Aplikasi *Behaviour Tree* Pada *Non-Player Character* Dalam *Video Game*

Muhammad Dzaki Razaan Faza 13519033
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13519033@std.stei.itb.ac.id

Abstract*—Pada revolusi industri 4.0 ini kecerdasan buatan menjadi salah satu komponen penting. Salah satu aplikasinya adalah dalam pembuatan *NPC* yang adaptif terhadap berbagai kondisi dan merespon secara spesifik. Teknik yang popular digunakan saat ini adalah *behaviour tree.

***Keywords*—Artificial Intelligence, Behaviour Tree, Tree Data Structure, Video Game**

I. Pendahuluan

Revolusi industri 4.0 membawa pengaruh besar dalam dunia industri serta kehidupan masyarakat saat ini. Dengan adanya otomasi dan digitalisasi penggunaan tenaga kerja manusia menjadi semakin berkurang, tetapi kualitas dan efektifitas produksi justru meningkat drastis. Salah satu komponen utama dalam pengembangan revolusi industri 4.0 adalah *artificial intelligence* atau kecerdasan buatan.

Kecerdasan buatan adalah model komputasi pada mesin yang memungkinkan dirinya untuk menyelesaikan permasalahan kompleks yang bahkan setara dengan manusia. Salah satu penerapannya terdapat dalam industri *video game.*

Kecerdasan buatan dalam *game* sudah ada bahkan sejak tahun 1950, pertama kali dikembangkan untuk membuat lawan dalam sebuah permainan *checkers* (mirip seperti permainan catur). Dengan berkembangnya teknologi industri yang sudah ada saat ini kecerdasan buatan pada *video games* digunakan terutama pada *non-player character (NPC)* untuk menciptakan entitas yang responsif, adaptif, dan memiliki kemampuan berpikir serta berperilaku mirip seperti manusia.

Video game seperti *Halo 2, Alien Isolation, Far Cry 4, BioShock Infinite* menggunakan kecerdasan buatan pada *NPC* sehingga dapat menganalisis kondisi sekitar dan memberikan respon secara spesifik. Teknik-teknik yang digunakan diantaranya adalah *Finite State Machine, Decision Tree,* dan *Behaviour Tree.*

Teknik *behaviour tree* merupakan teknik yang sering digunakan bahkan oleh perusahaan *game* AAA yang memiliki *development* dan *marketing budget* lebih tinggi. Pada makalah ini akan dijelaskan bagaimana implementasi teknik *behaviour tree* pada pengaturan perilaku berbagai macam *NPC* dan bagaimana teknik ini bisa menjadi popular bagi *developer*.

II. Teori Pohon

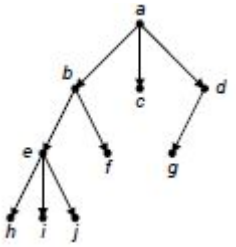
A. Definisi

Pohon adalah salah satu struktur data dalam *computer science* berupa graf tak berarah yang terhubung dan bersifat asiklik atau tidak memiliki sirkuit di dalamnya. Misalkan G = (V, E) adalah graf sederhana tak berarah dan simpulnya berjumlah n maka pohon memenuhi sifat-sifat sebagai berikut:

1. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal
2. G terhubung dan memiliki m n 1 buah sisi
3. G tidak mengandung sirkuit dan memiliki m n 1 buah sisi
4. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit
5. G terhubung dan semua sisinya adalah jembatan

B. Terminologi

Pohon yang satu buah simpulnya diibaratkan simpulnya sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah disebut pohon berakar.



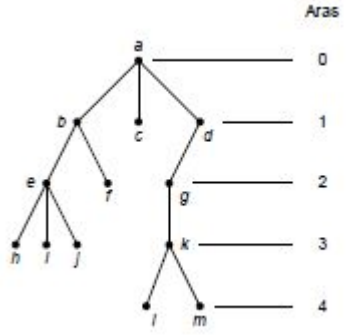
Gambar 1. Pohon Berakar
(sumber : Bahan Kuliah IF1210 Matematika Diskrit Pohon Bagian 2 Halaman 2)

Pada pohon berakar terdapat beberapa istilah-istilah atau terminologi yang sering digunakan antara lain:

1. Anak (*child*)
   Anak adalah simpul yang bukan merupakan akar artinya memiliki sisi yang masuk ke dalamnya yang berasal dari orangtuanya. Pada gambar 1 semua simpul selain a adalah anak dari sebuah simpul. Misal : simpul f dan e merupakan anak dari simpul b.
2. Orangtua (*parent*)
   Orangtua adalah simpul yang memiliki anak artinya simpul yang memiliki sisi keluar yang menunjuk ke

simpul lain yang akan menjadi anaknya. Pada gambar 1 simpul a adalah orangtua dari simpul b, c, dan d.

