim modern sejak dipopulerkan oleh gim *Halo 2* (2004) [2]. Hal ini karena pemanfaatan struktur data tree dalam BT menyebabkan BT lebih modular, lebih mudah di-*debug*, serta lebih mudah dibuat jika dibandingkan dengan FSM.

II. Landasan Teori

A. Non-playable character

Non-playable character (NPC) adalah karakter apapun di dalam gim yang tidak dikendalikan oleh pemain [7]. Istilah NPC sendiri diambil dari permainan papan *role-playing* seperti *Dungeons and Dragons*. Pada permainan tersebut, NPC dikendalikan oleh seorang moderator permainan yang disebut *game master*. Perbedaannya dengan gim adalah pada gim, NPC biasanya dikendalikan *game master* yang merupakan sebuah mesin/program atau NPC dikendalikan langsung oleh program dan algoritma.

B. Metode Pengendalian NPC

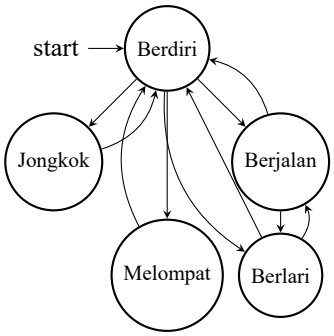
Ada beberapa metode untuk mengendalikan aksi NPC di gim. Metode-metode yamg populer untuk melakukan hal itu meliputi, tapi tidak terbatas pada:

- AI sutradara,
- *finite state machine*,
- dan *behavior tree*.

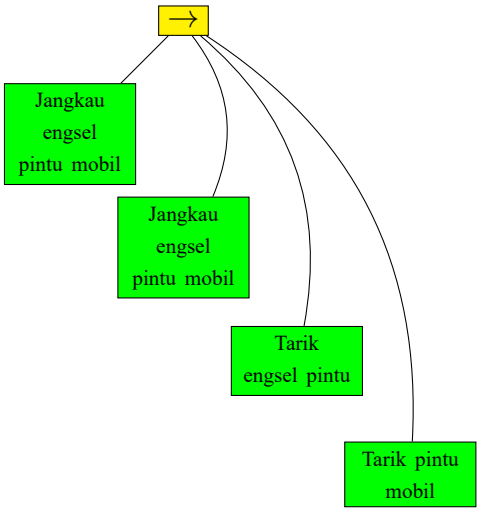
AI sutradara bekerja seperti halnya sutradara pada film dan teater. Sang "sutradara" menentukan dan mengendalikan kondisi permainan, misalnya menambahkan atau mengurangi jumlah musuh pada suatu waktu berdasarkan kondisi pemain dan skrip jika ada. Contoh gim yang menggunakan metode ini adalah *Left 4 Dead* (2008) [7].

Finite state machine (FSM) adalah salah satu metode yang mengendalikan aksi setiap NPC berdasarkan *state* (keadaan). Metode ini menentukan keadaan yang dimiliki oleh suatu NPC pada suatu waktu, sehingga aksi yang dapat dilakukan oleh NPC terbatas oleh keadaannya di waktu itu. Perpindahan keadaan NPC disebut transisi dan membutuhkan pemicu (*trigger*) sehingga fungsi transisi dilakukan dan NPC dapat mengganti *state*-nya. Contoh gim yang menggunakan metode ini adalah *Half-Life* (1998) [8].

Behavior Tree (BT) adalah metode lain untuk mengendalikan aksi NPC yang akan dibahas lebih lanjut pada bab selanjutnya. Metode ini memanfaatkan pohon berakar dengan setiap daun pohon menandakan aksi dan kondisi [4] dan setiap simpul lainnya disebut *decorator* (akan disebut "dekorator"),



Gambar 1. Contoh FSM sederhana



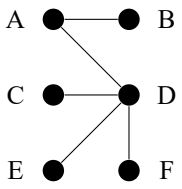
Gambar 2. Contoh BT membuka pintu mobil dengan akar sebagai simpul urutan

sequence (akan disebut "urutan"), atau *selector* (akan disebut "pemilih") [5] berfungsi untuk menentukan aksi apa saja yang dapat diambil atau harus dilakukan oleh NPC. Contoh gim yang menggunakan metode ini adalah *Halo 2* (2004).

