RANCANG BANGUN NPC (*NON-PLAYER CHARACTERS*) PADA *GAME* BERGENRE *TOWER DEFENSE*

SKRIPSI

LABORATORIUM *GAME*

Untuk memenuhi sebagian persyaratan untuk mencapai gelar Sarjana Komputer

Disusun Oleh :

HANAS SUBAKTI

NIM. 0910680019

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN

PROGRAM STUDI INFORMATIKA/ILMU KOMPUTER

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MALANG

2013

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

LEMBAR PERSETUJUAN

RANCANG BANGUN NPC (*NON-PLAYER CHARACTERS*) PADA *GAME* BERGENRE *TOWER DEFENSE*

SKRIPSI

LABORATORIUM *GAME*

Untuk memenuhi sebagian persyaratan untuk mencapai gelar Sarjana Komputer

Disusun Oleh :

HANAS SUBAKTI

NIM. 0910680019

Proposal skripsi ini telah diperikasa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Eriq M. Adams J, ST., M.Kom	**Novanto Yudistira, S.Kom., M.Sc.**
NIK. 850410 06 1 1 0027	**NIK. 83110 16 1 1 0425**

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN

RANCANG BANGUN NPC (*NON-PLAYER CHARACTERS*)
PADA *GAME* BERGENRE *TOWER DEFENSE*

SKRIPSI

LABORATORIUM *GAME*

Untuk memenuhi persyaratan memperoleh gelar Sarjana Komputer

Disusun oleh :

HANAS SUBAKTI

NIM. 0910680019

Skripsi ini telah diuji dan dinyatakan lulus pada

Tanggal 2 Januari 2014

Penguji I	Penguji II
Denny Sagita R., S.Kom., M.Kom	**Wibisono Sukmo Wardhono, ST., MT.**
NIP. 851124 06 1 1 0250	**NIP. 820404 06 1 1 0091**

Penguji III

Fajar Pradana, S.ST, M.Eng

NIK. 871121 16 1 1 0371

Mengetahui

Ketua Program Studi Teknik Informatika

Drs. Marji, M.T.

NIP. 19670801 199203 1 001

PERNYATAAN
ORISINALITAS SKRIPSI

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah SKRIPSI ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata di dalam naskah SKRIPSI ini dapat dibuktikan terdapat unsur-unsur PLAGIASI, saya bersedia SKRIPSI ini digugurkan dan gelar akademik yang telah saya peroleh (SARJANA) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku. (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 13 Januari 2014

Mahasiswa,

Hanas Subakti
NIM 0910680019

PRAKATA

Puji syukur saya ucapkan kepada Allah SWT yang senantiasa melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Proposal Skripsi yang berjudul, "Rancang Bangun NPC (*Non-Player Characters*) pada *Game* Bergenre *Tower Defense*".

Skripsi ini diajukan sebagai salah satu syarat untuk mendapatkan gelar sarjana S-1 Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya. Penyusun menyelesaikan Skripsi ini berdasarkan teori-teori yang telah di peroleh dalam perkuliahan, literature dari beberapa buku dan papper dan bimbingan dari dosen pembimbing serta pihak-pihak lain yang telah banyak memberikan semangat dan bantuan. Penyusun ingin menyampaikan penghargaan dan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Kedua orang tua penulis yang senantiasa memberikan dukungan, kasih sayang dan do'a demi terselesaikannya proposal skripsi ini.
2. Bapak Drs. Marji, M.Si. dan Issa Arwani, ST., MT.. selaku Ketua dan Sekretaris Program Studi Informatika serta segenap Bapak/Ibu Dosen, Staff Administrasi dan Perpustakaan Program Studi Teknik Informatika Universitas Brawijaya.
3. Bapak Eriq M. Adams J. ST,M.Kom dan Bapak Novanto Yudistira, S.Kom., M.Sc. selaku dosen pembimbing Penulis. Terima kasih atas semua bimbingan dan dorongan semangatnya.
4. Seluruh Dosen Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya yang telah membekali penulis dengan ilmu – ilmu yang bermanfaat.
5. Teman-temanku angkatan 2009 : Ika Kusumaning Putri S.Kom, Tierhy Rahman Azis S.Kom, Yusuf oktofani S.Kom, Himawan Aditya S.kom, Hoyi Ndadak Aji S.Kom, Antarangga Bhawika S.Kom, M Rizky Gus Sofyan S.Kom, Moh Halimi S.kom, M Rizky Aziz Kurnia S.Kom, Rizky Ananda Aditya S.Kom, Nizammudin

Ghonim S.Kom, Wildan Ryan S.Kom, Hafidz Rozaq S.Kom, Dwinta Ayu Pramita S.Kom, Dhimas Sawung S.Kom dll.

6. Teman Raion Studio : Arga Suwastika P.P S.Kom, Arik Ahmad S.Kom, Aang Samsudy S.Kom, Ika Kusumaning S.Kom, Fendy Gusta P. S.Kom, Grandis Mahendra S.kom, Andryanto S.Kom dll.
7. Serta semua pihak yang namanya tidak bisa penulis sebutkan satu persatu. Terima kasih atas do'a dan dukungannya.

Penyusun sadar bahwa masih banyak kesalahan dan kekurangan dalam penyusunan Skripsi ini, untuk itu penyusun mohon maaf dan mengharapkan kritik dan saran guna penyempurnaan selanjutnya.

Malang, 12 Januari 2014

Penulis

repository.ub.ac.id

ABSTRAK

Hanas Subakti. 2013. Rancang Bangun NPC (*Non-Player Characters*) Pada *Game* Bergenre *Tower Defense*. Skripsi Program Studi Informatika/ Ilmu Komputer, Program Teknologi Informasi dan Ilmu Komputer, Universitas Brawijaya. Pembimbing: Eriq M.Adams J, ST., M.Kom dan Novanto Yudistira, S.Kom., M.Sc.

Tower Defense merupakan *game* yang bertipe *Real Time Strategy*. Game *Tower Defense* mempunyai dua objek NPC yaitu *Enemy* yang bertugas menyerang daerah pertahanan pemain dan *Tower* untuk menghadang *Enemy*. Dalam membuat *game* bertipe *Real Time Strategy* faktor yang mempengaruhi ketertarikan *user* dalam memainkan *game* ini yaitu NPC. Oleh karena itu, perlu diperhatikan proses dalam membangun dan merancang NPC dalam *game Tower Defense*. Dalam penelitian ini, perancangan NPC pada *game Tower Defense* meliputi analisa *gameplay*, identifikasi NPC, Identifikasi *behaviour* NPC, dan merancang implementasi kecerdasan buatan pada NPC. Implementasi *behaviour* pada NPC *Enemy* menggunakan *pathfinding A Star* untuk mencari jalur terpendek yang dilewati menuju titik akhir. Pada NPC *Tower behaviour chasing* menggunakan teknik *Line of Sight* dan *behaviour state machine* menggunakan *Finite State Machine* agar *Tower* dapat mencari *Enemy* yang berada dalam jarak jangkaunya dan meningkatkan kemampuan dan kekuatan *Tower*. Pengujian *frame rate* dan *Memory* yang dibutuhkan pada 10 uji coba dalam 2 skenario, membuktikan NPC telah berjalan sesuai dengan perancangan dengan performa optimal. Penelitian ini akan memberikan panduan kepada pengembang *game* pemula untuk membangun NPC pada *game Tower Defense* pada *mobile game.*

Kata Kunci: *Tower, Enemy, Defense,* NPC, *behaviour.*

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

ABSTRACT

Hanas Subakti. 2013. ***Design and Implementation of NPC (Non-Player Characters) In the Tower Defense game genre Program of Informatics Technology and Computer Science, Brawijaya University*****. Advisors*****: Eriq M.Adams J, ST., M.Kom** ***and*** **Novanto Yudistira, S.Kom., M.Sc.**

Tower Defense genre is one of type Real Time Strategy games. Tower Defense games have two objects (NPC) that's called Enemy in charge of attacking the player's Defense and Tower that serves to block the Enemy. In making of Real Time Strategy game, one of the reasons that affect user's interest in playing this game is it's NPCs. Therefore, there are several things to consider in building and designing process of the NPC in the game Tower Defense. The design of this research includes the analysis of gameplay NPC, NPC identification, NPC behavior identification, and designing implementation Artificial Intelligence for NPC. Behaviour implementation on Enemy NPC using A star pathfinding, so Enemy can find the shortest path that is passed to reach the end point. At the Tower behaviour, for chasing behaviour using Line of Sight techniques and for state machine behaviour using Finite State Machine for Tower seeking Enemies within its range and to improve the capability and strength of Tower. Testing frame rate and memory usage required in 10 trials at 2 scenarios, prove that the NPCs has been run in accordance with the design optimally. This research will providing guidance to beginner game developers how to build games Tower Defense NPC in the mobile game.

Keywords**:** *Tower, Enemy, Defense,* NPC*, behaviour.*

UNIVERSITAS BRAWIJAYA

DAFTAR ISI

PRAKATA i
ABSTRAK iii
ABSTRACT iv
DAFTAR ISI v
DAFTAR GAMBAR viii
DAFTAR TABEL x
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Masalah 2
1.4 Tujuan 2
1.5 Manfaat 3
1.6 Sistematika Penulisan 3
BAB II DASAR TEORI 5
2.1 *Tower Defense* 5
2.2 Kecerdasan Buatan pada *Game* 6
2.3 *Non-Player Character* 9
2.4 *Movement* and *Pathfinding Algorithm* 9
2.5 Parameter Pengujian 11
2.5.1 *Frame per Second* (FPS) 11
2.5.2 Penggunaan *Memory (Memory Usage*) 11
BAB III METODOLOGI PENELITIAN DAN PERANCANGAN 12
3.1 Metode Penelitian 12
3.2 Perancangan 13

3.2.1 Analisa *Gameplay Tower Defense* 13
3.2.2 Identifikasi NPC *Tower Defense* 15
3.2.3 Identifikasi *Behaviour* NPC *Tower Defense* 16
3.2.4 Perancangan Implementasi *Behaviour* NPC *Tower Defense* 18
3.2.5 Perancangan Peta *Game Tower Defense* 26
3.3 Implementasi NPC *Tower Defense* 27
3.3.1 Penentuan Spesifikasi 28
3.3.2 Implementasi Peta Permainan *Tower Defense* 28
3.3.3 Implementasi Algoritma Pada NPC *Tower Defense* 29
3.4 Pengujian dan Analisis Kinerja Algoritma yang Diterapkan NPC Dalam Permainan *Tower Defense* 29
3.4.1 Menentukan Skenario Pengujian 29
3.4.2 Melakukan Pengujian 30
3.4.3 Menganalisis Hasil Pengujian 30
BAB IV IMPLEMENTASI 31
4.1 Penentuan Spesifikasi 31
4.1.1 Spesifikasi Perangkat Keras 31
4.1.2 Spesifikasi Perangkat Lunak 32
4.2 Implementasi Peta *Game Tower Defense* 32
4.2.1 Implementasi *Dataset* Peta 32
4.2.2 Implementasi *Enemy* 33
4.2.3 Implementasi *Tower* 34
4.3 Implementasi Algoritma Pada NPC *Tower Defense* 35
4.3.1 Implementasi *Behaviour Enemy* 35
4.3.2 Implementasi *Behaviour Tower* 39

BAB V PENGUJIAN DAN ANALISIS 45
5.1 Menentukan Skenario Pengujian 45
5.1.1 Pengujian Skenario 1 45
5.1.2 Pengujian Skenario 2 48
5.2 Analisis Hasil Pengujian 50
BAB VI PENUTUP 53
6.1 Kesimpulan 53
6.2 Saran 53
DAFTAR PUSTAKA DP-1

DAFTAR GAMBAR

Gambar 2. 1 *Flash Element Tower Defense* 6
Gambar 2. 2 *Desktop Tower Defense* 6
Gambar 2. 3 Berbagai Penjelasan Kecerdasan Buatan 8
Gambar 2. 4 Model Kecerdasan Buatan pada *Game* 8
Gambar 2. 5 Hirarki gerak perilaku 9
Gambar 3. 1 Diagram Alir Runtutan Metode Penelitian 12
Gambar 3. 2 Diagram Alir Perancangan Implementasi 13
Gambar 3. 3 Tampilan awal permainan *Desktop Tower Defense* 14
Gambar 3. 4 Tampilan saat permainan *Desktop Tower Defense* berjalan 15
Gambar 3. 5 Diagram Alir Implementasi *Behaviour Enemy* 19
Gambar 3. 6 Diagram alir algoritma *A** 23
Gambar 3. 7 Agen mengikuti jalur tujuan 23
Gambar 3. 8 Diagram Alir *Line Of Sight* 25
Gambar 3. 9 Rancangan *Dataset* Simulasi Permainan 26
Gambar 3. 10 Diagram alir Implementasi 28
Gambar 3. 11 Diagram alir Pengujian dan Analisis 29
Gambar 4. 1 Rancangan *Dataset* Permainan Keseluruhan 32
Gambar 4. 2 Implementasi *Dataset* Peta Permainan 33
Gambar 4. 3 Implementasi Peta dalam *array* 2D 33
Gambar 4. 4 Implementasi *Enemy* berupa *Spritesheet* 34
Gambar 4. 5 Implementasi *Enemy* dalam peta 34
Gambar 4. 6 Implementasi *Tower* dalam peta 35
Gambar 4. 7 Implementasi Algoritma *A** 37
Gambar 4. 8 *Pseudocode Method* TemukanNodeTerkecil 38
Gambar 4. 9 *Pseudocode Method* BuatFinalPath 39
Gambar 4. 10 *Pseudocode Line Of Sight* 41
Gambar 4. 11 *Pseudocode Method* MenghadapTarget 42
Gambar 4. 12 Implementasi *Finite State Machine* 43

Gambar 5. 1 Contoh Pengujian Skenario 1 46
Gambar 5. 2 Grafik Hasil Pengujian FPS Skenario 1 47
Gambar 5. 3 Grafik Hasil Pengujian *Memory* Skenario 1 47
Gambar 5. 4 Grafik Hasil Pengujian FPS Skenario 2 49
Gambar 5. 5 Grafik Hasil Pengujian *Memory* Skenario 2 49

DAFTAR TABEL

Tabel 3. 1 Identifikasi *Behaviour* Objek NPC 16
Tabel 3. 2 Perancangan Agen 27
Tabel 4. 1 Spesifikasi Perangkat Keras *Device* 31
Tabel 4. 2 Spesifikasi Perangkat Lunak *Device* 32
Tabel 4. 3 Kelas Musuh 35
Tabel 4. 4 Kelas *Tower* 39
Tabel 5. 1 Tabel Hasil Pengujian Skenario 1 46
Tabel 5. 2 Tabel hasil pengujian Skenario 2 48

BAB I
PENDAHULUAN

1.1 Latar Belakang

Game Tower Defense adalah *game* yang bertipe *Real Time Strategy* yang berfokus pada pengaturan sumber daya yang dimiliki pemain dan penempatan menara pertahanan (*Tower*). Dalam bentuk yang paling sederhana, sebuah *game Tower Defense* terdiri dari pemain yang bertugas membeli dan mengorganisir *Tower* yang akan menembak berbagai jenis musuh (*Enemy*) yang menyerang pertahanan pemain. [AVE-11:1]. Karena *game Tower Defense* bertipe *real time strategy* maka salah satu hal yang membuat *game* ini menantang dan adiktif yaitu NPC yang ada dalam *game* ini. Oleh karena itu perlu diperhatikan oleh *developer game* cara merancang NPC yang benar dalam pembuatan *Non-Player Characters* (NPC) *game Tower Defense*. Secara umum NPC pada *game Tower Defense* adalah *Tower* dan *Enemy*.