3. Lintasan (*path*)
   Lintasan adalah sisi-sisi yang harus dilalui dari satu simpul ke simpul lainnya. Pada gambar 1 lintasan dari a ke j adalah a,b,e,j.
4. Derajat (*degree*)
   Derajat adalah jumlah anak pada suatu simpul atau berapa banyak sisi yang keluar dari simpul tersebut. Pada gambar 1 simpul k berderajat 2, sedangkan simpul e berderajat 3.
5. Daun (*leaf*)
   Daun adalah simpul pada pohon yang berderajat nol atau tidak memiliki anak. Pada gambar 1 simpul h,i,j,l,m adalah daun.
6. Aras (*level*)
   Aras atau tingkat adalah panjang jalan dari akar sampai simpul tertentu.



Gambar 2. Pohon Berakar
(sumber : Bahan Kuliah IF1210 Matematika Diskrit Pohon Bagian 2 Halaman 9)

7. Tinggi (*height*)
   Tinggi adalah aras maksimum dari sebuah pohon. Pada gambar 2 maka tinggi pohonnya adalah empat(4).

III. Kecerdasan Buatan Dalam *Video Games*

Pada *game*s, kecerdasan buatan harus mampu bertindak dengan cara tertentu terhadap kondisi tertentu. Kecerdasan buatan harus dapat beradaptasi bukan hanya bertindak secara acak.

Secara umum bentuk dari kebanyakan kecerdasan buatan pada *game* sebagai terdiri dari beberapa layer sebagai berikut:

1. *Execution management* yaitu jenis kecerdasan buatan yang berkaitan dengan pengaturan dari *NPC*.
2. *Decision making (Finite State Machine, Behaviour Tree, Decision Tree)*
3. Grup kecerdasan buatan yang mengatur NPC secara jamak atau dalam jumlah besar, yang biasanya digunakan dalam *game* RTS untuk mengatur strategi dan posisi dari *NPC*.

Pada makalah ini akan membahas kecerdasan buatan dalam layer *decision making* terutama teknik *behaviour tree.*

IV. Behaviour Tree

Behavior Tree adalah pohon yang memiliki nodal-nodal yang memiliki aturan-aturan tersendiri. Ada dua aturan utama yaitu priority dan sequence. Dalam implementasi Behavior Tree, setiap simpul memiliki logika dan aturan tersendiri dan membatasi transisinya untuk tetap berada pada lingkup *state-state* tersebut. Daun pada *Behavior Tree* mendefinisikan aksi atau perilaku yang digunakan untuk membentuk aksi utama.

Berdasarkan fungsinya, simpul pada *Behavior Tree* dibagi menjadi dua bagian:

1. *Leaf*
   Leaf merupakan simpul terminal bertujuan untuk mengeksekusi bagian paling primitif yang dimiliki oleh *main behavior* dan mengembalikan nilai dari suatu state.
   a. Simpul Kondisi
      Digunakan untuk mengecek apakah suatu kondisi tertentu sudah terpenuhi.
   b. Simpul Aksi
      Digunakan untuk melakukan komputasi terhadap suatu state pada permainan tersebut untuk mengubahnya.

   Nilai yang dapat dikembalikan oleh simpul pada *Behavior Tree* antara lain adalah:
   a. *Success*
      Ketika persyaratan yang diajukan oleh simpul kondisi terpenuhi atau eksekusi dari simpul aksi telah diselesaikan.
   b. *Failure*
      Ketika persyaratan yang diajukan oleh simpul kondisi tidak terpenuhi atau eksekusi dari simpul aksi tidak bisa diselesaikan.
   c. *Running*
      Ketika eksekusi dari simpul aksi sudah dimulai, tetapi program belum dapat menarik kesimpulan untuk hasilnya.
2. *Composite*
   Composite digunakan untuk mendefinisikan dan mengatur hubungan antara simpul-simpul lainnya, seperti bagaimana atau kapan simpul tersebut harus diproses.
   a. *Selector*
      Selector memproses setiap anaknya secara bergantian. Ketika salah satu dari anaknya mengembalikan nilai *success*, maka *selector* akan segera mengembalikan nilai *success* tersebut. Apabila anak yang sedang diproses mengembalikan nilai *failure*, maka selector akan memproses anak berikutnya hingga tidak ada anak lagi yang tersisa dan mengembalikan nilai failure. Selector direpresentasikan dengan simbol tanda tanya dan kadang disebut sebagai priority.
   b. *Sequence*
      Sequence memproses setiap anaknya secara bergantian. Ketika salah satu dari anaknya mengembalikan nilai failure, maka selector