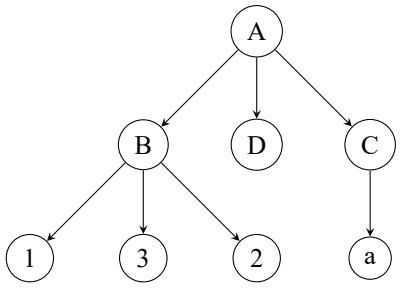
C. Pohon

Pohon (*tree*) adalah graf tak-berarah terhubung yang tidak mengandung sirkuit [1]. Di dalam dunia ilmu komputer dan aplikasinya, pohon memiliki banyak implementasi dan merupakan salah satu struktur data yang kerap digunakan.

Setiap pohon $G = (V, E)$ yang memliki n simpul, memiliki sifat [1]:



Gambar 3. Contoh pohon



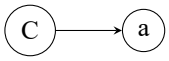
Gambar 4. Contoh pohon berakar T_1

- G adalah graf terhubung tak berarah sederhana,
- G memiliki $m = n - 1$ sisi,
- setiap pasang simpul G terhubung dengan lintasan tunggal,
- penambahan satu sisi pada G akan membuat sebuah sirkuit,
- dan semua sisi G adalah jembatan.

Salah satu jenis pohon yang paling sering digunakan adalah pohon berakar. Pohon berakar adalah pohon yang setiap sisinya merupakan sisi berarah dan sebuah simpulnya seperti "sumber" dari simpul lainnya. Gambar 4 adalah contoh pohon berakar. Pada penggambaran pohon berakar, biasanya sisinya digambarkan tidak berarah karena arahnya diimplikasikan di setiap sisi.

Pada pohon berakar, ada beberapa istilah atau terminologi yang sering digunakan. Istilah-istilah itu adalah [2]

- anak (*child*/*children*) dan orangtua (*parent*),
  pada pohon T_1 di gambar 4, simpul a adalah anak dari simpul C dan simpul A adalah orangtua dari simpul B, C, dan D.
- Lintasan (*path*),
  pada pohon T_1 di gambar 4, lintasan dari simpul A ke simpul 1 adalah A, B, 1 dan panjangnya adalah 2.
- Saudara kandung (*sibling*),
  pada pohon T_1 di gambar 4, saudara dari simpul B adalah D dan C.
- Upapohon (*subtree*),



Gambar 5. Sebuah upapohon T_1

  pada pohon T_1 di gambar 4, salah satu upapohonnya ditunjukkan pada gambar 5.
- Derajat (*degree*),
  pada pohon T_1 di gambar 4, simpul A memiliki derajat 3, simpul B memliki derajat 3, simpul D memiliki derajat 0, dan simpul C memiliki derajat 1. Derajat adalah banyak anak suatu simpul.
- Daun (*leaf*),
  pada pohon T_1 di gambar 4, daunnya adalah simpul 1, 2, 3, D, a. Daun adalah simpul berderajat 0.
- Simpul dalam (*internal nodes*),

pada pohon T_1 di gambar 4, simpul dalamnya adalah simpul A, B, C. Simpul dalam adalah simpul yang memiliki anak (bukan daun).
- Aras/tingkat (*level*),
  pada pohon T_1 di gambar 4, simpul A memiliki tingkat 0, simpul B, D, C memiliki tingkat 1, simpul 1, 2, 3, a memiliki tingkat 2.
- Tinggi (*height*)/kedalaman (*depth*).
  pada pohon T_1 di gambar 4, tingginya adalah 3. Tinggi adalah tingkat atau aras maksimum suatu pohon.

III. *Behavior Tree*

A. Dasar BT

Seperti sudah dibahas di bab-bab sebelumnya, *behavior tree* (BT) adalah salah satu bentuk pemanfaatan pohon untuk memecahkan masalah penentuan aksi NPC. BT dipopulerkan pada tahun 2004 oleh gim *Halo 2* dan merupakan pengganti untuk metode yang sebelumnya sering digunakan di insdustri ini, yaitu *finite state automaton* (FSM).