Seiring dengan terus berkembangnya industri *game* terutama *Tower Defense* pada *mobile games*, para pengembang *game* profesional berlomba-lomba untuk mengembangkan NPC pada *Tower Defense* untuk menarik lebih banyak penggunanya. Berbeda halnya dengan developer *game* pemula, mereka tidak mengetahui bagaimana merancang NPC serta memilih teknik *Artificial Intelligence* yang sesuai dengan *behaviour* NPC pada *game* bertipe *Tower Defense* pada *platform mobile.* Karena *platform* yang digunakan adalah *mobile game,* pengembang *game* harus memperhitungkan performa *game* tersebut supaya dapat berjalan lancar sesuai kemampuan *mobile device*.

Artificial Intelligence adalah teknik yang digunakan dalam permainan komputer dan permainan video yang menerapkan tiruan dari kecerdasan manusia terhadap perilaku dari NPC [MIL-09:4]. Dalam membangun sebuah *game Artificial Intelligence* ada beberapa langkah yang harus dilakukan. Hal pertama yang dilakukan adalah mengevaluasi perilaku NPC dalam *game* tersebut. Ketika

perilaku NPC sudah diketahui, langkah selanjutnya kita menentukan teknik *Artificial Intelligence* yang sesuai untuk diimplementasikan pada NPC [MIL-09].

Penelitian ini akan membahas bagaimana membangun objek NPC pada *game Tower Defense* dalam *platform mobile game*. Diharapkan penelitian ini dapat memberikan panduan kepada pengembang *game* pemula untuk membangun NPC pada *game Tower Defense* pada *mobile game* secara optimal dan sesuai perancangan.

1.2 Rumusan Masalah

Berdasarkan latar belakang diatas, penulis merumuskan beberapa masalah sebagai berikut :

1. Bagaimana merancang NPC pada *Game* Bergenre *Tower Defense*?
2 Bagaimana mengimplementasikan NPC pada *Game* Bergenre *Tower Defense*?
3 Bagaimana menguji NPC pada *Game* Bergenre *Tower Defense*?

1.3 Batasan Masalah

Agar permasalahan yang dirumuskan lebih terfokus, maka penelitian ini dibatasi oleh hal – hal sebagai berikut :

1. *Game* yang dibangun merupakan *game* dua dimensi (2D).
2. Jenis *game* adalah *Tower Defense*.
3. Menggunakan bahasa pemrograman c#.
4. Menggunakan *framework* XNA 4.0.
5. Perancangan dan implementasi *game* hanya dikhususkan pada OS *Windows phone.*
6. *Game* dapat diimplementasikan pada perangkat *mobile* dengan spesifikasi *full touch* dan ukuran layar 800 x 480 piksel.

1.4 Tujuan

Berdasarkan pada rumusan masalah yang telah dijelaskan, maka tujuan yang ingin dicapai dari skripsi ini adalah Membangun NPC pada *Game* Bergenre *Tower Defense*.

1.5 Manfaat

Manfaat yang dapat diberikan dari skripsi ini antara lain:

1. Bagi penulis:
   a. Mengaplikasikan ilmu yang didapat selama mengikuti perkuliahan di Informatika Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya.
   b. Mengembangkan penggunaan kecerdasan buatan dalam NPC pada *Game* Bergenre *Tower Defense*
2. Membantu pengembang *game* menentukan teknik kecerdasan buatan yang sesuai untuk *game* yang mempunyai *genre Tower Defense*.

1.6 Sistematika Penulisan

Sistematika isi dan penulisan dalam skripsi ini antara lain :

Bab I Pendahuluan

Berisi tentang latar belakang masalah, perumusan masalah dan pokok-pokok bahasan, tujuan dan manfaat dari penelitian serta sistematika pembahasan.

Bab II Dasar Teori

Bab ini berisi tentang teori dasar *Tower defense*, kecerdasan buatan dalam *game*, NPC, algoritma *movement* dan *pathfinding* dan parameter pengujian yang berisi *Frames Per Second* dan penggunaan *Memory*.

Bab III Metodologi Penelitian dan Perancangan

Bab ini berisi tentang gambaran umum langkah – langkah dalam pembuatan NPC pada *Game Tower Defense* dimulai dari tahap perancangan dan implementasi serta pengujian secara umum.

Bab IV Implementasi

Membahas tentang implementasi *behaviour* pada NPC yang ada dalam *game Tower Defense.*

Bab V **Pengujian dan Analisis**

Memuat hasil pengujian dan analisis terhadap NPC yang telah diimplementasikan.

Bab VI **Penutup**

Memuat kesimpulan yang diperoleh dari pembuatan dan pengujian NPC, serta saran – saran untuk pengembangan lebih lanjut.

BAB II
DASAR TEORI

Bab ini membahas dasar teori yang digunakan untuk menunjang penulisan skripsi mengenai Rancang Bangun *Non-Player characters* (NPC) Pada *Game Bergenre Tower Defense*. Beberapa dasar teori yang dimaksud adalah *Tower Defense*, kecerdasan buatan pada *game*, NPC, algoritma *Movement* dan parameter pengujian.

2.1 *Tower Defense*

Tower Defense game adalah *game* yang bertipe *Real Time Strategy* yang berfokus pada pengaturan sumber daya yang dimiliki pemain dan penempatan menara pertahanan (*Tower*). Dalam bentuk yang paling sederhana, sebuah *game Tower Defense* terdiri dari pemain yang bertugas membeli dan mengorganisir *Tower* yang akan menembak berbagai jenis musuh (*Enemy*) yang menyerang pertahanan pemain. Untuk setiap *Enemy* dihancurkan oleh *Tower*, pemain mendapatkan sumber daya yang sesuai dengan kesulitan *Enemy* [AVE-11:1]. Banyak *game*, seperti *Flash Element Tower Defense*, *Enemy* yang datang berjalan melalui jalur yang sudah disediakan, yang hanya memperbolehkan pemain untuk menyusun strategi yang optimal dan efektif dalam peletakan *Tower* di luar jalur *Enemy* dalam mengalahkan *Enemy*. Namun, beberapa versi dari tipe *game Tower Defense* memperbolehkan pemain untuk membuat labirin mereka sendiri dengan cara mengatur peletakan *Tower*, seperti *Desktop Tower Defense* [AVE-11]. Tampilan gambar dari *Flash Element Tower Defense* dan *Desktop Tower Defense* ditunjukkan pada gambar 2.1 dan 2.2.

Jika semua *Enemy* yang menyerang telah dihancurkan, maka pemain berhasil memenangkan ronde tersebut. Namun apabila ada *Enemy* yang berhasil mencapai akhir pertahanan pemain, *Enemy* menang. Pengelolaan *Tower* biasanya meliputi pembelian, peletakan dan *upgrade Tower* yang menembak *Enemy* secara otomatis [AVE-11:1]. Dalam penelitian ini jenis *Tower Defense* yang digunakan adalah *Desktop Tower Defense.*

Gambar 2. 1 ***Flash Element Tower Defense***
Sumber : [WIK-13]

Gambar 2. 2 ***Desktop Tower Defense***
Sumber : [WIK-13]

2.2 Kecerdasan Buatan pada *Game*

Kecerdasan memiliki beragam jenis dan derajat yang dialami oleh manusia, sebagian besar binatang dan beberapa mesin [MCJ-07]. Kecerdasan buatan adalah membuat komputer dapat melakukan tugas berpikir seperti yang

dapat dilakukan oleh hewan dan manusia [MIL-09:04]. Kecerdasan buatan merupakan sains dan teknik untuk membuat sebuah mesin yang cerdas, khusunya program komputer cerdas [MCJ-07]. Secara historis, terdapat beberapa pendekatan mengenai arti dari kecerdasan buatan seperti dijabarkan dalam Gambar 2.3 [RUS-10].

Kecerdasan buatan diaplikasikan ke berbagai macam disiplin ilmu, seperti klasifikasi *heuristic*, sistem pakar, *computer vision*, pengenalan suara, dan *game* [MCJ-07]. Kecerdasan buatan pada *game* memiliki sebuah model. Model tersebut terbagi menjadi tiga bagian: pergerakan, pengambilan keputusan dan strategi seperti yang digambarkan dalam Gambar 2.4 [MIL-09]. Kecerdasan buatan yang meliputi pergerakan (*movement*) dan pengambilan keputusan (*decision making*) menggunakan algoritma yang berkerja pada basis per-karakter, dan sisanya digunakan secara menyeluruh [MIL-09:09].

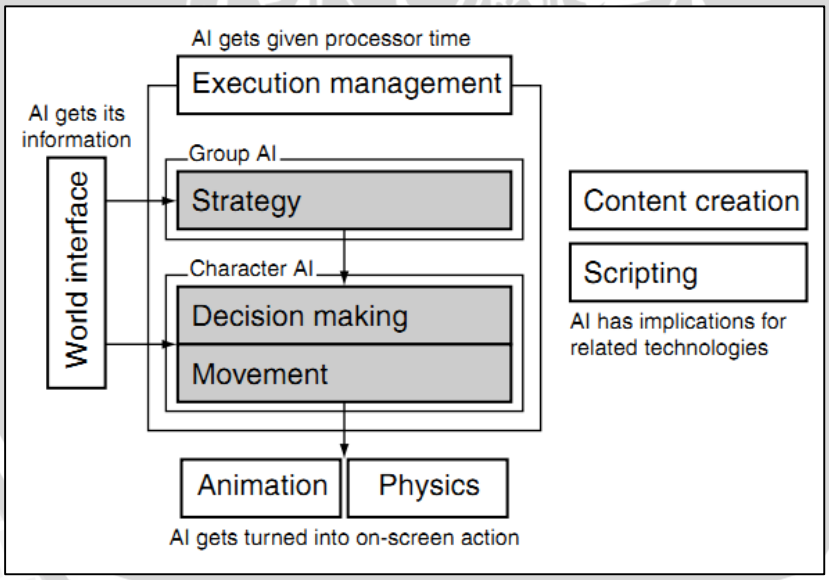
Kecerdasan buatan pada *game* untuk bertarung di semua area di peta, dibutuhkan pada *game* masa kini untuk mendukung pemain yang bertarung, pasukan yang diperintah oleh pemain, dan pertarungan *multiplayer* [DAL-09]. Tugas kecerdasan buatan tersebut harus dirancang semirip mungkin dengan manusia sehingga diharuskan memenuhi taktik dan hukum militer secara otentik, meskipun hal tersebut komplek dan berada di lingkungan dinamis [DAL-09].

Implementasi dari kecerdasan buatan pada *game* umumnya diimplentasikan pada *non-player character*, dan salah satu dari displin yang diteliti adalah pencarian jalur terpendek [JIE-11]. Menurut Kuiper, pencarian jalur di dunia nyata, jenis algoritma apapun yang digunakan, algoritma akan terbebani oleh proses perangkat keras untuk sensor [CHE-09]. Secara relatif, dunia virtual secara keseluruhan telah dirancang dan disimpan oleh komputer yang tidak membutuhkan pendeteksian melalui sensor [CHE-09].

Berpikir Manusiawi	**Berpikir Rasional**
"Usaha yang menarik untuk membuat komputer berpikir . . . *mesin dengan pikiran,* dalam arti sebenarnya." (Haugeland, 1985)	**"**Sebuah studi mengenai kecakapan mental melalui penggunaan model komputasi." (Charniak dan McDermott, 1985)

"[Otomatisasi dari] aktivitas yang berhubungan dengan pemikiran manusia, aktivitas seperti pengambilan keputusan, penyelesaian masalah, belajar . . ." (Hellman, 1978)	"Studi mengenai komputasi yang membuat mungkin untuk melihat, memberikan alasan, dan bertindak." (Winston, 1992)
Bertindak Manusiawi "Seni menciptakan yang melakukan fungsi intelegensi apabila dilakukan oleh manusia" (Kurzweil, 1990) "Studi tentang bagaimana membuat komputer melakukan sesuatu yang, saat ini, manusia melakukannya lebih baik." (Rich dan Knight, 1991)	**Bertindak Rasional** "*Computational Intelligence* merupakan studi untuk mendesain agen cerdas." (Poole, *et al*, 1998) "Kecerdasan buatan . . . berkaitan dengan perilaku cerdas dalam sebuah model." (Nilsson, 1998)

Gambar 2. 3 Berbagai Penjelasan Kecerdasan Buatan
Sumber : [RUS-10]



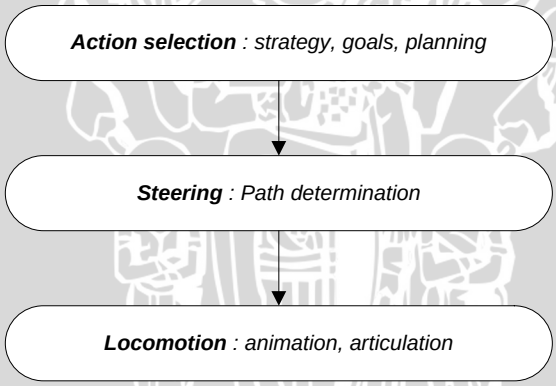
Gambar 2. 4 Model Kecerdasan Buatan pada *Game*
Sumber : [MIL-09:09]

Pada penelitian ini kecerdasan buatan akan diimplementasikan ke dalam NPC untuk memberikan *behaviour* pada NPC yang akan dirancang.

2.3 *Non-Player Character*

Autonomous character adalah jenis *otonomous agent* yang ditujukan untuk penggunaan komputer animasi dan media interaktif seperti *game*s dan *virtual reality*. Agen ini merepresentasikan sebuah karakter dalam cerita atau *game* dan mempunyai beberapa kemampuan untuk menentukan gerak mereka sendiri. Hal ini berbeda dengan karakter dalam sebuah animasi film yang tindakannya sudah diatur dalam skrip, dan juga berbeda dengan *"avatar"* dalam sebuah *game* atau *virtual reality* yang tindakannya langsung secara *real time* dilakukan oleh pemain. Di dalam *game*, *autonomous* karakter sering disebut dengan *Non-Player Characters*.

Perilaku karakter yang otonom dapat lebih baik dipahami dengan membaginya menjadi beberapa lapisan. Lapisan ini dimaksudkan hanya untuk kejelasan dan kekhususan dalam diskusi yang akan mengikuti. Gambar 2.5 menunjukkan sebuah divisi gerak perilaku otonom hirarki karakter menjadi tiga lapisan: seleksi tindakan, *steering*, dan penggerak [MIY-10:2].



Gambar 2. 5 Hirarki gerak perilaku
Sumber : [MIY-10:2]

Pada penelitian ini lebih difokuskan pada bagian *Action Selection* dan juga *Steering* yang di dalamnya berisi strategi pergerakan *Non-Player Characters.*

2.4 *Movement* and *Pathfinding Algorithm*

Movement atau sering kita sebut pergerakan adalah algoritma yang mengubah keputusan menjadi beberapa jenis gerakan. Ketika seorang karakter musuh tanpa senjata akan menyerang pemain dalam *game* Super Mario *Sunshine*

[Nintendo *Enter-tainment, Analysis and Development*, 2002], pertama-tama musuh menghadap langsung ke arah pemain. Ketika sudah cukup dekat, musuh itu bisa langsung menyerang pemain. Keputusan untuk menyerang membawa satu set algoritma *movement* yang mempunyai tujuan di lokasi dari pemain. Apabila kemudian animasi menyerang dijalankan akan membuat nyawa pemain berkurang. Algoritma *Movement* bisa menjadi lebih kompleks bukan hanya sekedar menuju ke suatu tempat saja. Karakter *movement* harus menghindari halangan yang ada di jalan atau melakukan pekerjaannya ketika melewati sebuah ruang. Pengawal dalam beberapa *level* di *Splinter Cell*[Ubisoft Montreal *Studios*, 2002] akan merespon pemain dengan mendengar dimana alarm bunyi berasal. Hal ini membutuhkan sistem navigasi nearest *wall-mounted* pada letak alarm yang bisa menjadi jarak yang jauh dan mungkin mempunyai navigasi yang kompleks untuk menghindari halangan dalam melewati koridor-koridor.

Banyak dari gerakan gerakan dalam *game* yang mempunyai animasi secara langsung. Misalnya saja *Sims*,di dalam *game The Sims*, sedang duduk dengan meja dan makanan di depannya dan ingin menjalankan animasi makan, dan animasi makan dengan mudah dijalankan. Sekali kecerdasan buatan telah memutuskan bahwa karakter harus makan, tidak ada lagi kecerdasan buatan yang dibutuhkan [MIL-09:10].

