DESAIN *NON PLAYABLE CHARACTER* SEBAGAI MUSUH PADA GAME SEPEDA MENGGUNAKAN METODE *MARKOV STATE MACHINE*

SKRIPSI

Oleh:
NAZAR PESONA WILDAN
NIM. 09650080

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2016

DESAIN *NON PLAYABLE CHARACTER* SEBAGAI MUSUH PADA GAME SEPEDA MENGGUNAKAN METODE *MARKOV STATE MACHINE*

SKRIPSI

Diajukan Kepada :
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang
Untuk Memenuhi Salah Satu Persyaratan Dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh:
NAZAR PESONA WILDAN
NIM. 09650080

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG
2016

DESAIN *NON PLAYABLE CHARACTER* SEBAGAI MUSUH PADA GAME SEPEDA MENGGUNAKAN METODE *MARKOV STATE MACHINE*

SKRIPSI

Oleh:

NAZAR PESONA WILDAN

NIM. 09650080

Telah Diperiksa dan Disetujui untuk Diuji:

Tanggal, 8 Juni 2016

Dosen Pembimbing I	Dosen Pembimbing II
Fresy Nugroho, MT	**Yunifa Miftachul A, MT**
NIP. 19710722 200101 1 001	**NIP. 19830616 201101 1 004**

Mengetahui,

Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian

NIP. 19740424 200901 1 008

HALAMAN PENGESAHAN

DESAIN *NON PLAYABLE CHARACTER* SEBAGAI MUSUH PADA GAME SEPEDA MENGGUNAKAN METODE *MARKOV STATE MACHINE*

SKRIPSI

Oleh:
NAZAR PESONA WILDAN
NIM.09650080

Telah Dipertahankan di Depan Dewan Penguji Skripsi
Dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer (S.Kom)

Tanggal, 27 Juni 2016

Susunan Dewan Penguji			**Tanda Tangan**
Penguji Utama	:	**Hani Nurhayati, M.T** **NIP. 19780625 200801 2 006**	(.....)
Ketua Penguji	:	**Fachrul Kurniawan, M.T** **NIP. 19771020 200901 1 001**	(.....)
Sekretaris Penguji	:	**Fresy Nugroho, M.T** **NIP. 19710722 201101 1 001**	(.....)
AnggotaPenguji	:	**Yunifa Miftachul Arif, M.T** **19830616 201101 1 004**	(.....)

Mengetahui,
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

PERNYATAAN

ORISINALITAS PENELITIAN

Saya yang bertanda tangan di bawah ini :

Nama : Nazar Pesona Wildan

NIM : 09650080

Fakultas / Jurusan : Sains dan Teknologi / Teknik Informatika

Judul Penelitian : Desain *Non Playable Character* Sebagai Musuh pada Game Sepeda Menggunakan Metode *Markov State Machine*

Menyatakan dengan sebenar-benarnya bahwa hasil penelitian saya ini tidak terdapat unsur-unsur penjiplakan karya penelitian atau karya ilmiah yang pernah dilakukan atau dibuat oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata hasil penelitian ini terbukti terdapat unsur-unsur jiplakan, maka saya bersedia untuk mempertanggung jawabkan, serta diproses sesuai peraturan yang berlaku.

Malang, 9 Juni 2016

Penulis

Nazar Pesona Wildan
NIM. 09650080

MOTTO

Allah Tidak Akan Membebani Seseorang Melainkan Sesuai dengan Kemampuannya

Terus Berusaha Karena dengan Usaha Yang Baik dan Sungguh- sungguh Akan Mendapat Hasil Yang Baik Pula

لَا يُكَلِّفُ ٱللَّهُ نَفْسًا إِلَّا وُسْعَهَا ... ٢٨٦

Allah tidak membebani seseorang melainkan sesuai dengan kesanggupannya.(Q.S Al-Baqoroh : 286)

PERSEMBAHAN

Dengan menyebut Asma-Mu yang Agung, syukurku akan segala karunia-Mu, serta shalawat serta salam kepada Muhammad SAW kekasih-Mu,

Ya Allah, semoga setiap langkah selalu Engkau ridhoi dengan segala rahmat-Mu

---

Saya Persembahkan karya ini kepada semua pihak yang telah membantu dalam menyelesaikan karya ini

Bapak Tercinta Alm. Busyro Fuad dan Ibu Tercinta Enik Sriyani, orang tua hebat yang selalu menyayangi dan sabar memberikan semangat
Serta do'a kepada saya.

Kakakku tercinta Dila Umnia Soraya yang telah banyak membantuku dan selalu memberikan motivasi,
Serta adikku Tersayang Alfa Sanatin Illa Romza yang menjadi penghibur hati

Bapak Fresy Nugroho, M.T dan Bapak Yunifa Miftachul A., M.T
Selaku Dosen pembimbing yang dengan sabar membimbing saya

Sahabat dan saudara kelas C Teknik Informatika 2009
Yang selalu menjadi kekuatan dalam semangat

Teman, rekan dan Sahabatku UIN Malang,
Khususnya teman-teman jurusan Teknik Informatika 2009
Fauzadin Gansala sebagai rekan tim pembuatan game

Teman seperjuangan di Malang,
Agil, Andang, Syamsul, Reza dan Bang Zakki

Kepada setiap orang yang telah membantu

Terima kasih.

KATA PENGANTAR

بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ

Segala puji syukur penulis panjatkan kehadirat Allah SWT, karena atas segala rahmat dan limpahan hidayah-Nya, skripsi yang berjudul "Desain *Non Playable Character* Sebagai Musuh pada Game Sepeda Menggunakan Metode *Markov State Machine*" ini dapat penulis selesaikan sebagai salah satu syarat untuk memperoleh gelar sarjana pada program studi Teknik Informatika jenjang Strata-1 Universitas Islam Negeri (UIN) Maulana Malik Ibrahim (Maliki) Malang. Sholawat serta salam semoga senantiasa Allah limpahkan kepada Nabi Muhammad SAW, keluarga, sahabat dan seluruh umatnya yang rela berkorban demi kemajuan Islam.

Dalam penyelesaian skripsi ini, banyak pihak yang telah memberikan bantuan baik moril maupun materiil. Atas segala bantuan yang telah diberikan, penulis ingin menyampaikan doa dan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Prof. DR. H. Mudjia Rahardjo, M.Si, selaku Rektor Universitas Islam Negeri (UIN) Maulana Malik Ibrahim Malang beserta seluruh staf. Dharma Bakti Bapak dan Ibu sekalian terhadap Universitas Islam Negeri Malang turut membesarkan dan mencerdaskan penulis.
2. Dr. Hj. Bayyinatul Muchtaromah., drh.,M.Si, selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Malang beserta staf. Bapak dan ibu

sekalian sangat berjasa memupuk dan menumbuhkan semangat untuk maju kepada penulis.

3. Dr. Cahyo Crysdian, selaku Ketua Jurusan Teknik Informatika, yang telah memotivasi, membantu dan mengarahkan penulis menyelesaikan penulisan skripsi ini.
4. Bapak Fresy Nugroho, M.T selaku dosen pembimbing satu dan Bapak Yunifa Miftachul Arif, M.T selaku dosen pembimbing dua, yang telah banyak memberikan bimbingan, nasihat, dan pengarahan dalam penyelesaian penulisan skripsi ini.
5. Seluruh Dosen Universitas Islam Negeri (UIN) Maliki Malang, khususnya Dosen Teknik Informatika dan staf yang telah memberikan ilmu kepada penulis, dan dukungan untuk menyelesaikan penulisan skripsi ini.
6. Bapak dan Ibuku tersayang, kakakku, adikku dan seluruh keluarga besar di Nganjuk yang telah banyak memberikan doa, motivasi dan dorongan dalam penyelesaian skripsi ini.
7. Semua sahabat yang telah membantu penulis hingga terselesaikannya skripsi ini, khususnya kepada teman-teman TI-UIN Malang angakatan 2009 semoga Allah SWT memberikan balasan yang setimpal atas jasa dan bantuan yang telah diberikan.
8. Dan kepada seluruh pihak yang mendukung penulisan skripsi ini, yang tidak dapat disebutkan satu persatu, penulis ucapkan terima kasih yang sebesar-besarnya.

Berbagai kekurangan dan kesalahan mungkin pembaca temukan dalam penulisan makalah ini, untuk itu penulis menerima segala kritik dan saran dari pembaca. Semoga penulisan skripsi ini bermanfaat bagi pembaca sekalian.

Wassalamualaikum Wr. Wb.

Malang, 9 Juni 2016

Penulis

Nazar Pesona Wildan

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Wildan, Nazar Pesona. 2016. **Desain *Non Playable Character* Sebagai Musuh pada Game Sepeda Menggunakan Metode *Markov State Machine*.** Skripsi. Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: (I) Fresy Nugroho, M.T (II) Yunifa Miftachul Arif, M.T

Kata Kunci: *Game, Simulasi, NPC, AI, Markov State Machine.*

Perkembangan game saat ini sudah sangat pesat. Setiap hari bisa bermunculan banyak game baru baik itu merupakan game android, PC maupun game console. Salah satu genre game yang mulai dikembangkan adalah game simulasi. Jenis game ini mensimulasikan suatu kegiatan tertentu dalam dunia nyata ke dalam komputer, sehingga para user seperti melakukan kegiatan nyata mereka dalam dunia virtual. Komponen yang harus ada dalam game simulasi adalah musuh atau pengganggu yang bertujuan untuk mempersulit permainan. Dalam game ini musuh yang dibuat adalah berupa NPC(*non playable character*) yang diberikan perilaku untuk mengganggu karakter utama dalam mencapai misi permainan. Metode yang digunakan untuk menentukan perilaku NPC adalah Markov *State Machine*. Metode ini mengatur perilaku NPC terhadap *player* untuk patrol, mengejar *player* atau menyerang *player*. Uji coba implementasi *Markov State Machine* pada perilaku NPC sesuai dengan desain output perilaku yang didesain sebelumnya, yaitu patrol, mengejar dan menyerang *player*.

ABSTRACT

Wildan, Nazar Pesona. 2016. **Design Non Playable Character in The Bike Game as an Enemy Using The Markov State Machine.** Thesis. Informatics Department of Faculty of Science and Technology. Maulana Malik Ibrahim State Islamic University, Malang. Adviser: (I) Fresy Nugroho, M.T (II) Yunifa Miftachul Arif, M.T

Keywords: *Game, Simulation, NPC enemies, AI, Markov State Machine.*

Development of the game is now very rapidly. Every day could be popping up a lot of games wether it's the android game, PC game or console game. One genre of game that was developed is a simulation game. This type of game simulate of a certain activity in the real word into the computer, so the user like doing real activities in the virtual world. Components that must be present in the simulation game is an enemy or an attacker aiming to make the game more harder. In this game the enemies that are created are in the form of a NPC(Non Playable Character) given behavior to interfere with the main character in achieving the mission of the game. The method used to determine the bahavior of the NPC is a Markov State Machine. This method regulate the behavior of the NPC to patrol, chasing player or attacking player. Result of Markov State Machine implementation on behavior of the NPC accordance with the design output behavior is designed previously there are patrol, chasing and attacking player.

الملخص

ولدان، نزار فسونا. 2016. **تصميم الشخصية غير قابلة اللعبة كالأعداء في لعبة الدراجة بالأسلوب آلة الدولة ماركوف**. البحث الجامعي. قسم التقنية المعلوماتية كلية العلوم والتكنولوجيا الجامعة الإسلامية الحكومية مولانا مالك إبراهيم مالانج. المشرف: (1) فريشي نوغروهو الماجيستر، (2) يونيفا مفتاح العارف الماجيستر

كلمات البحث: لعبة، محاكاة، (الشخصية غير قابلة) *NPC* عدو، *AI*, آلة الدولة ماركوف.

تطوير اللعبة هي الآن بسرعة كبيرة. كل يوم يمكن أن تظهر لعبة جديدة في الكثير سواء كان هي ألعاب الروبوت، أو أجهزة الكمبيوتر أو وحدة لعبة. احد النوع من اللعبة التي يجري تطويرها هي لعبة محاكاة. اللعبة من هذا النوع تحاكي النشاط المعين في العالم الحقيقي إلى الكمبيوتر، حتى أن المستخدم مثل الأنشطة الحقيقية في العالم الافتراضي. المكونات التي يجب أن تكون موجودة في لعبة المحاكاة هي أعداء أو مهاجم تهدف إلى تعقيد اللعبة. في هذه اللعبة الأعداء المصورة (الشخصية غير قابلة) *NPC* التى أعطيت الأعمال المشوسة على الشخصية الرئيسية في تحقيق مهمة من اللعبة. الطريقة المستخدمة لتحديد أعمال *NPC* هو آلة الدولة ماركوف. هذه الطريقة تنظم أعمال *NPC* للعدو ألى اللاعب للدورية، مطاردة اللاعب أو مهاجمة اللاعب. اختبارة تنفيذ محاكمة آلة الدولة ماركوف على *NPC* للعدو وفقا بتصميم الناتج الأعمال المصمم سابقا، وهي الدورية، والمطاردة والمهاجمة في اللاعب.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Salah satu cabang ilmu yang diterangkan dalam Al Qur'an adalah ilmu pengetahuan, yang menjelaskan tentang bagaimana bumi mengitari matahari dan bagaimana bulan mengitari bumi. Termasuk dalam ilmu pengetahuan adalah ilmu tentang komputer. Walaupun di dalam Al Qur'an tidak dijelaskan secara langsung, akan tetapi Al Qur'an menjelaskan bagaimana kita umat manusia diperintahkan untuk menggali ilmu yang ada di langit dan di bumi termasuk ilmu tentang pemrograman. Seperti yang telah dijelaskan dalam Surah Yunus ayat 101 berikut:

قُلِ ٱنظُرُواْ مَاذَا فِي ٱلسَّمَٰوَٰتِ وَٱلۡأَرۡضِۚ وَمَا تُغۡنِي ٱلۡأٓيَٰتُ وَٱلنُّذُرُ عَن قَوۡمٖ لَّا يُؤۡمِنُونَ ١٠١

Artinya: "Katakanlah: Perhatikanlah apa yang ada di langit dan di bumi. tidaklah bermanfaat tanda kekuasaan Allah dan Rasul-rasul yang memberi peringatan bagi orang-orang yang tidak beriman".

Ayat ini mendorong umat manusia untuk mengembangkan ilmu pengetahuan melalui kontemplasi, eksperimentasi dan pengamatan. Perintah Allah SWT dalam ayat tersebut di atas sudah jelas bahwa manusia diperintah untuk menggali semua tanda yang ada di langit dan bumi untuk menghasilkan

ilmu pengetahuan. Termasuk di dalamnya yaitu ilmu komputer yang menjelaskan tentang pemrograman. (Quraish Shihab)

Perkembangan *game* saat ini sudah sangat pesat. Setiap hari bisa bermunculan banyak *game* baru baik itu merupakan *game* android, PC maupun *game* console. Perkembangan *game* didukung oleh para user yang tidak hanya dari golongan anak-anak dan remaja, bahkan sampai orang dewasa juga memainkannya. Seiring perjalanan waktu, sudah banyak genre *game* yang bermunculan mulai dari *game action, fighting, adventure, racing* hingga simulasi.

Mengutip pernyataan Nario Baba bahwa perkembangan teknologi telah banyak mengubah struktur dasar dari perindustrian kita dalam kehidupan sehari-hari. Perindustrian beberapa dekade yang lalu telah membuat kemajuan luar biasa dan berkembang pesat. Salah satu contoh yang paling terlihat adalah industri *game* komputer (Nario Baba : 2007).

Salah satu genre *game* yang mulai dikembangkan adalah *game* simulasi. Jenis *game* ini mensimulasikan suatu kegiatan tertentu dalam dunia nyata ke dalam komputer, sehingga para user seolah- olah seperti melakukan kegiatan nyata mereka dalam dunia virtual. Jenis *game* ini disimulasikan ke dalam komputer berupa bentuk virtual *game* 3D yang sebagian merupakan gambaran dari bentuk- bentuk nyata dalam kehidupan manusia.

Komponen yang harus ada dalam *game* simulasi adalah musuh atau pengganggu yang bertujuan untuk mempersulit permainan, sehingga *game* akan terasa lebih menantang. Dalam *game* ini musuh yang dibuat adalah berupa NPC(*non playable character*) yang diberikan perilaku untuk mengganggu karakter utama dalam mencapai misi permainan. Musuh akan

mengejar karakter utama jika memasuki wilayah atau area patroli musuh. Penambahan NPC ini bertujuan untuk menjadikan *game* simulasi ini tidak cepat membosankan.

Metode yang digunakan untuk menentukan perilaku NPC adalah *Markov State Machine*. Ini adalah metode yang memasukkan konsep metode *Markov Chain* ke dalam *Finite State Machine*. Metode ini berfungsi menentukan perilaku NPC untuk mentarget karakter *user* jika memasuki area musuh. NPC akan dipasang di beberapa titik dan memiliki area masing-masing. Jika karakter *user* memasuki area NPC maka secara otomatis NPC akan aktif dan melakukan aksi yang telah dipasangkan sebelumnya dan sesuai dengan keadaan *player* saat itu.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dikemukakan dapat dirumuskan masalahnya adalah

1. Bagaimana cara mengimplementasikan metode *Markov State Machine* pada NPC sehingga dapat berperilaku terhadap *player*?

1.3 Batasan Masalah

Mengingat masih banyaknya pengembangan yang bisa dilakukan dalam *game* sepeda, maka dalam pembahasan dibatasi pada:

1. Perilaku musuh akan diaplikasikan kepada beberapa NPC.
2. Metode yang digunakan untuk menentukan perilaku NPC yaitu metode *Markov State Machine*.

3. *Engine* dalam pembuatan menggunakan unity 3D , karena unity mampunyai *performance* yang baik dalam hal mengolah grafis 3D tetapi mempunyai proses yang ringan, sehingga dapat di jalankan di komputer spesifikasi rendah.

1.4 Tujuan Penelitian

Tujuan dilakukannya penelitian ini adalah untuk:

1. Mengimplementasikan metode *Markov State Machine* pada NPC untuk menentukan perilaku terhadap *player*.

1.5 Manfaat Penelitian

Hasil dari penelitian ini diharapkan akan memberikan manfaat terhadap pengembangan *game* di indonesia antara lain :

1. Mengembangkan teknologi virtual untuk simulasi di Indonesia.
2. Membuat media belajar berupa *game* interaktif.

1.6 Metode Penelitian

Berikut adalah langkah-langkah metode yang yang di gunakan dalam penelitian,terdiri dari :

1. Studi Literatur
    1) Pengumpulan informasi tentang metode *Markov Chain*.
    2) Pengumpulan informasi untuk spesifikasi *hardware* yang dibutuhkan.
    3) Pengumpulan informasi mengenai NPC(*non playable character*)

2. Perumusan Masalah

    1) Menganalisa *software game* sepeda yang digunakan.

