MEMANGUN *GAME* 3D *SIDE SCROLL* DAN MENERAPKAN MODEL *BEHAVIOR TREE*PADA NPC *ENEMY* DALAM *GAME* "MAVERICK"

Heny Pratiwi[1)], Pajar Pahrudin[2)], M. Yuditia Arfiyanto T. Nassa[3)]

[1,2,3]Program Studi Teknik Informatika, STMIK Widya Cipta Dharma
[1,2,3]Jl. M. Yamin No.25, Samarinda, 75123
E-mail : Yuditian3@gmail.com[1)]

ABSTRAK

Untuk menjadikan *Game* dapat menarik maka perlu dikembangkan *Non Player Character* (NPC) yang mempunyai kemampuan seperti manusia, AI (*artificial intelligent*) yaitu kecerdasan buatan yang dibuat untuk dapat memberikan kesan pada npc yang terlihat lebih alami dan berprilaku mirip manusia, baik npc tersebut adalah tokoh sampingan yang membantu pemain dalam progres cerita atau sebagai musuh yang harus di kalahkan pemain.

Behavior tree adalah sebuah cara yang terstruktur dalam pengalihan tugas pada *autonom agent* atau AIseperti robot atau *virtual entity* dalam *game* komputer (npc). *Behavior Tree* adalah cara yang sangat efisien dalam membangun sistem kompleks yang modular dan reaktif.

Hasil dari penelitian ini adalah dibuatnya *game side scrolling* yang memnpunyai grapik 3 dimensi. Dalam *game* terdapat *non-playable character* (NPC) yang menggunakan *behavior tree* yang mengontrol prilaku npc tersebut. Perilaku npc tersebut akan bereaksi terhadap aksi yang dilakukan oleh pemain dalam *game*.

Kata Kunci : *Game* 3D, *Behavior Tree*, *Side Scroll*.

1. PENDAHULUAN

Teknologi informasi merupakan suatu teknologi yang dapat digunakan untuk mengelola suatu data, yaitu untuk menghasilkan informasi yang akurat dan tepat waktu yang digunakan untuk kepentingan hiburan, pribadi, bisnis, sampai pemerintahan, salah satu yang memenuhi kebutuhan akan hiburan adalah *game*.

Untuk menjadikan *Game* dapat menarik maka perlu dikembangkan *Non Player Character* (NPC) yang mempunyai kemampuan seperti manusia, AI (*artificial intelligent*) yaitu kecerdasan buatan yang dibuat untuk dapat memberikan kesan pada npc yang terlihat lebih alami dan berprilaku mirip manusia, baik npc tersebut adalah tokoh sampingan yang membantu pemain dalam progres cerita atau sebagai musuh yang harus di kalahkan pemain.

Non player character musuh yang mampu bereaksi dengan aksi yang dilakukan pemain menjadi salah satu faktor agar permainan tetap dapat menarik bagi pemain, terutama dalam *game* yang ber*genre action* yang memerlukan ketangkasan parapemainnya dalam melawan musuh yang sudah di rancang sedemikian rupa menggunakan kecerdasan buatan, *artificial intelligence agent* dibutuhkan untuk membantu mengatur perilaku npc dalam *game*, dibutuhkan suatu metode yang dapat berkoordinasi dengan *player* sehingga dapat mengatur posisi pergerakan dan perilaku sesuai dengan kondisi*player* saat menyerang agar dapat menghasilkan perilaku yang adaptif terhadap aksi *player*.

Perkembangan lain pada NPC musuhadalah pengembangan dari naluri bertahan hidup pada saat *battle* (bertarung). Dalam *game* ini, NPC dapat mengenali obyek yang berbeda pada lingkungan dan dapat menentukan apa yang harus dilakukan pada saat berhadapan dengan halangan di hadapannya.

Salah satu penerapan kecerdasan buatan di *game* untuk pengambilan keputusan yang cerdas adalah *behavior tree*. Metode *behavior tree* dipilih karna selain mudah untuk diterapkan juga mampu membuat sistem yang kompleks baik secara *modular* dan *reactive*, dimana kedua properti tersebut sangat penting dalam pengembangan *game*. Seperti namanya *behavior tree* adalah metode yang menggunakan model pohon sebagai modul eksekusi nya dimana dalam model pohon tersebut terdapat *root* (akar) dan beberapa macam node yang bisa digunakan sesuaidengan fungsinya masing – masing dan *leaf* (daun) node yang mengeksekusi *task* (tugas) dalam hirarki pohon tersebut.

2. RUANG LINGKUP PENELITIAN

2.1 RUMUSAN MASALAH

Berdasarkan latar belakang di atas, maka penulis membuat perumusan masalah sebagai isi dari laporan, rumusan dari masalah yang dikemukakan adalah: "Bagaimana Membangun *Game* 3d *side scroll* dan Menerapkan Model *Behavior Tree* pada NPC *enemy* dalam *game* Maverick?*"*.

2.2 BATASAN MASALAH

1. *Game* bersifat *single-player*
2. Menggunakan *side scroll view* sebagai *view point* nya
3. *Action gameplay*
4. *Story line* akan berlatar belakang Sci-fi
5. Tidak mempunyai *Score* akhir
6. *Game* bersifat *offline*
7. Bahasa Pemograman menggunakan Bahasa C++
8. *Graphic Game* berupa 3D
9. Menggunakan Unreal engine 4sebagai *Engine*
10. *Game* hanya untuk PC/Laptop
11. Target *Audience* untuk remaja - dewasa (17+)
12. Menggunakan model AI *Behavior tree*

2.3TUJUAN PENELITIAN

Tujuan penilitian ini bertujuan agar dapat menerapkan metode *behavior tree* pada NPCagar lebih adaptif terhadap prilaku pemain dalam permainan ini.

3. BAHAN DAN METODE

Adapun bahan dan metode yang gunakan dalam membangun penelitian ini yaitu:

3.1 *Game*

Game berasal dari bahasa inggris yang berarti permainan . Dalam setiap game terdapat peraturan yang berbeda - beda untuk memulai permainannya sehingga membuat jenis game semakin bervariasi . Karena salah satu fungsi game sebagai penghilang stress atau rasa jenuh maka hampir setiap orang senang bermain game baik anak kecil , remaja maupun dewasa , mungkin hanya berbeda dari jenis *game* yang dimainkannya saja .