Karakter pada sebuah *game* biasanya perlu untuk berjalan di sekitar area level mereka. Terkadang pergerakan mereka sudah di set oleh *programmer* dan sebagian lagi dihasilkan oleh algoritma *Pathfinding*. *Pathfinding* berada di antara AI movement dan juga *Decision Making*. Sebagian besar *game* menggunakan solusi *pathfinding* berdasar sebuah algoritma yang disebut *A**(*Star*) [MIL-09:198]. Dalam penelitian ini Algoritma *A** akan diimplementasikan dalam NPC untuk *pathfinding behaviour*.

2.5 Parameter Pengujian

Penelitian ini menggunakan 2 parameter dalam pengujian yaitu *Frame per second* dan penggunaan *Memory*.

2.5.1 *Frame per Second* (FPS)

Frame per second (FPS) pada tampilan simulasi adalah jumlah gambar fragmen, gambar-gambar berbeda ditampilkan tiap detik untuk mengelabui mata penonton menjadi objek yang bergerak [ALD-09]. FPS biasa disebut juga *frame rate*, FPS merupakan sebuah metrik fluiditas [ALD-09]. Acara pada TV NTSC disiarkan pada 30 FPS, sedangkan PAL adalah 25 FPS, dan film pada bioskop ditampilkan pada 24 FPS [ALD-09].

Kapasitas 20 FPS dianggap oleh sebagian besar pihak sebagai batas minimum untuk mengelabui mata [ALD-09]. *Frame rate* yang lebih tinggi (berkisar antara 60 FPS) menghasilkan animasi yang lebih memanjakan, dan beberapa konsol permainan mengorbankan elemen lain untuk menjaga *level* ini [ALD-09].

FPS yang lebih rendah disebabkan oleh keterbatasan sistem, pemrograman yang buruk, dan aplikasi lain yang menghabiskan sumber daya sistem (seperti kecerdasan buatan) [ALD-09]. *Detail* dari gambar, seperti resolusi layar, kompleksitas objek, dan pencahayaan juga menurunkan jumlah FPS [ALD-09].

2.5.2 Penggunaan *Memory (Memory Usage*)

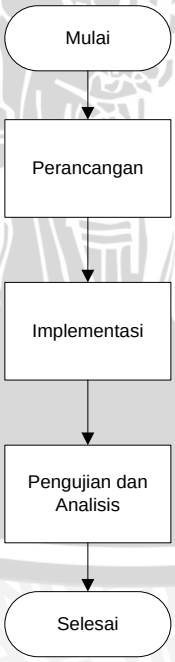
Penggunaan *memory* yang dimaksud adalah berapa banyak dari *memory* perangkat yang dipakai untuk menjalankan aplikasi. Jika kita ingin membuat aplikasi *windows phone* yang bagus, maka kita harus meluangkan waktu untuk meningkatkan kualitas dan performa. Walaupun kita sudah membuat ide yang besar untuk sebuah aplikasi, pengguna akan berhenti menggunakan aplikasi tersebut dan akan memberikan review negatif karena aplikasi yang kita buat lambat, kurang respon atau bahkan *crash* sewaktu-waktu. Untuk *device windows phone*, kita dapat mengetahui *memory* yang digunakan menggunakan *Windows Phone Application Analysis. Windows Phone Application Analysis* berfungsi memonitoring, mengevaluasi dan meningkatkan kualitas dan performa dari aplikasi *windows phone* yang telah dibuat[MIC-13].

BAB III
METODOLOGI PENELITIAN DAN PERANCANGAN

Bab ini menjelaskan langkah-langkah metodologi penelitian dan perancangan. Dalam metode penelitian akan dijabarkan mengenai metode atau langkah-langkah yang digunakan dalam penelitian skripsi. Dalam subbab perancangan menjelaskan perancangan implementasi dari penelitian ini

3.1 Metode Penelitian

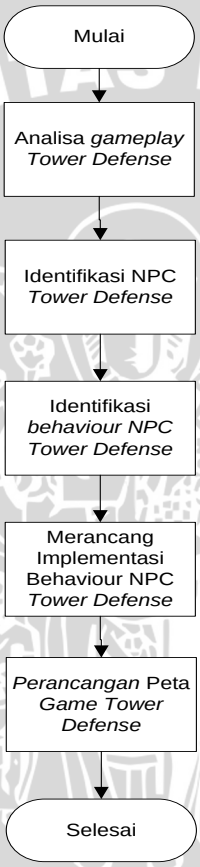
Pada Subbab ini menjelaskan langkah-langkah metodologi penelitian. Penelitian ini akan dilakukan dalam beberapa tahap, dijabarkan pada diagram alir dalam Gambar 3.1 yaitu melakukan perancangan untuk melakukan pendekatan tentang NPC yang akan dibangun. Setelah itu mengimplementasikan hasil dari perancangan terhadap NPC dan yang terakhir adalah pengujian yang bertujuan untuk membuktikan apakah alur pembuatan NPC yang dilakukan dalam penelitian berjalan dengan baik dan implementasi NPC berjalan sesuai dengan perancangan serta optimal.



Gambar 3. 1 Diagram Alir Runtutan Metode Penelitian
Sumber: Metode Penelitian

3.2 Perancangan

Bab perancangan terdiri dari lima tahap yaitu menganalisa *gameplay Tower Defense*, mengidentifikasi agen NPC dalam *game Tower Defense*, menentukan teknik AI yang tepat berdasarkan *behaviour* NPC *game Tower Defense*, implementasi AI *behaviour* pada NPC dan merancang *dataset game Tower Defense*. Tahap-tahap perancangan dijabarkan dalam diagram alir dalam Gambar 3.2.



Gambar 3. 2 Diagram Alir Perancangan Implementasi
Sumber: Metode Penelitian

3.2.1 Analisa *Gameplay Tower Defense*

Pada penelitian ini jenis *game Tower Defense* yang dianalisa adalah *Tower Defense* yang bertipe *Desktop Tower* [AVE-11]. Yang menjadi ciri khas dari *Desktop Tower Defense* adalah pada awal permainan dimulai tak ada satu pun penghalang di dalam peta. Untuk lebih jelasnya berikut ditampilkan ilustrasi peta permainan *Desktop Tower Defense* pada gambar 3.3.

Gambar 3. 3 Tampilan awal permainan *Desktop Tower Defense*
Sumber: [AVE-11]

Permainan dimulai ketika *Enemy* datang melalui gerbang awal dengan misinya untuk berjalan menuju ke gerbang akhir melewati peta permainan. Apabila *Enemy* telah sampai pada gerbang akhir maka *lives point* pemain akan dikurangi. Untuk mencegah *Enemy* sampai pada gerbang akhir pemain dapat membuat pertahanan dengan meletakkan *Tower* untuk menyerang *Enemy*. Setelah diletakkan *Tower* pada peta, *Enemy* akan mencari jalur terpendek untuk mencapai gerbang akhir. Permainan ini akan selesai apabila lives point dari pemain menjadi 0(nol). Gambar dari permainan *Desktop Tower Defense yang* ditunjukkan pada gambar 3.4.

Gambar 3. 4 Tampilan saat permainan *Desktop Tower Defense* berjalan
Sumber: [AVE-11]

3.2.2 Identifikasi NPC *Tower Defense*

Dari analisa *game*play *Desktop Tower Defense*, kita dapat mengetahui dalam permainan berjenis ini mempunyai 2 jenis NPC. Kedua NPC tersebut adalah *Enemy* dan *Tower*.

3.2.2.1 *Enemy*

Dalam permainan *Desktop Tower Defense*, *Enemy* bertugas untuk menyerang gerbang akhir dari pemain. *Enemy* harus mencari jalurnya apabila di peta ada *Tower* yang di pasang pemain untuk mengahadangnya. Apabila *Enemy* hancur sebelum sampai gerbang akhir maka pemain akan mendapat poin. Sebaliknya, kalau *Enemy* berhasil mencapai gerbang akhir maka *lives point* dari pemain akan dikurangi. Ada tiga properti yang dimiliki oleh semua *Enemy* dalam permainan ini. Yang pertama, *Enemy* mempunyai *lives point* untuk mengetahui jumlah nyawa yang dimiliki. Kemudian *Enemy* mempunyai letak jalur mana saja yang akan dilewati untuk menuju gerbang akhir dan mengalahkan pemain. Dan yang ketiga kecepatan dari *Enemy* bersifat konstan tanpa percepatan. Di dalam permainan *Desktop Tower Defense*, variasi *Enemy* hanya mencangkup pengaturan

ketiga komponen tersebut untuk membuat level permainan menjadi lebih menantang.

3.2.2.2 *Tower*

Tower bertugas menghadang jalan dari *Enemy*. *Tower* diletakkan di mana saja sesuai dengan pilihan pemain dalam mengatur strateginya. Tiap *Tower* dalam permainan ini mempunya beberapa properti umum. Diantaranya, setiap *Tower* memiliki jumlah peluru, kekuatan dan kecepatan peluru yang akan ditembakkan ke *Enemy*. *Tower* juga mempunyai target *Enemy* yang menjadi sasaran tembaknya. Selain itu, *Tower* juga mempunyai radius jangkauan tertentu, jadi hanya *Enemy* yang berada dalam jarak jangakaunya yang akan menjadi target. Untuk memasang *Tower* pada peta,pemain harus membeli *Tower* yang dipilih terlebih dahulu. Oleh karena itu, *Tower* mempunyai properti harga jual.

3.2.3 Identifikasi *Behaviour* NPC *Tower Defense*

Untuk menentukan dan mendesain teknik AI yang sesuai pada objek NPC hal pertama yang dilakukan adalah mengidentifikasi *behaviour* dari objek NPC tersebut. Penjelasan *Behaviour* NPC akan dijabarkan pada tabel 3.1.

Tabel 3. 1 Identifikasi *Behaviour* Objek NPC

No	Nama NPC	Kemampuan	Kebutuhan AI
1	*Enemy*	Mencari jalur terpendek melewati halangan *Tower* untuk menuju gerbang akhir dari permainan.	*Pathfinding*
2	*Tower*	Dapat berputar 360 derajat mengikuti posisi *Enemy* yang menjadi targetnya.	*Chasing*
		Dapat di-upgrade menjadi *Tower* yang mempunyai kapabilitas lebih tinggi daripada sebelumnya	*State Machine*

Sumber: Perancangan

3.2.3.1 *Enemy Behaviour*

Sama halnya dengan *pathfinding*, tujuan utama dari *Enemy* adalah menemuan jalur terbaik dari *vertex* awal dan *vertex* akhir. Secara umum algoritma

pathfinding digolongkan menjadi 2 jenis [RUS-10] yaitu algoritma *Uninformed search* dan juga algoritma *informed search*. Algoritma *Uninformed search* adalah algoritma yang tidak memiliki keterangan tentang jarak atau biaya dari *path* dan tidak memiliki pertimbangan akan *path* mana yang lebih baik contohnya adalah algoritma *breadth First Search*. Sedangkan algoritma *Informed search* memiliki keterangan tentang jarak atau biaya dari *path* dan memiliki pertimbangan berdasarkan pengetahuan akan *path* mana yang lebih baik contoh algoritma ini adalah algoritma *A** dan algoritma Djikstra.

Jika kita lihat dari *behaviour* yang dimiliki oleh *Enemy* maka algoritma *Informed search* adalah algoritma yang sesuai untuk di aplikasikan ke NPC *Enemy*. Namun algoritma *informed search* mempunyai dua algoritma dibawahnya yaitu *A** dan algoritma Djikstra. Perbedaan keduanya menurut Adi Wijaya dan Roi Gunawan [WIJ-01] antara lain:

1. Waktu pencarian menggunakan Algoritma *A** dalam menemukan rute lebih cepat daripada algoritma Djikstra.
2. Jumlah *loop A** lebih sedikit daripada Djikstra
3. Rute yang ditemukan berbeda tetapi mempunyai biaya yang sama.
4. Algoritma *A** lebih cocok untuk pengembangan *game* yang bersifat *real time*.

Berdasarkan hal tersebut diatas algoritma *A** adalah algoritma yang sesuai untuk NPC *Enemy*.

3.2.3.2 Tower Behaviour

Ada dua *behaviour* yang akan diimplementasikan ke dalam *Tower*. Yang pertama adalah kemapuan berputar 360 derajat sesuai letak *Enemy* yang menjadi target. *Behaviour* seperti ini dikelompokkan menjadi *Chasing behaviour*. Sedangkan metode yang sesuai yaitu menggunakan metode *Line Of Sight* [DMB-04]. *Line Of Sight* adalah istilah yang sering dijumpai pada war*game*s dan juga *role playing game*. *Line Of Sight* mengacu pada visibilitas (Apa yang dapat dilihat oleh *Tower*) di dalam peta permainan. Kebanyakan digunakan untuk mengalahkan musuh yang berada dalam garis radiusnya.

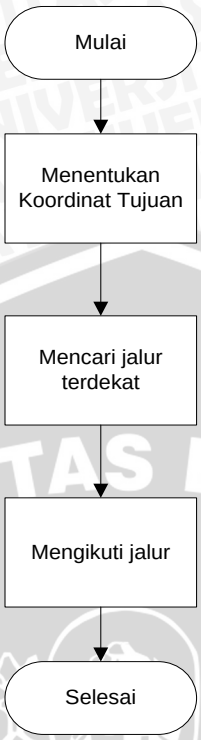
Behaviour yang berikutnya yang akan diimplementasikan ke *Tower* yaitu dapat ditingkatkan menjadi *Tower* yang mempunyai kapabilitas lebih tinggi daripada sebelumnya. *Tower* mempunyai *level/state* yang bermacam-macam. Misalkan saja, apabila *Tower* masih berada di tingkat/*level* 1 maka yang dijalankan adalah *state* ke-1. Begitupun juga jika *Tower* di-*upgrade* menjadi *level* 2, maka *Tower* akan menjalankan *behaviour* pada level ke-2. Berdasarkan kebutuhan behavior tersebut *Finite state Machine* adalah teknik AI yang sesuai. *Finite State Machine* pada dasarnya adalah melakukan pemecahan *behaviour* dari *object*/agen berdasarkan *state*-nya. Dan nantinya juga harus didefinisikan aturan-aturan transisi sehingga *state* dapat berubah dari yang satu ke yang lain degan mudah [DMB-04:228].

3.2.4 Perancangan Implementasi *Behaviour* NPC *Tower Defense*

Berdasarkan identifikasi *Behaviour* pada NPC akan digunakan 3 metode AI untuk membangun NPC pada *game* bertipe *Tower Defense* antara lain *A* pathfinding*, *Line Of Sight* dan *Finite State Machine*.

3.2.4.1 Perancangan *Behaviour Enemy*

Algoritma *A** memiliki tiga tahapan, mendapatkan koordinat titik tujuan, mencari jalur terdekat dan membuat jalurnya, dan mengikuti jalur yang sudah tercipta. Perancangan algoritma *pathfinding A** dalam NPC *game Tower Defense a*kan dijabarkan pada diagram alir gambar 3.5.



Gambar 3. 5 Diagram Alir Implementasi *Behaviour Enemy*
Sumber: Perancangan

Adapun langkah-langkah diagram alir implementasi *behaviour Enemy* akan dijabarkan sebagai berikut:

1. Mendapatkan Koordinat Titik Tujuan

   Di dalam penelitian ini teknik peta permainan yang digunakan adalah *Tile map*. Teknik ini memiliki versi yang bermacam-macam mulai dari dua dimensi maupun isometrik, tetapi teknik *Tile Map* sendiri pada dasarnya adalah sebuah teknik untuk menyusun gambar-gambar kecil yang disebut dengan *Tile* yang ditampilkan dengan urutan sedemikian rupa sehingga membentuk gambar yang utuh.