    2) Menganalisa kebutuhan *user*.

    3) Menganalisa waktu pembuatan NPC.

3. Analisis

    1) Identifikasi dan desain sistem.

    2) Menganalisa unity 3d dan bahasa pemrograman C#/ javascript.

    3) Menganalisa sistem virtualisasi unity 3d

    4) Menganalisa sistem *game* sepeda.

4. Perancangan Sistem

    1) Pembuatan desain *input*.

    2) Pembuatan desain proses.

    3) Pembuatan desain *output*.

    4) Perancangan perilaku NPC pada *game*.

5. Ujicoba dan Evaluasi

    Ujicoba dilakukan sampai sistem benar- benar *ready to use*. Kekurangan yang terjadi diperbaiki dalam lingkup batasan masalah. Evaluasi dilakukan untuk mengetahui apakah sistem yang dibangun sudah mendekati yang diharapkan.

6. Dokumentasi

    Penulisan laporan skripsi merupakan dokumentasi dari keseluruhan pelaksanaan penelitian. Diharapkan dokumentasi penelitian berguna dan bermanfaat untuk penelitian dan pengembangan lebih lanjut.

1.7 Sistematika Penulisan

Dalam penulisan skripsi ini, secara keseluruhan terdiri dari lima bab yang masing-masing bab disusun dengan sistematika sebagai berikut:

BAB I PENDAHULUAN

Pada bab ini merupakan bab pendahuluan, yang di dalamnya memuat latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian dan sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Pada bab tinjauan pustaka menjelaskan teori yang berhubungan dengan permasalahan penelitian yang meliputi:

BAB III ANALISIS DAN PERANCANGAN SISTEM

Bab ini menjelaskan tentang pembuatan desain *non playable character* (NPC) sebagai musuh pada *game* sepeda dengan implementasi *Markov State Machine* sebagai kecerdasan pada NPC yang meliputi metode penelitian yang digunakan, perancangan aplikasi dan desain aplikasi yang akan digunakan.

BAB IV HASIL DAN PEMBAHASAN

Pada bab ini menjelaskan hasil yang dicapai dari perancangan sistem dan implementasi program, yaitu dengan melakukan pengujian sehingga dapat ditarik kesimpulan.

BAB V PENUTUP

Pada bab terakhir berisi kesimpulan berdasarkan hasil yang telah dicapai dari pembahasan. Serta berisi saran yang diharapkan sebagai bahan pertimbangan oleh pihak-pihak yang akan melakukan pengembangan terhadap program ini.

BAB II

TINJAUAN PUSTAKA

2.1 Permainan/*Game*

Dalam bahasa Indonesia "*Game*" berarti "Permainan". Permainan yang dimaksud dalam *game* juga merujuk pada pengertian sebagai "kelincahan intelektual" (*intellectual playbility*). Sementara kata "*game*" bisa diartikan sebagai arena keputusan dan aksi permainanya. Ada target-target yang ingin dicapai pemainnya. Kelincahan intelektual, pada tingkat tertentu, merupakan ukuran sejauh mana *game* itu menarik untuk dimainkan secara maksimal. (Arix, 2011:5)

Menurut Salen dan Zimmerman (2004:80) dalam *Rules of Play: Game Design Fundamentals* bahwa *game* adalah sebuah sistem di mana pemain terlibat dalam konflik buatan, didefinisikan oleh aturan, yang menghasilkan hasil kuantitatif. Dalam permainan muncul dari hubungan antara tindakan pemain dan sistem hasil, itu adalah proses dimana seorang pemain mengambil tindakan dalam sistem yang dirancang dari sebuah permainan dan sistem merespon tindakan. Artinya dari suatu tindakan dalam permainan berada dalam hubungan antara tindakan dan hasil. *Game* edukasi merupakan bidang akademik baru dan bidang interdisipliner pembelajaran, yang berfokus pada *game*, bermain dan fenomena terkait.

2.2 Studi Literatur

Untuk membatu kelancaran dalam pembuatan sistem ini, maka di butuhkan pula literatur. Buku- buku yang berkaitan dengan algoritma, tentang pemrograman C# dan JavaScript. Buku tentang algoritma *Finite State Machine* juga menjadi rujukan untuk pengerjaan *game* ini, begitu pula dengan buku buku tentang *Script* bahasa pemrograman C# yang bisa di dapat pada

literatur manual *scripting* yang di sediakan *Game engine*, khususnya dalam fungsi yang terintegrasi dengan *game engine*, literatur *online* yang berupa jurnal, *e-book*, *proceding* juga dilakukan untuk menunjang keterbatasan buku-buku yang membahas tentang *Markov State Machine.*

Selain literatur dari buku dan jurnal, juga dibutuhkan literatur mengenai penelitian yang telah dilakukan sebelumnya. Seperti penelitian yang dilakukan Jun Chen, Prapun Suksompong dan Toby Berger yang berjudul "Communication through a Finite State Machine with Markov Property". Penelitian difokuskan pada pemodelan Finite State Machine yang pada bagian selanjutnya akan dikembangkan menjadi model sistem pengguna tunggal. Hal itu dipengaruhi oleh kemampuan timbal balik yang ada pada proses pengambilan keputusan Markov. Penelitian ini membahas tentang bagaimana properti dari rantai markov dapat digabungkan dengan Finite State Machine pada prakteknya.

2.3 *Non Playable Character*(NPC)

Non Playable Character merupakan jenis *autonomus agent* yang ditunjukan untuk penggunaan komputer animasi dan media interaktif seperti *games* dan *virtual reality*. Agen ini mewakili tokoh dalam cerita atau permainan dan memiliki kemampuan untuk improvisasi tindakan mereka. Ini adalah kebalikan dari seorang tokoh dalam sebuah film animasi, yang tindakannya ditulis di muka, dan untuk *"avatar"* dalam sebuah *game* atau *virtual reality*, tindakan yang diarahkan secara real time oleh *player*. (Reynolds, 1999)

Pada permainan komputer, perilaku agen berupa *non playable character*(NPC) hingga saat ini terus dikembangkan. Untuk *genre* permainan komputer tertentu misalnya *First Person Shooter*(FPS) dan *Real-Time Strategy*(RTS), perilaku NPC merujuk pada kemampuan permainan untuk menantang pemain pada tingkat taktis dan strategis. (Supeno Mardi dkk, 2011)

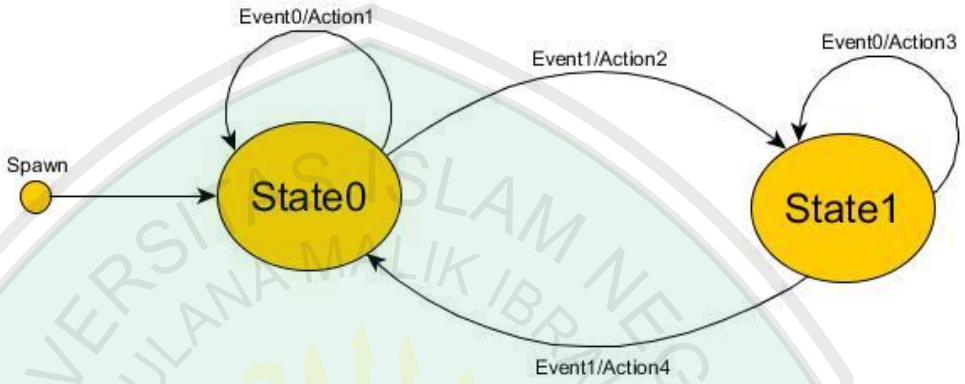
Kedudukan NPC dalam suatu *game* adalah sebagai komponen pelengkap yang dapat menjadikan *game* tersebut tidak membosankan. Biasanya dalam sebuah *game,* perilaku NPC ditentukan sebagai musuh atau pengganggu dari karakter yang dimainkan *user*. Banyak teknik yang digunakan untuk membuat NPC yang dapat berperilaku realistis dan bervariasi.

2.4 *Finite State Machine*

FSM (*Finite State Machine*) adalah sebuah metodologi perancangan sistem kontrol yang menggambarkan tingkah laku atau prinsip kerja sistem dengan menggunakan tiga hal berikut: *state*(keadaan), *event*(kejadian) dan *action*(aksi). Menurut Ian Millington (2006) dalam bukunya yang berjudul *Artificial Itelligence for Games* menyebutkan bahwa *Finite State Machines* (FSM) masuk dalam ranah *Decision Making* (pembuat keputusan) pada *Artificial Intelligence* (AI).

Pada satu saat dalam periode yang cukup signifikan, sistem akan berada pada salah satu *state* yang aktif. Sistem dapat beralih atau bertransisi menuju *state* lain jika mendapat masukan atau *event* tertentu, baik yang berasal dari perangkat luar atau komponen dalam sistemnya itu sendiri.

Transisi keadan ini umumnya juga disertai oleh aksi yang dilakukan sistem ketika menanggapi masukan yang terjadi. Aksi yang dilakukan tersebut dapat berupa aksi yang sederhana atau melibatkan rangkaian proses yang relatif kompleks.



Gambar 2.1 Contoh diagram *state* sederhana

(Sumber: Setiawan, 2006)

Diagram tersebut memperlihatkan FSM dengan dua buah *state* dan dua buah *input* serta empat buah aksi output yang berbeda. Pada saat sistem dimulai, sistem akan berada pada *state0,* ketika mendapat masukan *event0* maka akan menghasilkan keluaran *action1*. Hal itu terjadi dikarenakan sistem berada di keadaan awal dan belum mendapat kejadian apapun. Kemudian sistem akan mulai bertransisi ke *state1* ketika mendapat masukan kejadian/*event1* dan akan melakukan *action1* yang merupakan bentuk aksi dari adanya *event1*. Begitu seterusnya sistem akan terus bertransisi berdasarkan masukan yang terjadi dan akan merespon dengan keluaran aksi.

Finite State Machine(FSM) dalam *game* berfungsi sebagai pembuat cerita atau skenario bagi NPC. Meliputi apa yang akan dilakukan NPC ketika bertemu dengan *player*.

2.5 *Markov Chain*

Analisa *Markov Chain* adalah suatu metode yang mempelajari sifat-sifat suatu variabel pada masa sekarang yang didasarkan pada sifat- sifatnya di masa lalu dalam usaha menaksir sifat- sifat variabel tersebut di masa yang akan datang. Konsep dasar Markov Chain baru diperkenalkan sekitar tahun 1907, oleh seorang Matematisi Rusia Andrei A. Markov(1856-1922) (Edi Abdurachman, 1999). Dalam analisis Markov yang dihasilkan adalah suatu informasi probabilistik yang dapat digunakan untuk membantu pembuatan keputusan. Jadi, analisis ini bukan suatu teknik optimisasi melainkan suatu teknik deskriptif. Analisis Markov merupakan suatu bentuk khusus dari model probabilistik yang lebih umum dikenal sebagai proses stokastik(*Stochastic process*).

Proses stokastik adalah suatu himpunan variabel acak X_1 yang terdefinisi pada suatu ruang sampel (Howard M. Taylor, 1998). kata stokastik(*stochastics*) merupakan jargon untuk keacakan. Oxford Dictionary menafsirkan proses stokastik sebagai suatu barisan kejadian yang memenuhi hukum- hukum peluang. Apabila suatu kejadian tertentu dari suatu rangkaian eksperimen tergantung dari beberapa kemungkinan kejadian, maka rangkaian eksperimen tersebut disebut Proses Stokastik.

Konsep dasar analisis markov adalah *state* dari sistem atau *state* transisi, sifat dari proses ini adalah apabila diketahui proses berada dalam suatu keadaan tertentu, maka peluang berkembangnya proses di masa mendatang hanya tergantung pada keadaan saat ini dan tidak tergantung pada keadaan sebelumnya, atau dengan kata lain rantai Markov adalah rangkaian

proses kejadian dimana peluang bersyarat kejadian yang akan datang tergantung pada kejadian sekarang.

Analisis Markov ini sangat sering digunakan untuk membantu pembuatan keputusan dalam bisnis dan industri, misalnya dalam masalah ganti merek, masalah hutang-piutang, masalah operasi mesin, analisis pengawasan dan lain-lain. Informasi yang dihasilkan tidak mutlak menjadi suatu keputusan, karena sifatnya yang hanya memberikan bantuan dalam proses pengambilan keputusan.

2.6 Probabilitas Transisi

Probabilitas Transisi adalah perubahan dari satu status ke status yang lain pada periode (waktu) berikutnya dan merupakan suatu proses random yang dinyatakan dalam probabilitas. Matriks probabilitas adalah matriks bujur sangkar dengan elemen riil tak negatif pada selang [0, 1], dan jumlah tiap barisnya = 1(Siang, Jong Jek: 2009). Untuk lebih jelasnya dapat dilihat pada matriks probabilitas transisi berikut ini :

$$P = \begin{Vmatrix} P_{00} & P_{01} & P_{02} & \dots & P_{0j} \\ P_{10} & P_{11} & P_{12} & \dots & P_{1j} \\ P_{20} & P_{21} & P_{22} & \dots & P_{2j} \\ \dots & \dots & \dots & \dots & \dots \\ P_{i0} & P_{i1} & P_{i2} & \dots & P_{ij} \end{Vmatrix}$$

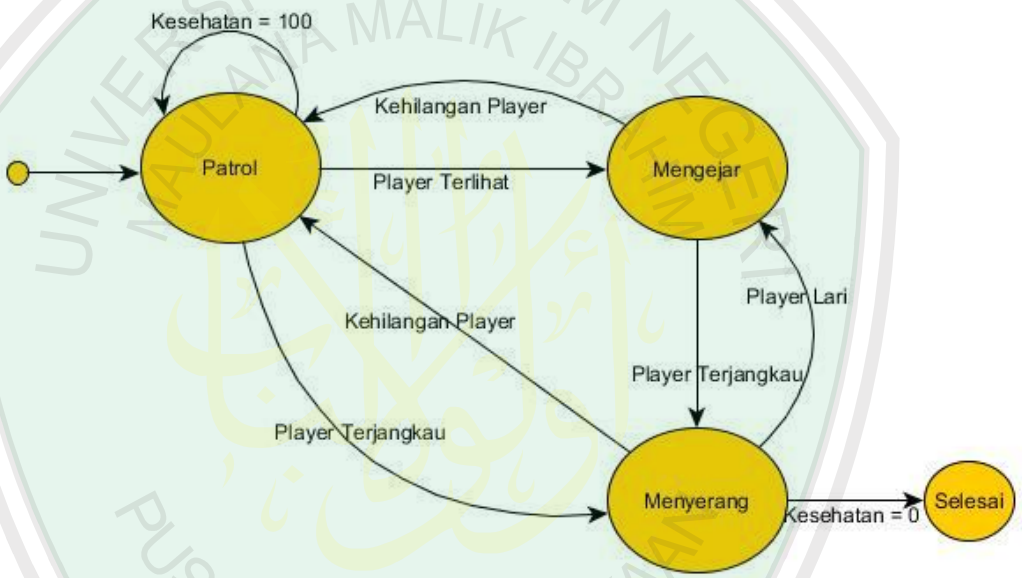
P_{ij} merupakan probabilitas bersyarat dimana nilai state mengalami transisi dari i ke j dalam satu bidang. Oleh karena angka tersebut melambangkan kemungkinan, maka semuanya merupakan bilangan non negatif dan tidak lebih dari satu. Secara matematis :

$$0 < P_{ij} < 1 \qquad i, j = 0,1,2,\dots$$

$$\sum_{j=0}^{\infty} Pij = 1 \qquad i = 0,1,2,\ldots,$$

2.7 *Markov State Machine*

Pada bagian berikut ini akan dijelaskan tentang perubahan proses dari *Finite State Machine* menjadi *Markov State Machine.* Skenario dasar permainan adalah menggunakan *Finite State Machine,* dengan proses sebagai berikut:

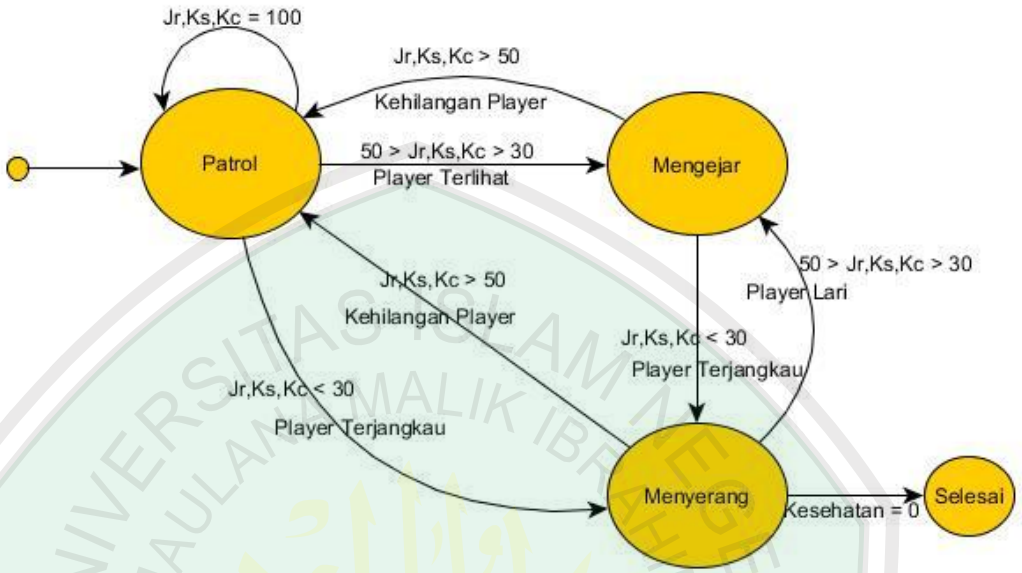


Gambar 2.2 *Finite State Machine* NPC

(Sumber: Matahari Bhakti, dkk. 2011)

Proses pada gambar 2.2 di atas merupakan skenario permainan menggunakan *Finite State Machine.* Dalam proses tersebut masih merupakan skenario umum dengan keadaan yang belum dijelaskan. Untuk menjelaskan atau menjabarkan keadaan yang merupakan masukan maka perlu dimasukkan algoritma *Markov Chain* ke dalam proses. Perubahan proses inilah yang akan

memunculkan proses baru yaitu *Markov State Machine.* Berikut adalah proses *Markov State Machine*:



Gambar 2.3 *Markov State Machine* NPC

Dalam proses *Markov State Machine* keadaan *player* didefinisikan dengan tiga masukan yaitu:

Jr = Jarak

Ks = Kesehatan

Kc = Kecepatan

Aksi dari NPC akan terjadi sesuai dengan keadaan player yang telah didefinisikan di atas. Sehinggan transisi *state* atau kejadian dari proses ini tergantung dari *event* atau keadaan player pada *state* saat ini.