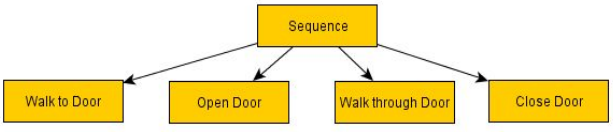
akan segera mengembalikan nilai failure tersebut. Apabila anak yang sedang diproses mengembalikan nilai success, maka selector akan memproses anak berikutnya hingga tidak ada anak lagi yang tersisa dan mengembalikan nilai success. Sequence direpresentasikan dengan simbol anak panah.

c. Decorator
Setiap decorator hanya memiliki anak tunggal dan berfungsi untuk memodifikasi behavior anak tersebut (wrapped task) dengan cara memanipulasi nilai yang dikembalikan atau mengubah frekuensi. Tipe-tipe *decorator* diantaranya adalah :

- Always Fail
  Selalu mengembalikan nilai failure tanpa menghiraukan nilai kembali yang sesungguhnya.
- Always Succeed
  Selalu mengembalikan nilai success tanpa menghiraukan nilai kembali yang sesungguhnya.
- Include
  Mencantumkan sebuah upapohon eksternal.
- Invert (negation)
  Mengembalikan nilai succeed apabila wrapped task gagal dan mengembalikan nilai failure apabila wrapped task berhasil.
- Limit
  Membatasi berapa kali wrapped task dapat diproses.
- Repeat
  Mengulangi pemrosesan wrapped task sebanyak n kali.
- UntilFail
  Mengulangi pemrosesan wrapped task hingga wrapped task bernilai failure, agar decorator dapat mengembalikan nilai success.
- UntilSuccess
  Mengulangi pemrosesan wrapped task hingga wrapped task bernilai success, agar decorator dapat mengembalikan nilai success.

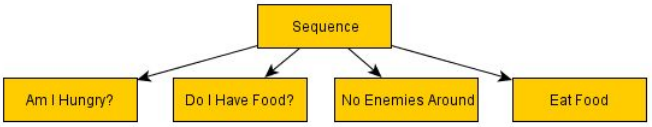
III. Aplikasi Behaviour Tree Pada Non-Player Character

Behaviour tree dapat digunakan untuk memodelkan berbagai macam perilaku NPC dalam *video games.* Salah satu aplikasi sederhana adalah dengan menggunakan simpul komposit *sequence* adalah sebagai berikut :



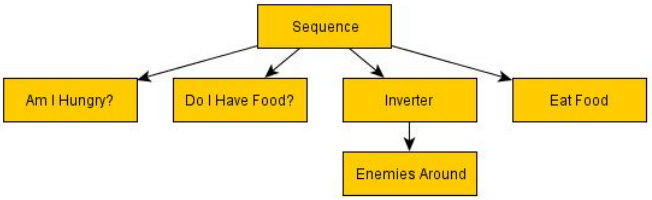
Gambar 3. *Sequence* Sederhana
(sumber :
https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php)

Sequence ini memprogram karakter untuk berjalan ke pintu, membukanya, melewati pintu, lalu menutup pintu. Simpul *leaf* dieksekusi secara berurutan dari kiri ke kanan jika setiap *leaf* mengembalikan nilai *success.* Bila ada *leaf* yang mengembalikan nilai *failure* misalnya ketika karakter sedang berjalan ke pintu ada sesuatu yang menghalangi seperti tembok maka *sequence* akan berhenti dan mengembalikan nilai *failure. Behaviour tree* ini digunakan untuk melakukan urutan suatu tindakan untuk melakukan sesuatu dalam contoh ini untuk membuka pintu. Selain itu, *sequence* juga dapat digunakan untuk mengecek deretan *list* kondisi sebelum melakukan sesatu.