   Dalam *game Desktop Tower Defense* koordinat tujuan ditentukan oleh *game* desainer. Kebanyakan titik koordinat tujuan berada jauh dari koordinat awal *Enemy*. Dalam implementasi algoritma *A** setiap koordinat dari *tile* akan dijadikan *node* yang mempunyai nilai untuk menentukan apakah *node* tersebut adalah yang dapat dilewati oleh *Enemy* atau justru merupakan halangan.

2. Mencari Jalur Terdekat

Algoritma *A** menyelesaikan masalah yang menggunakan graf untuk perluasan ruang statusnya. Dengan kata lain algoritma ini digunakan untuk menyelesaikan permasalahan yang bisa direpresentasikan dengan graf. Dengan menerapkan suatu heuristik, algoritma ini membuang langkah-langkah yang tidak perlu dengan pertimbangan bahwa langkah-langkah yang dibuang sudah pasti merupakan langkah yang tidak akan mencapai solusi yang diinginkan.

Sebelum menjabarkan lebih lagi tentang kerja Algoritma *A**, akan dijelaskan beberapa kata kunci untuk memudahkan algoritma dengan lebih mudah.

1. Nilai *Greedy* G(n)

   Tiap *node* akan di asosiasikan dengan nilai G atau sering disebut nilai *Greedy*, nilai G menjelaskan seberapa jauh jarak yang ditempuh dari *node* awal ke *node* ini.

2. Fungsi *Heuristik* H(n)

   Fungsi ini digunakan untuk mengestimasi seberapa jauh yang akan dilalui untuk berpindah dari *node* satu ke *node* yang lainnya.

3. Nilai F(n)

   Tiap *Node* juga akan diasosiasikan dengan nilai F ini. Nilai F merupakan estimasi kasar dari seberapa jauh yang harus dilalui dari *node* awal menuju *node* akhir jika jalur yang akan dibuat melintasi *node* ini. Nilai F berasal dari penjumlahan antara nilai G dengan Heuristik.

4. *Open List*

   Merupakan *List* besar untuk menampung semua *node* yang mempunyai kemungkinan masuk ke dalam jalur *Final*.

5. Closed *List*

   Merupakan *list* yang menyimpan semua *node* yang telah diperiksa. *List* ini digunakan untuk memastikan bahwa *node* yang sudah diperiksa tidak diperiksa kembali.

6. *Node Parent*

   Setiap *node* akan mempunyai referensi/*parent* dari *node* yang menambahkan *node* tersebut ke dalam *Open List*. Saat algoritma telah selesai, kita akan menggunakan *Node Parent* dari *node* saat itu untuk mengikuti jejak jalur akhir dari smpul terakhir ke *node* awal.

Algoritma *A** membangkitkan *node* yang paling mendekati solusi. *Node* ini kemudian menyimpan *Node Parent* ke dalam *list* sesuai dengan urutan yang paling mendekati solusi terbaik. Kemudian, *node* pertama pada *list* diambil, dibangkitkan *Node Parent node* tersebut dan kemudian suksesor ini disimpan ke dalam *Open list* sesuai dengan urutan yang terbaik untuk menghasilkan solusi akhir.

Node pada *list* bisa berasal dari kedalaman berapapun dari graf. Algoritma ini akan mengunjungi secara mendalam (mirip *Depth First Search*) selama *node* tersebut merupakan *node* yang membuat bobot terkecil. Jika *node* yang sedang dikunjungi ternyata tidak mengarah kepada solusi yang diinginkan, maka akan melakukan runut balik ke arah *node parent* untuk mencari *node* anak lainnya yang lebih menjanjikan dari pada *node* yang terakhir dikunjungi. Bila tidak ada juga, maka akan terus mengulang mencari ke arah *node parent* sampai ditemukan *node* yang lebih baik untuk dibangkitkan suksesornya. Strategi ini berkebalikan dengan algoritma *Depth First Search* yang mencari sampai kedalaman yang terdalam sampai tidak ada lagi suksesor yang bisa dibangkitkan sebelum melakukan runut balik, dan algoritma *Depth First Search* tidak akan melakukan pencarian secara mendalam sebelum pencarian secara melebar selesai. *A** baru berhenti ketika

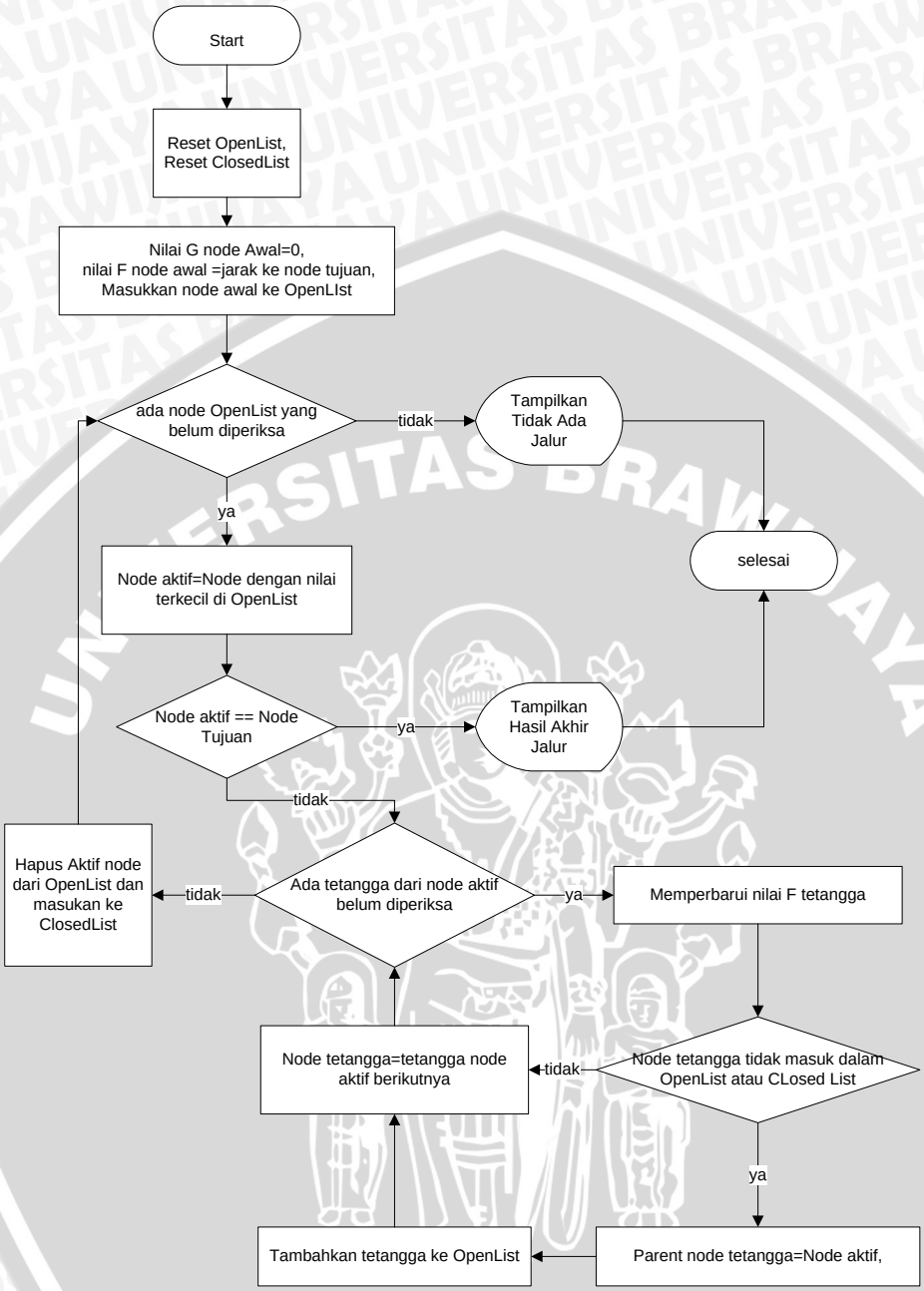
mendapatkan solusi yang dianggap solusi terbaik. Algoritma *A** menerapkan teknik *heuristik* dalam membantu penyelesaian persoalan. *Heuristik* adalah penilaian yang memberi harga pada tiap *node* yang memandu *A** mendapatkan solusi yang diinginkan. Dengan heuristik yang benar, maka *A** pasti akan mendapatkan solusi (jika memang ada solusinya) yang dicari. Cara bekerja dari algortima *A** akan diajabarkan dalam diagram alir pada gambar 3.6.

3. Mengikuti Jalur

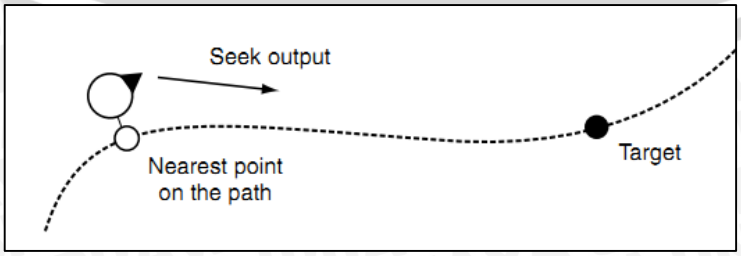
Jalur berupa kumpulan hubungan *node-node* yang sudah tercipta dari pencarian berdasarkan bobot terkecil, akan diikuti oleh AI Agen untuk menuju ke titik tujuan dari titik awal. Pencarian jalur merupakan sebuah perilaku yang didelegasikan [MIL-09:76]. Pencarian jalur *Enemy* akan menghitung jarak terdekat menuju titik akhir secara aktual berdasarkan letak *Tower* yang dibuat sebelum *Enemy* berada di peta maupun ketika *Enemy* sudah berada dalam peta.

Berdasarkan penghitungan tersebut, kemudian AI *Enemy* akan mencari tiap-tiap ujung *node*, yang disebut *corner*, yang sudah terbentuk. Proses pencarian *corner* oleh AI Agen disebut *seek*, dimana proses *seek* berusaha mengarahkan agen menuju titik tujuan [REY-13]. Titik tujuan yang dicapai bukan merupakan titik tujuan akhir, melainkan *corner* atau ujung dari *node*.

AI Agen akan berjalan menuju ke arah *corner* dan akan merubah tujuan ke *corner* berikutnya hingga mencapai titik tujuan akhir. Agen yang mengikuti jalur dijelaskan seperti dalam Gambar 3.7.



Gambar 3. 6 Diagram alir algoritma *A
Sumber: Perancangan



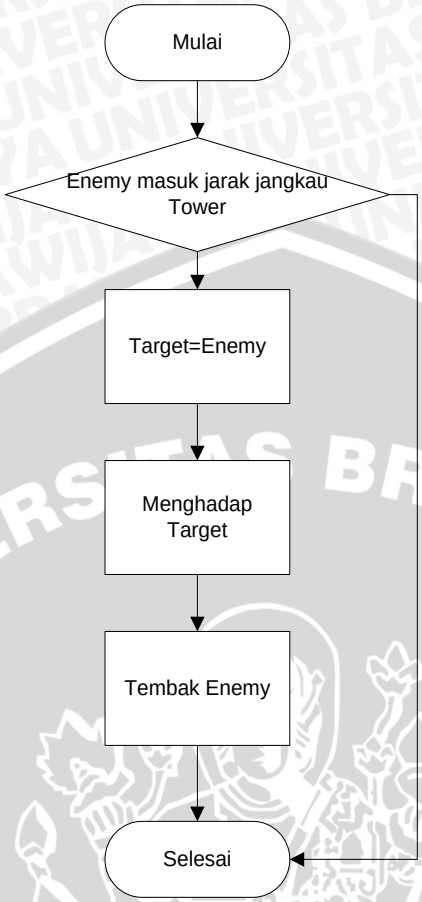
Gambar 3. 7 Agen mengikuti jalur tujuan
Sumber : [MIL-09]

3.2.4.2 Perancangan *Behaviour Tower*

Tower akan menggunakan gabungan dari dua jenis teknik AI, yaitu *Line Of Sight* untuk *Tower* menentukan target *Enemy* dan *Finite State Machine* untuk *meningkatkan* kemampuan *Tower*. Perancangan teknik AI pada *behaviour Tower* dijelaskan dibawah ini yaitu:

1. *Line Of Sight*

   Teknik *Line Of Sight* dalam *Tower* berfungsi untuk menentukan musuh yang menjadi target tembak *Tower*. Setiap *Tower* mempunyai jangkauan area masing-masing. Dengan menggunakan teknik *Line Of Sight Tower* akan selalu menghadap ke arah dimanapun *Enemy* yang masih dalam jangkauannya. Setelah menentukan *Enemy* yang menjadi target, *Tower* akan menembak tepat ke *Enemy*. Untuk menentukan sudut perputaran *Tower* diperoleh dari hasil tangen dari posisi koordinat X dan Y dari selisih letak antara *Enemy* dan *Tower*. Diagram alir *Line Of Sight* yang akan diterapkan ditunjukkan pada gambar 3.8.



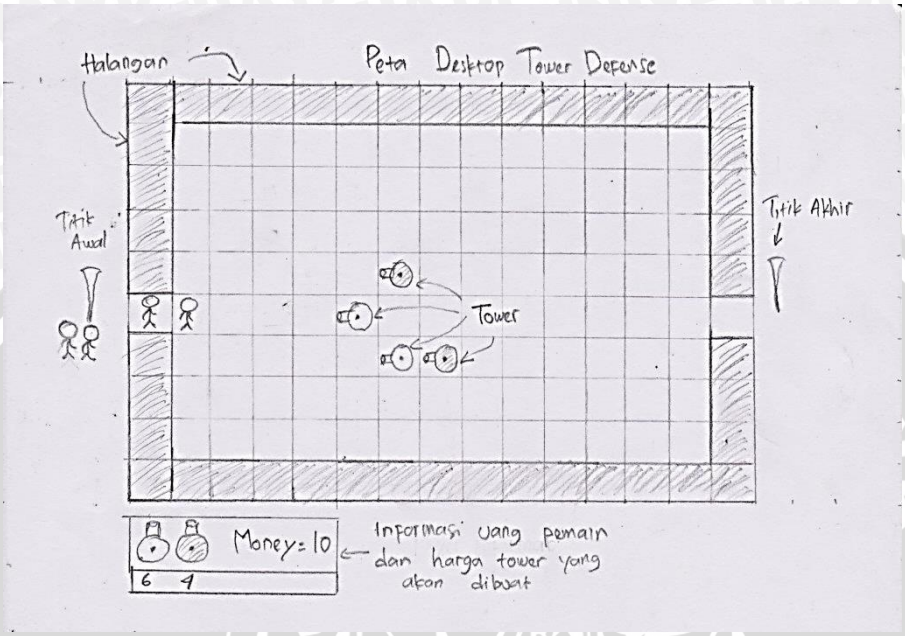
Gambar 3. 8 Diagram Alir *Line Of Sight*
Sumber: Perancangan

2. Finite State Machine

Sebuah *Finite State Machine* adalah sebuah mesin abstrak yang dapat berada di satu dari beberapa *state* yang berbeda dan yang sudah ditentukan. *Finite State Machine* juga dapat mendefinisikan sebuah kumpulan dari kondisi yang menentukan kapan *state* seharusnya berganti. *State* yang sebenarnya menentukan bagaimana perilaku *State* Machine[DMB-04:228]. *Tower* membutuhkan *Finite State Machine* untuk kemampuannya yang bisa ditingkatkan. Ketika *Tower* sudah dibangun di dalam peta, apabila pemain menginginkan *Tower* tersebut ditingkatkan kemampuannya maka *Tower* harus merubah tingkatan dari perilakunya. Dalam penelitian ini *Tower* mempunyai 2 *state*. Yaitu *state* ketika *Tower* awal dibuat dan yang berikutnya yaitu ketika *Tower* ditingkatkan kemampuannya.

3.2.5 Perancangan Peta *Game Tower Defense*

Rancangan untuk tampilan simulasi pada peta permainan ditunjukkan dalam Gambar 3.9. Simulasi pada peta permainan terdiri dari NPC *Enemy*, NPC *Tower*, titik awal dan titik tujuan.