Sesuai dengan namanya, *behavior tree* menggunakan pohon berakar atau *rooted tree* untuk merepresentasikan dan menyimpan data dan proses. Penggunaan pohon ini menyebabkan BT memiliki beberapa keuntungan jika dibandingkan dengan FSM, yaitu [5]

- modular,
- memiliki hirarki,
- memiliki representasi yang mudah dibaca, dan
- ada penanganan yang gamblang untuk
  - *sequence* (aksi yang harus dilakukan berurutan),
  - *fallback*/*selector* (memilih satu dari beberapa aksi), dan
  - *interruptions* (interupsi aksi).

B. Alur Pemrosesan BT

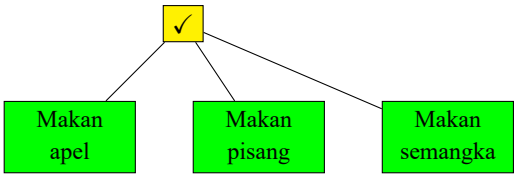
Pemrosesan pohon pada BT dimulai dari akar pohon, lalu tergantung jenis simpul pada akar, akan dilakukan salah satu atau semua aksi pada simpul (simpul diproses) di cabang-cabang akar. Lalu simpul yang diproses akan mengembalikan tiga status [10][1]:

- *success* (berhasil),
- *failure* (gagal),
- *running* (diproses).

Status-status ini menandakan hasil dari proses yang terjadi pada salah satu simpul pohon. Status berhasil memiliki arti proses yang dijalankan berhasil, status gagal memiliki arti proses yang dijalankan gagal. Terakhir, status *running* atau diproses memiliki arti proses yang dikerjakan oleh simpul masih belum selesai dan ketika pohon "dipanggil" lagi oleh program, proses pada simpul akan dilanjutkan sampai mengembalikan berhasil atau gagal.

Untuk menggambarkan hasil dari status, lihat BT pada gambar 2. Misalkan akar pada BT itu disebut sebagai simpul proses membuka pintu mobil, maka ketika proses berjalan dan

[1]Sebenarnya bisa lebih, tapi pada umumnnya hanya dibutuhkan tiga buah status.



Gambar 6. BT dengan simpul akar selektor

sampai pada simpul "Jangkau engsel pintu mobil," simpul tersebut akan mengembalikan status diproses kemudian status itu akan dikembalikan lagi ke simpul orang tua (*parent node*) dari BT itu. Jika proses ketika membuka pintu gagal, misalnya karena engsel pada pintu mobil hilang, maka simpul "Jangkau engsel pintu mobil" akan mengembalikan gagal (*failure*). Akan tetapi, jika proses itu berhasil dan mengembalikan diproses ke simpul orang tuanya, maka proses akan dilanjutkan dengan tahapan yang kurang-lebih sama sampai mencapai simpul "Tarik pintu mobil," dan pintu mobil sudah berhasil dibuka, maka simpul tersebut akan mengembalikan status berhasil (*success*) ke simpul orang tuanya (akar BT) dan status akan dikembalikan lagi ke simpul orang tua dari akar BT.

C. Jenis Simpul BT

Pohon berakar pada BT dibentuk oleh tiga buah jenis simpul, yaitu [10]:

- *composite* (gabungan),
- *decorator* (penghias), dan
- *leaf* (daun).

Simpul *composite* pada BT adalah simpul yang dapat memiliki satu atau lebih anak. Karena memiliki beberapa anak, maka simpul ini disebut simpul *composite* atau gabungan. Jenis simpul ini adalah jenis simpul yang paling sering muncul pada BT. Simpul gabungan yang paling sering muncul dan mendasar ada dua jenis, yaitu

- *sequence node* (simpul urutan aksi), dan
- *selector node* (simpul pemilih aksi).

Kedua simpul ini memproses simpul anak-anaknya dengan cara yang berbeda dan mirip dengan *logic gate*.