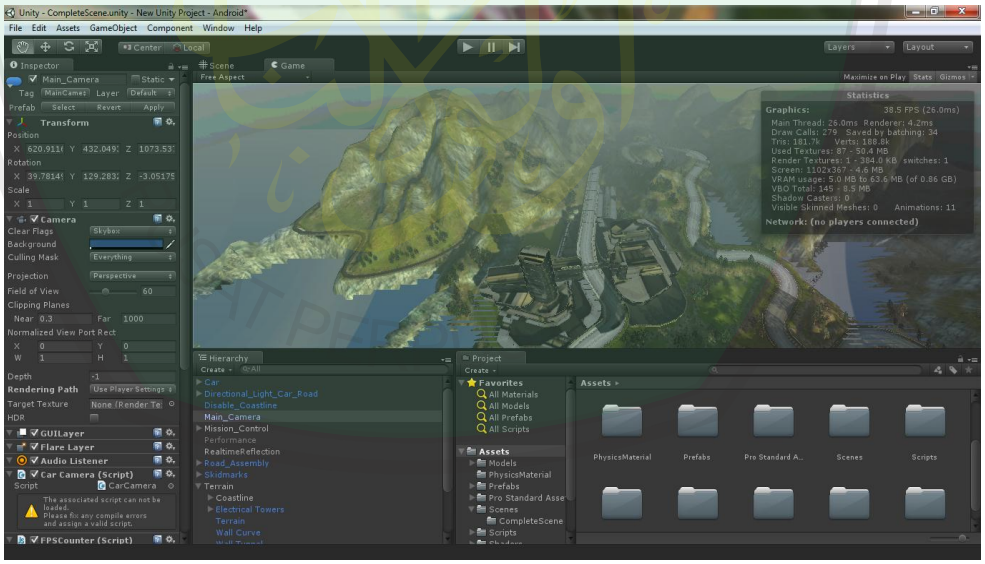
2.8 *Game Engine*

Banyaknya *game* baru yang bermunculan banyak di pengaruhi beberapa faktor salah satunya adalah kemajuan teknologi di bidang *game*,

membuat *game* saat ini saat ini berbeda dengan pada zaman dahulu ,kemudahan dalam membuat *game* pada saat ini di dukung oleh bermunculan *engine* untuk pembuatan *game*, selain dirasa mudah dan juga mempermudah untuk menganalisis kesalahan yang terjadi pada *game* tersebut.

Game engine yang digunakan mendukung tiga bahasa dengan *framework Mono open source*, C#, JavaScript dan Phyton. Pada *Game engine* tersebut terdapat fasilitas yang memudahkan untuk penggolongan seperti, *asset*, animasi, tekstur, suara dan juga memungkinkan untuk meng-*import* model 3D dari aplikasi modeller lain, seperti Blender, untuk membuat *real-time* grafis menggunakan mesin *rendering* buatan sendiri dan juga dikombinasikan dengan nVidia PhysX *physics engine.*

2.7.1 Antar muka dan kontrol *game engine*



Gambar 2.4 Antar muka keseluruhan *game engine*

(Sumber: Clifford Peters. dkk, 2013)

2.7.1.1 *Inspector*

Berisi tentang semua detail objek yang di sorot pada scene,di dalamnya kita bisa mengatur berbagai hal , seperti ukuran

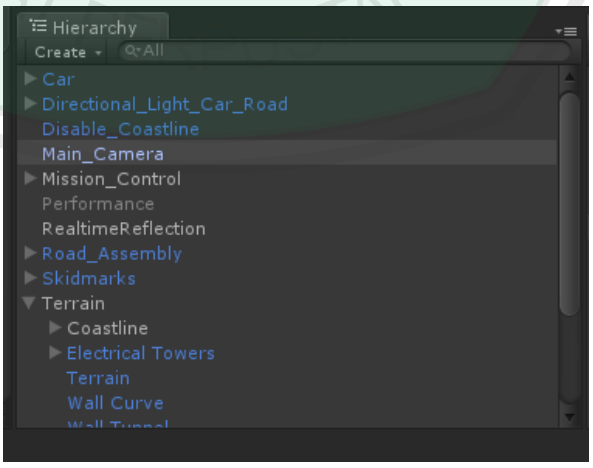
objek, letak koordinat objek, model kamera Dll. *Control* objek bisa menggunakan C# yang mana dapat mengendalikan dari masukan yang telah di berikan.

Gambar 2.5 Tampilan *inspector* pada *game engine*

(Sumber: Clifford Peters. dkk, 2013)

2.7.1.2 *Hierarchy*

Hierarchy berisi tentang semua yang ada pada *game*, dan juga menggabungkan antara objek satu dengan yang lainnya, seperti halnya menggambarkan turunan dari suatu objek tertentu



Gambar 2.6 Tampilan *hierarchy* pada *game engine*

(Sumber: Clifford Peters. dkk, 2013)

2.7.1.3 *Project*

Project menampilkan semua *file* yang sudah kita buat pada proyek, pada gambar 4 dapat dilihat pada folder *asset* terdapat *model, texture, scene, script* dan lain lain.

Gambar 2.7 Tampilan *project* pada *game engine*

(Sumber: Clifford Peters. dkk, 2013)

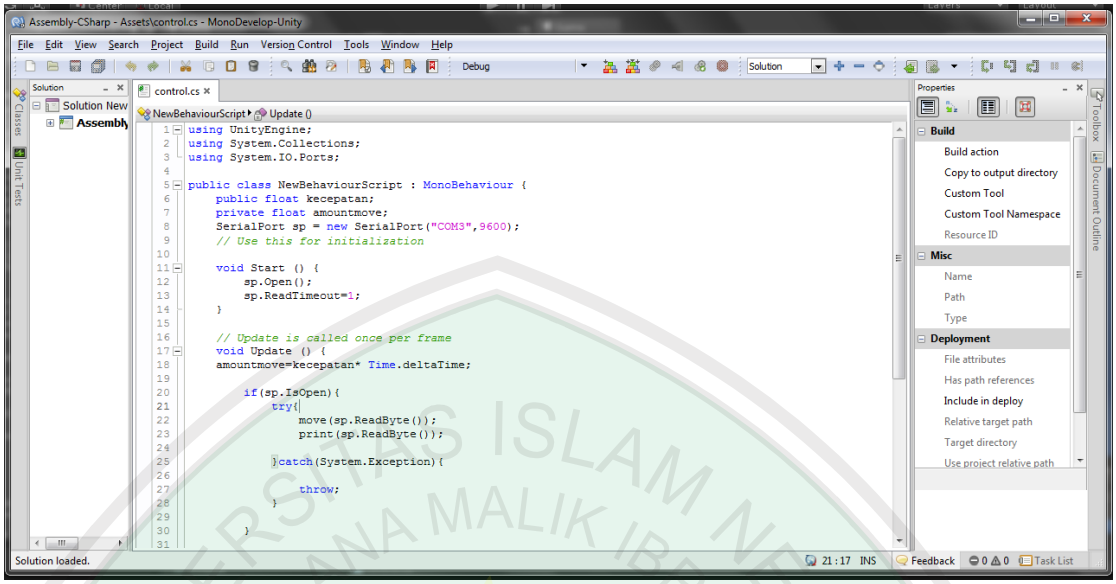
2.7.1.4 *Scene*

Scene menampilkan lembar kerja untuk memanipulasi objek yang ada pada tampilan, meliputi melakukan penggeseran objek, rotasi objek edit ukuran , perspektif objek.

2.7.1.5 *Game*

Untuk menampilkan objek hasil dari manipulasi scene secara *realtime.*

2.7.2 Antar muka *Mono Develop*



Gambar 2.8 Antarmuka *mono develop* untuk kode program

(Sumber: Clifford Peters. dkk, 2013)

BAB III

ANALISIS DAN PERANCANGAN GAME

3.1 Perancangan Sistem

Game yang dibangun adalah *game single player* yang berjenis *simulation game* atau *game* simulasi. Dalam permainan ini terdapat sebuah karakter sebagai pemain utama yang mengendarai sepeda dan terdapat karakter musuh berupa *Non Playable Character* (NPC) yang cerdas yang dimainkan oleh komputer. Objek penelitian dalam permainan ini adalah desain perilaku NPC yang dikendalikan komputer.

3.1.1 Keterangan Umum *Game*

Game gowes ini merupakan sebuah *game* simulasi yang menggambarkan seseorang mengendarai sepeda mengelilingi lingkungan hutan. Dalam *game* ini pemain diharuskan berkeliling mengendarai sepeda untuk mencari *obstacle* yang mempunyai poin. Setiap *obstacle* memiliki poin 10 yang akan diakumulasikan di akhir permainan.

Pada sisi *game* yang akan di dibangun, dengan background lingkungan hutan atau pedesaan dengan jalan yang masih berupa tanah, dan juga dengan tambahan misi *game* untuk mengkoleksi *point*, dan misi tertentu,dengan model di buat mendekati dengan aslinya. Selain itu *game* ini akan dibangun dengan menambahkan NPC sebagai musuh yang bertugas untuk mengganggu pemain dalam mencapai misi. NPC dalam *game* akan ditempakan di beberapa posisi yang ditentukan.

3.1.2 Deskripsi NPC

Dalam *game* ini NPC yang dibuat berupa karakter orang yang berpakaian kuli dan membawa kunci pas. NPC ini akan melakukan *patrol* pada saat *game* dimulai dan akan berubah megejar ketika didekati *player*. NPC juga bisa menyerang apabila kondisi *player* sangat dekat dan tidak berhasil kabur. NPC mempunyai 3 bentuk animasi yaitu pada saat patrol, menyerang dan mengejar.

1. **Rancangan NPC patrol**

   NPC yang sedang patrol dimodelkan seperti seorang pekerja yang sedang berjalan bolak-balik ke arah yang sudah ditentukan. Model NPC yang sedang patrol dapat dilihat pada gambar 3.1 berikut ini:

Gambar 3.1 Model NPC sedang patrol

2. **Rancangan NPC mengejar**

   NPC yang sedang mengejar *player* dimodelkan sebagai seorang pekerja yang sedang berlari menuju *player*. NPC berlari mendekati *player* tetapi tidak sampai menyentuhnya. Model NPC yang sedang mengejar *player* dapat dilihat pada gambar 3.2 berikut ini:

Gambar 3.2 Model NPC mengejar *player*

3. **Rancangan NPC menyerang**

NPC yang sedang menyerang dimodelkan seperti seorang pekerja yang berlari menuju *player*. Animasinya sama seperti pada saat mengejar, akan tetapi pada saat menyerang NPC memiliki kecepatan yang lebih tinggi dan mendekati *player* sampai menyentuhnya. Model NPC pada saat menyerang dapat dilihat pada gambar 3.3 berikut ini:

Gambar 3.3 Model NPC menyerang *player*

3.1.3 Algoritma *Game*

3.1.3.1 Skenario Perilaku Menyerang untuk NPC

Dalam penelitian ini, dibuat skenario pada *game* untuk dijadikan simulasi atau uji coba. Karakter dalam *game* dibedakan menjadi dua yaitu pemain(*player*) dan NPC sebagai musuh yang menjadi objek penelitian. Dalam menentukan perilaku NPC ketika bertemu *player* maka dibuatlah tabel yang berisi variabel *input* dan variabel *output* sebagai acuan untuk aksi NPC terhadap *player*. Berikut adalah tabel perilaku NPC:

Tabel 3.1 Perilaku NPC

NPC	Variabel *Input* Perilaku	Variabel *Output* Perilaku
NPC	Jarak terhadap *player*, kesehatan, kecepatan	Patrol, mengejar *player*, menyerang *player*

3.1.3.2 Rancang FSM Perilaku NPC

State utama yang tersusun dalam FSM dapat digambarkan sebagai berikut :

1. *Spawn / start*

   Merupakan *state* posisi awal NPC berada

2. *Walk / patrol state* berjalan / patroli

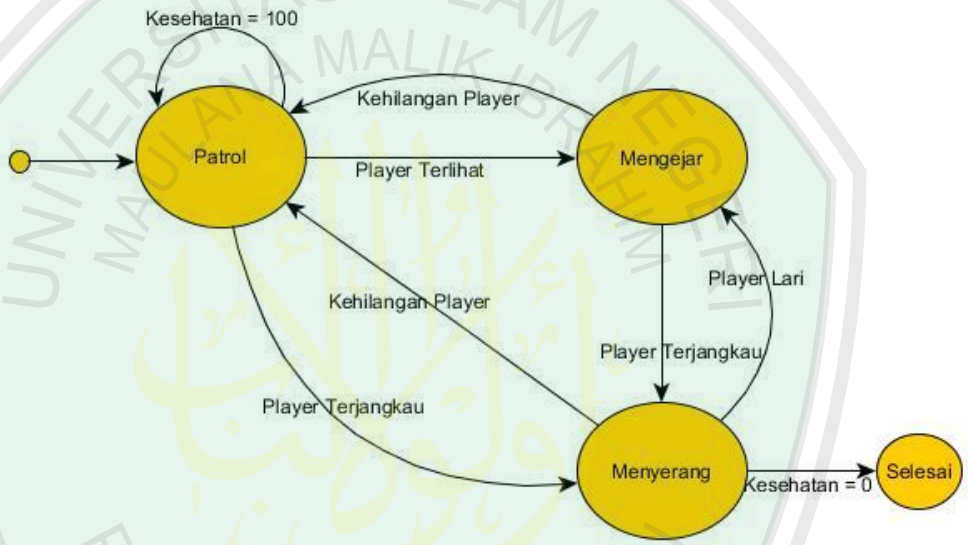
   NPC bergerak menuju *state* yang sudah ditentukan sebelumnya

3. Menyerang

   State NPC terlibat pertempuran dipicu jangkauan NPC terhadapa *player*. Terdapat *state* menyerang dan mengejar dalam *state* ini.

4. Mati / *Game* Over

   Nilai kesehatan *player* = 0



Gambar 3.4 Rancang FSM perilaku NPC

(Sumber: Matahari Bhakti, dkk. 2011)

3.1.3.3 *Markov State Machine*

Pada tahap ini merupakan tahap yang akan memasukkan konsep metode *Markov Chain* ke dalam *Finite State Machine* yang telah dirancang di atas. Proses ini nantinya akan memunculkan model perhitungan yang disebut *Markov State Machine*. Penerapan *Markov State Machine* dalam permainan ini ditujukan pada

perilaku NPC, sehingga NPC mempunyai beberapa variabel sebagai parameter dalam mengeksekusi perilakunya.

Variabel yang digunakan dibagi menjadi dua yaitu variabel *input* dan variabel *output*. Variabel input meliputi kondisi *player* saat mendekati NPC berupa jarak, kesehatan dan kecepatan. Sedangkan variabel *output*nya adalah aksi yang akan dilakukan oleh NPC terhadap *player* berupa patrol, mengejar dan menyerang. Dengan adanya variabel tersebut maka akan mempermudah proses perhitungan untuk menentukan perilaku NPC, kapan harus patrol, mengejar dan menyerang. Nilai dari variabel tersebut ditentukan berupa parameter batas atas dan bawah, sehingga dapat dituliskan sebagai berikut:

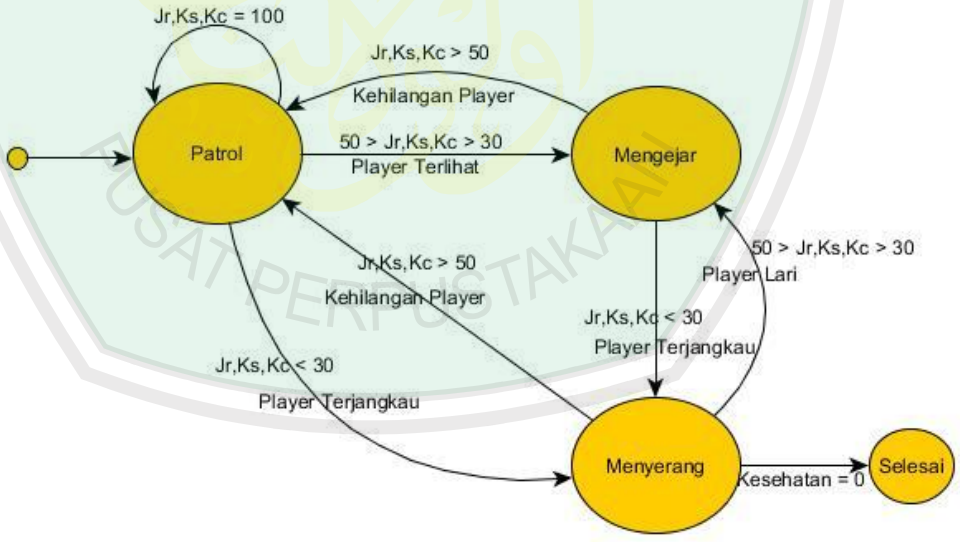
Tabel 3.2 Batas minimal dan batas maksimal nilai parameter variabel *output*

Parameter Perilaku NPC		
Variabel	**Notasi**	**Nilai**
P	Patrol	50 ─ 100
K	Kejar	30 ─ 50
S	Serang	0 ─ 30

Tabel tersebut di atas merupakan parameter batas maksimal dan batas minimal nilai aksi NPC. Variabel patrol mendapat range nilai yang paling besar dikarenakan patrol adalah kondisi dimana

NPC tidak melakukan aksi apapun terhadap *player*. Variabel kejar berada pada *range* nilai antara patrol dan serang dikarenakan NPC akan melakukan aksi kejar jika kondisi *player* berada dalam jangkauan NPC tetapi tidak dapat ditangkap. Variabel serang memiliki *range* nilai terkecil dikarenakan NPC akan menyerang *player* pada saat kondisi *player* berada dalam jangkauan kejar dan jangkauan tangkap/serang NPC.

Sehingga dengan melihat Tabel 3.2, perhitungan *Markov Chain* dapat dimasukkan ke dalam *Finite State Machine*. Rancangan FSM gerak NPC yang telah dijelaskan dalam Gambar 3.4 digabungkan dengan variabel *Markov Chain* menjadi sebuah algoritma *Markov State Machine*, seperti pada gambar 3.5 berikut:



Gambar 3.5 Gerak NPC menggunakan *Markov State Machine*

(Sumber: Matahari Bhakti, dkk. 2011)

Dengan menggunakan algoritma *Markov State Machine* maka eksekusi setiap *state* akan dipengaruhi oleh 3 variabel *input* yaitu jarak(Jr), kesehatan(Ks) dan kecepatan(Kc). Ketiga variabel *input* itulah yang nantinya akan menjadi parameter untuk memutuskan perilaku NPC terhadap *player* berupa patrol, mengejar atau menyerang.