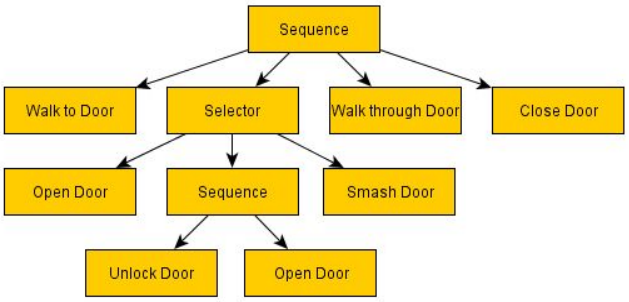
Gambar 4. *Sequence Condition Checking*
(sumber :
https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php)

Berbeda dari contoh sebelumnya yang melakukan urutan aksi, contoh ini mengecek *list* kondisi yang harus dipenuhi sebelum melakukan suatu aksi yaitu makan. Karakter harus memenuhi syarat seperti apakah karakter lapar?, apakah ada persediaan makanan?, apakah ada musuh di sekitar sebelum dapat melakukan aksi makan. Pada *sequence* pengecekan kondisi dapat juga dimanfaatkan *decorator* seperti *inverter* untuk membuat *condition checking* yang lebih baik.



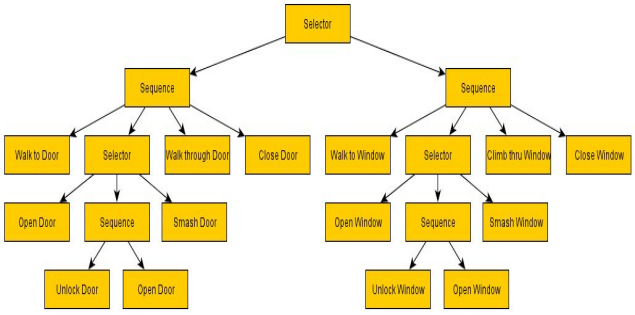
Gambar 5. *Sequence Condition Checking Using Inverter*
(sumber :
https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php)

Pada algoritma yang lebih kompleks dapat juga ditambahkan *selector* pada *behaviour tree. Sequence* membuka pintu yang dicontohkan pada gambar 3 dapat dimodifikasi menggunakan *selector* tersebut.



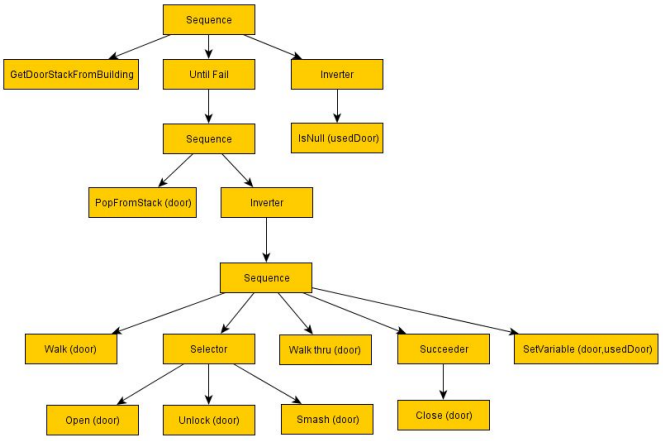
Gambar 6. *Opening Door Sequence Using Selector*
(sumber :
https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php)

Pada gambar 6 aksi membuka pintu didetailkan lagi menjadi metode-metode yang dapat dilakukan. Pertama adalah mengecek apabila pintunya bisa langsung dibuka artinya tidak terkunci. Jika mengembalikan nilai *success, leaf unlock door* dan *smash door* tidak akan dijalankan melainkan akan melanjutkan ke *leaf walk through door.* Jika mengembalikan nilai *failure,* maka karakter akan melanjutkan ke *leaf* selanjutnya yaitu mencoba membuka pintu menggunakan kunci. Begitu seterusnya mencoba satu-satu anak dari *selector* hingga menemukan nilai *success* atau sudah dicek semua dan tidak ada yang berhasil maka *selector* akan mengembalikan nilai *failure* pada orangtuanya. Pemanfaatan *sequence* dan *selector* yang lebih kompleks dapat membuat perilaku NPC untuk mencari jalan masuk ke sebuah bangunan. Apabila tidak bisa melalui pintu maka akan dicari jendela seperti yang digambarkan berikut ini:



Gambar 7. *Entering Building Behaviour Tree*
(sumber :
https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php)

Dengan menggunakan berbagai *decorator* dalam *behaviour tree* yang dibuat dapat menciptakan perilaku NPC yang lebih kompleks dan adaptif. Selain itu, dapat juga dimanfaatkan fungsi-fungsi eksternal seperti *set variable, isNull* dan struktur data lain seperti *Stack* memungkinkan *behaviour tree* untuk menerima dan mengembalikan sebuah nilai. Berikut adalah contoh *opening door behavior tree* yang memanfaatkannya.