Simpul *sequence* dapat dibayangkan seperti gerbang AND pada gerbang logika karena simpul ini akan menghasilkan status sukses jika dan hanya jika semua simpul anak dari simpul ini menghasilkan status sukses juga. Artinya, simpul *sequence* akan menjalankan semua proses pada simpul anaknya. Contoh dari simpul ini adalah BT pada gambar 2. Simpul akar dari BT itu adalah simpul *sequence* dan ditandakan dengan simbol $\rightarrow$.

Simpul *selector* mirip dengan gerbang XOR pada gerbang logika karena simpul ini akan menghasilkan status sukses jika hanya salah satu simpulnya menghasilkan status sukses. Simpul ini memproses anak simpulnya satu per satu hingga didapatkan simpul yang mengembalikan status berhasil, berbeda dengan simpul *sequence* yang harus menjalankan semua simpulnya sampai semua simpul mengembalikan status sukses.

Contoh BT dengan simpul selektor dapat dilihat pada gambar 6. Simpul *selector*-nya terdapat pada simpul akar, ditandai oleh simbol ✓. Pada BT tersebut, ingin dipilih sebuah buah untuk dimakan. Diberikan tiga pilihan: apel, pisang, semangka. Jika apel dipilih dan dimakan, maka artinya simpul "Makan apel," mengembalikan status sukses, setelah itu proses oleh simpul akar akan berhenti dan mengembalikan status sukses ke simpul orang tua dari simpul akar. Akan tetapi, jika simpul "Makan apel" gagal, maka akan dilanjutkan ke simpul selanjutnya, yaitu "Makan pisang." Jika simpul itu mengembalikan status gagal lagi, maka simpul *selector* akan melanjutkan ke simpul anak selanjutnya.

Simpul lainnya adalah simpul *decorator*, yaitu simpul yang mirip dengan simpul *composite* karena dapat memiliki anak. Akan tetapi, simpul ini hanya dapat memiliki satu anak. Simpul ini biasanya digunakan untuk [10]:

- mengubah status anak simpulnya,
- menghentikan proses anak simpulnya, atau
- mengulang proses anak simpulnya sampai kondisi tertentu atau sampai dihentikan

tergantung jenis simpul dekoratornya. Jenis-jenis simpul dekrator adalah [10]

- *inverter* (pembalik),
- *succeeder* (pemberhasil),
- *repeater* (pengulang),
- *repeat until fail* (pengulang sampai gagal),

Simpul dekorator *inverter* akan "memutarbalikan" status dari simpul anaknya. Jika simpul anaknya mengembalikan sukses, maka akan dikembalikan gagal oleh simpul *inverter*. Simpul ini seperti gerbang NOT pada gerbang logika.

Simpul dekorator *succeeder* adalah simpul dekorator yang akan selalu mengembalikan status sukses, apapun yang terjadi. Simpul ini berguna jika pada suatu simpul *sequence* tidak ingin prosesnya berhenti karena ada anak dari simpul *sequence* yang mengembalikan status gagal. Simpul ini dapat digabungkan dengan simpul *inverter* untuk menghasilkan simpul yang selalu mengembalikan status gagal.

Simpul dekorator *repeater* adalah simpul dekorator yang akan mengulang proses dari simpul anaknya sampai dihentikan atau sampai beberapa kali iterasi. Bisa dibayangan seperti "while loop" dalam programming. Berguna untuk akar dari BT agar BT terus berjalan selama gim dimainkan.

Simpul dekorator *repeat until fail* bekerja sama seperti simpul *repeater*, simpul ini akan terus mengulang proses dari simpul anaknya. Bedanya dengan simpul *repeater*, simpul ini hanya akan bekerja sampai simpul anaknya mengembalikan status gagal.

Simpul pembuat BT terakhir adalah simpul daun (*leaf*). Sama seperti di pohon, simpul ini adalah simpul yang tidak memiliki anak. Aksi dan kondisi aksi NPC pada gim dituliskan pada simpul ini. Contoh simpul daun ditunjukkan oleh simpul "Makan apel," "Makan pisang," dan simpul "Makan semangka" pada gambar 6.

Jika BT dianalogikan sebagai sebuah program atau kode [10], maka simpul-simpul *composite* dan *decorator* dapat dipandang sebagai pernyataan `if` atau `while`, sedangkan simpul daun seperti kode lainnya yang dapat membuat program/kode menghasilkan atau memproses sesuatu.