Algoritma *Markov State Machine* diimplementasikan ke dalam NPC yang bertujuan untuk menentukan perilakunya terhadap *player*. Karakter *player* memiliki atribut berupa jarak, kesehatan dan kecepatan. Atribut itulah yang nantinya akan menjadi nilai *input* untuk menentukan perilaku NPC terhadap *player*. Perilaku NPC akan tergantung pada kondisi *player* saat bertemu NPC. Jika *player* berada pada jarak yang jauh dari NPC, kondisi kesehatan *player* masih seratus persen dan *player* berkendara dalam kecepatan tinggi, maka NPC akan melakukan patrol, tidak mengejar atau menyerang.

Kondisi *player* setiap waktu bisa berubah sesuai dengan perjalanan permainan, sehingga nialai *input* akan berubah setiap waktu dan perilaku NPC terhadap *player* juga akan berubah pula. Dikarenakan nilai acuan aksi NPC terhadap *player* masih berupa *range* nilai maka perlu dibuat tabel berupa nilai yang mutlak sebagai parameter perilaku NPC terhadap *player*. Sehingga dapat dibuat beberapa tabel kemungkinan sebagai berikut:

1. Tabel nilai variabel parameter yang pertama

Tabel 3.3 Nilai variabel parameter pertama

Variabel *Input*	Variabel *Output*			**Jumlah**
	Patrol	Kejar	Serang	
Jarak	60	30	10	**100**
Kesehatan	50	30	20	**100**
Kecepatan	55	25	20	**100**

Jumlah pada ketiga baris adalah seratus, tetapi jumlah kolom tidak. Informasi ini digunakan untuk membuat matriks probabilitas transisi. Nilai dari tabel tersebut di atas adalah parameter yang menjelaskan bahwa pada jarak, kesehatan dan kecepatan berapa NPC akan melakukan aksi patrol, kejar dan serang. Karena syarat nilai untuk perhitungan rantai markov adalah harus berjumlah satu. Dengan demikian matriks kemungkinan transisinya adalah:

Tabel 3.4 Matriks Probabilitas Transisi parameter pertama

Variabel *Input*	Variabel *Output*		
	Patrol	Kejar	Serang
Jarak	60/100 = 0.6	30/100 = 0.3	10/100 = 0.1
Kesehatan	50/100 = 0.5	30/100 = 0.3	20/100 = 0.2
Kecepatan	55/100 = 0.55	25/100 = 0.25	20/100 = 0.2

Tabel 3.5 Matriks Probabilitas Transisi parameter pertama

Variabel *Input*	Variabel *Output*		
	Patrol	Kejar	Serang
Jarak	0.6	0.3	0.1
Kesehatan	0.5	0.3	0.2
Kecepatan	0.55	0.25	0.2

Dari 3 variabel input yaitu jarak, kesehatan dan kecepatan akan dibagi lagi menjadi 3 tingkatan pada setiap variabel yaitu:

1. Jarak = - Jauh
   - Sedang
   - Dekat
2. Kesehatan = - Kuat
   - Sedang
   - Lemah
3. Kecepatan = - Cepat
   - Sedang
   - Lambat

Merujuk pada tabel probabilitas transisi maka *value* dari tabel diatas merupakan nilai maksimal pada masing- masing sub variabel. Nilai terbesar merupakan parameter kondisi *player* saat ini mempunyai jarak yang jauh, kesehatan yang masih prima dan kecepatan tinggi, sehingga aksi dari NPC adalah patrol. Sedangkan nilai terkecil adalah kondisi dimana *player* saat ini dekat dengan

musuh, lemah dan berjalan lambat, maka aksi dari NPC adalah serang.

Value atau nilai dari tabel tersebut diatas merupakan matriks probabilitas transisi, sehingga dapat dituliskan sebagai berikut:

$$\text{Matriks Probabilitas Transisi} = \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.5 & 0.3 & 0.2 \\ 0.55 & 0.25 & 0.2 \end{bmatrix}$$

2. Tabel nilai variabel parameter kedua

Tabel 3.6 Nilai variabel parameter kedua

Variabel *Input*	Variabel *Output*			**Jumlah**
	Patrol	Kejar	Serang	
Jarak	50	40	10	**100**
Kesehatan	60	40	20	**120**
Kecepatan	70	30	30	**130**

Dari tabel parameter di atas dijadikan bilangan riil tak negatif sebagai syarat matriks probabilitas transisi menjadi:

Tabel 3.7 Matriks Probabilitas Transisi parameter kedua

Variabel *Input*	Variabel *Output*		
	Patrol	Kejar	Serang
Jarak	50/100 = 0.5	40/100 = 0.4	10/100 = 0.1
Kesehatan	60/120 = 0.5	40/120 = 0.333	20/120 = 0.167
Kecepatan	70/130 = 0.54	30/130 = 0.23	30/130 = 0.23

Berdasarkan tabel tersebut di atas maka didapatkan matriks probabilitas transisinya sebagai berikut:

$$\text{Matriks Probabilitas Transisi} = \begin{bmatrix} 0.5 & 0.4 & 0.1 \\ 0.5 & 0.333 & 0.167 \\ 0.54 & 0.23 & 0.23 \end{bmatrix}$$

Dalam prakteknya matriks probabilitas transisi digunakan dalam rumus untuk menghitung kemungkinan yang terjadi di masa mendatang ketika *player* dalam kondisi yang saat ini. Sehingga dalam menghitung kemungkinan aksi NPC dapat dirumuskan sebagai berikut:

[Jr Ks Kc] = [Jr Ks Kc] x Matriks Probabilitas Transisi

Keterangan:

Jr = Jarak

Ks = Kesehatan

Kc = Kecepatan

Sebagai pembuktian dari metode Markov *State Machine* ini, maka terdapat beberapa contoh dengan kondisi *player* yang berbeda-beda, berikut adalah contohnya:

1. **Menggunakan tabel parameter yang pertama(tabel 3.3)**

   Diketahui *player* dengan kondisi sebagai berikut:

   Jarak = 70, Kesehatan = 50 dan Kecepatan = 30

   Maka akan didapatkan nilai variabelnya adalah

   Jr = $\frac{70}{150}$ = 0.467

Ks = $\frac{50}{150}$ = 0.333

Kc = $\frac{30}{150}$ = 0.2

[Jr Ks Kc] = [Jr Ks Kc] x Matriks Probabilitas Transisi

$$[\text{Jr Ks Kc}] = [0.467 \quad 0.333 \quad 0.2] \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.5 & 0.3 & 0.2 \\ 0.55 & 0.25 & 0.2 \end{bmatrix}$$

$$= [0.5567 \quad 0.29 \quad 0.1533]$$

Dikarenakan perhitungan di atas untuk menentukan kemungkinan perilaku NPC pada saat *player* dalam kondisi tersebut, maka hasil dari perhitungan di atas akan dikalikan dengan jumlah keseluruhan dari nilai kondisi awal *player* yaitu 150, jadi kemungkinan kondisi *player* berikutnya dituliskan sebagai berikut:

Jr = 0.5567 x 150 = 83.5

Ks = 0.29 x 150 = 43.5

Kc = 0.1533 x 150 = 23

Dan dengan merujuk pada tabel parameter nilai variabel maka penentuan perilaku NPC terhadap *player* dapat dilakukan dengan menghitung nilai terdekat dari parameter. Dengan cara mengurangkan parameter setiap *input* terhadap nilai hasil perhitungan di atas, sehingga didapatkan selisih terkecil. Selisih terkecil itulah yang merupakan nilai terdekat dari parameter. Perhitungannya dapat dilihat pada tabel berikut:

Tabel 3.8 Perhitungan hasil untuk mengetahui nilai selisih

Variabel *Input*	Variabel *Output*		
	Patrol	Kejar	Serang
Jarak	60 – 83.5 = -23.5	30 – 83.5 = -53.5	10 – 83.5 = -73.5
Kesehatan	50 – 43.5 = 6.5	30 – 43.5 = -13.5	20 – 43.5 = -23.5
Kecepatan	55 – 23 = 32	25 – 23 = 2	20 – 23 = -3

Tabel 3.9 Hasil perhitungan nilai terdekat terhadap parameter

Variabel *Input*	Variabel *Output*		
	Patrol	Kejar	Serang
Jarak	-23.5	-53.5	-73.5
Kesehatan	6.5	-13.5	-23.5
Kecepatan	32	2	-3

Pemutusan perilaku NPC terhadap *player* dilakukan dengan melihat kondisi *player* yang memiliki nilai paling dekat terhadap parameter pada setiap nilai *input*. Dalam hal ini dapat dituliskan nilai terdekat pada masing- masing *input* adalah sebagai berikut:

Jarak mempunyai nilai selisih terkecil yaitu -23.5 dan berada pada posisi patrol.

Kesehatan mempunyai nilai selisih terkecil yaitu 6.5 dan berada pada posisi patrol.

Kecepatan mempunyai nilai selisih terkecil yaitu 2 dan berada pada posisi mengejar.

Sehingga NPC akan patrol dikarenakan kondisi jarak dan kesehatan player berada dalam posisi patrol.

2. Menggunakan tabel parameter kedua(tabel 3.6)

Dengan studi kasus yang sama sebagai berikut:

Diketahui *player* dengan kondisi sebagai berikut:

Jarak = 70, Kesehatan = 50 dan Kecepatan = 30

Maka akan didapatkan nilai variabelnya adalah

Jr = $\frac{70}{150}$ = 0.467

Ks = $\frac{50}{150}$ = 0.333

Kc = $\frac{30}{150}$ = 0.2

[Jr Ks Kc] = [Jr Ks Kc] x Matriks Probabilitas Transisi

$$[\text{Jr Ks Kc}] = [0.467 \quad 0.333 \quad 0.2] \begin{bmatrix} 0.5 & 0.4 & 0.1 \\ 0.5 & 0.333 & 0.167 \\ 0.54 & 0.23 & 0.23 \end{bmatrix}$$

$$= [0.508 \quad 0.344 \quad 0.148]$$

Dikarenakan perhitungan di atas untuk menentukan kemungkinan perilaku NPC pada saat *player* dalam kondisi tersebut, maka hasil dari perhitungan di atas akan dikalikan dengan jumlah keseluruhan dari nilai kondisi awal *player* yaitu 150, jadi kemungkinan kondisi *player* berikutnya dituliskan sebagai berikut:

Jr = 0.508 x 150 = 76.2

Ks = 0.344 x 150 = 51.6

Kc = 0.148 x 150 = 22.2

Kemudian dengan melihat tabel parameter kedua maka akan didapat kesimpulan sebagai berikut:

Tabel 3.10 Perhitungan hasil untuk mengetahui nilai selisih

Variabel *Input*	Variabel *Output*		
	Patrol	Kejar	Serang
Jarak	50 – 76.2 = -26.2	40 – 76.2 = -36.2	10 – 76.2 = -66.2
Kesehatan	60 – 51.6 = 8.4	40 – 51.6 = -11.6	20 – 51.6 = -31.6
Kecepatan	70 – 22.2 = 47.8	30 – 22.2 = 7.8	30 – 22.2 = 7.8

Tabel 3.11 Hasil perhitungan nilai terdekat terhadap parameter

Variabel *Input*	Variabel *Output*		
	Patrol	Kejar	Serang
Jarak	-26.2	-36.2	-66.2
Kesehatan	8.4	-11.6	-31.6
Kecepatan	47.8	7.8	7.8

Pemutusan perilaku NPC terhadap *player* dilakukan dengan melihat kondisi *player* yang memiliki nilai paling dekat terhadap parameter pada setiap nilai *input*. Dalam hal ini dapat dituliskan nilai terdekat pada masing- masing *input* adalah sebagai berikut:

Jarak mempunyai nilai selisih terkecil yaitu -26.2 dan berada pada posisi patrol.

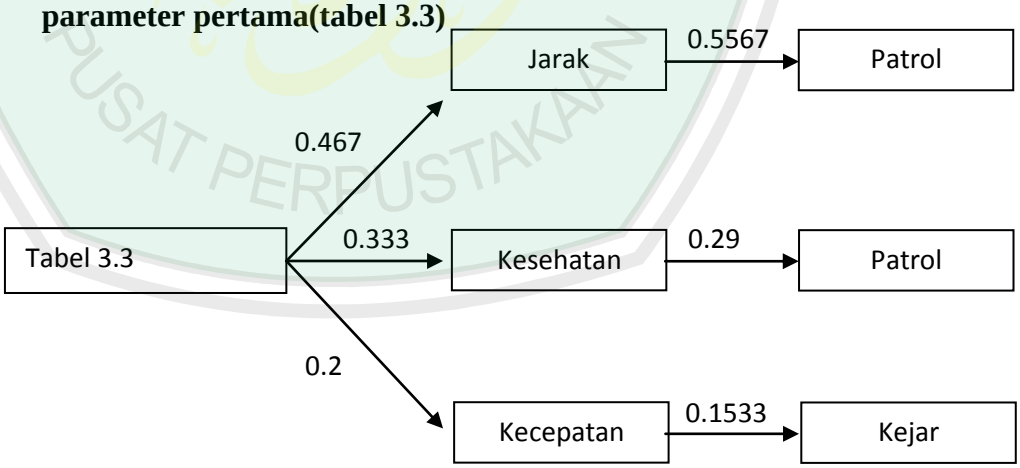
Kesehatan mempunyai nilai selisih terkecil yaitu 8.4 dan berada pada posisi patrol.

Kecepatan mempunyai nilai selisih terkecil yaitu 7.8 dan berada pada posisi mengejar dan menyerang.

Sehingga kesimpulannya NPC akan patrol dikarenakan kondisi jarak dan kesehatan player berada dalam posisi patrol.

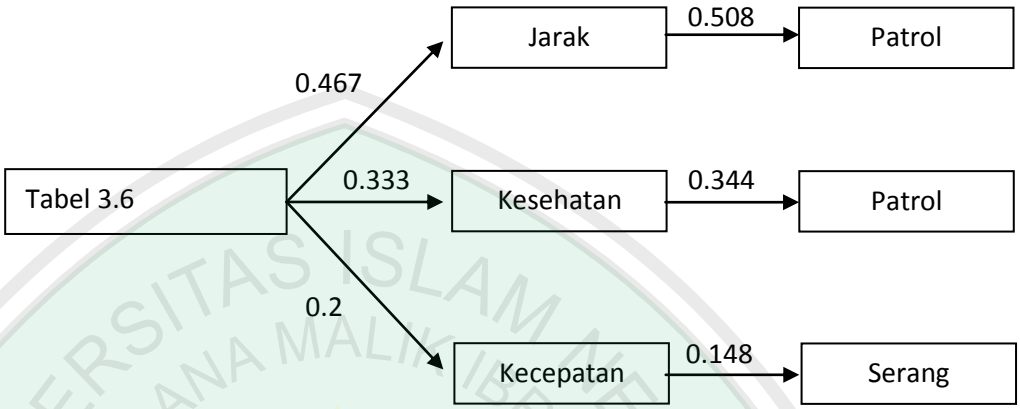
Dari dua contoh kasus dengan tabel parameter yang berbeda di atas maka dapat disimpulkan melalui bagan perbandingan kemungkinan perilaku NPC terhadap *player* di masa yang akan datang sebagai berikut:

1. **Probabilitas perilaku NPC terhadap *player* menggunakan tabel parameter pertama(tabel 3.3)**



Gambar 3.6 probabilitas perilaku NPC berdasarkan tabel pertama

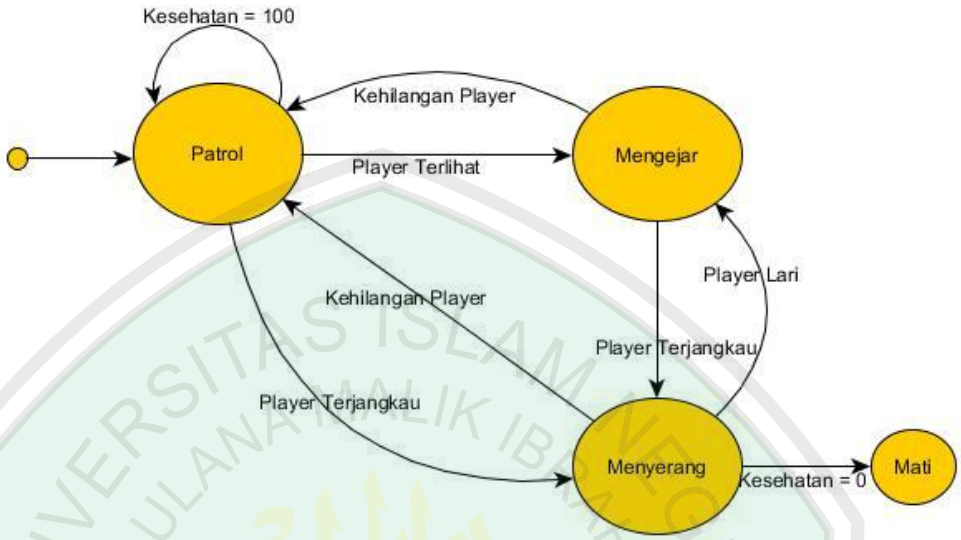
2. **Probabilitas perilaku NPC terhadap *player* menggunakan tabel parameter kedua(tabel 3.6)**



Gambar 3.7 probabilitas perilaku NPC berdasarkan tabel kedua

Dari dua perhitungan menggunakan tabel parameter pertama dan kedua, hasil akhir atau kesimpulan perilaku *player* yang akan terjadi kurang lebih sama. Hal tersebut bisa terjadi dikarenakan sebelumnya sudah dibuat fungsi keanggotaan yang berisi batas minimal dan maksimal nilai dari masing- masing variabel output.

3.1.3.4 Rancangan FSM Perilaku NPC



Gambar 3.8 Rancangan FSM Perilaku NPC

BAB IV

HASIL DAN PEMBAHASAN

Bab ini membahas tentang implementasi dari perancangan yang dibuat sebelumnya. Selain itu juga melakukan pengujian terhadap aplikasi yang dibuat untuk mengetahui apakah aplikasi tersebut telah berjalan sesuai dengan tujuan yang ingin dicapai.