Menurut Yunus (2015), mengungkapkan: "*Game* merupakan suatu jenis model permainan atau pertandingan. *game* bisa di artikan sebagai aktivitas terstruktur atau semi terstruktur, yang biasanya dilakukan untuk *fun* dan kadang digunakan sebagai alat pembelajaran".

3.2 Jenis *Game (Genre Game)*

Menurut Adams (2014), *Game* terbagi ke beberapa jenis atau yang biasa disebut *genre* yang mendefinisikan *genre game* adalah sebagai berikut :

"GENRES are categories of games characterized by particular kinds of challenges, regardless of settings or game-world contents".

Terjemahan *: "Genre* adalah kategori dari karakteristik *game* yang berdasar pada beberapa jenis tantangan, terlepas dari aturan atau isi dari dunia *game* tersebut*".*

3.3 Animasi

Menurut Mardiyasa (2016), Animasi adalah suatu proses dalam menciptakan efek gerakan atau perubahan dalam jangka waktu tertentu, dapat juga berupa perubahan warna dari suatu objek dalan jangka waktu tertentu dan bisa juga dikatakan berupa perubahan bentuk dari suatu objek ke objek lainnya dalam jangka waktu tertentu. Pengertian lain tentang animasi adalah pembuatan gambar atau isi yang berbeda-beda pada setiap frame, kemudian dijalankan rangkain frame tersebut menjadi sebuah motion atau gerakan sehingga terlihat seperti sebuah film. Sedangkan menurut Andreas Andi Suciadi, animasi adalah sebuah objek atau beberapa objek yang tampil bergerak melintasi stage atau berubah bentuk, berubah ukuran, berubah warna, berubah putaran, berubah properti-properti lainya. Secara garis besar animasi adalah suatu tampilan menarik, grafis statis maupun dinamis, yang disebabkan oleh perubahan tiap frame (frame by frame), perubahan posisi bergerak (motion tween) maupun perubahan bentuk diikuti pergerakan (motion shape).

3.4 *Object Oriented Programming* (OOP)

Menurut Sukamto (2014), mendefinisikan bahwa," *Object Oriented Programming* (OOP) adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya".

Pemrograman berorientasi objek (*Object Oriented Programming* atau disingkat OOP) adalah paradigma pemrograman yang berorientasikan kepada objek yang merupakan suatu metode dalam pembuatan program, dengan tujuan untuk menyelesaikan kompleksnya berbagai masalah program yang terus meningkat. Objek adalah entitas yang memiliki atribut, karakter (*behavior*) dan kadang kala disertai kondisi (*state*) (Douglas).

Pemrograman berorientasi objek ditemukan pada Tahun 1960, dimana berawal dari suatu pembuatan program yang terstruktur (*structured programming*). Metode ini dikembangkan dari bahsa C dan Pascal. Dengan program yang terstruktur inilah untuk pertama kalinya kita mampu menulis program yang begitu sulit dengan lebih mudah.

Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik piranti lunak skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

OOP dalam melakukan pemecahan suatu masalah kita tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh anggap kita memiliki sebuah departemen yang memiliki *manager*, sekretaris, petugas administrasi data dan lainnya. Misal *manager* tersebut ingin memperoleh data dari *bag* administrasi maka *manager* tersebut tidak harus mengambilnya langsung tetapi dapat menyuruh petugas *bag* administrasi untuk mengambilnya. Pada kasus tersebut seorang *manager* tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi *manager* bisa mendapatkan data tersebut melalui objek petugas adminiistrasi. Jadi untuk

menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri.

Pemrograman orientasi-objek menekankan konsep sebagai berikut :

a. Kelas (*Class*) — kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebuah *class* adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi objek. Sebuah *class* secara tipikal sebaiknya dapat dikenali oleh seorang *non-programmer* sekalipun terkait dengan domain permasalahan yang ada, dan kode yang terdapat dalam sebuah *class* sebaiknya (relatif) bersifat mandiri dan independen (sebagaimana kode tersebut digunakan jika tidak menggunakan OOP). Dengan modularitas, struktur dari sebuah program akan terkait dengan aspek-aspek dalam masalah yang akan diselesaikan melalui program tersebut. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya.

b. Objek (*Object*) - membungkus data dan fungsi bersama menjadi suatu unit dalam sebuah program komputer. Objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.

c. Abstraksi (*Abstract*) - Kemampuan sebuah program untuk melewati aspek informasi yang diproses olehnya, yaitu kemampuan untuk memfokus pada inti. Setiap objek dalam sistem melayani sebagai model dari "pelaku" abstrak yang dapat melakukan kerja, laporan dan perubahan keadaannya, dan berkomunikasi dengan objek lainnya dalam sistem, tanpa mengungkapkan bagaimana kelebihan ini diterapkan. Proses, fungsi atau metode dapat juga dibuat abstrak, dan beberapa teknik digunakan untuk mengembangkan sebuah pengabstrakan.

d. Enkapsulasi (*Encapsulation*) - Memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam dari sebuah objek dengan cara yang tidak layak; hanya metode dalam objek tersebut yang diberi ijin untuk mengakses keadaannya. Setiap objek mengakses *interface* yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

e. Polimorfisme (*Polimorfism*) melalui pengiriman pesan. Tidak bergantung kepada pemanggilan subrutin, bahasa orientasi objek dapat mengirim pesan; metode tertentu yang berhubungan dengan sebuah pengiriman pesan tergantung kepada objek tertentu di mana pesa tersebut dikirim. Contohnya, bila sebuah burung menerima pesan "gerak cepat", dia akan menggerakan sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, dia akan menggerakkan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama, namun yang sesuai dengan kemampuan hewan tersebut. Ini disebut polimorfisme karena sebuah variabel tungal dalam program dapat memegang berbagai jenis objek yang berbeda selagi program berjalan, dan teks program yang sama dapat memanggil beberapa metode yang berbeda di saat yang berbeda dalam pemanggilan yang sama. Hal ini berlawanan dengan bahasa fungsional yang mencapai polimorfisme melalui penggunaan fungsi kelas pertama.

f. Inheritas (*Inheritance*) - Mengatur polimorfisme dan enkapsulasi dengan mengijinkan objek didefinisikan dan diciptakan dengan jenis khusus dari objek yang sudah ada - objek-objek ini dapat membagi (dan memperluas) perilaku mereka tanpa haru mengimplementasi ulang perilaku tersebut (bahasa berbasis-objek tidak selalu memiliki inheritas).