D. Penggabungan Simpul Composite

Simpul-simpul *composite* pada sebuah BT dapat digabungkan jenis-jenisnya. Maksudnya adalah anak dari simpul *composite* bisa saja simpul *composite* lainnya. Perhatikan gambar 8. BT itu adalah BT untuk membuka pintu yang menggunakan simpul *selector* sebagai simpul akarnya dan simpul *sequence* untuk salah satu anaknya.

BT tersebut bekerja dari simpul akarnya. BT pertama akan ke simpul anak pertama, "Buka pintu." Jika simpul itu gagal, misalnya karena pintu dikunci, maka proses akan dilanjutkan ke simpul kedua. Simpul kedua adalah simpul *sequence* untuk proses membuka pintu yang dikunci. Proses pada simpul kedua dikerjakan secara berurutan, dimulai dengan mengambil kunci pintu, lalu membuka kunci pintu, dan diakhiri dengan membuka pintu. Jika salah satu proses itu ada yang gagal, misalnya kunci tidak cocok dengan pintu, maka simpul *sequence* akan mengembalikan status gagal, sehingga akan dilanjutkan dengan simpul "Hancurkan pintu." Jika simpul tersebut juga gagal, maka simpul *selector* akan gagal dan pintu gagal dibuka.

Akan tetapi, jika salah satu simpul anak dari simpul *selector* berhasil, maka proses pada simpul akar akan berhenti karena salah satu anaknya sudah mengembalikan status berhasil. Status berhasil atau gagal itu akan dikembalikan ke orang tua simpul *selector* (simpul akar).

Jadi, simpul *composite* yang merupakan simpul anak dapat dianggap seperti simpul daun pada biasanya yang akan mengembalikan status berhasil atau gagal. Bedanya hanya pada cara pemrosesannya yang mungkin saja lebih sulit.

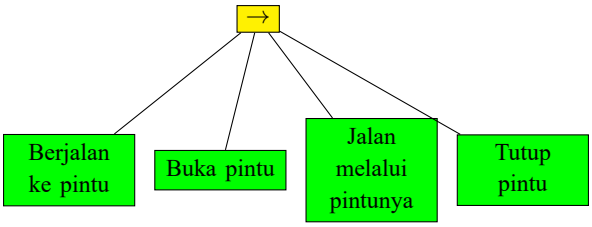
E. Modularitas BT

Setiap BT dapat dianggap seperti modul untuk BT lainnya. Hal ini karena sifat yang dijelaskan pada bab III-D dan merupakan salah satu keunggulan besar BT di atas FSM.

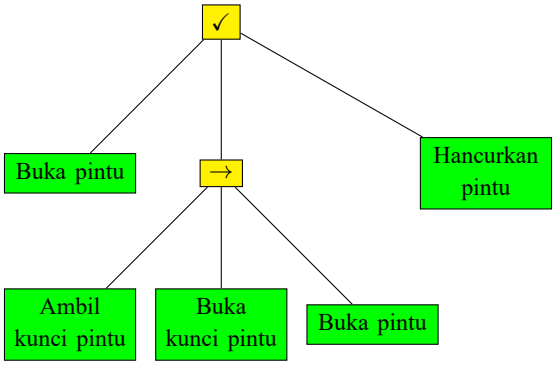
Untuk menambahkan atau menggabungkan suatu BT dengan BT lain, hanya perlu disambungkan simpul salah satu BT dengan akar salah satu BT, sehingga BT yang akarnya dihubungkan ke simpul BT lain menjadi anak dari simpul BT lain.

Contohnya dapat dilihat pada gambar 9 yang merupakan perpanjangan dari BT pada gambar 7. BT tersebut menggabungkan BT gambar 7 dengan BT gambar 8. Cara penggabungannya adalah menggantikan simpul "Buka pintu" pada gambar 7 dengan BT untuk proses membuka pintu. Penggabungan ini menghasilkan sistem BT yang lebih kompleks, tapi tetap mudah dibaca.