4.1 *Game* Dalam Perspektif Islam

Dalam Al Qur'an memang tidak dijelaskan secara spesifik tentang informatika apalagi tentang *game*. Al Qur'an adalah mukjizat Nabi Muhammad SAW yang di dalamnya berisi tuntunan bagi umat muslim meliputi keimanan, aqidah dan alam semesta. Selain itu, Al Qur'an juga merupakan petunjuk bagi umat manusia yang ada di bumi. Dikarenakan Al Qur'an adalah petunjuk, maka di dalamnya terdapat semua hal yang ada di bumi ini. Mulai dari makanan, minuman, tata surya dan lain sebagainya. Termasuk di dalam Al Qur'an dijelaskan tentang keutamaan ilmu, mulai dari ilmu agama sampai ilmu pengetahuan.

Salah satu cabang ilmu yang diterangkan dalam Al Qur'an adalah ilmu pengetahuan, yang menjelaskan tentang bagaimana bumi mengitari matahari dan bagaimana bulan mengitari bumi. Termasuk dalam ilmu pengetahuan adalah ilmu tentang komputer. Walaupun di dalam Al Qur'an tidak dijelaskan secara langsung, akan tetapi Al Qur'an menjelaskan bagaimana kita umat manusia diperintahkan untuk menggali ilmu yang ada di

langit dan di bumi termasuk ilmu tentang pemrograman. Seperti yang telah dijelaskan dalam Surah Yunus ayat 101 berikut:

قُلِ ٱنظُرُواْ مَاذَا فِي ٱلسَّمَٰوَٰتِ وَٱلۡأَرۡضِۚ وَمَا تُغۡنِي ٱلۡأٓيَٰتُ وَٱلنُّذُرُ عَن قَوۡمٖ

لَّا يُؤۡمِنُونَ ١٠١

Artinya: "Katakanlah: Perhatikanlah apa yang ada di langit dan di bumi. tidaklah bermanfaat tanda kekuasaan Allah dan Rasul-rasul yang memberi peringatan bagi orang-orang yang tidak beriman".

Tafsir Al Mishbah Oleh Prof. Quraish Shihab:

Ayat ini mendorong umat manusia untuk mengembangkan ilmu pengetahuan melalui kontemplasi, eksperimentasi dan pengamatan. Ayat ini juga mengajak untuk menggali pengetahuan yang berhubungan dengan alam raya beserta isinya. Sebab, alam raya yang diciptakan untuk kepentingan manusia ini, hanya dapat dieksplorasi melalui pengamatan indrawi.

قل (*qul*) berasal dari akar kata (قل يقل قولا قل), kata *qul* adalah kata perintah (*fi'il amar)* yang secara harfiyah dipahami "katakanlah". Kata *qul* secara tekstual diperintahkan kepada Nabi Muhammad saw, tetapi dalam konteks ditujukan kepada seluruh manusia, dalam istilah bahasa arab disebut "*mukhathab ghair mu'ayan".*

أنظروا (*unzuru*) bentuk jama' dari *unzur* (أنظر) yang secara*harifah* bermakna lihat, perhatikan, renungkan. Kata *unzur* termasuk kata perintah (*fi'il amar*) yang besal dari akar kata (نظرينظرأنظر). Adapun kata *al-samawat* (السموات) bentuk jama' dari*al-samaa* (السماء) yang dalam kamus bahasa arab diartikan;

langit, awan, hujan. الارض(*ardu/i*) dipahami sebagai bumi, sesuatu yang di bawah.

Dengan memperhatikan kosakata di atas, dapat dipahami kita dianjurkan untuk membaca, merenungkan seluruh ayat Allah SWT yang tercipta yakni dengan memperhatikan atau meneliti apa yang ada di langit dan apa yang ada di bumi. Ayat tersebut di atas menjelaskan tentang manusia dianjurkan untuk memperhatikan alam sekitar (langit dan bumi). Dengan memperhatikan alam sekitar yang melahirkan berbagai disiplin ilmu. Dengan memperhatikan bintang melahirkan ilmu astronomi, memperhatikan angin melahirkan ilmu komunikasi, telepon dan lain sebagainya. Memperhatikan bumi menghasilkan ilmu geografi dan banyak ilmu- ilmu yang lainnya.

Perintah Allah SWT dalam ayat tersebut di atas sudah jelas bahwa manusia diperintah untuk menggali semua tanda yang ada di langit dan bumi untuk menghasilkan ilmu pengetahuan. Termasuk di dalamnya yaitu ilmu komputer yang menjelaskan tentang pemrograman. Memang tidak dijelaskan secara detail tentang *game*, tetapi perintah Allah SWT jelas bahwa kita harus mengembangkan ilmu itu untuk kebaikan.

4.2 Kebutuhan Perangkat Keras

Spesifikasi Kebutuhan perangkat Keras yang di gunakan dalam penelitian ini adalah:

1. Prosesor intel Core i3 3217U @1.8 GHz
2. HardDisk 500 GB - 5600 RPM
3. RAM 2GB

4. VGA IntelHD Graphics 4000 1 Gb Shared memory
5. Keyboard
6. Mouse Laser
7. Monitor 14’’

4.3 Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak yang di gunakan dalam penelitian ini adalah :

1. *Game Engine*
2. Monodevelop
3. Windows 7 Ultimate Edition x64

Spesifikasi *hardware* dan *software* pada pc ataupun laptop,sudah mumpuni dan sudah di coba secara langsung untuk menjalankan *game* simulasi, hal yang berpengaruh dan menjadi prioritas adalah pada spesifikasi *hardware*, yang harus setara atau lebih pada spesifikasi di atas , karena hal ini berpengaruh pada pengolahan grafis yang menggunakan animasi 3D.

4.4 Implementasi Antarmuka

Pada sub bab implementasi antarmuka ini akan dijelaskan komponen antarmuka *game* yang dilihat user.

4.4.1 Main Menu

Adalah menu utama sebelum melakukan permainan, terdapat beberapa pilihan diantaranya sebagai berikut:

Gambar 4.1 Menu Utama

1. Play

   Tombol untuk memulai *game*

2. Option

   Menu untuk beberapa pilihan yang telah disediakan, diantaranya :

   a. Control

      Tombol ini Berisi halaman Kontrol pada *game* yang mana *player* dapat mengetahui control yang ada pada *game*.

   b. Credit

      Tombol ini Berisi tentang hal hal seputar pembuat *game* ini

   c. Back

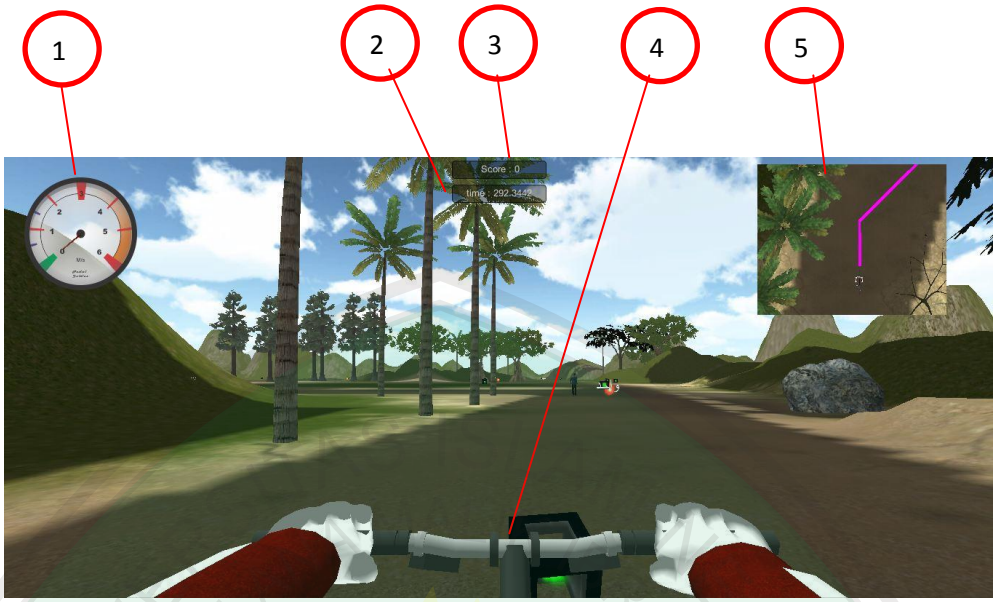
      Tombol ini akan membawa Kembali ke menu utama

3. Exit

   Tombol untuk keluar dari *game*

4.4.2 *Game*

Tampilan pada *game* ini menampilkan bebrapa item diantaranya:



Gambar 4.2 Tampilan permainan

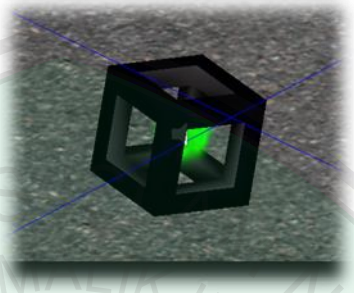
1. Spedo meter : Menampilkan kecepatan sepeda
2. Score : Menampilkan score yang di dapat
3. Timer : Menampilkan waktu yang tersisa menjalankan suatu misi
4. Point pickup : Berupa kubus yang apabila di koleksi / berbenturan pada sepeda , akan menambah score
5. Minimaps : Berisi map kecil , yang menampilkan jalan rute jalan.

4.5 Implementasi Sistem *Game*

Sistem *game* yang akan di implementasikan pada *game* sepeda ini menggunakan bahasa pemrograman C# dan javascript sebagai core dari *game*, dan menggunakan editor monodevelop. Sedangkan desain visual dan interface mengunakan *Game engine*.

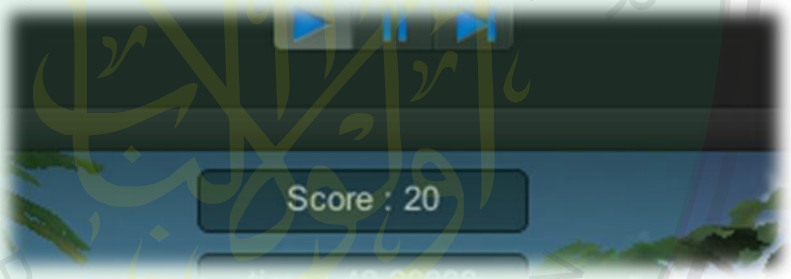
4.5.1 Pengaturan Score

Game pada umumnya terdapat scoring, begitu juga pada *game* ini , proses penambahan *score* ini dilakukan pada saat sepeda berbenturan dengan objek yang bernama pickup, berbentuk kubus (Gambar4.4).



Gambar 4.3 Pickup untuk point

Berikut adalah GUI dari *Score* yang ada pada *game* (Gambar 4.4)



Gambar 4.4 Gui *Score*

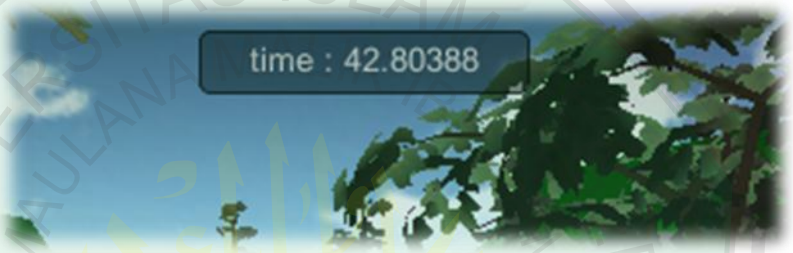
Untuk implementasinya mengunakan kelas *cbpickup*.cs. berikut adalah baris kodenya :

```
#pragma strict
function OnTriggerEnter (info : Collider){
      if (info.tag == "Sepeda"){
      updateScore.currentscore += 10;
      //yield  WaitForSeconds(5);
      Destroy(gameObject);
      }
}
```

Kelas di atas menangani proses *scoring* pada *game*, apabila objek yang menyentuh adalah sepeda maka poin bertambah.

4.5.2 Pengaturan *Timer*

Pada *game* ini suatu misi dibatasi oleh waktu, sehingga *player* bermain berdasarkan waktu yang di tentukan, waktu yang di sediakan di hitung mundur dan di tampilkan pada layar seperti (gambar 4.5)



Gambar 4.5 GUI *Timer*

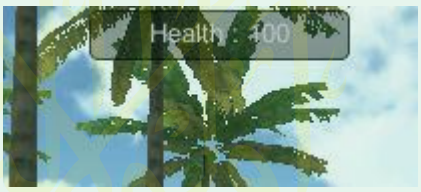
Berikut adalah implementasi kedalam kode C dalam kelas *coundowntimer.cs:*

```
   public float currentTime = 90;
   public float offsety     = 40;
   public float sizex       = 100;
   public float sizey       = 40;
   public string nextLevelToLoad   ;
   // Use this for initialization

void FixedUpdate ()
   {
   if (currentTime <= 0) {

AutoFade.LoadLevel(nextLevelToLoad,1,1,Color.white);
         }
         currentTime -= Time.deltaTime;
   }
   void OnGUI(){
      GUI.Box(new Rect(Screen.width/2-sizex/2,offsety,
sizex,sizey),"time : " +currentTime );
   }
}
```

Dari potongan kode di atas , sekilas kita mengetahui dapat mengatur waktu yang di tentukan melalui variabel *currentTime.* Apabila kita melebihi waktu yang di tentukan maka secara otomatis misi gagal dan permainan akan berakhir.

4.5.3 Pengaturan Kesehatan

Pada *game* ini *player* diberikan atribut kesehatan dalam menyelesaikan suatu misi. Kesehatan *player* akan berkurang saat berbenturan dengan NPC dan *player* akan mati saat kesehatannya bernilai 0. Berikut tampilan GUI kesehatan *player* pada layar permainan:



Gambar 4.6 GUI Kesehatan

Berikut adalah implementasi kedalam kode C dalam kelas *coundowntimer.cs:*

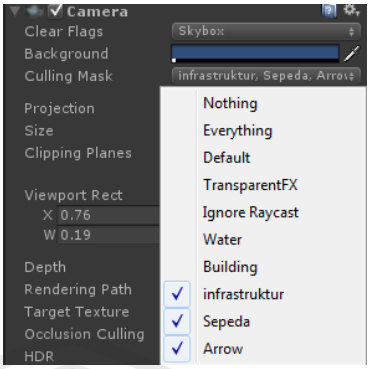
```
public float currentTime = 90;
public float offsety     = 40;
public float sizex        = 100;
public float sizey        = 40;
public string nextLevelToLoad   ;
// Use this for initialization

void OnGUI(){

        GUI.Box(new Rect(Screen.width/2-
sizex/2,offsety+30,sizex,sizey),"Health : "
+PlayerItems.Kesehatan );

   }
```

4.5.4 Mini Map

Mini map adalah Map kecil yang di tempatkan pada kanan atas layar yang berfunsi sebagai komponen pembatu navigasi , yang nantinya berisi garis garis rute yang akan di lalui *player*. Mini map ini akan mengikuti kemanapun arah pergerakan *player* dan juga memberikan arah setiap pergerakannya secara realtime. Berikut adalah tampilan minimap pada layar *game* (Gambar 4.7)

Gambar 4.7 Mini map

Pembuatan Mini map menggunakan *Second Camera,* yang di tempatkan pada atas *player* sehingga dapat terlihat lingkungan sekitar *player*, untuk pengaturan objek apa saja yang hanya bisa di lihat oleh mini map adalah layer infrastruktur, sepeda, dan arrow yang dapat dilihat pada gambar 4.8 berikut:



Gambar 4.8 Layer pada kamera

4.5.5 *Obstacle*/NPC

Obstacle adalah halangan atau rintangan, yang dalam *game* ini *obstacle* berbentuk sebuah NPC(Non Playable Character). NPC berwujud seorang manusia yang bertugas sebagai musuh atau pengganggu *player* yang sedang mengendarai sepeda. *Obstacle* atau NPC akan berada di titik- titik tertentu pada map, dengan bergerak patrol. NPC akan bereaksi apabila *player* mendekatinya pada jarak tertentu untuk mengganggu perjalanan *player*. Berikut adalah bentuk NPC yang ada dalam permainan:

Gambar 4.9 Tampilan NPC

4.6 Implementasi *Markov State Machine* Perilaku NPC

Proses Implementasi adalah proses pembangunan komponen-komponen pokok suatu sistem yang didasarkan pada desain dan rancangan yang telah dibuat sebelumnya. Implementasi perancangan *Artificial Intelligence* pada penelitian ini diterapkan pada pengaturan respon perilaku NPC dengan metode *Markov State Machine*.