3.5 Unreal Engine 4

Unreal Engine 4 adalah salah satu dari banyak *software* yang digunakan dalam pembuatan *game*. Menurut Shah (2015), *engine* ini adalah penyempurnaan dari *engine* sebelumnya dan menjadi *tool* untuk pengembangan *game* yang ideal, baik secara indie atau komersil, Epic Game sendiri telah meningkatkan kualitas Unreal Engine dan menjadikannya salah satu *software* pilihan dalam pengembangan *game* baik itu hanya projek kecil atau besar. Epic Game juga menyediakan *content* gratis yang dapat digunakan dalam pembuatan *game*, Unreal Engine 4 sendiri memberikan fitur - fitur yang dapat membantu *developer* dalam pembuatan *game* seperti *Blueprint scripting* yang sangat membantu mempersingkat waktu pengerjaan projek, bukan hanya itu, dalam hal grafis Unreal Engine juga menggunakan teknik iluminasi *global real-time* menggunakan *voxel* kerucut *tracing* yang dapat menghilangkan kebutuhan untuk perhitungan tiap penerangan.

3.6 *Unified Modeling Language* (UML)

Rosa (2014), UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataannya UML paling banyak digunakan pada metodologi berorientasi objek.

3.7 Diagram UML

Rosa (2014), pada UML terdiri dari 13 macam diagram yang dikelompokkan dalam 3 kategori. Berikut ini penjelasan singkat dari pembagian kategori tersebut.

1. *Structure diagram*, yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang di modelkan. *Structure diagram* terdiri dari *class diagram, object diagram, component diagram, composite structure diagram, package diagram dan deployment diagram.*
2. *Behavior diagram* yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada sebuah sistem. *Behavior diagram* terdiri dari *Use case diagram, Activity diagram, State Machine System*.
3. *Interaction diagram* yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antar subsistem pada suatu sistem. *Interaction* diagram terdiri dari *Sequence Diagram, Communication Diagram, Timing Diagram, Interaction Overview Diagram*.

3.8 *Use Case Diagram*

Rosa (2014), use case atau diagram use case merupakan pemodelan untuk kelakuan (*behavior*) sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu. Berikut adalah simbol-simbol yang ada pada diagram *use case* :

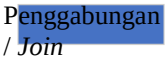
Tabel 1Simbol – simbol *Use case diagram*

No.	Simbol	Deskripsi
1	*Use case* 	Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor, biasanya dinyatakan dengan menggunakan kata kerja di awal frase nama *use case*
2	Actor 	Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang, biasanya dinyatakan menggunakan kata benda di awal frase nama *actor*.
3	*Association*	Komunikasi antara aktor dan *use case* yang berpartisilpasi pada *use case* atau *use case* memiliki interaksi dengan aktor.
4	*Extend*	Relasi *use case* tambahan kesebuah *use case* dimana *use case* yang ditambahkan dapat berdiri sendiri walau tanpa *use case* tambahan itu, mirip dengan prinsip *inheritance* pada pemrograman berorientasi objek, biasanya *use case* tambahan memiliki nama depan yang sama dengan *use case* yang ditambahkan, misal  Arah panah mengarah pada *use case* yang ditambahkan, biasanya *use case* yang menjadi *extend*-nyamerupakan jenis yang sama dengan *use case* yang menjadi induknya.
5	*Generalization*	Hubungan generalisasi dan spesialisasi (umum-khusus) antara dua buah *use case* dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya
6	*Include / Uses*	Relasi *use case* tambahan ke sebuah *usecase* dimana *use case* yang ditambahkan memerlukan *use case* ini untuk menjalankan fungsinya atau sebagai syarat dijalankan *use case* ini Ada dua sudut pandang yang cukup besar mengenai *include* di *use case* : - *Include* berarti *use case* yang ditambahkan akan selalu di panggil saat *use case* tambahan dijalankan. - *uses* berarti *use case* tambahan akan selalu melakukan pengecekan apakah *usecase* yang di tambahkan telah dijalankan sebelum *use case* tambahan di jalankan.

3.9 *Activity Diagram*

Rosa (2014), diagram aktivitas atau activity diagram menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak. Yang perlu di perhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem.

Tabel 2 Simbol-simbol *activity diagram*

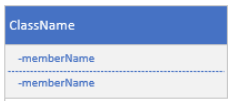
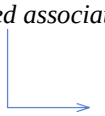
No.	Simbol	Deskripsi
1	Stasus awal	Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal
2	Aktivitas 	Aktivitas yang dilakukan sistem, aktivitas biasanya diawali dengan kata kerja.
3	Percabangan/ *Decision* 	Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu.
4	Penggabungan / *Join* 	Asosiasi penggabungan dimana lebih dari satu aktivitas digabungkan menjadi satu.
5	Status Akhir 	Status akhir yang dilakukan oleh sistem, sebuah diagram aktivitas memiliki sebuah status akhir.
6	*Swimlane* 	Memisahkan organisasi bisnis yang bertanggunga jawab terhadap aktivitas yang terjadi.

3.10*Class Diagram*

Rosa (2014), diagram kelas atau class diagram menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan method atau operasi. Berikut penjelasan atribut dan metode :

1. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas.
2. Operasi atau method adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

Tabel 3 Simbol - simbol *class diagram*

No.	Simbol	Deskripsi
1	Kelas 	Kelas pada struktur sistem
2	Antarmuka / *interface* 	Sama dengan konsep interface dalam pemrograman berorientasi objek
3	Asosiasi / *associaton* 	Relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan *multiplicity*
4	Asosiasi berarah / *directed association* 	Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan multiplicity
5	Generalisasi 	Relasi antar kelas dengan makna generalisasi-spesialisasi (umum khusus)
6	Kebergantungan/ *dependency* 	Relasi antar kelas dengan makna kebergantungan antar kelas
7	Agregasi/ *aggregation* 	Relasi antar kelas dengan makna semua-bagian (*whole-part*)

3.11 *Sequence Diagram*

Rosa (2014), diagram sekuen menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dengan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu. Membuat diagram sekuen juga dibutuhkan untuk melihat skenario yang ada pada *use case*. Banyaknya diagram sekuen yang harus digambar adalah minimal sebanyak pendefinisian use case yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup dalam diagram sekuen sehingga semakin banyak *use case* yang didefinisikan maka diagram sekuen yang harus dibuat juga semakin banyak.