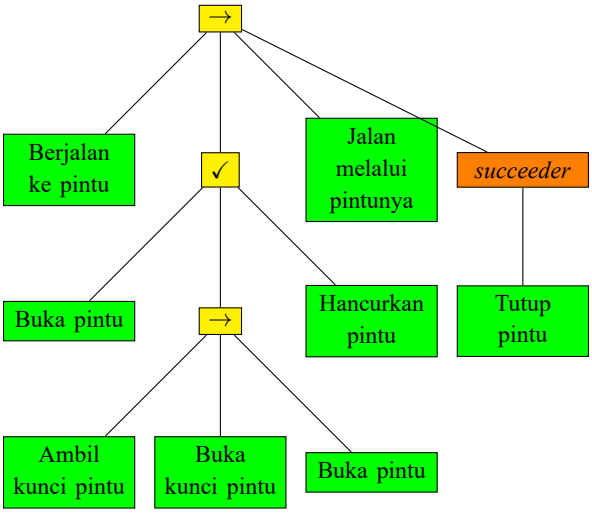
Selain itu, BT yang merupakan produk penggabungan memungkinkan NPC yang menggunakannya membuka pintu yang terkunci. Karena pada BT aslinya (gambar 7), jika buka pintu gagal, NPC akan membatalkan aksinya untuk masuk ke gedung. Akan tetapi pada BT gabungan, NPC dapat mencoba



Gambar 7. Contoh BT lain yang menggunakan node *sequence*. Digunakan untuk proses memasuki suatu gedung/rumah. Sumber: [10]



Gambar 8. Contoh BT yang menggabungkan simpul *selector* dengan *sequence*. Digunakan untuk membuka pintu. Sumber: [10]



Gambar 9. Contoh BT dengan modularitas yang menggabungkan BT pada gambar 7 dengan BT pada gambar 8. Merupakan BT yang digunakan untuk masuk ke gedung/rumah. Sumber: [10]

untuk membuka kunci pintunya atau mencoba menghancurkan pintu terlebih dahulu.

Perhatikan juga BT gabungan, ditambahkan sebuah simpul *decorator* "*succeeder*" pada BT tersebut. Penambahannya mudah karena hanya tinggal menyisipkan simpul itu di antara simpul akar dengan simpul "Tutup pintu." Simpul ini digunakan agar proses menutup pintu selalu berhasil, karena jika pintu dihancurkan pada simpul sebelumnya maka simpul "Tutup pintu" akan mengembalikan status gagal kalau tidak disisipkan simpul *decorator succeeder*.

IV. Kesimpulan

Pemanfaatan pohon untuk menentukan aksi yang akan dilakukan oleh sebuah NPC di dalam gim dapat menggunakan *behavior tree*. Penggunaan *behavior tree* dapat mempermudah pemrograman aksi NPC serta mempermudah pembacaan hasil pemrograman NPC. Selain itu, *behavior tree* juga sangat modular, sehingga penambahan dan pengurangan aksi NPC hanya semudah memutuskan atau menghubungkan simpul ke pohon.

Hal-hal itu menjadi sebuah keunggulan besar *behavior*

tree terhadap *finite state machine*—yang merupakan metode pengaturan aksi NPC terpopuler sebelum dikalahkan oleh *behavior tree*.

Dalam penentuan aksi NPC, seorang pemrogram *artificial intelligence* NPC dapat menentukan aksi yang akan dilakukan oleh NPC-nya dengan memanfaatkan kendali program pada *behavior tree* dan simpul yang beragam jenis.

Kendali program pada *behavior tree* memanfaatkan kondisi/status yang dikembalikan oleh simpul anaknya. Suatu simpul anak dapat mengembalikan status *success* (berhasil), *failure* (gagal), atau *running* (diproses). Status berhasil dan gagal sudah cukup jelas maknanya, sedangkan status diproses memiliki arti proses yang dikerjakan oleh simpul anak mash belum selesai atau masih ada simpul anak lain yang masih harus dikerjakan sebelum dapat mengembalikan status berhasil.