Gambar 3. 9 Rancangan *Dataset* Simulasi Permainan
Sumber: Perancangan

3.2.5.1 *Dataset* Peta

Peta yang digunakan yaitu merupakan peta dengan tipe *Grid*, yaitu digambarkan dengan kotakan yang disusun sesuai koordinat. Halangan dalam peta ditentukan oleh *game* desain untuk mengatur *Enemy* tetap dalam peta sampai sebelum sampai pada titik akhir. Pada mulanya *Enemy* berjalan dari titik awal menuju titik tujuan tanpa hambatan. Sebelum akhirnya setelah pemain membuat *Tower* di dalam peta, maka *Tower* tersebut akan menjadi halangan bagi *Enemy*. Dikarenakan ada halangan maka *Enemy* harus membuat jalur baru untuk menghindari *Tower*. DI bagian kiri bawah juga terdapat informasi tentang uang yang dimiliki oleh pemain dan juga harga *Tower* yang akan dipilih.

3.2.5.2 *Dataset* Agen NPC

Perancangan dari agen *Non-Player Character* dijelaskan pada tabel 3.2.

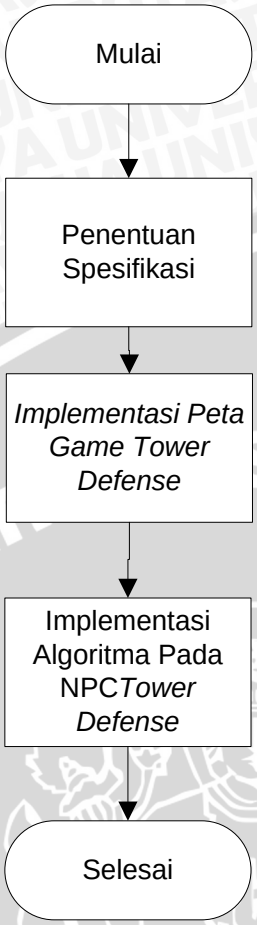
Tabel 3. 2 Perancangan Agen

No.	Nama	Rancangan Tampilan	Kemampuan
1.	*Enemy*		1. Berjalan ke semua arah 360°, namun tidak dapat melompat 2. Percepatan bernilai nol, dan kecepatan diatur oleh *user* 3. Memiliki *health bar*
2.	*Tower*		1. Moncong senjata dapat berputar 360° mengikuti posisi *Enemy* target. 2. Menembakkan peluru dengan rasio yang ditentukan. 3. Menembak AI Agen terdekat dan mengubah target apabila AI Agen tersebut sudah diluar jangkauan

Sumber: Perancangan

3.3 Implementasi NPC *Tower Defense*

Bab implementasi terdiri dari tiga tahap, yaitu spesifikasi sistem, implementasi *dataset*, implementasi agen yang akan menjalankan simulasi, yaitu AI *Tower* dan AI *Enemy*, dan implementasi algoritma dan penanaman algoritma pada agen *non-player character*. Tahap-tahap implementasi dijabarkan dalam diagram alir dalam Gambar 3.10.



Gambar 3. 10 Diagram alir Implementasi
Sumber: Metode Penelitian

3.3.1 Penentuan Spesifikasi

Subbab ini menjabarkan spesifikasi dari lingkungan sistem tempat simulasi dijalankan. Spesifikasi sistem yang dimaksud adalah lingkungan perangkat keras dan perangkat lunak yang digunakan dalam menjalankan simulasi. Subbab ini bertujuan untuk mengetahui dalam spesifikasi tersebut, seberapa baik simulasi dijalankan untuk pengolahan data selanjutnya.

3.3.2 Implementasi Peta Permainan *Tower Defense*

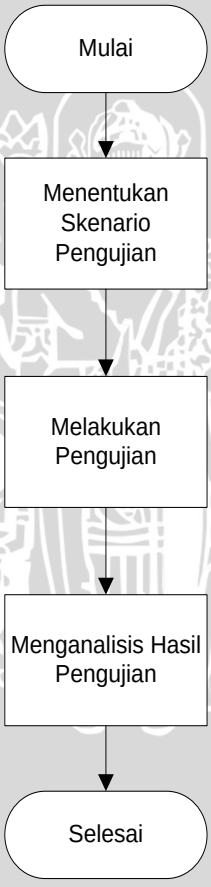
Pada Subbab ini akan diimplementasikan *dataset* yang akan digunakan. Meliputi gambar *environment* pada *game*, gambar NPC *Enemy* dan juga gambar NPC *Tower*.

3.3.3 Implementasi Algoritma Pada NPC *Tower Defense*

Subbab yang berisi penerapan dari algoritma yang telah dirancang dan akan dipakai dalam permainan *Tower Defense*. Metode-metode yang sudah dirancang dalam bab perancangan juga akan dijabarkan dalam subbab ini.

3.4 Pengujian dan Analisis Kinerja Algoritma yang Diterapkan NPC Dalam Permainan *Tower Defense*

Pengujian yang dilakukan berdasarkan implementasi yang telah dibuat berdasarkan metriks pengujian yang telah ditentukan, yaitu FPS dan pengujian penggunaan *Memory*. Alur implementasi dari pengujian dan analisis dijabarkan dalam diagram alir pada gambar 3.11.



Gambar 3. 11 Diagram alir Pengujian dan Analisis
Sumber: Metode Penelitian

3.4.1 Menentukan Skenario Pengujian

Skenario pengujian algoritma dilakukan dengan beberapa alternatif, yang pertama adalah pengujian performa dengan menggunakan metode *stress testing*

dengan mengubah jumlah *Enemy* dan *Tower* pada permainan untuk mengetahui tingkat stabilitas dari *game*, kedua memastikan bahwa *behaviour* yang dimiliki oleh *Enemy* sudah sesuai dengan yang sudah dirancang dengan cara mengubah titik awal dan juga titik akhir dari *game*. Dan yang terakhir memastikan bahwa *behaviour* yang dimiliki oleh *Tower* sudah sesuai dengan perancangan dengan cara meletakkan *Tower* di tempat yang dilewati oleh *Enemy*.

3.4.2 Melakukan Pengujian

Setiap simulasi akan ditampilkan hasil dari FPS dan juga penggunaan *Memory*. Untuk melihat hasil dari FPS dengan cara menampilkan nilai FPS di dalam simulasi. Untuk mengetahui banyak *memory* yang digunakan, *software* yang digunakan yaitu *Visual Studio 2012*. Setelah simulasi dijalankan maka akan ada analisis *memory* yang dipakai pada saat simulasi. Berdasarkan hasil tersebut akan dicatat dan dianalisa tingkat stabilitas dari implementasi *game* ini.

3.4.3 Menganalisis Hasil Pengujian

Hasil pengujian skenario pertama akan di analisis performa dari *game* menggunakan metode *stress testing* berdasar hasil FPS dari *game*. Sedangkan skenario kedua dan ketiga bertujuan untuk menguji *behaviour* pada masing - masing agen AI. Hasil dari tiap-tiap skenario akan dianalisis. Analisis dibutuhkan untuk Membangun NPC pada *Game* Bergenre *Tower Defense*. Gambar 3.4 menjabarkan agen NPC diuji pada *dataset* peta permainan dengan parameter pengujian yang telah ditentukan.

BAB IV
IMPLEMENTASI

Bab ini membahas mengenai implementasi algoritma berdasarkan hasil yang telah didapatkan dari analisis kebutuhan dan proses perancangan. Pembahasan terdiri dari penjelasan tentang spesifikasi sistem, batasan-batasan dalam implementasi, implementasi *dataset* dan implementasi Algoritma pada NPC.

4.1 Penentuan Spesifikasi

Hasil analisis kebutuhan dan perancangan perangkat lunak yang telah dijelaskan pada bab 4 menjadi acuan untuk melakukan implementasi NPC (*Non-Player Characters*) pada *Game* Bergenre *Tower Defense* yang dapat berfungsi sesuai dengan kebutuhan. Spesifikasi sistem untuk implementasi algoritma dijelaskan pada spesifikasi perangkat keras dan perangkat lunak.

4.1.1 Spesifikasi Perangkat Keras

Spesifikasi perangkat keras yang dipakai dalam proses pengembangan dijelaskan dalam Tabel 4.1.

Tabel 4. 1 Spesifikasi Perangkat Keras *Device*

Nama Komponen	Spesifikasi
Processor	*Dual core, 1500 MHz, Krait*
Memory	*1024 MB RAM*
Storage	*8 GB*
Graphics processor	*Adreno 225*
System Chip	*Qualcomm Snapdragon S4 Plus MSM8960*

Sumber: Implementasi

4.1.2 Spesifikasi Perangkat Lunak

Spesifikasi perangkat lunak yang dipakai dalam proses pengembangan dijelaskan dalam Tabel 4.2.

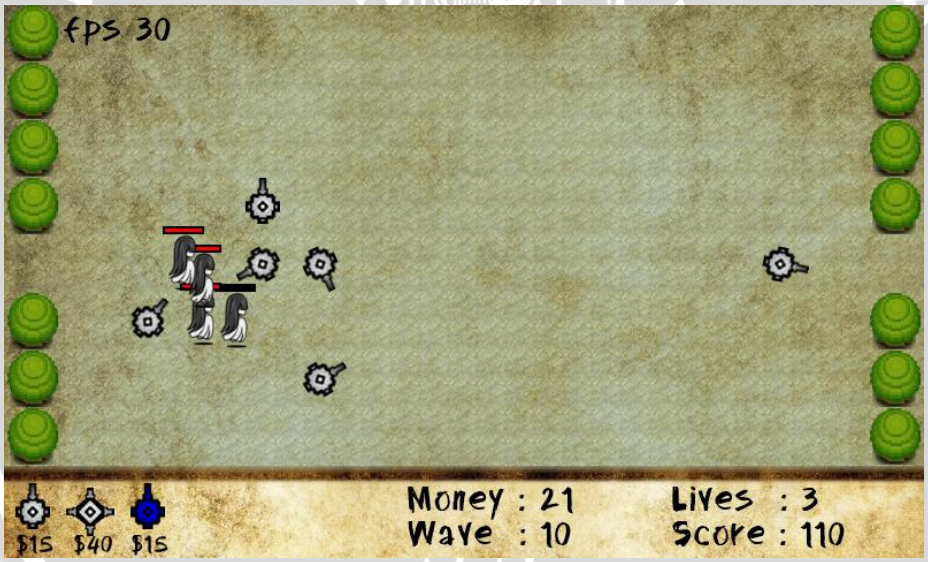
Tabel 4. 2 Spesifikasi Perangkat Lunak *Device*

Sistem Operasi	*Lumia Amber*
Game Engine	XNA *framework*

Sumber: Implementasi

4.2 Implementasi Peta *Game Tower Defense*

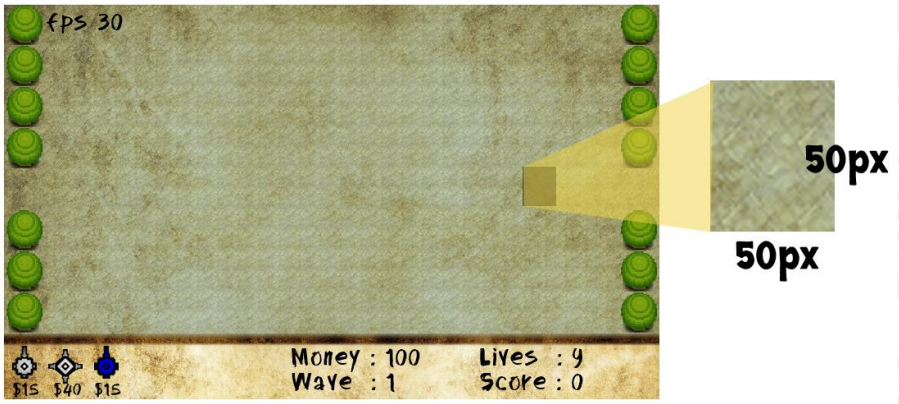
Implementasi dari *dataset* pada permainan *game Tower Defense* ditunjukkan pada gambar 4.1. Terdapat tiga macam komponen dalam permainan yang diimplementasikan sebagai *dataset* untuk simulasi.



Gambar 4. 1 Rancangan *Dataset* Permainan Keseluruhan
Sumber: Implementasi

4.2.1 Implementasi *Dataset* Peta

Peta dalam permainan ini berupa *grid*. Masing-masing *grid* mempunyai nilai yang mewakili apakah *grid* tersebut merupakan jalan atau halangan bagi *Enemy*. Ukuran tiap *grid* peta yaitu 50 piksel kali 50 piksel. Gambar implementasi peta ditunjukkan pada gambar 4.2.



Gambar 4. 2 Implementasi *Dataset* Peta Permainan
Sumber: Implementasi

Pengaturan rintangan pada peta diatur menggunakan *Array* dua dimensi. Untuk *grid* yang dapat dilintasi bernilai 0(nol) sedangkan *grid* yang tidak dapat dilewati bernilai 1(satu). Kode sumber dari peta yang ditunjukkan pada gambar 4.2 diperlihatkan oleh gambar 4.3.

```
//digunakan untuk membuat peta
    int[,] peta = new int[,]
  {
    //ada 0-15 kesamping
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},//baris 0
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},//baris 1
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},//baris 2
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},//baris 3
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},//baris 4
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},//baris 5
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},//baris 6
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},//baris 7
  };
```

Gambar 4. 3 Implementasi Peta dalam *array* 2D
Sumber: Implementasi

4.2.2 Implementasi *Enemy*

Dalam permainan *Desktop Tower Defense*, *Enemy* adalah NPC yang dapat berjalan melewati peta dengan kecepatan yang dapat diatur, namun dengan kecepatan konstan, dan agen tidak dapat melompat. Oleh karena itu agen dibuat berupa Spritesheet agar jalannya *Enemy* tidak kaku dan mempunyai animasi. Tampilan *Enemy* dalam bentuk *Spritesheet* ditunjukkan pada gambar 4.4.

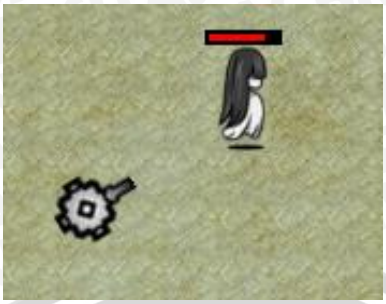
Gambar 4. 4 Implementasi *Enemy* berupa *Spritesheet*
Sumber: Implementasi

Ketika dalam permainan, *Enemy* mempunyai atribut lain yaitu berupa health bar. Health bar menunjukkan nyawa yang tersisa dari *Enemy*. Apabila *Enemy* dapat melewati peta dengan health bar yang masih sisa maka poin dari pemainakan dikurangi. Implementasi *Enemy* dalam peta ditunjukkan dalam gambar 4.5 berikut ini.

Gambar 4. 5 Implementasi *Enemy* dalam peta
Sumber: Implementasi

4.2.3 Implementasi *Tower*

Tampilan NPC *Tower* ditunjukkan dalam Gambar 4.6. *Tower* merupakan objek 2D statis, dengan satu moncong senjata yang bisa berputar 360 derajat, bertujuan untuk mempermudah membidik, yaitu ke arah *Enemy* yang menjadi target.



Gambar 4. 6 Implementasi *Tower* dalam peta
Sumber: Implementasi

4.3 Implementasi Algoritma Pada NPC *Tower Defense*

Implementasi algoritma AI ke dalam NPC diperlukan untuk menjadikan NPC bergerak sesuai dengan perilaku yang sudah dirancang sebelumnya. Oleh karena itu, algoritma AI yang sudah dirancang akan diterapkan ke *Enemy* dan *Tower*.

4.3.1 Implementasi *Behaviour Enemy*

Enemy mempunyai beberapa atribut yang tertanam terkait kemampuannya. Seperti halnya saja atribut nyawa, atribut ini berfungsi untuk menyimpan informasi apakah *Enemy* masih hidup atau justru sudah mati. Atribut berikutnya yaitu Lintasan, Atribut ini berisi jalur yang harus dilewati *Enemy* utuk mencapai titik tujuan. Dalam permainan, *Enemy* hanya bertugas untuk berjalan melewati peta untuk menuju titik tujuan dari *game Tower Defense*. Kelas yang ditanamkan ke *Enemy* akan dijabarkan dalam tabel 4.3.

