

## **BAB 3**

### **ANALISIS DAN PERANCANGAN SISTEM**

#### **3.1 Analisis Sistem**

Analisis sistem merupakan penguraian dari suatu sistem informasi yang utuh ke dalam bagian-bagian komponennya yang dimaksudkan untuk mengidentifikasi dan mengevaluasi segala permasalahan dan hambatan yang terjadi serta kebutuhan yang diharapkan dapat menjadi acuan untuk diusulkannya perbaikan-perbaikan.

##### **3.1.1 Analisis Masalah**

Masalah yang diangkat dalam penulisan skripsi ini adalah tentang penerapan algoritma SMA\*. Algoritma SMA\* (*Simplified Memory-Bounded A\**) merupakan pengembangan dari algoritma A\* yang mengatasi masalah *storage problem* dengan meletakkan *limit* tetap pada ukuran *open list*. Ketika sebuah *node* baru diproses, jika memiliki nilai yang lebih besar dari setiap *node* yang ada di *list*, maka akan dibuang. Jika tidak, maka akan ditambahkan, dan *node* yang sudah dalam *list* dengan nilai terbesar dihapus. Dengan ini diharapkan algoritma SMA\* dapat menghemat penggunaan memori yang harus dihemat pada perangkat *mobile*, sehingga *game* dapat berjalan dengan lancar dengan penggunaan memori yang minimum.

##### **3.1.2 Storyline/Pengenalan Game**

*Game “Monster Nest”* ini dimulai dengan mengikuti karakter utama yang sudah berhasil masuk ke dalam terowongan kereta bawah tanah yang sudah lama tak berfungsi. Terowongan ini dipercaya sebagai sarang monster yang menyerang kota, ia ingin memusnahkan semua *monster* tersebut sehingga kota kembali aman. Dilengkapi dengan senjata *shotgun* dan *assault rifle* yang diperoleh dari toko senjata lokal, dia sudah siap untuk menghadapi semua kengerian yang ada.

Kereta bawah tanah tua ini keadaannya gelap, lembah dan penuh dengan sampah dan puing-puing lainnya. Sebelum menjadi sarang *monster*, terowongan ini sering digunakan dalam beberapa tahun. Pencahayaan minimal dan terbatas. Ini memberikan nuansa yang menakutkan ditambah lagi karakter tidak tahu dari mana *monster* akan muncul.

*Monster* yang akan dibasmi oleh karakter memiliki banyak kaki dan sedikit menyerupai kutu. Ukurannya sama dengan anjing berukuran sedang. *Monster* ini berada memenuhi terowongan dan mengejar karakter. Jika *monster* mencapai karakter sebelum dibunuh, mereka akan melompat dan menyerangnya dengan meledakkan diri mengeluarkan cairan beracun.

### 3.1.3 Analisis Tingkat Kesulitan

Tingkat kesulitan dalam *game Monster Nest* ini, terdiri dari 3 *level* yaitu *easy*, *medium*, dan *hard* dengan spesifikasi sebagai berikut:

#### 1. *Level easy*

Jumlah *monster* akan bertambah 1 *monster* setiap *player* menyelesaikan 1 *ronde*. Waktu awal yang diberikan dalam 1 *ronde* untuk *player* menyelesaikan misi membunuh *monster* adalah 30 detik dan akan bertambah sesuai dengan jumlah *ronde* yang dimenangkan dikali dengan waktu awal.

#### 2. *Level medium*

Jumlah *monster* akan bertambah 2 *monster* setiap *player* menyelesaikan 1 *ronde*. Kecepatan *monster* akan menjadi satu setengah kali lipat dari sebelumnya. Waktu awal yang diberikan dalam 1 *ronde* untuk *player* menyelesaikan misi membunuh *monster* adalah 30 detik dan akan bertambah sesuai dengan jumlah *ronde* yang dimenangkan dikali dengan waktu awal.

#### 3. *Level hard*

Jumlah *monster* akan bertambah 3 *monster* setiap *player* menyelesaikan 1 *ronde*. Kecepatan *monster* akan menjadi dua kali lipat dari sebelumnya. Waktu awal yang diberikan dalam 1 *ronde* untuk *player* menyelesaikan misi membunuh *monster* adalah 30 detik dan akan bertambah sesuai dengan jumlah *ronde* yang dimenangkan dikali dengan waktu awal.