Tabel 4 Simbol – simbol *sequence diagram*

No	Simbol	Deskripsi
1	Aktor 	Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat diluar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang, biasanya dinyatakan dalam menggunakan kata benda diawal frase nama aktor.
2	Garis Hidup /*lifetime* 	Menyatakan kehidupan suatu objek.
3	Objek 	Menyatakan objek yang berinteraksi pesan
4	Waktu aktif 	Menyatakan objek dalam keadaan aktif dan berinteraksi, semuanya yang terhubungdengan waktu aktif ini adalah sebuah tahapan yang dilakukan di dalamnya
5	Pesan tipe *create* 	Menyatakan suatu objek membuat objek yang lain, arah panah mengarah pada objek yang dibuat
6	Pesan tipe *call* 	Menyatakan suatu objek memanggil operasi/metode yang ada pada objek lain atau dirinya sendiri
7	Pesan Tipe Masukkan 	Menyatakan bahwa suatu objek mengirimkan data/masukkan/informasi ke objek lainnya, arah panah mengarah pada objek yang dikirim
8	Pesan tipe *return* 	Menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode menghasilkan suatu kembalian ke objek tertentu, arah panah mengarah pada objek yang menerima kembalian
9	Pesan tipe *destroy* 	Menyatakan suatu objek mengakhiri hidup objek yang lain, arah panah mengarah pada objek yang diakhiri, sebaliknya jika ada *create* maka ada *destroy*

3.12 Metode Pengembangan Multimedia

Menurut Munir (2012), "Metodologi dalam pengembangan *software* sangat dikaitkan dengan susunan kerja atau *framework* karena menggunakan pendekatan sistem informasi". Tujuan mewujudkan susunan kerja ialah untuk menuntun peneliti dalam mengembangkan *software* tersebut.

Metodologi pengembangan multimedia yang terdiri dari 6 tahapan, yaitu *concept, design, material collecting, assembly, testing* dan *distribution*.

1.Tahap *concept* (pengonsepan) adalah tahapan untuk menentukan tujuan dan siapa pengguna program.
2.Tahap *Design* (perancangan) adalah tahap pembuatan spesifikasi mengenai antarmuka program, gaya, tampilan maupun kebutuhan materil/bahan untuk program.
3. Tahap pengumpulan data adalah tahap pengumpulan bahan yang sesuai dengan kebutuhan yang dikerjakan.
4. Tahap *Assembly* (pembuatan) adalah tahap pembuatan semua objek atau bahan multimedia.
5. Tahap Pengujian adalah tahap yang dilakukan setelah menyelesaikan tahap pembuatan (assembly) dengan menjalankan aplikasi/program dan melihat apakah ada kesalahan atau tidak.
6. Tahap Distribusi Pada tahap ini, aplikasi yang akan disimpan dalam suatu media penyimpanan.

3.13 *Artificial Intellegence* (AI/Kecerdasan buatan)

Menurut Poole dan Macworth (2017), Kecerdasan buatan adalah bidang yang mempelajari sintesisdan analisis agen komputasi yang bertindak cerdas. Agen adalah sesuatu yang bertindak dalam suatu lingkungan dan dapat bereaksi terhadap lingkungan nya.

Agen cerdas bertindak cerdas Ketika :

1. Apa yang dilakukan agen tersebut sesuai dengan keadaan dan tujuannya, dengan mempertimbangkan konsekuansi jangka pendek dan jangka Panjang dari tindakannya.
2. Fleksibel untuk mengubah lingkungan dan tujuannya
3. Belajar dari pengalaman
4. Membuat pilihan yang tepat mengingat keterbatasan persepsi dan komputasinya.

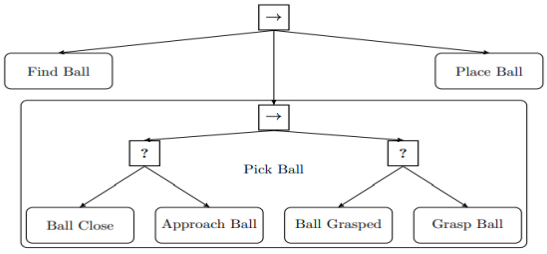
Agen komputasi adalah agen yang keputusannya tentang tindakannya dapat dijelaskan dalam hal perhitungan. Artinya, keputusan dapat dipecah menjadi operasi primitif yang dapat diimplementasikan dalam perangkat fisik.Perhitungan ini dapat mengambil banyak bentuk. Di komputer itu dilakukan dalam "perangkat keras." Meskipun ada beberapa agen yang bisa dibilang tidak komputasional, seperti angin dan hujan mengikis lanskap, itu adalah pertanyaan terbuka apakah semua agen cerdas adalah komputasi.

3.14*Behavior Tree*

Menurut Colledanchise dan Orgen (2018), *Behavior tree* adalah sebuah cara yang terstruktur dalam pengalihan tugas pada *autonom agent* atau AIseperti robot atau *virtual entity* dalam *game* komputer (npc). *Behavior Tree* adalah cara yang sangat efisien dalam membangun sistem kompleks yang modular dan reaktif.

3.15Formulasi Dasar *Behavior Tree*

Dalam*Behavior Tree* terdapat node yang disebut*root, control-flow nodes* dan *execution node*. Pada node yang terhubungsatusama lain dapatdianggapsebagai*parent – child*, *root node* tidakmemiliki*parent node*, namun*control-flow node*memilikisatu*parent node* dansetidaknyasatu*child node* yang diletakkandibawah*node* tersebut, sedangkan*execution node*tidakbisamemiliki*children node* namunberperansangatpentingkarna*node* inilah yang akanmengeksekusitugas yang adapadamodul*behavior tree* tersebut, contohnyasepertipadagambardibawahini.