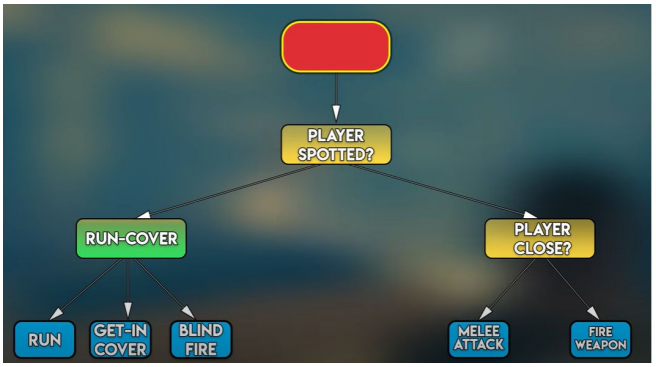


Gambar 8. *Opening Door Behaviour Tree Using Stack*
(sumber :
https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php)

Behaviour tree pada gambar 8 memanfaatkan *stack* untuk membuat karakter yang dapat mencari jalan masuk ke dalam sebuah bangunan melalui pintu. Dengan cara mengambil informasi pintu yang ada dalam bangunan lalu memasukkannya ke dalam *stack* dalam *leaf GetDoorStackFromBuilding.* Untuk setiap pintu yang akan dicek di-*pop* dari *stack* lalu dilakukan pengecekan apakah bisa dibuka. Bila ada pintu yang tidak terkunci atau bisa terbuka menggunakan kunci atau kekuatan *player* dapat mendobrak pintu maka dengan adanya *inverter, sequnce* akan mengembalikan nilai *failure* dan menghentikan *loop until fail.* Ketika keluar dari *loop,* maka hanya ada dua kemungkinan yaitu *stack* pintu dalam bangunan sudah habis, artinya tidak pintu yang bisa dibuka, atau ada pintu yang terbuka. Oleh karena itu, akan dicek menggunakan fungsi *isNull(usedDoor)* yang diberi *inverter* juga.

Pada *behaviour tree* yang lebih kompleks lagi dapat diaplikasikan kecerdasan buatan pada NPC untuk memasuki sebuah bangunan baik melalui pintu ataupun jendela. Beberapa NPC yang bisa memanfaatkan *behaviour tree* diantaranya adalah *zombie* yang ingin mencari pintu masuk bangunan dimana *player* bersembunyi, simulasi NPC yang dapat menerobos rumah *player* untuk mencuri barang, dan masih banyak lagi.

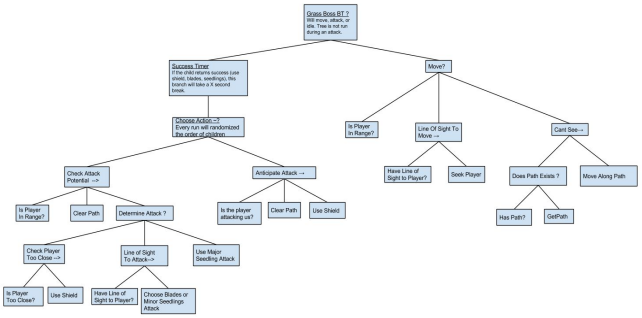
Beberapa contoh aplikasi *behaviour tree* yang lain terdapat pada kecerdasan buatan bagi mush *NPC* dalam *First Person Shooter*(*FPS*) seperti *Call of Duty, Halo,* dan masih banyak lagi. Berikut ini adalah contoh sederhananya:



Gambar 9. *FPS Games Enemy*
(sumber :
https://www.youtube.com/watch?v=6VBCXvfNlCM)

Dengan menggunakan *selector* yang lebih kompleks *NPC* dapat mendeteksi apakah *player* berada didekatnya. Hal pertama yang akan dilakukan adalah dengan mencari perlindungan di balik tembok atau semacamnya yaitu dengan menjalankan *sequence* lari, memasuki perlindungan, dan menembak secara liar(tanpa melihat posisi *player*). Apabila *player* berada sangat dekat dengan *NPC* maka *sequence* yang dilakukan adalah serangan jarak pendek lalu menembakkan senjata yang dibawa.