Jenis-jenis simpul *behavior tree* dibagi berdasarkan jumlah anak simpul yang dapat dimilikinya. Simpul dari *behavior tree* dibagi menjadi simpul *composite* (1 atau lebih anak), *decorator* (1 anak), *leaf* (0 anak). Simpul *composite* dibagi menjadi dua, yaitu *selector* (hanya akan menjalankan salah satu simpul anak) dan *sequence* (menjalankan semua simpul anak). Karena dapat memiliki lebih dari dua anak, simpul ini biasanya digunakan untuk menggabungkan beberapa aksi dan kondisi menjadi sebuah pohon atau upapohon. Aksi atau kondisi disimpan pada simpul *leaf*. Sedangkan simpul *decorator* biasanya digunakan untuk mengubah hasil suatu simpul atau untuk melakukan pengulangan.

Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada Tuhan YME karena atas berkat dan rahmat-Nya Penulis dapat menyelesaikan makalah Matematika Diskrit ini. Selain itu, terima kasih juga Penulis ucapkan kepada orang tua Penulis karena selalu memberikan dukungan kepada penulis dan sudah memungkinkan Penulis untuk melanjutkan pendidikan jenjang universitas.

Terima kasih kepada seluruh dosen pengajar IF2120 Matematika Diskrit Semester 1 Tahun 2020/2021, terutama Bapak Dr. Rinaldi Munir karena sudah menyediakan materi perkuliahan di situs pribadinya serta Ibu Dr. Nur Ulfa Maulidevi karena sudah mengajar dan memberikan kuliah IF2120 di kelas saya.

Penulis juga berterima kasih kepada para pemberi solusi mengenai masalah LaTeX dan pemberi tutorial mengenai LaTeX di internet. Karena mereka sudah banyak membantu Penulis dalam menyusun makalah ini dan menjawab masalah Penulis yang berhubungan dengan LaTeX.

Tidak lupa penulis juga berterima kasih kepada teman-teman Penulis, terutama kepada teman-teman Teknik Informatika ITB 2019 yang sudah memberikan Penulis dukungan moral untuk menyelesaikan makalah ini dan sudah menjadi teman diskusi Penulis selama penyusunan makalah ini.

Daftar Pustaka

[1] R. Munir, "Pohon (Bag. 1)," Institut Teknologi Bandung, 2020.
[2] R. Munir, "Pohon (Bag. 2)," Institut Teknologi Bandung, 2020.
[3] AI and Games, "Behaviour Trees: The Cornerstone of Modern Games AI | AI 101," *youtube.com*, 2, Jan. 2019. [Daring]. Tersedia: https://www.youtube.com/watch?v=6VBCXvfNlCM [Diakses 7 Desember 2020]
[4] Holistic3d, "Introduction to Behaviour Trees," *youtube.com*, 15, Okt. 2017. [Daring]. Tersedia: https://www.youtube.com/watch?v=uq8hnnkAxsw [Diakses 8 Desember 2020]
[5] P. Ögren, "What is a Behavior Tree and How do they work? (BT intro part 1)," *youtube.com*, 8, Sep. 2019. [Daring]. Tersedia: https://www.youtube.com/watch?v=DCZJUvTQV5Q [Diakses 8 Desember 2020]
[6] P. Ögren, "What is a Behavior Tree and How do they work? (BT intro part 2)," *youtube.com*, 28, Okt. 2019. [Daring]. Tersedia: https://www.youtube.com/watch?v=dB7ZSz890cw [Diakses 9 Desember 2020]
[7] "The Next Generation 1996 Lexicon A to Z," *Next Generation*, Mar., p. 39, 1996.
[8] AI and Games, "The Director AI of Left 4 Dead | AI and Games," *youtube.com*, 2, Des. 2014. [Daring]. Tersedia: https://www.youtube.com/watch?v=WbHMxo11HcU [Diakses 9 Desember 2020]
[9] AI and Games, "The AI of Half-Life: Finite State Machines | AI 101," *youtube.com*, 29, Mei 2014. [Daring]. Tersedia: https://www.youtube.com/watch?v=JyF0oyarz4U [Diakses 9 Desember 2020]
[10] C. Simpson, "Gamasutra: Chris Simpson's Blog - Behavior trees for AI How they work," Juli, 2014. [Daring]. Tersedia: https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php [Diakses 9 Desember 2020]

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.

Tangerang Selatan, 11 Desember 2020

Josep Marcello/13519164