Tabel 4. 3 Kelas Musuh

Musuh.cs
void SetLintasan () **Tujuan**: mengatur jalur *Enemy* yang harus dilewati. Jalur yang dikirimkan berasal dari pencarian jarak terpendek menggunakan algoritma *A**. void Update() **Tujuan**:Menjalankan logika dari *Enemy*. Mulai dari efek yang terjadi apabila ditembak oleh *Tower* sampai dengan apabila

Enemy sudah mencapai titik tujuan pada peta. Selain itu di fungsi ini juga akan mengubah jalur baru dari *Enemy* apabila ada halangan baru berupa *Tower* yang pemain buat. void Draw () **Tujuan**: menampilkan *Enemy* ke dalam permainan.

Sumber: Implementasi

Sesuai dengan perancangan, untuk mencari jalur terpendek akan digunakan algoritma *A**. Algoritma *A** merepresentasikan peta ke dalam *node-node*. Setiap *node* mempunyai *node* tetangga yang nantinya akan menjadi kandidat *node* aktif berikutnya. Implementasi algoritma *A** akan dijelaskan pada gambar 4.7.

Pseudocode algoritma *A**	
Deklarasi openList, nilaiGbaru, nilaiHbaru, nodeTetangga Deskripsi Inputan: nodeAwal, nodeAkhir	

	Proses :
	Start
	If nodeAwal == nodeAkhir then
	Return emptyPath
	End If
	nodeAwal.DalamOpenList = true
	nodeAwal.nilaiF = Heuristik (nodeAwal, nodeAkhir)
	openList.add(nodeAwal)
	while openList.Count > 0
	nodeAktif = TemukanNodeTerkecil()
	If nodeAktif == null Then
	Break while
	End If
	If nodeAktif == nodeAkhir then
	Return BuatFinalPath()
	End If
	For i=0 to nodeAktif.SemuaTetangga.Length Do
	nodeTetangga = nodeAktif.SemuaTetangga[i]
	if nodeTetangga == null or nodeTetangga.bisaDilalui == false
	continue for
	end if
	nilaiGbaru = nodeAktif.nilaiG + 1
	nilaiHbaru = Heuristik (nodeTetangga , nodeAkhir)
	if nodeTetangga.DalamOpenList == false or nodeTetangga.DalamClosedList == false
	nodeTetangga.nilaiG = nilaiGbaru

```
24                          nodeTetangga.nilaiF = nilaiHbaru + nilaiGbaru
25                          nodeTetangga.Parent = nodeAktif
26                          nodeTetangga.DalamOpenList = true
27                          openList.add(nodeTetangga)
28                  end if
29                  if  nodeTetangga.DalamOpenList  ==  true  or
    nodeTetangga.DalamClosedList == true
30                          nodeTetangga.nilaiG = nilaiGbaru
31                          nodeTetangga.nilaiF = nilaiHbaru + nilaiGbaru
32                          nodeTetangga.Parent = nodeAktif
33  end if
34  End for
35  openList.remove ( nodeAktif )
36  nodeAktif.DalamClosedList = true
37  end while
38  return EmptyPath
39  end
```

Gambar 4. 7 Implementasi Algoritma *A
Sumber: Implementasi

Penjelasan gambar 4.7:

1. Baris 1 sampai 3, apabila variabel nodeAwal sama dengan variabel nodeAkhir pencarian tidak dilakukan dan mengembalikan *Empty Path* atau jalur yang kosong.
2. Baris 4 sampai 6 persiapan awal pencarian dengan memasukkan variabel nodeAwal ke dalam *list openList* untuk memulai pencarian.
3. Baris 7 sampai 37 memeriksa semua *node* yang ada didalam *list openList*. Adapun proses yang dilakukan antara lain:

a. Baris 8 mencari *node* dengan nilai F terkecil untuk dijadikan nodeAktif dengan menggunakan *method* TemukanNodeTerkecil. *Method* TemukanNodeTerkecil akan dijelaskan pada gambar 5.8.

b. Baris 9 dan 10 bermakna apabila tidak ada hasil kembalian dari *method* TemukanNodeTerkecil maka proses pencarian dihentikan

c. Baris 12 dan 13, apabila nodeAktif sama dengan nodeAkhir maka akan mengembalikan hasil akhir dari pencarian algoritma *A**. Hasil akhir berupa jalur merupakan hasil kembalian dari *method* BuatFinalPath yang akan dijabarkan pada gambar 5.9.

d. Pada baris 15 sampai 34 mencari *node* terkecil dari *node-node* yang bertetangga dengan nodeAktif. Apabila sudah ditemukan

nodeTetangga yang mempunyai nilai bobot terkecil maka akan dimasukkan kedalam *list openList* untuk diperiksa selanjutnya.

4. Baris 35 dan 36 menghapus nodeAktif sebelumnya dari *openList* dan menjadikan nodeAktif ke dalam *closedList.*
5. Pada baris 37 dan 38, mengembalikan *list* dengan jalur kosong apabila tidak ada hasil akhir dari pencarian.

Pada gambar 4.8 akan dijelaskan mengenai *method* TemukanNodeTerkecil. *Method* ini berfungsi mencari *node* dengan bobot terkecil didalam openList.

Pseudocode method TemukanNodeTerkecil	
Deklarasi tempNode, jarakTerdekat Deskripsi Inputan: openList	
	Start
1	tempNode = openList[0]
2	For j=0 to openList.Count do
3	If openList[j] < jarakTerdekat
4	tempNode = openList[i]
5	jarakTerdekat = tempNode.nilaiF
6	End if
7	End for
8	Return tempNode
9	End

Gambar 4. 8 *Pseudocode Method* TemukanNodeTerkecil
Sumber: Implementasi

Penjelasan gambar 4.8:

1. Baris 1, menyimpan *node* pertama di *openList* ke *tempNode.*
2. Baris 2 sampai 7, mencari *node* dengan bobot jarak terkecil.

Pada gambar 4.9 akan dijelaskan mengenai *method* TemukanNodeTerkecil. *Method* ini berfungsi mencari *node* dengan bobot terkecil didalam *openList.*

Pseudocode method BuatFinalPath
Deklarasi nodeIndukTemp, jalurFix Deskripsi

Inputan: nodeAwal, nodeAkhir, closedList	
	Start
1	closedList.add(nodeAkhir)
2	nodeIndukTemp = nodeAkhir.*Parent*
3	while nodeIndukTemp != nodeAwal do
4	closedList.add(nodeIndukTemp)
5	nodeIndukTemp=nodeIndukTemp.*Parent*
6	end while
7	jalurFix.add(nodeAwal)
8	for i=closedList.Count-1 to i>=0 do
9	jaluFix.add(closedList[i])
10	end for
11	return jalurFix
12	end

Gambar 4. 9 *Pseudocode Method* BuatFinalPath
Sumber: Implementasi

Penjelasan gambar 4.9:

1. Baris 1, menambahkan nodeAkhir ke dalam *closedList*
2. Baris 2 menyimpan *node* induk dari nodeAkhir ke nodeIndukTemp
3. Baris 3 sampai 6 menelusuri *parent* dari nodeIndukTemp sampai nodeIndukTemp sama dengan nodeAwal setelah dilakukan pemeriksaan semua *node* dalam pencarian algortima *A**.
4. Baris 7 sampai baris 12 memasukkan semua hasil akhir jalur ke dalam jalurFix kemudian mengembalikannya sebagai hasil dari pencarian algoritma *A**.

4.3.2 Implementasi *Behaviour Tower*

Tower merupakan salah satu NPC dari *Desktop Tower Defense* yang mempunyai tugas untuk menghambat *Enemy* sampai pada titik akhir dari peta. Ada dua teknik AI yang ditanamkan kepada *Tower* yaitu *Line Of Sight* dan juga *Finite State Machine*. Implementasi dari *Tower* ditentukan oleh kode sumber dari kelas *Tower*. Penjelasan tentang kelas *Tower* akan dijabarkan pada tabel 4.4

Tabel 4. 4 Kelas *Tower*

Tower.cs
bool IsDalamRadius () **Tujuan**: Mengecek apakah *Tower* mempunyai target *Enemy* yang berada dalam radiusnya/jarak jangkau. Apabila ada target *Enemy* yang masih berada pada jarak jangkaunya maka

akan mengembalikan nilai *true*.

void GetMusuhTerdekat()

Tujuan: Mencari target *Enemy* dari list*Enemy* yang ada. *Enemy* yang menjadi target adalah *Enemy* yang berada dalam jarak jangkau dari *Tower*. *Tower* kan mencari target *Enemy* baru apabila target *Enemy* sebelumnya sudah diluar jarak jangkau *Tower*.

void UpgradeLevel2()

Tujuan: meningkatkan kemampuan *Tower* menjadi tingkat ke 2 yang sebelumnya masih tingkat 1.

void MenghadapTarget()

Tujuan: membuat moncong *Tower* selalu menghadap ke *Enemy* yang menjadi targetnya.

void Update()

Tujuan: untuk menjalankan logika-logika dari *Tower*. Selain itu mekanisme *Tower* menembak juga berada pada *method* ini.

void Draw()

Tujuan: menampilkan *Tower* dan peluru ke dalam permainan.

Sumber: Implementasi

4.3.2.1 Implementasi *Line Of Sight*

Seperti yang sudah dijabarkan dalam perancangan, teknik *Line Of Sight* dalam *Tower* berfungsi untuk menentukan musuh yang menjadi target tembak *Tower*. Setiap *Tower* mempunyai jangkauan area masing-masing. Dengan menggunakan teknik *Line Of Sight Tower* akan selalu menghadap ke arah dimanapun *Enemy* yang masih dalam jangkauannya. Setelah menentukan *Enemy* yang menjadi target, *Tower* akan menembak tepat ke *Enemy*. Implementasi *Line Of Sight* ditunjukkan pada gambar 4.10.

Pseudocode Line Of Sight

Deklarasi

Tower, listPeluru

Deskripsi

Inputan: semuaMusuh

```
        Proses :
1       Start
2       If Tower.PunyaTargetMusuh == false then
3               Foreach Enemy in semuaMusuh do
4                       If Distance(Tower,musuh) < Tower.Radius then
                                Tower.Target = Enemy
5                       End if
6               End foreach
7       Else
8               Tower.MenghadapTarget()
9               if Tower.target.IsMati == true or Tower.IsDalamRadius == true  then
10                      Tower.Target=null
                end if
11              if Tower.Target != null  then
12                      listPeluru.add(new peluru(Tower.Target))
13              end if
14      End if
15      End
16
17
```

Gambar 4. 10 *Pseudocode Line Of Sight*
Sumber: Implementasi

Penjelasan gambar 4.10:

1. Baris 2 sampai 7, apabila *Tower* belum mempunyai target *Enemy* maka akan mencari *Enemy* yang berada dalam semuaMusuh dengan pertimbangan *Enemy* harus berada dalam jarak jangkaunya.
2. Baris 9 membuat moncong *Tower* menghadap ke arah dimana letak *Enemy* targetnya berada. *Method* MenghadapTarget akan dijelaskan pada gambar 4.11.
3. Baris 10 dan 11 mengecek apakah *Enemy* target sudah mati atau sudah berada dalam jarak jangkau. Kalau salah satu kondisi tersebut terpenuhi maka *Tower* tidak mempunyai target *Enemy*.
4. Baris 13 dan 14 jika musuh masih mempunyai target *Enemy* maka akan ditembakkan peluru mengarah ke target *Enemy*.

Pada gambar 4.11 akan dijelaskan mengenai *method* MenghadapTarget. *Method* ini berfungsi membuat moncong *Tower* menghadap ke posisi target *Enemy*.

Pseudocode Method MenghadapTarget	
Deklarasi Rotasi, arahVektor Deskripsi Inputan: *Tower*	
	Proses : Start arahVektor = *Tower*.posisi – *Tower*.target.posisi arahVektor.Normalize() rotasi=atan2(arahVektor) *Tower*.setRotasi(rotasi) End

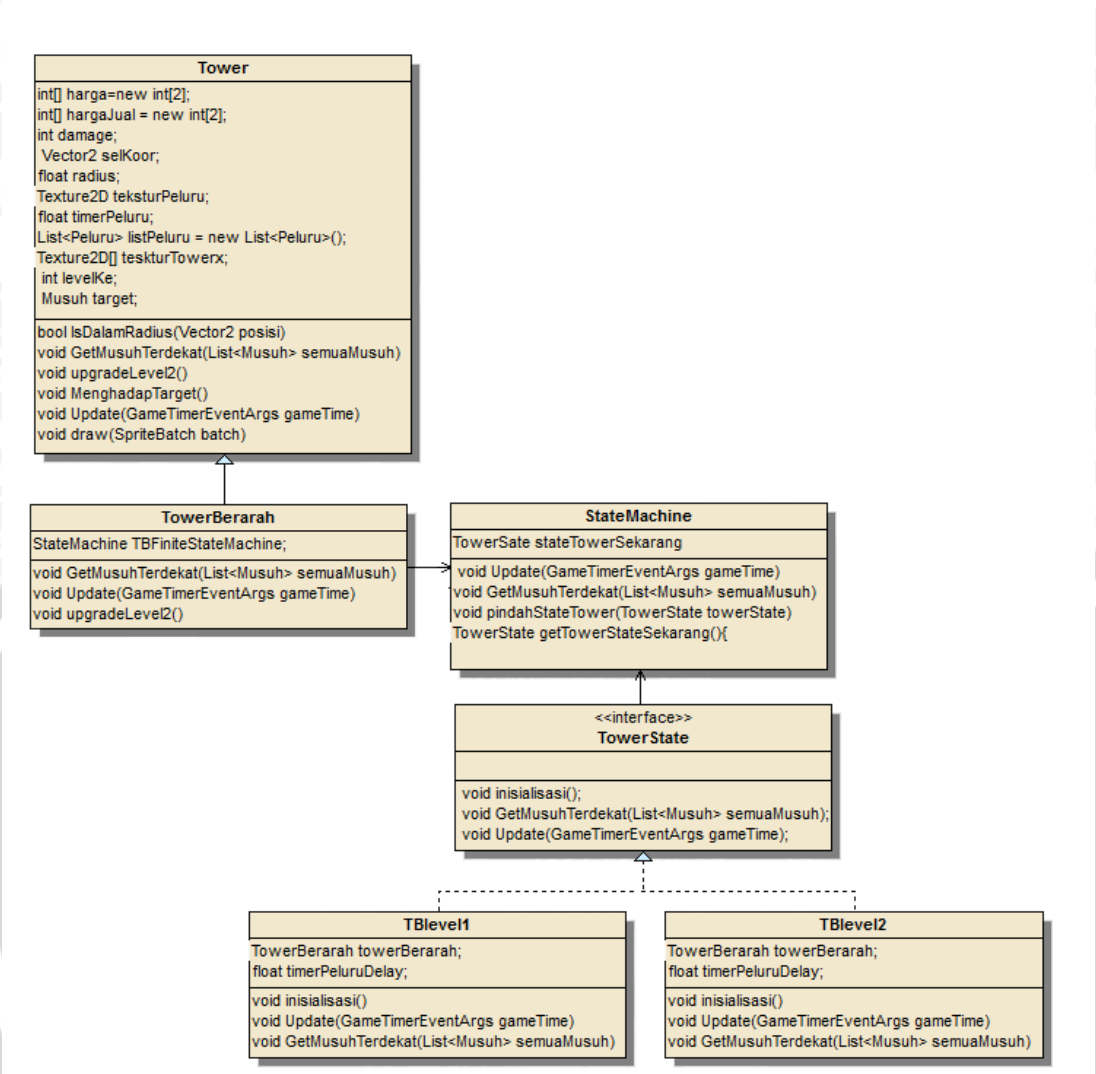
Gambar 4. 11 *Pseudocode Method* MenghadapTarget
Sumber: Implementasi

Penjelasan gambar 4.11:

1. Baris 2 dan 3 mencari sudut/kemiringan berupa vektor berasal dari selisih antara posisi *Tower* dan target *Enemy* serta menormalisasikan vektor tersebut.
2. Baris 4 menghitung besar derajat sudut dari arahVektor
3. Baris 5 mengeset rotasi *Tower* sesuai dengan sudut yang nantinya berpengaruh terhadap tampilan arah moncong *Tower*.