Gambar 1. Formulasi dasar *behavior tree*

Root node akan mengirimkan sinyal yang disebut *ticks* kepada *children node* nya (*control-flow* ataupun *execution node*) untuk memulai eksekusi tugas pada modul *behavior tree* tersebut, kemudian *control-flow node* yang menerima *ticks* lalu akan memulai eksekusi sesuai dengan fungsi *control-flow node* tersebut dan mengirimkan sinyal *ticks* kepada *children node* nya (*control-flow* ataupun *execution node*). *Control-flow* sendiri memiliki beberapa jenis dengan fungsinya masing – masing yang akan dijelaskan sebagai berikut :

1. *Sequence Node*
   Sequence Node adalah *node* yang fungsinya mengirimkan *ticks* pada *child*-nya secara berurut dimulai dari *child node* paling kiri dan akan melanjutkan pada *child* berikutnya, jika *child node* nya mengembalikan *ticks* dengan status *fail* atau *running*, *Sequence node* kemudian akan mengembalikan status berdasarkan *child* nya kepada *parent node* nya, *Sequence node* hanya akan mengembalikan *ticks* dengan status *success* hanya jika semua *child* nya memberikan *ticks* dengan status *success*.
2. *Fallback Node*
   Node ini bekerja hampir seperti *sequence node* yang mengirimkan *ticks* kepada *child* nya yang berada paling kiri, perbedaaannya adalah *control-flow node* ini akan terus mengirimkan sinyal *ticks* sampai menemukan salah satu *child* nya yang mengembalikan *ticks* dengan stasus *success* atau *running*, dan kemudian mengembalikan nya pada *parent* nya, *fallback*

node hanya akan mengembalikan status *fail* hanya jika semua *child* nya mengembalikan status *fail*. *Node* ini tidak akan mengirimkan *tick* pada *child* selanjutnya jika ada *child* yang mengembalikan status *success.*

3. *Parallel Node*
   Berbeda dengan *control-flow node* yang lain *node* ini dapat memberikan *ticks* kepada *child* nya secara serentak, namun dalam hal mengembalikan status pada *parent* nya memiliki kesamaan dengan *sequence* dimana akan mengembalikan *success* jika semua *child* nya mengembalikan *success* dan mengembalikan *fail* jika salah satu *child* nya mengembalikan *fail*.

3.16 Pengujian *Black box*

Menurut Pressman (2012), *Black Box testing* atau pengujian kotak hitam atau juga disebut *Behavioral Testing*, berfokus pada persyaratan fungsional dari perangkat lunak. Artinya teknik *Black Boxtesting* memungkinkan untuk mendapatkan set kondisi masukan yang sepenuhnya akan melaksanakan semua persyaratan fungsional untuk suatu program. *Black Box testing* adalah pendekatan komplementer yang mungkin untuk mengungkap kelas yang berbeda dari kesalahan daripada metode *white box testing*. Pengujian *black box* berusaha menemukan kesalahan dalam kategori sebagai berikut :

1. Fungsi–fungsi yang tidak benar atau hilang
2. Kesalahan *Interface*
3. Kesalahan dalam struktur data atau akses *databaseeksternal*
4. Kesalahan kinerja
5. Inisialisasi dan kesalahan terminasi

3.17 Pengujian *Beta*

Menurut pressman (2012), pengujian *beta* adalah pengujian yang dilakukan pada satu atau lebih pelanggan oleh pemakai akhir perangkat lunak. Tidak seperti pengujian *alpha*, pengembang biasanya tidak ada sehingga pengujian *beta* merupakan sebuah aplikasi "*live*" dari perangkat lunak didalam suatu lingkungan yang tidak dapat dikontrol oleh pengembang. Pengguna mengingat semua masalah (*real* atau imajiner) yang mereka temui selama pengujian *beta* lalu melaporkannya kepada pengembang. Pelanggan merekam semua masalah yang mereka temui selama pengujian *beta* melaporkannya kepada pengembang. Sebagai hasil dari pelaporan masalah selama pengujian *beta* ini, pengembang perangkat lunak melakukan modifikasi dan kemudian mempersiapkan pelepasan produk perangkat lunak ke seluruh pelanggan.

Pengujian *beta* merupakan jenis pengujian yang paling produktif dari perangkat lunak(*Software*) non-rilis yang ada, tahap *beta* dimulai ketika sebuah produk didorong dari fungsional. *Bug* sudah diperbaiki, fitur ditingkatkan atau dirubah untuk penggunaan yang maksimal, antarmuka dan grafis telah diperbaiki dan masalah kinerja dioptimalkan. Meskipun pengujian *beta* adalah langkah ketiga dalam pengembangan produk, namun seringkali menjadi tahap yang terpanjang karena ada begitu banyak aspek yang harus diuji. Tahap *beta* biasanya dimulai ketika pengembang membuka produk kepada mereka yang belum terlibat.

Pengujian *beta* merupakan pengujian yang dilakukan secara objektif yang diuji secara langsung lewat kuisioner, untuk mencari nilai yang tepat maka menggunakan 2 rumus, pertama nilai persentase dari responden dan untuk mencari nilai rata-rata keseluruhan niai persentase dari responden:

$$P = \frac{S}{\text{Jumlah Responden}} x\ 100\%$$

Keteranagn :
P = Nilai Persentasi
S = Jumlah Frekuensi Dikali Dengan Skor
Jumlah Responden= Nilai Tertinggi Dikalikan Dengan Jumlah Responden

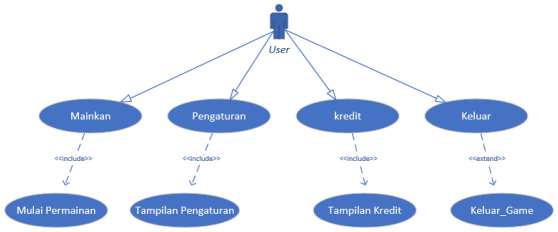
$$X = \frac{(n1 + n2 \ldots . nn)}{f}$$

Keterangan :
X= Nil'ai rata-rata kelompok.
n = Kumpulan data nilai persentase responden.
f= Nilai frekuensi pertanyaan / banyaknya pertanyaan.

4. RANCANGAN SISTEM ATAU APLIKASI

Pada tahap rancangan sistem / aplikasi ini bertujuan untuk memberikan suatu gambaran umum tentang *Design dan Flowchart*.