4.6.1 NPC Patrol

Pada bagian ini membahas mengenai implementasi *Markov State Machine* untuk pengaturan perilaku NPC yaitu patrol. NPC akan patrol ketika kondisi *player* masih berada pada nilai tertinggi. Pada saat *player* berada pada jarak di luar jangkauan NPC dan masih memiliki kesehatan yang penuh. Berikut ini adalah *script* untuk perilaku patrol NPC dalam kelas *Orang.cs*:

```
public Transform[] posisi;
public float speed = 10f;
Transform _transform;
int curPosisi = 0;

public enum STATUS { KEJAR, PATROL,SERANG  };

public STATUS statusMusuh ;
int maxSol = 0;
int idxSol = 0;
   for (int i =0; i<3; i++) {
         if (hasil[i]!=0)  {
               if (hasil[i] > maxSol ){
                     maxSol =hasil[i];
                     idxSol = i;
               }
         }
   }

   if (idxSol == 0) {
         statusMusuh = STATUS.PATROL;
   }else if (idxSol == 1) {
         statusMusuh = STATUS.KEJAR;
   }else if (idxSol == 2) {
         statusMusuh = STATUS.SERANG;
```

```
   }
void Start(){
        PlayerItems.Kesehatan = 100;
        statusMusuh = STATUS.PATROL;
        hitungMatrix ();
        hitungPlayer (Mathf.CeilToInt( Jarak),
Mathf.CeilToInt(PlayerItems.Kesehatan),
Mathf.CeilToInt(PlayerItems.Kecepatan));
}
void Update(){

   PlayerItems.Kecepatan = (int.Parse
(textSpeed.guiText.text) + 1) * 10;

   hitungPlayer (Mathf.CeilToInt( Jarak),
Mathf.CeilToInt(PlayerItems.Kesehatan),
Mathf.CeilToInt(PlayerItems.Kecepatan));

   if (statusMusuh == STATUS.PATROL) {
        Vector3 targetPos = posisi [curPosisi].position;
              Quaternion targetRotation =
Quaternion.LookRotation (new Vector3 (targetPos.x,
_transform.position.y, targetPos.z) -
_transform.position);
        _transform.rotation = Quaternion.Slerp
(_transform.rotation, targetRotation, Time.deltaTime *
2f);
        _transform.Translate (Vector3.forward * speed *
Time.deltaTime);
        if ((targetPos -
_transform.position).sqrMagnitude < 700f) {
                  GantiPosisi ();
                  print ("ganti");
             }
   }
```

Script di atas mengatur NPC untuk patrol pada posisi yang sudah ditentukan. NPC akan berjalan dari posisi pertama ke posisi kedua dan kembali lagi ke posisi pertama begitu seterusnya. NPC baru akan mengejar *player* pada saat jarak *player* berada pada jangkauan NPC. Berikut merupakan bentuk NPC yang sedang patrol:

Gambar 4.10 Tampilan NPC sedang patrol

4.6.2 NPC Mengejar

Pada bagian ini membahas mengenai implementasi *Markov State Machine* untuk pengaturan perilaku NPC yaitu mengejar. Berdasarkan perhitungan parameter pada skenario permainan, NPC akan mengejar *player* apabila jarak *player* berdada pada jangkauan NPC. Sesuai dengan skenario tersebut maka dapat dibuat *script* untuk mengatur perilaku mengejar NPC. Berikut *script* yang dibuat dalam kelas *Orang.cs*:

```
public Transform[] posisi;
public float speed = 10f;
Transform _transform;
int curPosisi = 0;

public enum STATUS { KEJAR, PATROL,SERANG  };

public STATUS statusMusuh ;
int maxSol = 0;
int idxSol = 0;
   for (int i =0; i<3; i++) {
         if (hasil[i]!=0)  {
               if (hasil[i] > maxSol ){
                     maxSol =hasil[i];
                     idxSol = i;
```

```
                }
            }
    }

    if (idxSol == 0) {
            statusMusuh = STATUS.PATROL;
    }else if (idxSol == 1) {
            statusMusuh = STATUS.KEJAR;
    }else if (idxSol == 2) {
            statusMusuh = STATUS.SERANG;
    }

void Update(){
    PlayerItems.Kecepatan = (int.Parse
(textSpeed.guiText.text) + 1) * 10;

            hitungPlayer (Mathf.CeilToInt( Jarak),
Mathf.CeilToInt(PlayerItems.Kesehatan),
Mathf.CeilToInt(PlayerItems.Kecepatan));


    if (statusMusuh == STATUS.KEJAR) {
            Vector3 targetPos = Player.transform.position;
                Quaternion targetRotation =
Quaternion.LookRotation (new Vector3 (targetPos.x,
_transform.position.y, targetPos.z) -
_transform.position);
                _transform.rotation = Quaternion.Slerp
(_transform.rotation, targetRotation, Time.deltaTime *
2f);
                _transform.Translate (Vector3.forward *
speed * Time.deltaTime);
    }
```

Script tersebut di atas mengatur perilaku NPC untuk mengejar *player* pada saat kondisi *player* berada pada jarak ≤ 50. NPC akan terus mengejar *player* dan akan berlanjut pada aksi berikutnya yaitu patrol atau menyerang. NPC akan kembali patrol apabila *player* berhasil kabur atau menjauh dan keluar dari jangkauan NPC. Sedangkan NPC akan menyerang apabila *player* tidak dapat keluar dari jangkauan NPC. Berikut merupakan bentuk NPC pada saat melakukan aksi mengejar:

Gambar 4.11 Tampilan NPC sedang mengejar

4.6.3 NPC Menyerang

Pada bagian ini membahas mengenai implementasi *Markov State Machine* untuk mengatur perilaku NPC menyerang *player*. Ketika NPC mengejar *player* dan pada saat yang bersamaan *player* masih dalam jangkauan NPC bahkan semakin mendekat, maka perilaku NPC selanjutnya yaitu menyerang. Apabila NPC berhasil menyentuh *player*, maka kesehatan *player* akan berkurang. Berikut *script* untuk mengatur berkurangnya kesehatan *player* dalam kelas *Orang.cs*:

```
public Transform[] posisi;
public float speed = 10f;
Transform _transform;
int curPosisi = 0;

public enum STATUS { KEJAR, PATROL,SERANG  };

public STATUS statusMusuh ;

void OnTriggerExit(Collider other) {
if (other.gameObject.name.Equals ("adepes") ||
other.gameObject.name.Equals ("Node")) {
```

```
    {
            PlayerItems.Kesehatan -= 5;
            if (PlayerItems.Kesehatan <=0){
                  PlayerItems.Kesehatan =0;
                  }

            }
    }
}
```

Script tersebut di atas mengatur berkurangnya kesehatan *player* setelah bertabrakan dengan NPC. Setiap sekali bertabrakan kesehatan *player* akan berkurang 5 dan NPC akan terus-menerus menyerang *player* sampai *player* mati atau *player* berhasil lari dan menjauh dari NPC. Berikut adalah bentuk NPC pada saat menyerang:

Gambar 4.12 Tampilan NPC sedang menyerang

4.6.4 Permainan Selesai

Pada bagian ini membahas tentang bagaimana permainan berakhir. Dalam menentukan permainan selesai terdapat 3 kondisi yang mampu mengeksekusi agar permainan berhenti. Tiga kondisi

tersebut adalah apabila pertama *player* menyelesaikan misi, kedua waktu permainan habis dan yang ketiga apabila kesehatan *player* = 0.

Berikut adalah *script* yang dibuat untuk mengatur berakhirnya permainan disebabkan misi selesai dalam kelas *checkpoint.cs*:

```
public float sizex= 200;
public float  sizey = 130;
public float currentTime = 1;
public string misi = "Misi Sukses";
static int currentscore = 0;
public float xp, yp, zp;
public Texture gambar;
private bool trigerx  = false;
public bool destroy = true;
public bool kunci = false; // centang pada point pembuka
/ point1
  //public string pointskarang;
public string nextpoint;

void setkunci (bool x){
        kunci = x;
   }
   void OnTriggerEnter(Collider col){
         if (kunci==true && col.tag == "Sepeda"){


               //yield  WaitForSeconds(5);
               if(destroy){
               arrow = GameObject.FindWithTag("Start");
               nav = arrow.GetComponent("navigasi");
                     nav.SendMessage("setEnd",nextpoint);

         pickup = GameObject.FindWithTag(nextpoint);
         chekpoint = pickup.GetComponent("checkpoint");
         chekpoint.SendMessage("setkunci",true);
               Destroy(this.gameObject,0.1f);

               }else{
                     trigerx = true;
               }
         }
   }
void TheMainMenu (int x){
   GUILayout.Space(10);

   GUILayout.Label(score);
```

```
   GUILayout.Space(10);
   if(GUILayout.Button("MainMenu")){
         Application.LoadLevel(1);
   }

   if(GUILayout.Button("Restart")){
         Application.LoadLevel(2);
   }

   if(GUILayout.Button("Quit")){
         Application.Quit();
   }
}
void OnGUI() {
   if (trigerx) {

         GUI.DrawTexture (new Rect (Screen.width/2-
sizex/2, sizey/2, sizex, sizey), gambar);
               GUI.Window(0,new Rect (Screen.width/2-
sizex/2, Screen.height/2-sizey/2, sizex, sizey),
TheMainMenu, misi);
   }
}
```

Script di atas mengatur tentang bagaimana permainan selesai apabila misi *player* sukses. Misi *player* dikatakan sukses apabila *player* telah berhasil menemukan semua *obstacle* yang berupa angka dalam huruf arab. Berikut gambar yang menunjukkan misi sukses:

Gambar 4.13 Tampilan misi sukses

Berikut adalah *script* yang dibuat untuk mengatur berakhirnya permainan disebabkan waktu permainan telah habis dalam kelas *countdownTimer.cs*:

```
public static float currentTime = 300;
GameObject misi;
Component misix;

void FixedUpdate ()
   {

   if (currentTime <= 0) {
         misi = GameObject.Find("gagal");

         misix = misi.GetComponent("misi_notifikasi");
         misix.SendMessage("settrigger",true);

         Destroy(this.gameObject);
   }

   currentTime -= Time.deltaTime;

}
```

Serta pada kelas *misi_notifikasi.cs* untuk menampilakn notifikasi bahwa misi selesai. Berikut *script* yang ada pada kelas *misi_notifikasi.cs:*

```
public float sizex= 200;
public float  sizey = 130;
public float currentTime = 1;
static int currentscore = 0;
public float xp, yp, zp;
public Texture gambar;
private bool trigerx  = false;

void TheMainMenu (int x){
   GUILayout.Space(10);

   GUILayout.Label(score);

   GUILayout.Space(10);
   if(GUILayout.Button("MainMenu")){
         Application.LoadLevel(1);
   }

   if(GUILayout.Button("Restart")){
         Application.LoadLevel(2);
   }

   if(GUILayout.Button("Quit")){
         Application.Quit();
   }
}
void OnGUI() {
   if (trigerx) {

GUI.DrawTexture (new Rect (Screen.width/2-sizex/2,
sizey/2, sizex, sizey), gambar);
         GUI.Window(0,new Rect (Screen.width/2-sizex/2,
Screen.height/2-sizey/2, sizex, sizey), TheMainMenu,
misi);
         }
}
```

Script tersebut di atas mengatur untuk menghentikan permainan apabila waktu yang ditentukan telah habis. Lama waktu dalam permainan ini adalah 300 detik dan berhitung mundur. Permainan

dinyatakan gagal dan berhenti saat *player* belum menyelesaikan misi dan waktu = 0. Berikut adalah gambar yang menunjukkan misi gagal karena waktu habis:

Gambar 4.14 Tampilan misi gagal karena waktu habis

Selanjutnya adalah *script* yang dibuat untuk mengatur berakhirnya permainan disebabkan kesehatan *player* telah habis/ kesehatan = 0 dalam kelas *Orang.cs*:

```
public Transform[] posisi;
public float speed = 10f;
Transform _transform;
int curPosisi = 0;
GameObject misi;
Component misix;

void FixedUpdate(){
if (Player!=null)Jarak =
Vector3.Distance(gameObject.transform.position,
Player.transform.position);
   if (PlayerItems.Kesehatan <= 0) {
```

```
            Time.timeScale = 0;
            Debug.Log("Misi gagal");

            misi = GameObject.Find("gagal");

            misix = misi.GetComponent("misi_notifikasi");
            misix.SendMessage("settrigger",true);

            Destroy(this.gameObject);
    }
}
```

Script di atas mengatur untuk menghentikan permainan pada saat kondisi kesehatan *player* = 0. Kesehatan *player* bernilai 100 dan akan berkurang 5 apabila menabrak NPC. Ketika kesehatan *player* habis dan *player* belum menyelesaikan misi maka misi gagal dan permainan berakhir. Berikut gambar yang menunjukkan misi gagal karena kesehatan *player* = 0:

Gambar 4.15 Tampilan misi gagal karena kesehatan = 0

4.6.5 Implementasi *Markov State Machine* pada *game*

Pada bagian ini membahas tentang implementasi *Markov State Machine* dalam permainan. Perhitungan *Markov State Machine* dalam permainan disesuaikan dengan perhitungan algoritma permainan yang ada pada bab iii yaitu dengan menentukan nilai matriks probabilitas terlebih dahulu. Matriks probabilitas tersebut yang nantinya akan menentukan perilaku NPC terhadap *player*, kapan NPC akan patrol, mengejar dan menyerang.

Dalam program yang dibuat, algoritma *Markov State Machine* diletakkan pada NPC. *Script* yang dibuat untuk menentukan perilaku NPC terhadap player disesuaikan dengan perhitungan algoritma *Markov State Machine.* Algoritma *Markov State Machine* dalam game menghitung jarak, kesehatan dan kecepatan *player* untuk nantinya akan dilakukan perhitungan dan hasil akhirnya akan menentukan perilaku NPC. NPC akan mempengaruhi kesehatan *player* jika NPC sampai menyerang dan menentuh pemain.

Berikut adalah *script* yang menjelaskan implementasi *Markov State Machine* untuk menentukan perilaku NPC terhadap *player* dalam kelas *Orang.cs:*

```
using UnityEngine;
using System.Collections;

public class Orang : MonoBehaviour {

   public Transform[] posisi;
   public float speed = 10f;
   Transform _transform;
   int curPosisi = 0;

   public GameObject Player;
```

```
public AudioSource audio;
public GameObject textSpeed;
public float Jarak   ;
public AnimationClip walkAnimation ;
public AnimationClip runAnimation ;

GameObject misi;
Component misix;
```

Script di bawah ini menentukan nilai parameter awal yang akan menjadi acuan hasil perilaku NPC. Nilai parameter awal ini juga nantinya akan menjadi matriks probabilitas transisi untuk menghitung kemungkinan perilaku NPC terhadap *player*.

```
public int[,] parameter =  {
      {50,40,10},
      {60,40,20},
      {70,30,30}};

public double[,] matrixProb =   {
      {0,0,0},
      {0,0,0},
      {0,0,0}};

public double[,] statusPlayer =   {
      {0,0,0},
      {0,0,0},
      {0,0,0}};
```

Setelah dibuat nilai parameter awal, maka proses selanjutnya yaitu dengan merubah nilai parameter tersebut menjadi matriks probabilitas transisi. Perhitungan dilakukan dengan cara menghitung jumlah total setiap baris parameter yang sudah dibuat. *Script* perhitungannya dapat dilihat di bawah ini:

```
public enum STATUS { KEJAR, PATROL,SERANG  };

public STATUS statusMusuh ;

int totalJarak,totalKesehatan,totalKecepatan;
void hitungTotal(){
      totalJarak     = parameter [0,0] + parameter
[0,1] + parameter [0,2];
```

```
        totalKesehatan = parameter [1,0] + parameter
[1,1] + parameter [1,2];
        totalKecepatan = parameter [2,0] + parameter
[2,1] + parameter [2,2];
   }
```

Setelah dihitung nilai total setiap baris parameter, langkah selanjutnya yaitu dengan membagi nilai setiap parameter dengan nilai total agar mendapat nilai bilangan non negatif yang tidak lebih dari satu sebagai syarat matriks probabilitas transisi. Perhitungan yang dilakukan ini untuk membuat matriks probabilitas transisi. Berikut adalah *script* perhitungannya:

```
  void hitungMatrix() {
       hitungTotal ();

       for (int i =0; i<3; i++) {
            matrixProb[0,i] = (double) parameter
[0,i]/totalJarak;
            matrixProb[1,i] = (double) parameter
[1,i]/totalKesehatan;
            matrixProb[2,i] =  (double) parameter
[2,i]/totalKecepatan;

       }
  }
```

Selanjutnya, *script* di bawah ini akan menghitung nilai *input* berupa kondisi *player* pada saat bertemu NPC. Nilai *input* yang telah dijadikan bilangan riil tak negatif dan kurang dari satu kemudian dikalikan dengan matriks probabilitas transisi. Hasil perhitungan ini akan mendapatkan status *player* pada saat setelah bertemu NPC. Berikut adalah *script* yang dibuat:

```
void hitung Player(int Jarak,int Kesehatan,int
Kecepatan){

  int total Player = Jarak + Kesehatan + Kecepatan;
  double Jr = (double)Jarak / total Player;
  double Ks = (double)Kesehatan / total Player;
```

```
   double Kc = (double)Kecepatan / total Player;

   double finJr = 0;
   double finKs = 0;
   double finKc = 0;

   finJr  = (double) Jr *  matrixProb[0,0]  + (double) Ks
*  matrixProb[1,0] + (double) Kc *  matrixProb[2,0] ;
   finKs  = (double) Jr *  matrixProb[0,1]  + (double) Ks
*  matrixProb[1,1] + (double) Kc *  matrixProb[2,1] ;
   finKc  = (double) Jr *  matrixProb[0,2]  + (double) Ks
*  matrixProb[1,2] + (double) Kc *  matrixProb[2,2] ;
   finJr = (double) finJr * total Player;
   finKs = (double) finKs * total Player;
   finKc = (double) finKc * total Player;

   status Player = new double[3,3]  {
         {0,0,0},
         {0,0,0},
         {0,0,0}};

   int idJr=0, idKs=0, idKc=0;

   double lastJr=100, lastKs=100, lastKc=100;
   for (int i =0; i<3; i++) {
         double diffJr=(double)Mathf.Abs((float)finJr -
parameter [0,i]),

   diffKs=(double)Mathf.Abs((float)finKs - parameter
[1,i]),

   diffKc=(double)Mathf.Abs((float)finKc - parameter
[2,i]);

               if (lastJr > diffJr ){
                     lastJr = diffJr;
                     idJr = i;
               }
               if (lastKs > diffKs ){
                     lastKs = diffKs;
                     idKs = i;
               }
               if (lastKc > diffKc ){
                     lastKc = diffKc;
                     idKc = i;
               }

         }

         status Player[0,idJr] =1;
         status Player[1,idKs ] = 1;
```

```
status[2,idKc] = 1;
int[] hasil =  {0,0,0};

hasil[idJr]++;
hasil[idKs]++;
hasil[idKc]++;
```

Hasil perhitungan tersebut di atas akan dijadikan parameter untuk menentukan perilaku apa yang akan terjadi pada NPC. *Script* untuk menentukan perilaku NPC dapat dilihat di bawah ini:

```
int maxSol = 0;
int idxSol = 0;
for (int i =0; i<3; i++) {
     if (hasil[i]!=0)  {
          if (hasil[i] > maxSol ){
               maxSol =hasil[i];
               idxSol = i;
          }
     }
}

if (idxSol == 0) {
     statusMusuh = STATUS.PATROL;
}else if (idxSol == 1) {
     statusMusuh = STATUS.KEJAR;
}else if (idxSol == 2) {
     statusMusuh = STATUS.SERANG;
}

}
```

Script berikut dibuat untuk menentukan berakhirnya permainan dikarenakan kesehatan *player* habis/kesehatan = 0.

```
  void FixedUpdate(){
       if (Player!=null)  Jarak = Vector3.Distance
(gameObject.transform.position,
Player.transform.position);
  if (PlayerItems.Kesehatan <= 0) {
       Time.timeScale = 0;
       Debug.Log("Misi gagal");

       misi = GameObject.Find("gagal");
       misix = misi.GetComponent("misi_notifikasi");
       misix.SendMessage("settrigger",true);
```

```
        Destroy(this.gameObject);
    }

    void OnCollisionEnter(Collision collision) {


    }
```

Script berikut dibuat untuk mengatur berkurangnya kesehatan *player* apabila diserang oleh NPC. Dalam *script* ini diatur jika *player* diserang oleh NPC, maka kesehatan akan berkurang 5.

```
    void OnTriggerEnter(Collider other) {

        if (other.gameObject.name.Equals ("adepes") ||
other.gameObject.name.Equals ("Node")) {

        audio.Play();
        other.transform.position = new
Vector3(other.transform.position.x ,
        other.transform.position.y ,
         other.transform.position.z-20);

        }
    }


    void OnTriggerExit(Collider other) {
        if (other.gameObject.name.Equals ("adepes") ||
other.gameObject.name.Equals ("Node")) {

        {

              PlayerItems.Kesehatan -= 5;
              if (PlayerItems.Kesehatan <=0){
                  PlayerItems.Kesehatan =0;
              }
        }
    }
}
void Awake(){
    _transform = transform;
}
```