4.3.2.2 Implementasi *Finite State Machine*

Finite State Machine merupakan teknik AI yang bertujuan untuk mempermudah agen AI berpindah *state*. Di dalam implementasi *Tower*, *Finite State Machine* berfungsi untuk mengatur perpindahan *Tower* mulai dari level 1 menjadi level 2. Teknik ini menggunakan *state* pattern dalam pemrograman berorientasi objek untuk implementasinya. Diagram kelas dari implementasi *Finite State Machine* untuk *Tower* ditunjukkan pada gambar 4.12.



Gambar 4. 12 Implementasi *Finite State Machine*
Sumber: Implementasi

Penjelasan gambar 4.12:

1. Kelas *Tower* merupakan kelas *general* untuk *Tower*. Kelas ini mempunyai standar dari *Tower* yang nantinya bisa divariasikan kemampuan dari masing-masing *Tower*.
2. Kelas TowerBerarah adalah kelas yang berisi tentang kemampuan dari *Tower*. Di dalam kelas ini ada atribut StateMachine agar *Tower* jenis TowerBerarah mempunyai *Finite State Machine* di dalam dirinya.

3. Kelas StateMachine merupakan sebuah *context*, namun supaya dapat digunakan lagi di kelas lainnya maka dibuatkan kelas tersendiri.
4. *Interface TowerState* mempunyai tiga *method* yang harus dipunya oleh objek dari *TowerState*. Ada tiga *method* antara lain:
    a. void inisialisasi() berisi tentang inisialisasi dari variabel-variabel yang akan dipakai.
    b. void GetMusuhTerdekat(List<Musuh> semuaMusuh) bertugas untuk mencari *Enemy* terdekat untuk dijadikan target.
    c. void Update(*Game*TimerEventArgs *game*Time) menjalankan logika dari masing masing *TowerState*.
5. Kelas TBLevel1 dan TBLevel2 merupakan tempat di mana *state behaviour Tower* akan diletakkan.

BAB V
PENGUJIAN DAN ANALISIS

Pada bab ini dilakukan proses pengujian dan analisis terhadap implementasi *Non-Player Characters* (NPC) pada *Game* Bergenre *Tower Defense*. Pengujian dilakukan dengan menguji tiap skenario dan pencatatan hasil berdasarkan parameter pengujian. Parameter pengujian mempunyai tujuan untuk memastikan apakah NPC sudah berjalan seperti yang sudah dirancang. Selain itu juga dilakukan pengujian Kinerja berupa jumlah rata-rata FPS dan *Memory* yang dibutuhkan saat menjalankan *game* dengan jumlah NPC yang divariasikan.

5.1 Menentukan Skenario Pengujian

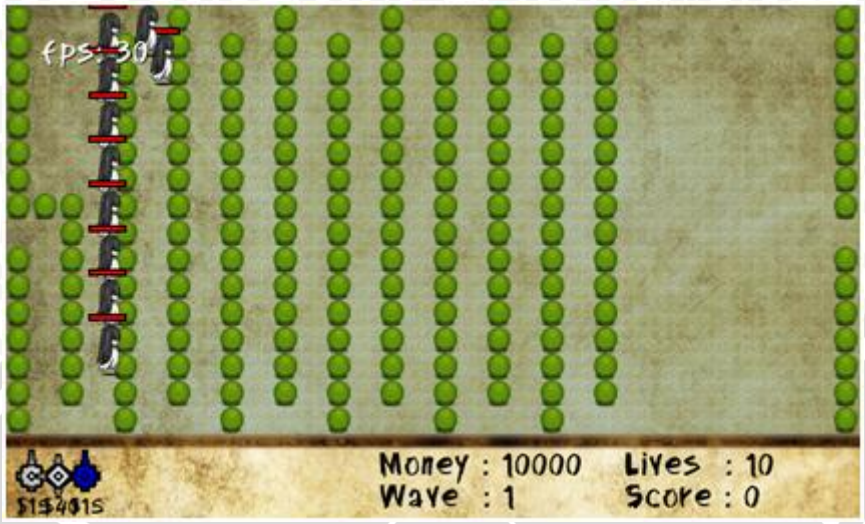
Pengujian dilakukan untuk mengetahui valid atau tidaknya *behaviour* yang sudah dirancang dengan implementasinya. Selain itu, pengujian juga dilakukan untuk mengetahui performa dari sistem dalam menjalankan algoritma *NPC*. Performa dari sistem dapat ditinjau dari rata-rata FPS per simulasi. Uji coba dibagi menjadi 2 skenario:

1. Uji coba skenario 1, simulasi dilakukan dengan cara menaruh NPC *Enemy* dengan jumlah bervariasi pada peta permainan. Tujuan dari skenario ini untuk memastikan apakah *behaviour* pada *Enemy* sudah berjalan sesuai perancangan.
2. Uji coba skenario 2, simulasi dilakukan dengan cara menaruh NPC *Tower* dan NPC *Enemy* dengan jumlah bervariasi pada peta permainan. Tujuan dari skenario ini untuk memastikan apakah *behaviour* pada *Tower* sudah berjalan sesuai perancangan.

5.1.1 Pengujian Skenario 1

Uji coba skenario 1 adalah pengujian untuk memastikan *behaviour* pada *Enemy* sudah berjalan secara benar dan optimal. Pengujian dilakukan dengan cara memvariasikan jumlah *Enemy* yang ada di dalam peta tanpa ada *Tower*.Adapun ketentuan atribut yang sama setiap tes yaitu kecepatan *Enemy* berjalan 0,5 piksel dan jeda untuk pembuatan *Enemy* baru yaitu 50 Milidetik. Untuk memastikan

*behaviour pathfinding A** berjalan dengan benar akan diberikan halangan di dalam peta. Contoh pengujian skenario 1 dapat dilihat pada gambar 5.1.



Gambar 5. 1 Contoh Pengujian Skenario 1
Sumber : Pengujian dan Analisis

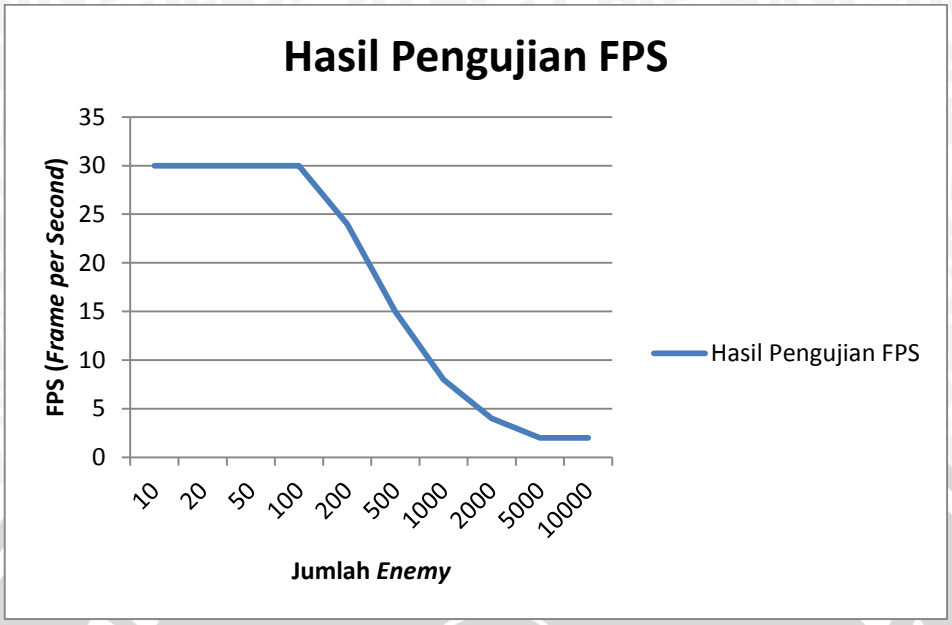
Hasil dari pengujian skenario 1 menggunakan matriks FPS dan *Memory* yang dibutuhkan dapat dilihat pada tabel 5.1.

Tabel 5. 1 Tabel Hasil Pengujian Skenario 1

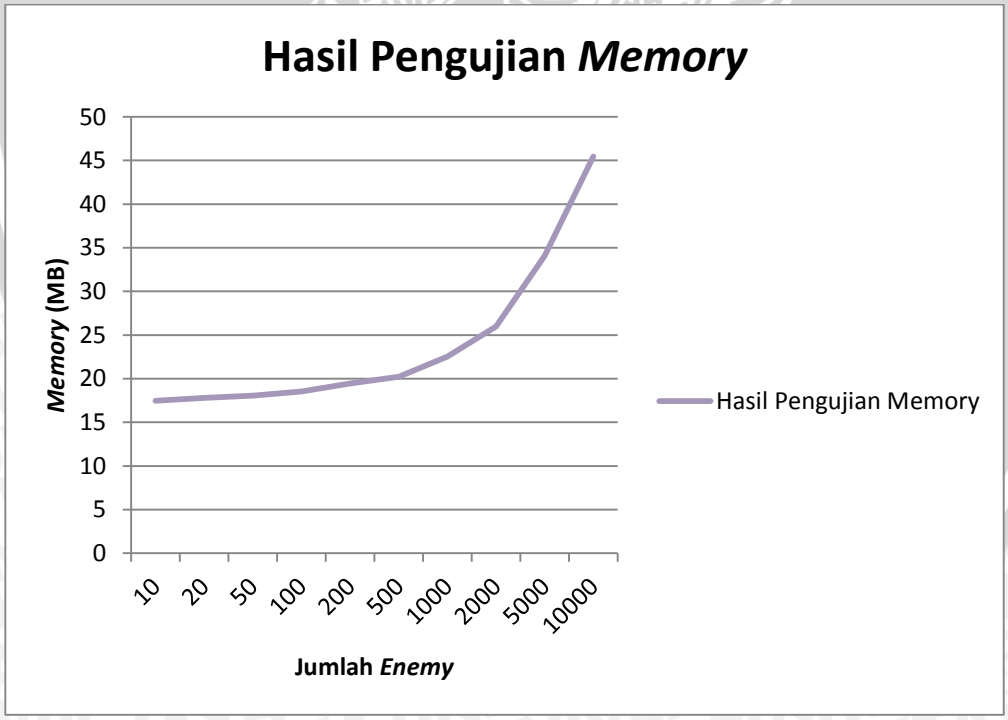
No. tes	Jumlah *Enemy*	FPS	Rataan *Memory* (Mb)
1	10	30	17,46
2	20	30	17,80
3	50	30	18,07
4	100	30	18,54
5	200	24	19,43
6	500	15	20,22
7	1000	8	22,54
8	2000	4	25,97
9	5000	2	34,08
10	10000	2	45,44

Sumber : Pengujian dan Analisis

Agar lebih mudah untuk melihat data maka tampilan grafik dari hasil pengujian Skenario 1 ditampilkan pada gambar 5.2 dan 5.3.



Gambar 5. 2 Grafik Hasil Pengujian FPS Skenario 1
Sumber : Pengujian dan Analisis



Gambar 5. 3 Grafik Hasil Pengujian *Memory* Skenario 1
Sumber : Pengujian dan Analisis

5.1.2 Pengujian Skenario 2

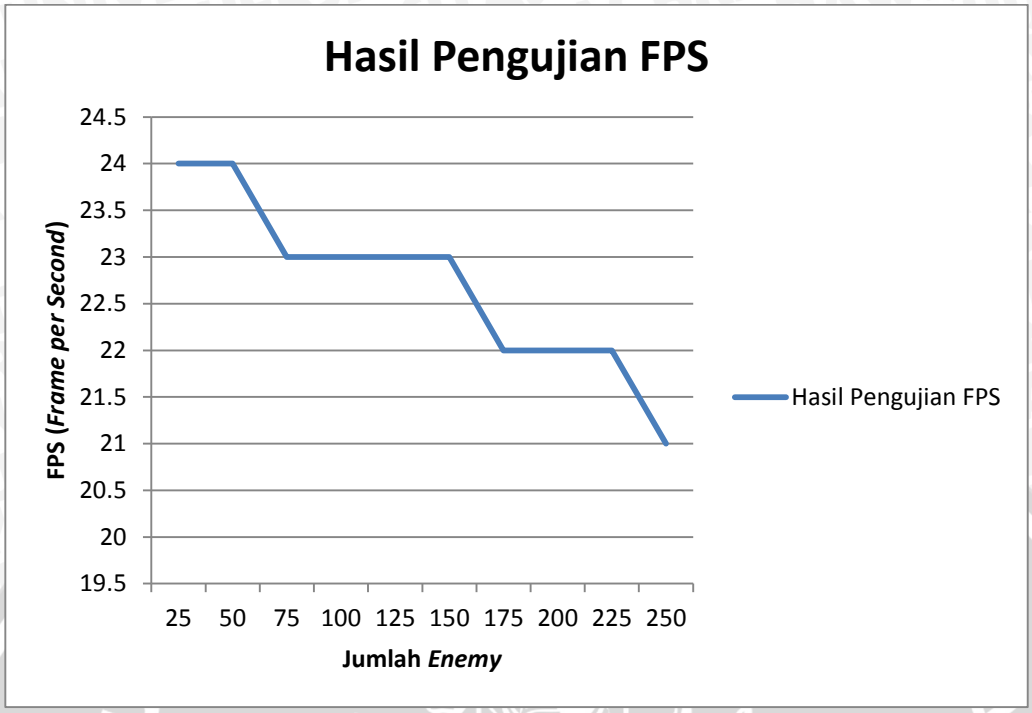
Uji coba skenario 2 adalah pengujian untuk memastikan *behaviour* pada *Tower* sudah berjalan secara benar dan optimal. Pengujian dilakukan dengan cara memvariasikan jumlah *Tower* yang ada di dalam. Adapun ketentuan atribut yang sama setiap tes yaitu jumlah *Enemy* 200, kecepatan *Enemy* berjalan 1,7 piksel dan jeda untuk pembuatan *Enemy* baru yaitu 100 Milidetik. Hasil dari pengujian skenario 2 menggunakan matriks FPS dan *Memory* yang dibutuhkan dapat dilihat pada tabel 5.2.

Tabel 5. 2 Tabel hasil pengujian Skenario 2

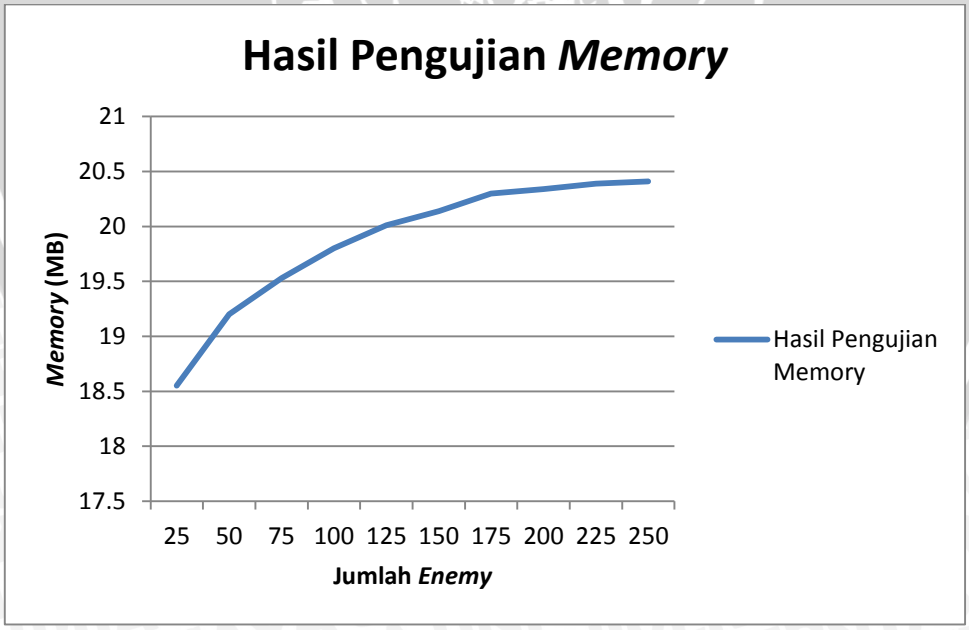
No. tes	Jumlah *Tower*	FPS	Rataan *Memory* (Mb)
1	25	24	18,55
2	50	24	19,20
3	75	23	19,53
4	100	23	19,80
5	125	23	20,01
6	150	23	20,14
7	175	22	20,30
8	200	22	20,34
9	225	22	20,39
10	250	21	20,41

Sumber : Pengujian dan Analisis

Agar lebih mudah untuk melihat data maka tampilan grafik dari hasil pengujian Skenario 2 ditampilkan pada gambar 5.4 dan 5.5.