4.1 *Use Case Diagram Main Menu*



Gambar 2.*Use Case iagram Main Menu*

Pada *Flowchart* gambar 2 *user* memiliki beberapa pilihan yaitu *main menu* dimana pada saat *game* pertama kali dibuka maka akan keluar tampilan *main menu*, setiap tombol memberikan fungsi yang berdbeda seperti tombol Mainkan yang akan memulai permainan, tombol Pengaturan yang akan membuka tampilan pengaturan pada *game*, tombol Kredit untuk menampilkan kredit, dan tombol Keluar untuk menutup aplikasi atau keluar dari *game.*

4.2 Use Case Specification

Use case specification menjelaskan interaksi timbal balik antara sistem dan *user*. Berikut merupakan detailnya :

1) *Use case specification main menu*

Tabel 5 *Use case specification main menu*

Nama Use case	Use Case *Main Menu*

Pelaku	*User*	
Deskripsi	*Use Case* ini mendeskripsikan mengenai fungsi pada tombol – tombol main menu	
Tujuan	*User* dapat memulai permainan ataupun keluar dari permainan	
Bidang khas suatu *event*	Kegiatan Pemain	Respon *system*
	User mengklik tombol mainkan	Sistem merespon dengan mengarahkan ke dalam *scene* dimana permainan dimulai
	User mengklik tombol Pengaturan	Sistem mengarahkan ke *scene* dimana *user* dapat mengakses pengaturan *game* untuk menunjang jalannya permainan
	User mengklik tombol kredit	Sistem merespon dengan mengarahkan *user* ke *scene* dimana terdapat informasi tentang *game*
	User mengklik tombol keluar	Sistem merespon dengan mengarahkan *user* keluaar dari aplikasi *game*

2) *Use case specification* Mainkan

Tabel 6 *Use case specification* Mainkan

Nama *Use Case*	*Use Case* Mainkan	
Pelaku	*User*	
Deskripsi	*Use Case* ini mendiskripsikan mengenai apa – apa saja yang dapat dilakukan setelah *user* mengklik tombol mainkan	
Tujuan	*User* dapat memulai permainan	
Bidang Khas Suatu *Event*	Kegiatan Pemain	Respon Sistem
	User mengklik tombol mainkan	Sistem merespon dengan mengarahkan ke *scene* dimana permainan dimulai

3) *Use case specification* Pengaturan

Tabel 6 *Use case specification* Mainkan

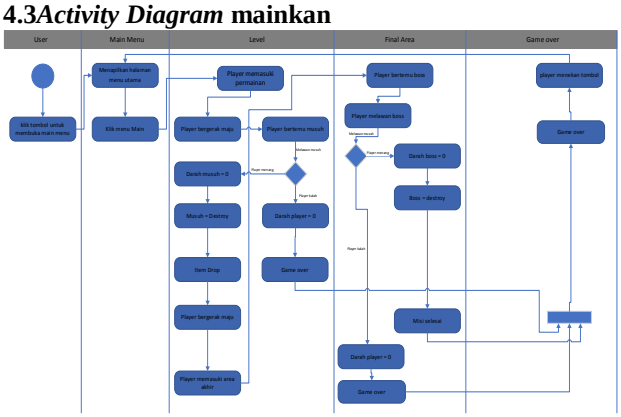
Nama *Use case*	*Use Case* Pengaturan	
Pelaku	*User*	
Deskripsi	*Use case* ini mendeskripsikan mengenai pengaturan *game* berupa pengaturan suara, grapik, dan control	
Tujuan	*User* dapat mengatur fitur *game* berupa *volume* suara, kualitas grafik dan tombol pada *keyboard*	
Bidang khas suatu *event*	Kegiatan Permainan	Respon Sistem
	User mengatur grafis dan fungsi tombol pada menu pengaturan	Sistem merespon dengan menyesuaikan pengaturan sesuai dengan keinginan *user*

4) *Use case specification credit*

Tabel 7 *Use case specification credit*

Nama *Use Case*	*Use case* kredit	
Pelaku	*User*	
Deskripsi	*Use case* ini mendeskripsikan tentang informasi mengenai *developers*	
Tujuan	*User* dapat mengetahui tentang pengembang	
Bidang khas suatu *event*	Kegiatan Pemain	Respon Sistem
	User mengklik tombol kredit pada menu utama	Sistem akan merespon dengan menampilkan *scene* kredit

4.3*Activity Diagram* mainkan

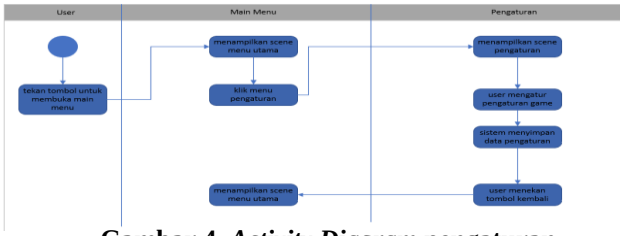


Gambar 3.*Activity diagram* mainkan

Activity diagram mainkan adalah sebuah rancangan yang bertujuan untuk menunjukkan beberapa proses yang terjadi saat *user* mengontrol karakter pada saat permainan dimulai.

4.4 *Activity Diagram* pengaturan

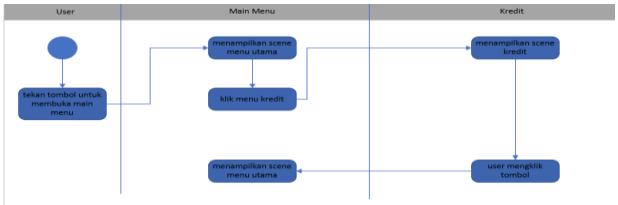
Activity diagram pengaturan menjelaskan tentang proses yang terjadi pada saat *user* mengklik tombol pengaturan.



Gambar 4. *Activity Diagram* pengaturan

4.5 *Activity Diagram* Kredit

Proses ini berlangsung saat pengguna mekan tombol kredit, dimana tombol kredit akan menampilkan informasii terkait dengan pengembang dan *game* tersebut.



Gambar 5. *Activity Diagram* Kredit

4.6 *Activity diagram* keluar

Proses ini berlangsung ketika *user* berada pada *scene main menu* lalu mengklik tombol keluar.

Gambar 6. *Activity diagram* keluar

4.7 IMPLEMENTASI

Pada tahap ini menampilkan tampilan beberapa interface dalam*game*maverick

1. Tampilan *Main Menu*

Gambar 7.Tampilan Halaman Utama

Pada Gambar 7adalah tampilan *menu* dimana pada tampilan ini terdapat 4 buah tombol dimana masing masing tombol akan membuka *sub-menu* dan menampilkan *interface* berdasarkan pilihan *menu* yang di klik.

Masing – masing tombol akan mengarahkan pemain pada antarmuka yang berbeda sesuai dengan fungsinya.

a) *Start Game* akan mengarahkan pemain kedalam permainan dan mulai menggerakkna karakter.
b) *Option* akan membuka pilihan *Setting* pada *game* seperti resolusi layar, kontrol, dan suara.
c) *Credit* menampilkan kredit
d) *Quit* akan menutup aplikasi

2. Tampilan dalam permainan

Gambar 8. Tampilan dalam permainan

Pada Gambar 8 Merupakan gambaran dalam *level* dimana *player* akan mengontrol karakter dan mencoba untuk menyelesaikan misi di dalam permainan.