Script di bawah berikut mengatur perilaku NPC yaitu patrol apabila kesehatan *player* masih penuh/kesehatan = 100.

```
void Start(){
   PlayerItems.Kesehatan = 100;
   statusMusuh = STATUS.PATROL;
   hitungMatrix ();
   hitungPlayer (Mathf.CeilToInt( Jarak),
Mathf.CeilToInt(PlayerItems.Kesehatan),
Mathf.CeilToInt(PlayerItems.Kecepatan));

}

void GantiPosisi(){
   if(curPosisi+1 == posisi.Length){
               curPosisi = 0;
   }else{
         curPosisi++;
   }
}
```

Berikut adalah *script* yang mengatur kecepatan serta animasi NPC pada masing- masing perilaku. Dalam *script* ini diatur pada saat NPC patrol maka animasinya akan berjalan dengan kecepatan 7. Sedangkan pada saat mengejar *player*, NPC akan berlari dengan kecepatan 15. Dan pada saat NPC mengejar *player*, maka akan berlari dengan kecepatan 20.

```
void Update(){

   PlayerItems.Kecepatan = (int.Parse
(textSpeed.guiText.text) + 1) * 10;

   hitungPlayer (Mathf.CeilToInt( Jarak),
Mathf.CeilToInt(PlayerItems.Kesehatan),
Mathf.CeilToInt(PlayerItems.Kecepatan));

   if (statusMusuh == STATUS.KEJAR) {
         speed = 15;
         gameObject.GetComponent<Animation>().Play("run")
   ;
         gameObject.GetComponent<Animation>()["run"].spee
   d =  0.5f;

   } else if (statusMusuh == STATUS.SERANG) {
         speed = 20;
         gameObject.GetComponent<Animation>().Play("run")
   ;
```

```
        gameObject.GetComponent<Animation>()["run"].spee
   d =  0.5f;
   } else {
        speed =7;
        gameObject.GetComponent<Animation>().Play
      ("walk");
   }

   if (statusMusuh == STATUS.PATROL) {
        Vector3 targetPos = posisi [curPosisi].position;
        Quaternion targetRotation =
Quaternion.LookRotation (new Vector3 (targetPos.x,
_transform.position.y, targetPos.z) -
_transform.position);
        _transform.rotation = Quaternion.Slerp
(_transform.rotation, targetRotation, Time.deltaTime *
2f);
        _transform.Translate (Vector3.forward * speed *
Time.deltaTime);
        if ((targetPos -
_transform.position).sqrMagnitude < 700f) {
             GantiPosisi ();
             print ("ganti");
        }
   } else if (statusMusuh == STATUS.KEJAR) {
        Vector3 targetPos = Player.transform.position;
        Quaternion targetRotation =
Quaternion.LookRotation (new Vector3 (targetPos.x,
_transform.position.y, targetPos.z) -
_transform.position);
        _transform.rotation = Quaternion.Slerp
(_transform.rotation, targetRotation, Time.deltaTime *
2f);
        _transform.Translate (Vector3.forward * speed *
Time.deltaTime);

        }
   }

}
```

4.7 Uji Coba

Untuk mengetahui sejauh mana implementasi algoritma terhadap sistem permainan, maka perlu dilakukan pengujian. Pengujian dijalankan pada computer dengan spesifikasi sebagai berikut:

8. Prosesor intel Core i3 3217U @1.8 GHz
9. HardDisk 500 GB - 5600 RPM
10. RAM 2GB
11. VGA IntelHD Graphics 4000 1 Gb Shared memory
12. Keyboard
13. Mouse Laser
14. Monitor 14’’

4.7.1 Hasil pengujian perilaku NPC terhadap *player*

Pada pengujian permainan kali ini dilakukan dengan menjalankan permainan dan melihat perilaku NPC pada saat *player* berada pada kondisi awal sampai kondisi *player* mendekati NPC dan mendapat serangan. Dengan cara perhitungan tersebut di atas didapatkan hasil dari beberapa input kondisi awal *player* menggunakan ***tabel 3.3*** sebagai berikut:

Tabel 4.1 Nilai awal kondisi *player*

No	Variabel *Input*			Jumlah
	Jarak	Kesehatan	Kecepatan	
1	80	80	80	240
2	80	70	70	220
3	80	70	60	210
4	80	60	50	190
5	80	50	50	180
6	70	60	40	170
7	70	60	70	200
8	70	80	80	230
9	70	50	30	150
10	70	50	40	160
11	60	70	60	190
12	60	70	70	200
13	60	60	60	180
14	60	50	50	160
15	60	50	30	140
16	50	50	50	150
17	50	30	50	130
18	50	30	30	110
19	50	30	40	120
20	50	30	20	100
21	40	30	40	110
22	30	30	40	100

Tabel 4.1 Nilai awal kondisi *player* (sambungan)

No	Variabel *input*			Jumlah
	Jarak	Kesehatan	Kecepatan	
23	40	30	40	110
24	20	30	40	90
25	20	20	20	60
26	20	20	10	50
27	10	20	10	40
28	20	10	10	40
29	10	10	20	40
30	10	10	10	30

Tabel 4.2 Hasil perkalian dengan Matriks Probabilitas Transisi

Hasil kali Matriks probabilitas transisi		
Jr	Ks	Kc
0.55	0.283333	0.166667
0.552273	0.284091	0.163636
0.552381	0.285714	0.161905
0.555263	0.286842	0.157895
0.558333	0.286111	0.155556
0.552941	0.288235	0.158824
0.5525	0.2825	0.165
0.547826	0.282609	0.169565
0.556667	0.29	0.153333
0.55625	0.2875	0.15625

Tabel 4.2 Hasil perkalian dengan Matriks Probabilitas Transisi (sambungan)

Hasil kali Matriks probabilitas transisi		
Jr	Ks	Kc
0.547368	0.284211	0.168421
0.5475	0.2825	0.17
0.55	0.283333	0.166667
0.553125	0.284375	0.1625
0.553571	0.289286	0.157143
0.55	0.283333	0.166667
0.557692	0.280769	0.161538
0.559091	0.286364	0.154545
0.558333	0.283333	0.158333
0.56	0.29	0.15
0.554545	0.281818	0.163636
0.55	0.28	0.17
0.554545	0.281818	0.163636
0.544444	0.277778	0.177778
0.55	0.283333	0.166667
0.55	0.29	0.16
0.5375	0.2875	0.175
0.5625	0.2875	0.15
0.55	0.275	0.175
0.55	0.283333	0.166667

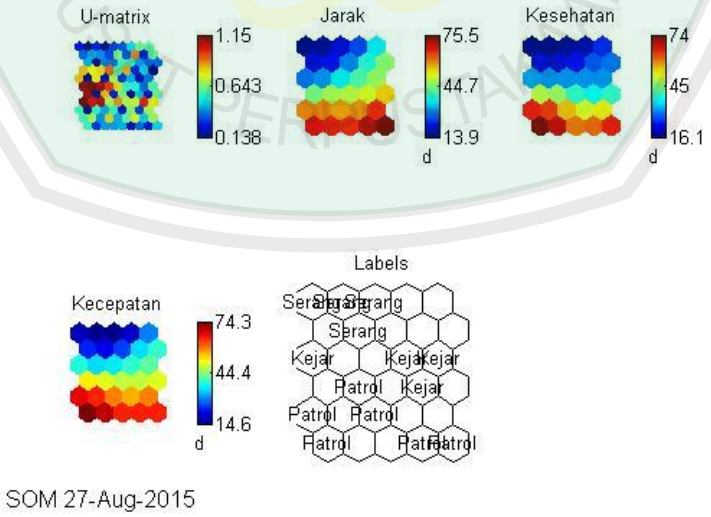
Tabel 4.3 *Output* aksi NPC terhadap *player*

Hasil Kali Jumlah			
Jarak	Kesehatan	Kecepatan	Keterangan
132	68	40	Patrol
121.5	62.5	36	Patrol
116	60	34	Patrol
105.5	54.5	30	Patrol
100.5	51.5	28	Patrol
94	49	27	Patrol
110.5	56.5	33	Patrol
126	65	39	Patrol
83.5	43.5	23	Patrol
89	46	25	Patrol
104	54	32	Patrol
109.5	56.5	34	Patrol
99	51	30	Patrol
88.5	45.5	26	Patrol
77.5	40.5	22	Patrol
82.5	42.5	25	Patrol
72.5	36.5	21	Kejar
61.5	31.5	17	Kejar
67	34	19	Kejar
56	29	15	Kejar
61	31	18	Kejar
55	28	17	Kejar

Tabel 4.3 *Output* aksi NPC terhadap *player* (sambungan)

Hasil Kali Jumlah			
Jarak	Kesehatan	Kecepatan	Keterangan
61	31	18	Kejar
49	25	16	Kejar
33	17	10	Serang
27.5	14.5	8	Serang
21.5	11.5	7	Serang
22.5	11.5	6	Serang
22	11	7	Serang
16.5	8.5	5	Serang

Berdasarkan sampel variabel *input* tersebut di atas dapat dibuat grafik yang menunjukkan kemungkinan aksi NPC terhadap *player* sebagai berikut

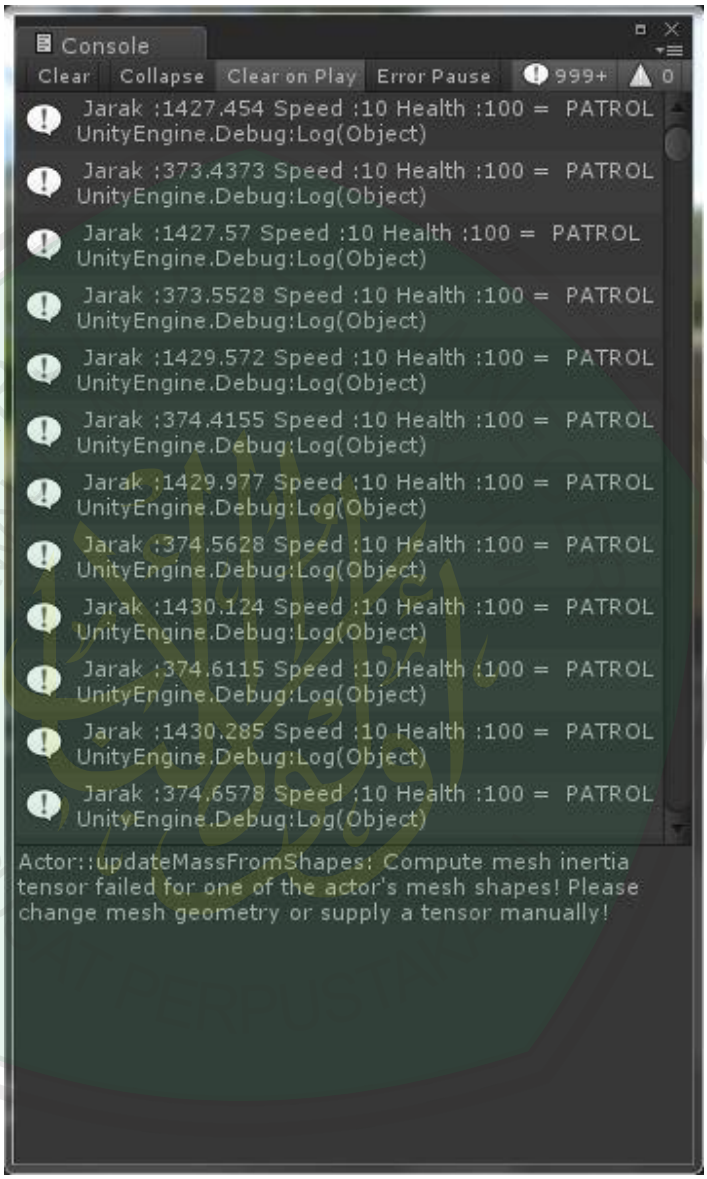


Gambar 4.16 Grafik aksi NPC terhadap *player*

Grafik tersebut di atas merupakan grafik hexagonal yang memetakan perilaku NPC terhadap *player*. Dalam grafik dilambangkan dengan warna dengan penempatan angka yang terkecil berada pada pojok kiri atas sampai yang terbesar berada pada pojok kanan bawah. U-matrix yang berada pada kiri atas, komponen variabel input dan label map unit pada pojok kanan bawah, semuanya terhubung dengan posisi data yang diolah. Pewarnaan hexagon diumpamakan dengan panas yang artinya semakin tinggi nilai input maka warnanya akan semakin merah. Dalam grafik warna biru berada di kiri atas karena area itu merupakan area unit yang memiliki nilai yang rendah. Sedangkan pada posisi kanan bawah berwarna merah yang menandakan posisi tersebut mempunyai nilai yang paling tinggi.

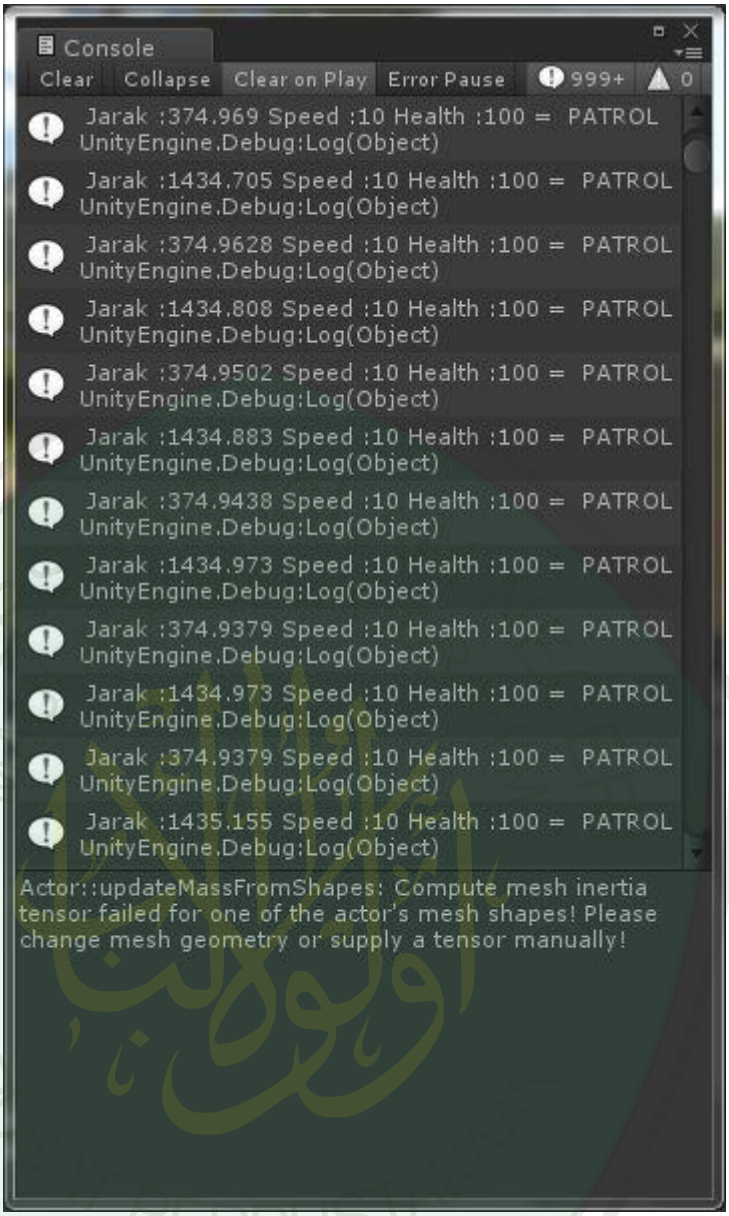
Pada masing- masing pola, hexagon pada beberapa posisi cocok dengan yang ada di map unit. Pada U-matrix hexagon tambahan berada diantara semua pasangan dari map unit yang berdekatan. Dalam U-matrix merupakan gambaran pola yang ada dalam label map unit. Sehingga penggambaran berdasarkan warna dilakukan dengan memunculkan warna dari *output* yang berdekatan nilainya. Sebagai contoh, pada label map unit kiri atas merupakan representasi nilai rendah dari jarak, kesehatan dan kecepatan. Posisi tersebut dalam map unit dituliskan sebagai *serang* dan pada U-matrix dapat dilihat bahwa posisi tersebut sangat dekat dengan unit lain yang memiliki nilai yang berdekatan.

Perubahan perilaku NPC terhadap *player* juga ditampilkan pada *console game engine* dalam *debug mode.* Berikut merupakan *screenshot* perilaku NPC dengan kondisi *player* yang berbeda:



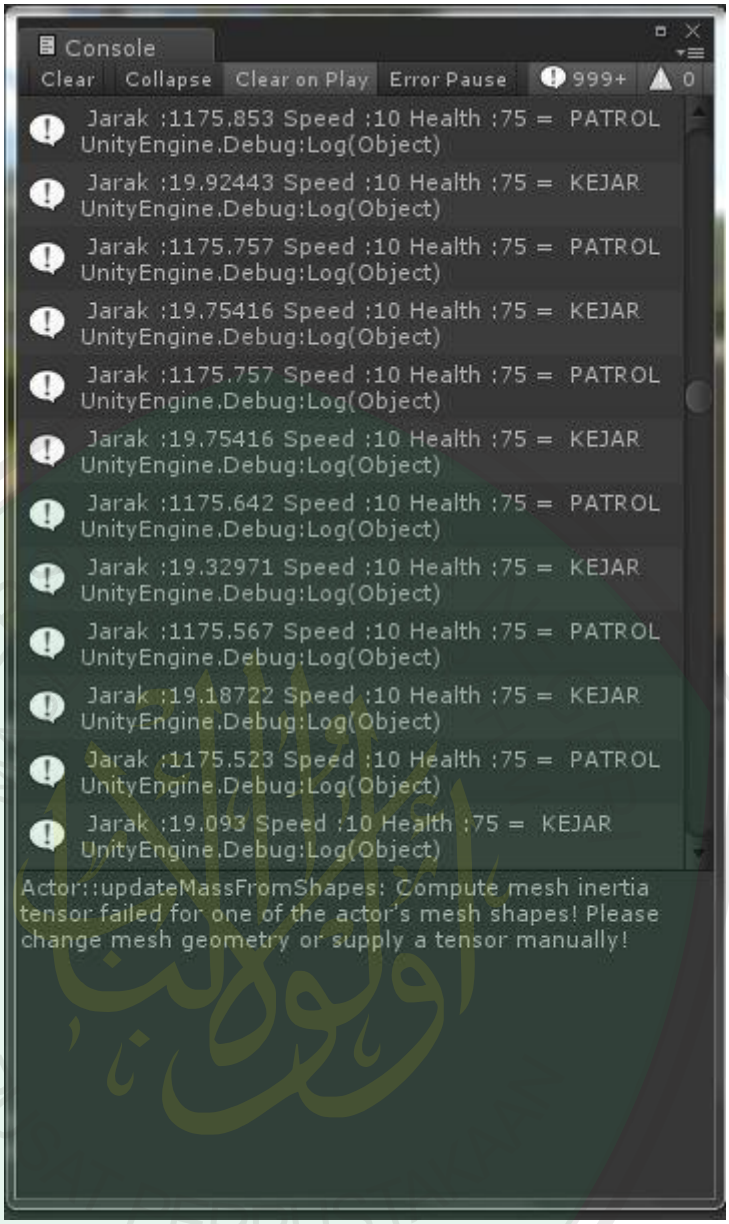
Gambar 4.17 NPC patrol

Pada gambar 4.17 di atas menunjukkan perilaku NPC yaitu patrol dikarenakan kondisi kesehatan *player* masih penuh dan jarak *player* terhadap NPC masih jauh.