Gambar 5. 4 Grafik Hasil Pengujian FPS Skenario 2
Sumber : Pengujian dan Analisis

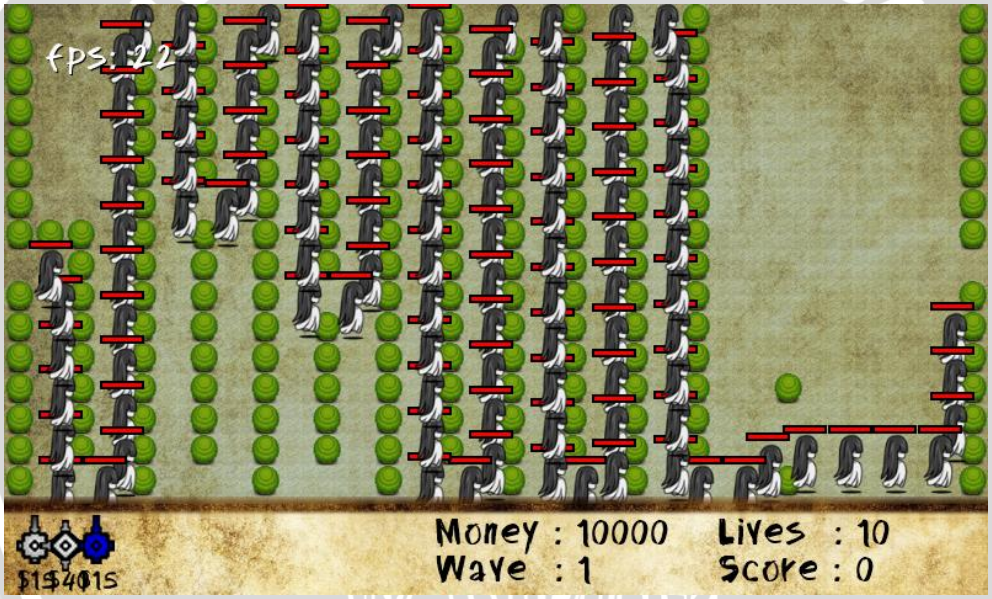


Gambar 5. 5 Grafik Hasil Pengujian *Memory* Skenario 2
Sumber : Pengujian dan Analisis

5.2 Analisis Hasil Pengujian

Analisis bertujuan untuk mendapatkan kesimpulan dari hasil pengujian implementasi NPC pada *Game Bergenre Tower Defense*. Proses analisis terhadap hasil pengujian performa dilakukan dengan melihat hubungan antara FPS yang dihasilkan dan penghitungan penggunaan *Memory*.

Pada uji coba skenario 1 mempunyai tujuan untuk memastikan apakah *behaviour* pada *Enemy* sudah berjalan sesuai dengan perancangan dan memiliki performa yang optimal. Sesuai dengan *screenshot* pengujian pada gambar 5.6 *Enemy* berhasil berjalan mulai dari titik awal sampai titik akhir permainan melewati halangan yang dipasang.

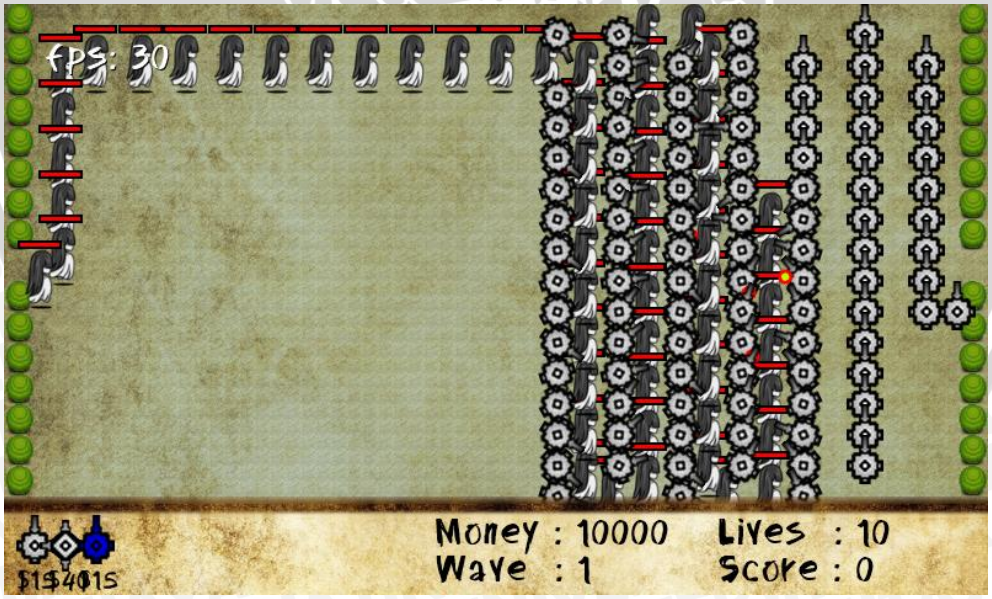


Gambar 5. 6 Gambar *Enemy* menuju titik akhir
Sumber : Pengujian dan Analisis

Enemy akan mencari jalur baru apabila ada perubahan pada peta permainan yang menutupi jalan yang sudah diperhitungkannya sebelumnya. Hal ini membuktikan bahwa pengimplementasian *behaviour* berupa *pathfinding* A *star* yang sudah dirancang dan ditanamkan ke dalam NPC *Enemy* berhasil. Berdasarkan hasil pengujian pada gambar 5.2 dan 5.3, FPS berada dalam keadaan stabil yaitu 30 FPS ketika *Enemy* berjumlah 10,20,50 dan 100. Hal ini dikarenakan komputasi dari *Enemy* belum terlalu banyak sehingga tidak mempengaruhi kinerja dan pemrosesan dari *device testing*. Baru setelah pengujian

dengan jumlah *Enemy* 200 ke atas sampai 10.000 FPS mengalami penurunan dikarenakan proses komputasi *Enemy* yang sudah mulai mempengaruhi dari kinerja *device testing*. Walaupun FPS pada saat *Enemy* berjumlah 10,20,50 dan 100 stabil, namun tetap terjadi peningkatan untuk banyaknya *Memory* yang digunakan dalam permainan. Peningkatan penggunaan *Memory* yang tertinggi adalah ketika jumlah *Enemy* 5000 kemudian ditingkatkan menjadi 10.000 *Enemy*, ini dikarenakan jumlah dari peningkatan jumlah musuh lebih banyak dari peningkatan-peningkatan pada jumlah *Enemy* sebelumnya. Dari data-data pengujian dapat diketahui, semakin banyak *Enemy* yang ada di dalam permainan semakin banyak *Memory* yang digunakan dan semakin sedikit FPS dari permainan.

Uji coba skenario 2 mempunyai tujuan untuk memastikan apakah *behaviour* pada *Tower* sudah berjalan sesuai dengan perancangan dan memiliki performa yang optimal. Dari *screenshot* pengujian pada gambar 5.7 , *Tower* selalu menghadap ke arah *Enemy* yang menjadi target sesuai dengan radius yang dimilikinya.



Gambar 5. 7 Gambar *Tower* menghadap *Enemy*
Sumber : Pengujian dan Analisis

Setelah *Tower* mendapatkan *Enemy* yang menjadi targetnya, *Tower* akan menembak *Enemy* tersebut dan mengurangi nyawa dari *Enemy*. Hal ini membuktikan bahwa pengimplementasian *behaviour* yang sudah dirancang kepada NPC *Tower* berhasil. Pengujian banyaknya *Tower* dengan jumlah terbesar yakni 250 *Tower* dikarenakan jumlah maksimal *Tower* yang dapat dibangun dalam peta permainan hanya 256 *Tower* untuk ukuran *grid* 32x16. Pada tes pertama nilai FPS hanya mencapai 24 FPS, kalah tinggi dibanding percobaan pertama dari *Enemy* yang FPS mencapai 30. Hal tersebut disebabkan penggunaan 200 *Enemy* untuk pengujian pada skenario 2 ini. Berdasarkan gambar 5.4, FPS berada dalam keadaan relatif stabil dengan selisih dari tes pertama yang memakai 25 *Tower* dengan tes terakhir yang memakai 250 *Tower* yaitu 3 FPS dengan jumlah rata-rata 22,7 FPS. Sama halnya dengan performa, *Memory* yang dibutuhkan oleh *Tower* pun juga tidak terlalu banyak terbukti dengan selisih antara tes pertama dengan tes terakhir hanya 1,86 Mb. Dari data-data pengujian tersebut, dapat disimpulkan bahwa semakin banyak *Tower* juga mempengaruhi penggunaan *Memory* walaupun tidak sesignifikan pada *Enemy* karena proses komputasi yang dibutuhkan *tower* tidak serumit dari komputasi yang dilakukan untuk *Enemy*.

Secara keseluruhan baik dari pengujian skenario 1 maupun skenario 2 telah membuktikan NPC telah berjalan sesuai dengan perancangan. Kuantitas *Tower* dan *Enemy* berbanding terbalik dengan banyaknya FPS dan berbanding lurus dengan banyaknya *Memory* yang dibutuhkan. Antara *Tower* dan *Enemy* kebutuhan *memory* lebih banyak dibutuhkan oleh *enemy*, hal ini dikarenakan komputasi *pathfinding* yang dilakukan *enemy* lebih kompleks dibanding komputasi *line of sight* dan *finite state machine* yang diimplementasikan ke *tower*.

BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan hasil pengujian dan analisis yang dilakukan terhadap implementasi dari *Non-Player Character*(NPC) pada *Game bergenre Tower Defense*, dapat diambil kesimpulan sebagai berikut:

1. Ada beberapa tahap yang dilakukan bagi *developer* pemula untuk merancang NPC pada *game Tower Defense* yaitu menganalisa gameplay *Tower Defense*, mengidentifikasi NPC *game Tower Defense*, mengidentifikasi *behaviour* NPC *game Tower Defense*, merancang implementasi *behaviour* NPC pada *game Tower Defense*.
2. Implementasi algoritma untuk NPC *Enemy* yang akan diberi *behaviour* berupa *pathfinding* dalam permainan *Desktop Tower Defense* dilakukan berdasarkan algoritma pencarian jalur berdasarkan *A** secara *real time* pada peta permainan berbasis *Grid*.
3. Implementasi algoritma untuk NPC *Tower* dalam permainan *Desktop Tower Defense* menggunakan teknik *Line Of Sight* untuk menyerang *Enemy* terdekat(*Chasing*) yang menjadi targetnya dan teknik *Finite State Machine* agar *Tower* memiliki kemampun untuk ditingkatkan tingkat kekuatannya.
4. Implementasi *behaviour* pada NPC *Enemy* cukup optimal dilihat dari hasil pengujian yang menunjukkan *frame rate* stabil ketika jumlah *Enemy* 100 dan hanya menggunakan *Memory* 18,54 Mb.
5. Implementasi *behaviour* pada NPC *Tower* cukup optimal dilihat dari hasil pengujian yang menunjukkan *frame rate* relatif stabil berada diantara 21 sampai 24 FPS dan hanya menggunakan *Memory* 20**,**41Mb untuk jumlah 250 *Tower*.

6.2 Saran

Untuk meningkatkan hasil yang telah dicapai dari penelitian ini dapat dilakukan beberapa perbaikan sebagai berikut:

1. Diharapkan pada penelitian selanjutnya, penggunaan *pathfinding* dapat di variasikan supaya ada macam-macam *Enemy*, misalnya ada *Enemy* yang berjalan dengan memperhitungkan bobot *Tower* yang menghadang yang berada dalam peta permainan.
2. Menggunakan *Hierarchical Finite State Machine* untuk pengombinasian *behaviour tower* yang semakin variatif dan pengelolaan *State Machine* yang lebih baik dan terstruktur[SAM-13].

DAFTAR PUSTAKA

[ALD-09] Aldrich, Clark. 2009. *The Complete Guide to Simulations and Serious Games: How the Most Valuable Content Will be Created in the Age Beyond Gutenberg to Google*. John Wiley and Sons. Hoboken, New Jersey.

[AVE-11] Avery, Phillipa. 2011. Computational Intelligence and Tower Defence Games. Department of Computer Science and Engineering University of Nevada, Reno, USA

[CHE-09] Chen, Siyuan, et al. 2009. *Fast Path Searching in Real Time 3D Game.* 2009 Global Congress on Intelligent Systems.

[DAL-09] Daly, Kyle. 2009. *Design, Implement and Evaluate the Effectiveness and Performance of a Dynamic Tactical Game Agent Position Evaluation System.* University of Bolton Computer Games Software Development Dissertation .

[DMB-04] David M. Bourg, Glenn Seeman. 2004, AI for *Game* Developers. O'Reilly

[FRE-04] Freeman, Eric dkk. 2004, *Head First Design Patterns*. O'Reilly Media, Sebstopol.

[HER-11] Hernández, Carlos, et al. 2011, *Fast Subgoaling for Pathfinding via Real-Time Search.* Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling, Freiburg.

[JIE-11] Jie, Jiang, et al. 2011, *The Application of AI for the Non Player Character in Computer Games*, International Conference on Computational and Information Sciences.

[MCJ-07] McCarthy, John. 2007, *What is Artificial Intelligence?* , Stanford University Computer Science Department. http://www-formal.stanford.edu/jmc/whatisai/whatisai.html

[MIC-13] Microsoft. 2013. Windows Phone Application Analysis. http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202934(v=vs.105).aspx.[22 November 2013].

[MIL-09] Millington, Ian. Funge, John. 2009, *Artificial Intelligence for Games Second Edition.* Morgan Kaufmann. Burlington, Massachusetts.

[MIY-10] Miftachul, Yunifa, et al. 2010, Strategi Menyerang pada Game FPS Menggunakan Hierarchy *Finite State Machine* dan Logika Fuzzy. 2009 Global Congress on Intelligent Systems. Institut Sepuluh Nopember, Surabaya

[REY-13] Reynold, Craig. 2004. *Seek and Flee steering behaviors.* http://www.red3d.com/cwr/steer/SeekFlee.html. [4 April 2013]

[RUS-10] Russel, Stuart. Norvig, Peter. 2010. *Artificial Intelligence. A Modern Approach. Third Edition.* Prentice Hall, New Jersey 2010.

[SAM-13] Reynold, Craig. 2013. Introduction to Hierarchical *State* Machines (HSMs). http://www.barrgroup.com/Embedded-Systems/How-To/Introduction-Hierarchical-*State*-Machines.[22 November 2013].

[TIL-11] Tilawah, Hapsari. 2011. Penerapan Algoritma A-star (*A**) Untuk Menyelesaikan Masalah Maze. Bandung: Institut Teknologi Bandung

[WIJ-01] Wijaya, Adi And Gunawan, Roi.2001, Analisis Perbandingan Algoritma Path Finding *A** Dan Dijkstra Pada Real Time Game. Undergraduate Thesis, Binus.

[WIK-13] Izno. 2013. *Tower Defense.* http://en.wikipedia.org/wiki/Tower_defense. Diakses tanggal 24 Oktober 2013