Dalam *level* juga terdapat *item* yang akan di kumpulkan oleh pemain dan juga terdapat musuh yang akan menghalangi pemain dalam menyelesaikan misi dalam *game*.

3. Tampilan *Game over*

Gambar 9. Tampilan *Game over*

Pada Gambar 9 Merupakantampilanpada saat HP karakter pemain menjadi 0 atau kalah akan menampilkan beberapa atribut seperti *time, damage, Collected item,* dan *rank.*

4. Tampilan Misi berhasil

Gambar 10.Tampilan Misi berhasil

Pada Gambar 10ini akan menampilkan *interface* saat pemain berhasil menyelesaikan misi dalam *game.*

4.8 Implementasi *Behavior Tree*

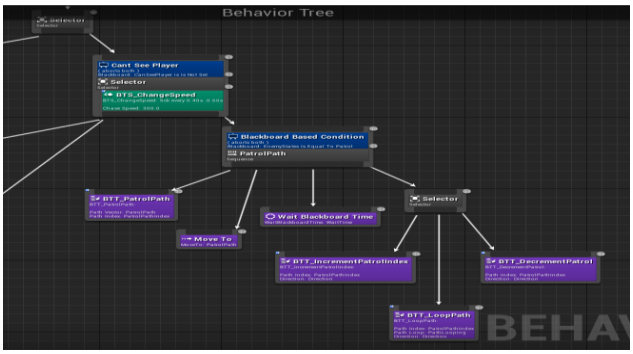
Dalam permainan ini *Autonom Agent* akan menggunakan *behaviour tree* sebagai pengendali perilaku *agent* dalam *game.* Adapun perubahan – perubahan perilaku npc berdasarkan kondisi yang sedang terjadi dalam *game.*

Dalam hal ini Unreal Engine 4 menyediakan *tool* yang dapat membantu dalam mendesain pembuatan *behaviour tree* , Bahasa pemograman yang digunakan dalam Unreal Engine 4 *editor* adalah bahasa C++ dalam bentuk *visual scripting* yang dapat membantu dalam membangun *behaviour tree*.

Berikut adalah adalah gambaran Implementasi *behaviour tree* pada *game* Maverick.

1) *Patrol path behavior*

Perilaku dari npc dalam *game* ini pada saat tidak mendeteksi adanya karakter pemain yang berada di dekatnya digambarkan pada gambar berikut.

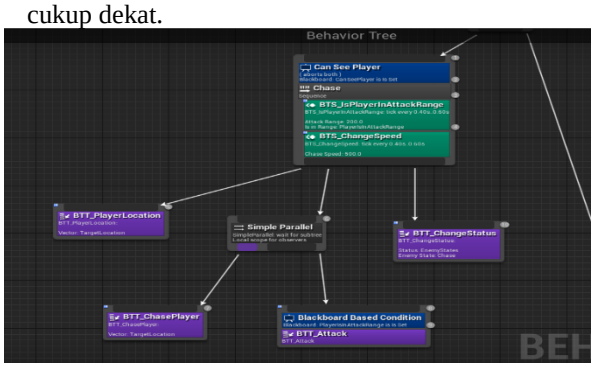


Gambar 11.*Patrol Path Behavior*

Pada Gambar 11menggambarkan bagaimana npc berprilaku saat pemain tidak terdeteksi, *Behaviour* dimulai pada *Root node yang* terhubung dengan *child node control flow* dibawah nya yang disebut s*elector, selector node* ini kemudian akan mengeksekusi *child node* yang memberikan *status true.* Pada *control flow node* dalam *behaviour patrol path* ini diberikan *sequence* yang pada dasarnya akan mengeksekusi semua *child* nya mulai dari kiri ke kanan sampai ada *node* yang mengirimkan *statusfail* dan *decorator node* yang berfungsi sebagai pengkondisian *behaviour* yang akan dieksekusi dimana dalam hal ini *parent node selector* dapat mengetahui bahwa *child node* ini mengirimkan *status true* dan akan dieksekusi. *Behaviour Tree* kemudian akan mengeksekusi setiap *task node* yang ada pada hirarki *patrol path behaviour* sampai kemudian salah satu *node* mengirimkan *status fail* dan akan mencoba mengeksekusi *child node* yang lainnya, *node* yang ada pada *behaviour* ini dimulai dari *sequence node* yang akan mengeksekusi *child node* paling kiri yaitu *node BTT_PatrolPath* yang sebelum nya sudah di buat dan akan menentukan titik kemana npc akan berjalan, berikut nya adala *move to node* yang akan memberikan perintah kepada npc untuk berjalan ke titik koordinasi yang di tentukan, selanjutnya adalah *wait blackboard time node* yang akan memberikan jeda agar *behaviour* ini tidak langsung mengeksekusi *node* lainnya yang dapat menyebabkan tingkah npc terus menerus berjalan menuju titik koordinasi berikutnya tanpa berhenti, setelah itu *child node* yang ada pada *sequence* adalah *selector node* yang hanya akan mengeksekusi salah satu *child node* nya yang mengirimkan *status true,child node* yang ada pada *selector node* adalah *BTT_IncrementPatrolIndex* yang mempunyai fungsi dalam memberikan titik koordinasi selanjutnya pada npc berdasarkan berapa jumlah titik koordinat yang diberikan secara bertahap, selanjutnya ada *BTT_LoopPath* yang akan memberikan instruksi agar npc berjalan kembali ke titik koordinat sebelumnya, kemuian ada *BTT_DecrementPatrolIndex* yang fungsinya hampir sama dengan *node incrementpatrolindex* namun akan memulai pemberian titik koordinat dari paling akhir sampai kembali ke titik koordinat awal.

2) *Chase and attack*

Perilaku npc berikut nya adalah dimana karakter pemain terlihat oleh npc kemudian melakukan pengejaran terhadap pemain dan akan menyrang pemain jika jarak antara npc dan pemain cukup dekat.