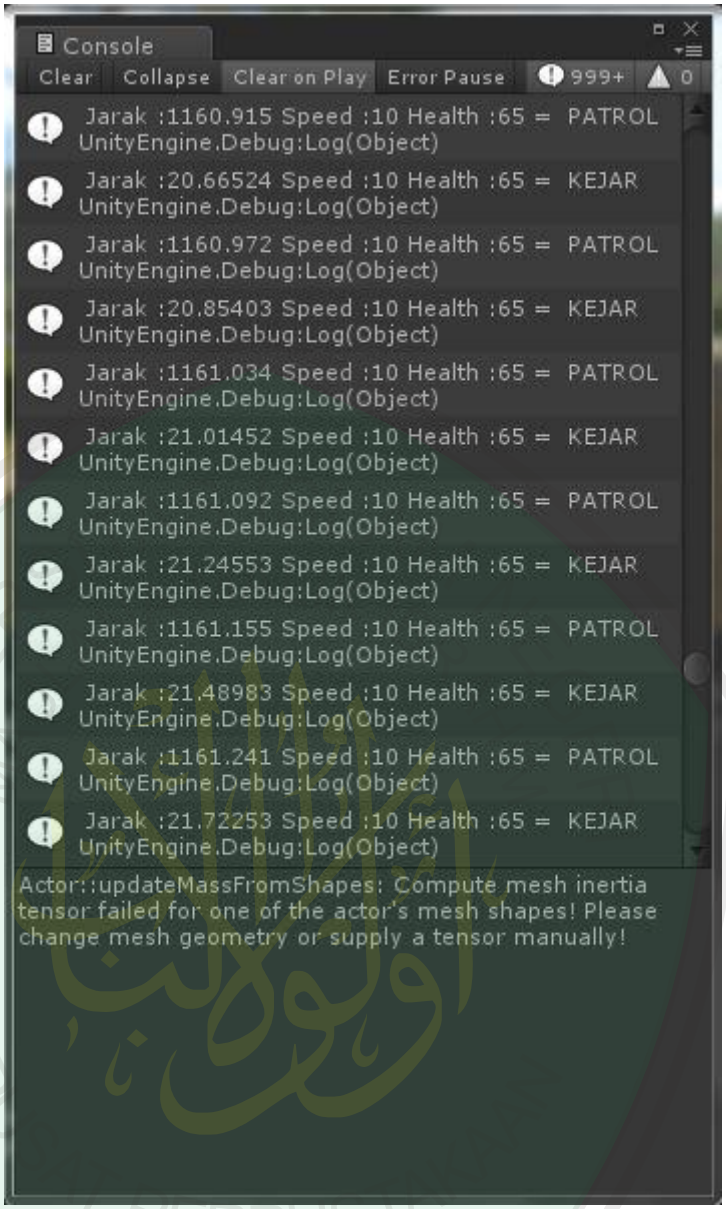
Gambar 4.18 NPC patrol

Pada gambar 4.18 di atas menunjukkan perilaku NPC yaitu patrol dikarenakan kondisi kesehatan *player* masih penuh dan jarak *player* terhadap NPC masih jauh.



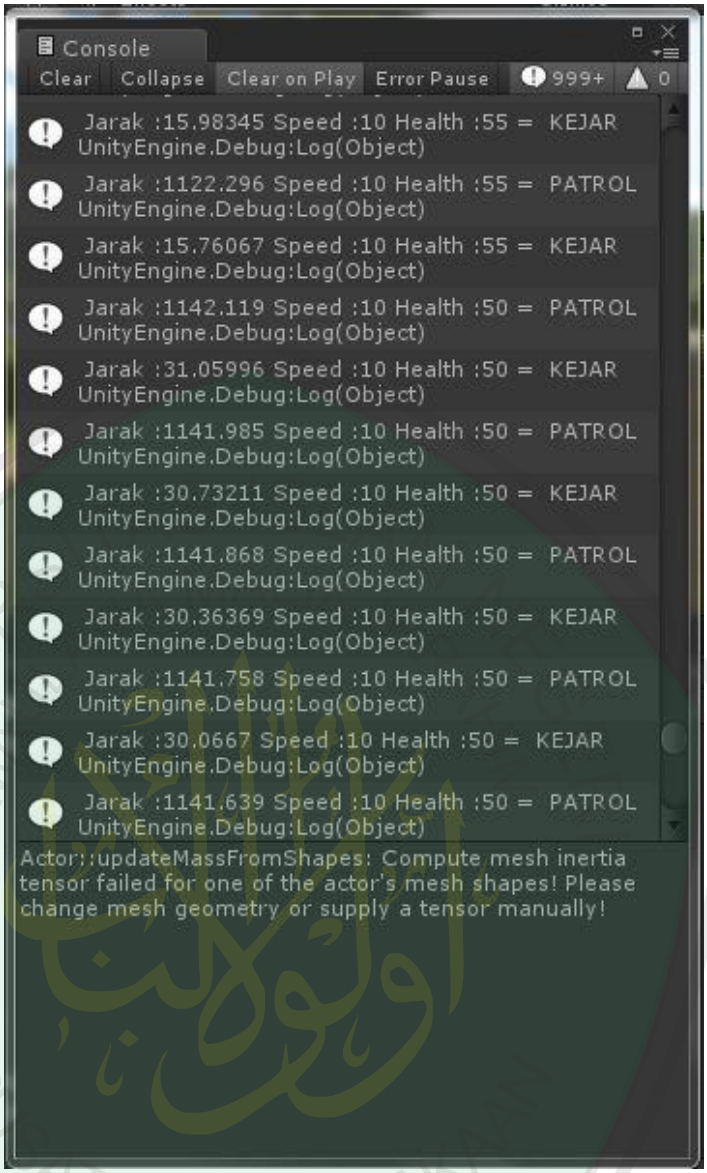
Gambar 4.19 NPC mengejar *player*

Pada gambar 4.19 di atas menunjukkan perilaku NPC mengejar *player* dikarenakan jarak *player* terhadap NPC sangat dekat dan NPC dapat menjangkaunya. Selain itu kesehatan *player* juga sudah mulai berkurang menjadi 75.



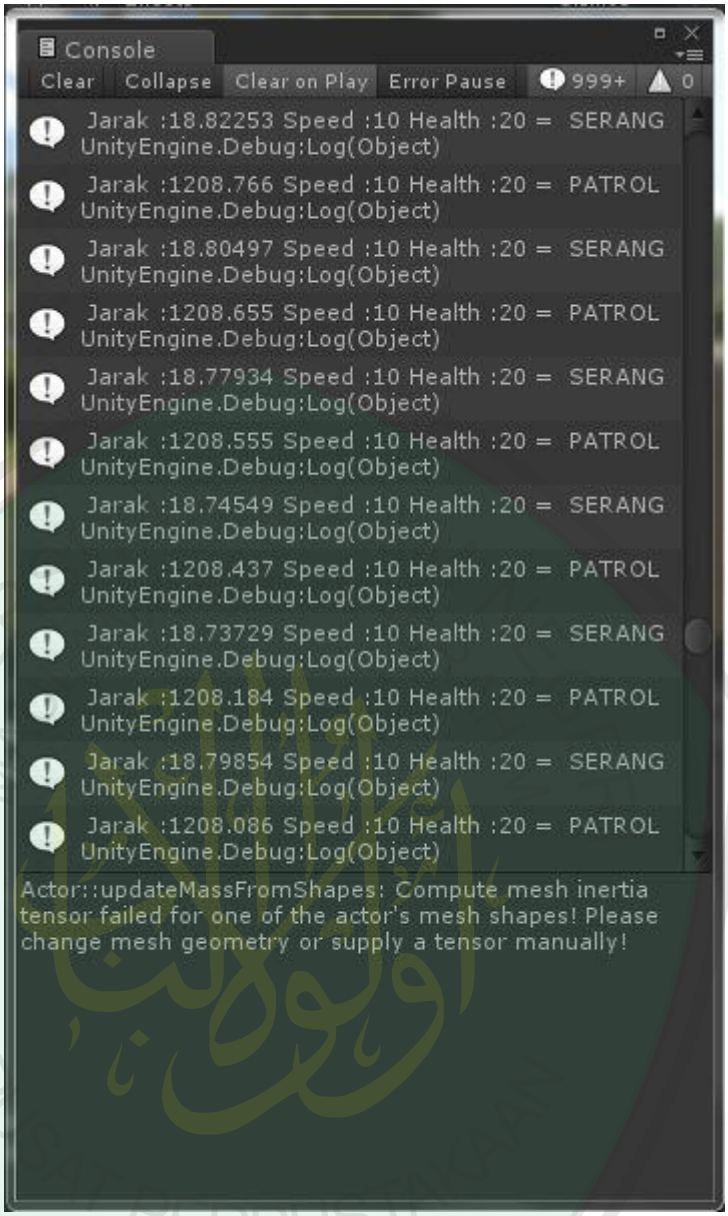
Gambar 4.20 NPC mengejar *player*

Pada gambar 4.20 di atas menunjukkan perilaku NPC mengejar *player* dikarenakan jarak *player* terhadap NPC sangat dekat dan NPC dapat menjangkaunya. Selain itu kesehatan *player* juga sudah mulai berkurang menjadi 65.



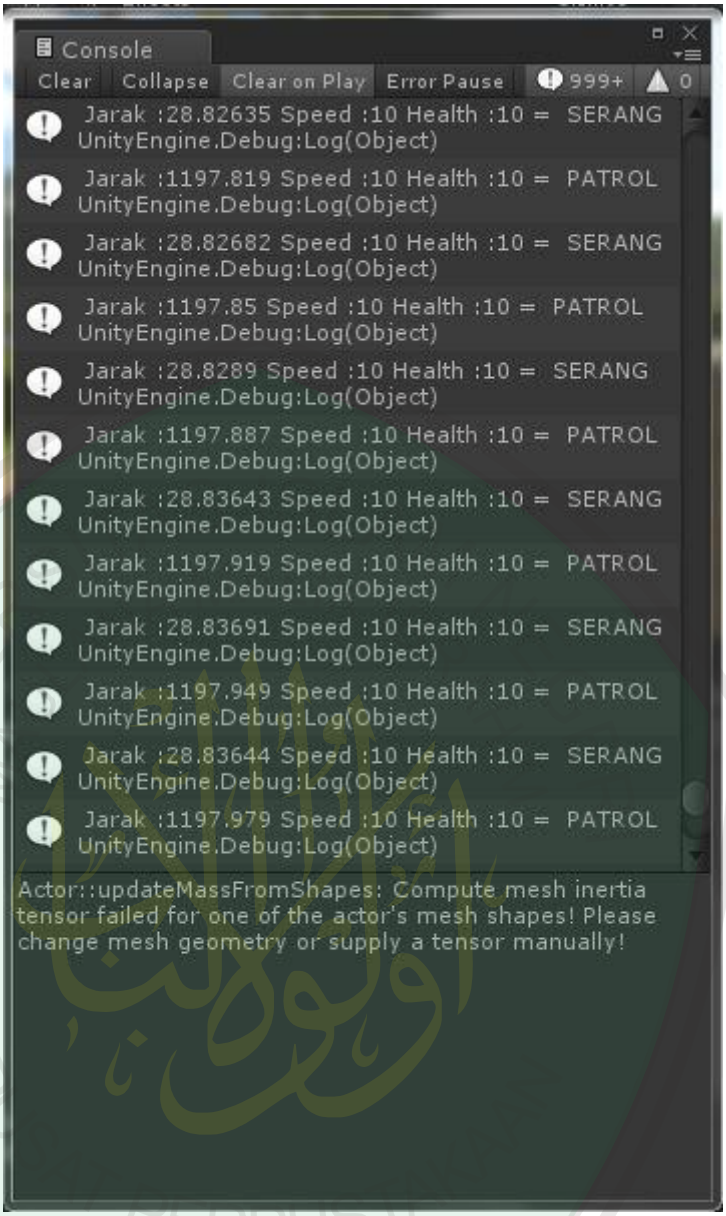
Gambar 4.21 NPC mengejar *player*

Pada gambar 4.21 di atas menunjukkan perilaku NPC mengejar *player* dikarenakan jarak *player* terhadap NPC sangat dekat dan NPC dapat menjangkaunya. Selain itu kesehatan *player* juga sudah mulai berkurang menjadi 50. Dalam gambar 4.19, 4.20 dan 4.21, NPC memiliki perilaku patrol dan kejar. Hal tersebut dikarekanan *player* berusaha menjauh dari NPC, sehingga terlihat jarak *player* terhadap NPC selalu berubah.



Gambar 4.22 NPC menyerang *player*

Pada gambar 4.22 di atas menunjukkan perilaku NPC menyerang *player* dikarenakan jarak *player* terhadap NPC sangat dekat dan NPC dapat menjangkaunya. Selain itu faktor yang mempengaruhi juga kesehatan *player* sudah semakin kecil yaitu 20.



Gambar 4.23 NPC menyerang *player*

Pada gambar 4.23 di atas menunjukkan perilaku NPC menyerang *player* dikarenakan jarak *player* terhadap NPC sangat dekat dan NPC dapat menjangkaunya. Selain itu faktor yang mempengaruhi juga kesehatan *player* sudah semakin kecil yaitu 10.

Dalam gambar 4.22 dan 4.23, NPC memiliki perilaku patrol dan serang. Hal tersebut dikarekanan *player* berusaha menjauh dari

NPC, sehingga terlihat jarak *player* terhadap NPC selalu berubah. Dan nilai kesehatan *player* terus berkurang ketika NPC menyerang dan *player* tidak berhasil menjauh.

BAB V

PENUTUP

3.2 Kesimpulan

Berdasarkan game yang telah di buat dan uji coba yang telah dilakukan, maka dapat ditarik kesimpulan bahwa algoritma *Markov State Machine* dapat diimplementasikan pada NPC untuk menentukan perilakunya terhadap *player* berdasarkan kondisi *player* pada saat bertemu NPC. Berdasarkan hasil uji coba dari 30 data kondisi *player* yang berbeda, perilaku NPC dapat dipresentasekan yaitu 53,3% NPC patrol, 26,7% NPC mengejar dan 20% NPC menyerang.

3.3 Saran

Berdasarkan kesimpulan di atas, terdapat beberapa saran untuk pengembangan game ini selanjutnya:

1. *Game* yang di bangun lebih memiliki tantangan yang kompleks dan menantang, sehingga dapat menarik.
2. Penambahan algoritma yang lebih bagus agar game berjalan lebih baik.
3. Penambahan NPC pada beberapa titik dan memiliki kemampuan yang berbeda.

DAFTAR PUSTAKA

Abdurachman Edi, *Konsep Dasar Markov Chain serta Kemungkinan Penerapannya di Bidang Pertanian*, Journal Informatika Pertanian *volume 8*, Desember 1999

Chen, Jun., Suksompong, Prapun., Berger, Toby. *Communication through a Finite State Machine with Markov Property.* School of Electrical and Computer Engineering. Cornell University. Itacha, New York.

Craig, W. Reynolds. *Steering Behaviors For Autonomous Characters*. Sony Computer Entertainment America.1999

El Jinar, Bilqis. *Pemodelan Permainan Monopoli Menggunakan Rantai Markov.* Skripsi Fakultas Sains dan Teknologi. UIN Syarif Hidayatullah Jakarta. 2011

Millington, Ian. *Artifial Intelligence for Games.*San Francisco, U.S.A.: Morgan Kaufmann Publishers. 2006

Monga, Ana., Singh, Balwinder. *Finite State Machine Based Vending Machine Controller with Auto-Billing Features.* International Journal of VLSI design & Communication Systems (VLSICS). Vol.3, No.2. April 2012

Nandya, Matahari Bhakti., Gunanto, Samuel Gandang., Santosa, R. Gunawan. *Pemetaan Perilaku Non Playable Character pada Permainan Berbasis Role Playing Game Menggunakan Metode Finite State Machine.* Jurnal Teknik Informatika. 2011

Nofiantoro, Arix. *Analsis dan Perancangan Game "Bermain Bersama Dito & Dola"*. Skripsi Tidak Diterbitkan. Yogyakarta: AMIKOM. 2011

Nugroho, Supeno Mardi Susiki. dkk. 2011. *Perilaku Taktis Untuk Non-Player Characters Di Game Peperangan Meniru Strategi Manusia Menggunakan Fuzzy Logic Dan Hierarchical Finite State Machine*. Surabaya: Jurnal Ilmiah Kursor Vol. 6, No. 1, Januari 2011.

Peters, Clifford, Kyaw, A. S., & Swe, T. N. (2013). *Unity 4.x Game AI Programming.* BIRMINGHAM - MUMBAI: Packt Publishing.

Salen, Katie and Eric Zimmerman. *Rules of Play: Game Design Fundamentals*.Cambridge (MA): The MIT Press.2004

Setiawan, Iwan. *Perancangan Software Embedded System Berbasis FSM.* Jurnal Teknik Elektro UNDIP. 2006

Siang, Jong Jek. *Jaringan Syaraf Tiruan dan Pemrogramannya Menggunakan MATLAB.* ANDI. Yogyakarta. 2009

Subaktil, Hanas., Adams, Eriq Muhammad., Yudistira, Novanto. *Rancang Bangun NPC (Non-Player Characters) pada Game Bergenre Tower Defense.* Jurnal Teknik Informatika. Universitas Brawijaya. Malang. 2013

Sweetser, Penelope., Wiles, Janet. *Current AI in Games: a Review.* Australian Journal of Intelligent Information Processing Systems. 8(1). pp. 24-42. 2002

Taylor, Howard M. *An Introduction To Stochastic Modeling Revised Edition.* Academic Press Limited. London. 1998