Dalam algortima yang lebih kompleks lagi *behaviour tree* juga dapat dimanfaatkan pada perilaku *boss* dalam sebuah permainan *role playing game* seperti berikut ini:



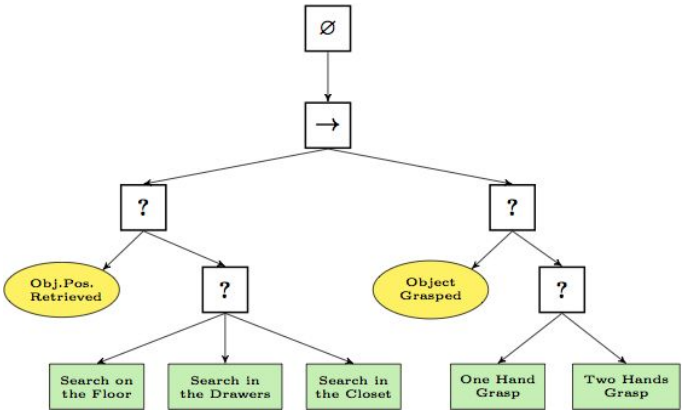
Gambar 10.*Advanced Behaviour Tree*
(sumber :
https://www.gamasutra.com/blogs/CodyOlivier/20150227/237519/Using_Behavior_Trees_to_Create_Retro_Boss_AI.php)

IV. Kelebihan Teknik Behaviour Tree

Behaviour tree memiliki banyak kesimpulan dibandingkan dengan teknik-teknik kecerdasan buatan lainnya untuk *NPC*. Pertama adalah mudah di-*trace* semakin kompleks perilaku *NPC* yang ingin dibuat maka akan semakin besar pula pohon yang diciptakan. Oleh karena itu, dibandingkan *finite state machine* yang transisi dari satu kondisi ke kondisi lainnya sangat kompleks, *behaviour tree* dapat dilakukan penelusuran dari ujung *leaf* ke simpul-simpul orangtuanya. Kedua adalah *designer frienly* karena tidak perlu pengetahuan *pemrograman* yang tinggi untuk mendesain *behaviour tree*. Sehingga, pendesain *NPC behaviour* dapat memberikan gambaran melalui *behaviour tree* baru akan diproses oleh *programmer* untuk diaplikasikan dan diperbaiki dari sisi algoritma. Ketiga adalah *reusable* artinya dapat digunakan kembali untuk *behaviour-behaviour NPC* lainnya yang mirip, tidak seperti *finite state* yang biasanya didesain untuk perilaku tertentu. Keempat adalah *behaviour tree* tidak memberikan beban yang terlalu berat pada komputer.

V. Simpulan

Pada revolusi industri 4.0 ini kecerdasan buatan menjadi salah satu komponen penting. Salah satu aplikasinya adalah dalam pembuatan *NPC* yang adaptif terhadap berbagai kondisi dan merespon secara spesifik. Teknik yang popular digunakan saat ini adalah *behaviour tree*. Model umumnya digambarkan dalam gambar berikut ini:



Gambar 11. Model Umum *Behaviour Tree*
(sumber :
https://en.wikipedia.org/wiki/Behavior_tree_(artificial_intelligence,_robotics_and_control))

VI. Ucapan Terima Kasih

Saya mengucapkan terima kasih pada Tuhan Yang Maha Esa (YME) untuk segala rahmat yang diberikan-Nya dalam pengerjaan makalah ini sehingga dapat terselesaikan tepat waktu. Saya juga ingin berterima kasih pada orang tua saya yang selalu memberikan dukungan. Terakhir, saya berterima kasih pada Dr. Ir. Rinaldi Munir, MT. yang telah membimbing saya selama satu semester sebagai dosen pengampu IF2120 Matematika Diskrit kelas 1.

References

[1] https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php Waktu akses: 10 Desember 2020 pukul 21.50.

[2] https://www.gamasutra.com/blogs/CodyOlivier/20150227/237519/Using_Behavior_Trees_to_Create_Retro_Boss_AI.php Waktu akses: 10 Desember 2020 pukul 21.53.

[3] R. Munir, "Pohon," in Matematika Diskrit IF2120 2020

[4] http://etheses.uin-malang.ac.id/20471/1/15650072.pdf Waktu akses: 10 Desember 2020 pukul 22.20

[5] https://www.youtube.com/watch?v=6VBCXvfNlCM Waktu akses 10 Desember 2020 pukul 19.20

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, 11 Desember 2020

Ttd

Muhammad Dzaki Razaan Faza 13519033