Gambar 12.*Chase and attack*

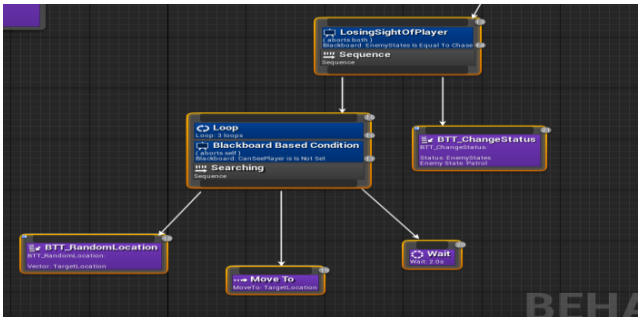
Pada Gambar 12 Menggambarkan tentangperilaku npc berubah dari mengejar pemain kemudian menyerang jika berada pada jarak yang dekat.

Dimulai dari *parent node selector* yang memilih *child node* nya yang mengirimkan *statustrue,* jika *node* yang ada pada hirarki *behaviour chase & attack* memberikan *status true* maka akan dieksekusi *task node* yang ada pada pada *behaviour* dimulai dari *Sequence node* yang akan mengeksekusi *child node* nya dari kiri kenan, *child node* yang akan dieksekusi pertama adalah *BTT_PlayerLocation task node* yang akan memberikan koordinat pemain, kemudian *node* selanjutnya adalah *control flow node* yang disebut sebagai *simple parallel node* yang dimana *node* ini dapat menjalankan semua *childnode* yang ada pada *node ini,* dalam hal ini *simple parallel node* diberikan *task node BTT_ChasePlayer* dan *BTT_Attack* dimana *BTT_ChasePlayer* akan memjalankan tugas agar mendekati pemain dan menyerang pemain pada saat yang sama jika jarak pemain dan npc sudah dekat, berikutnya adalah *task node BTT_ChangeStatus* dimana tugas dari *node* ini sebagai

indikasi *status* pada *decorator node* (kondisi) *chase true* agar *parent node* bisa mengeksekusi *behaviour* ini.

3) *Losing sight*

Perilaku *losing sight* terjadi jika musuh kehilangan pemain atau jika pemain berhasil kabur dari npc yang akan membuat npc mencari pemain disekitar lokasi dimana pemain terakhir terlihat.



Gambar 13.*Losing sight*

Pada Gambar 13diatas memperlihatkan hirarki *behaviour tree* pada saat kehilangan karakter pemain dan membuatnya mencari pemain di lokasi terakhir npc melihat pemain.

Dalam hirarki *behaviour* ini dimulai dari *sequence node* yang mempunyai 2 *child node* yang berupa *control flow node sequence* dan *task node*, eksekusi dimulai dari *sequence node* yang juga mempunyai *child node* berupa *task node*, dari kiri terdapat *task BTT_RandomLocation* yang memnpunyai fungsi untuk mendapatkan titik koordinat *random* dalam *map* dan kemudian di melanjutkan eksekusi pada *Move To task node* yang menggerakkan karakter npc menuju kordinat tersebut dan *task* terakhir adalah *wait* yang memberikan instruksi untuk menunggu dalam beberapa saat sebelum kemudian melanjutkan eksekusi pada *node* selanjutnya dan kemudian kembali pada *parent node*, berikut nya adalah *task node BTT_ChangeStatus* yang akan mengganti *status* pada kondisi dari masing - *behaviour*.

Keselurahan bagian *behaviour tree* dalam *game* maverick dapat dilihat pada gambar berikut.

Gambar 14.*Behavior tree* Maverick

5 KESIMPULAN

Berdasarkan uraian dari masing-masing bab dan hasil pembahasan dapat ditarik kesimpulan sebagai berikut :

1. Menggunakan *Behaviour Tree* dalam agen cerdas dalam *game* mampu memberikan perilaku yang adaptif terhadap pemain.
2. *Behaviour Tree* memberikan *output* yang sesuai dalam perilaku *patrol path, chase & attack player,* dan juga *losing sight* terhadap pemain.

5.1 SARAN

Saran – saran yang dapat diberikan untuk mengembangkan *game Maverick* yaitu :

1. Menambahkan karakter musuh agar lebih beragam
2. Membuat level tambahan dan memberikan tantangan yang berbeda pada setiap level nya
3. Menambahkan *item* agar *gameplay* tidak monoton
4. *Design UI* yang lebih menarik
5. Tingkat kesulitan *game* yang dapat diatur

6 DAFTAR PUSTAKA

Adams, E. (2014). *Pengertian Genre Game.* Jakarta: PT. Grasindo.

Alif, I. (2015). *3D Wayang Adventure Game untuk pengenalan budaya wayang nusantara menggunakan A* Algorithm sebagai pembangkit perilaku pencarian pada NPC. Thesis.* Malang: Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Colledanchise, M., & Orgen, P. (2018). *Behavior Tree in Rootics and AI.* Florida: CRC Press.

Fahrezi, I. A. (2018). *Pengembangan game platform "Adventure Of Tom The Black Hair" Dengan mengimplementasikan metode finite state machine. Tesis.* Yogyakarta: Universitas Islam Indonesia.

Mardiyasa, P. A. (2016). *Film animasi Pembelajaran Sistem Pencernaan Manusia Pada Kelas VIII SMP Negeri 3 Banjar Tahun Ajaran 2015/2016.* Bali: Universitas Pendidikan Ganesha Singaraja.

Munir. (2012). *Multimedia Konsep & Aplikasi dalam pendidikan.*Bandung: Alfabeta.

Poole, D. L., & Macworth, A. K. (2017). *Artificial Intellegence : Foundation of Computational Agents, 2nd Edition.*Cambridge: Cambridge University Press.

Pressman, R. S. (2012). *Rekayasa Perangkat Lunak Pendekatan Praktisi Edisi 7.* Yogyakarta: ANDI.

Rosa, A. (2014). *Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Objek. Informatika.* Bandung: Informatika.

Shah, R. (2015). *Master the Art of Unreal Engine 4 – Blueprint* . United Kingdom: Ebook.

Sugito, A., Hariadi, M., & Fanani, A. Z. (2017). *Pergerakan Adaptif Nonplayer Character Teman Berbasis Event menggunakan Behavior Tree dan Rule Based System . Thesis.* Semarang: UNiversitas Dian Nuswantoro.

Sukamto, R. (2014). *Analisa dan Desain Sistem Informasi.* Yogyakarta: Andi Offset.

Yunus, M. (2015). *Game Edukasi Matematika untuk Sekolah Dasar*. *Jurnal Ilmiah Ilmu Komputer*, Vol 10, No. 2 . Samarinda : Universitas Mulawarman.