### 3.1.4 *Gameplay*

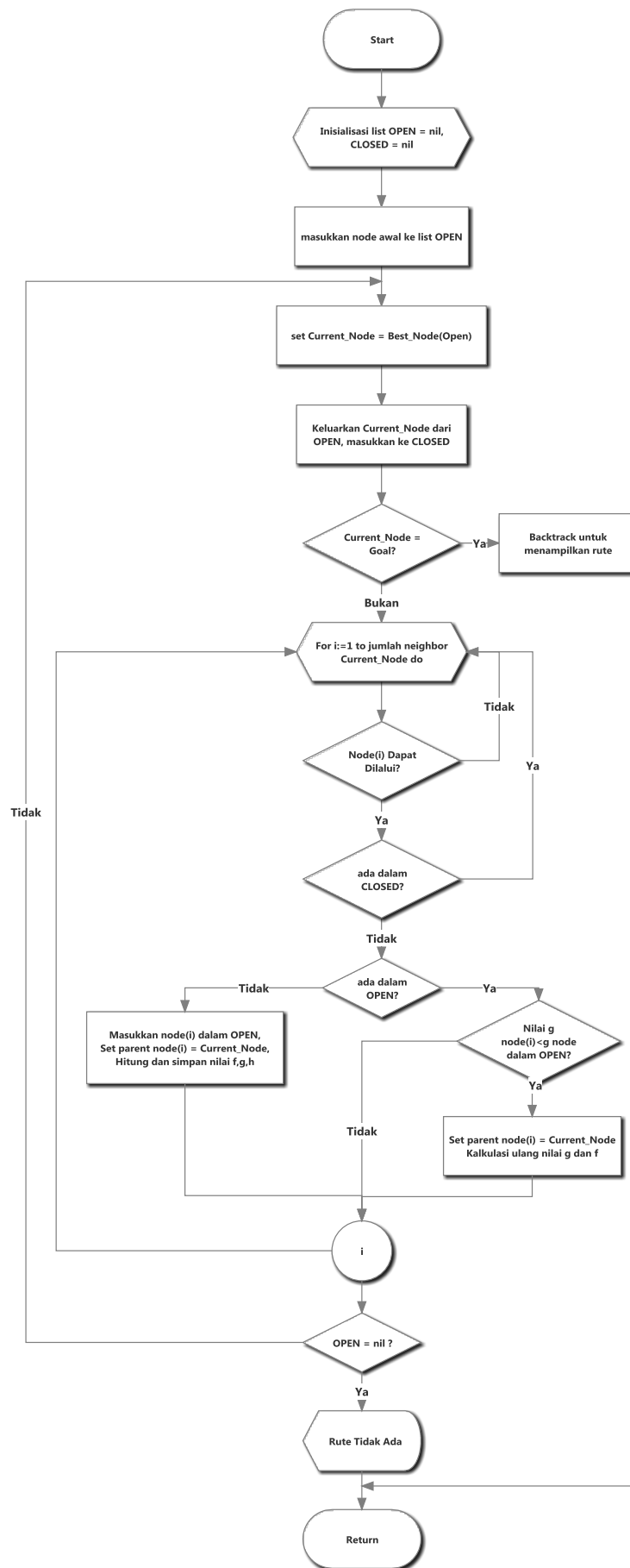
Pada awal permainan akan ditampilkan menu utama yang terdapat pilihan *play game*, *set level* dan *highscore*. Sebelum memulai permainan pemain dapat memilih tingkatan *level* yaitu memilih tingkatan *level* dengan permainan mudah, permainan sedang dan permainan terakhir yaitu permainan sulit, dengan menekan menu *set level*. Setelah pemain memilih salah satu pilihan tingkatan permainan, pemain dapat memulai dengan memilih menu *play game* kemudian pemain akan langsung dihadapkan dengan musuh, disini pemain diharuskan untuk membunuh semua *monster* yang ada untuk mendapatkan skor tertinggi, selama permainan pemain dapat menggerakkan *player* dengan menggerakkan tombol yang ada di layar, menembak, mengganti senjata, mengisi peluru dan dapat juga melakukan *pause* untuk menghentikan permainan sementara dengan menekan tombol *pause*. Pada menu *pause*, pemain dapat melanjutkan permainan, memulai permainan dari awal atau menuju ke menu utama, ketika pemain mati, otomatis akan menuju ke menu skor tertinggi. Setelah memasuki menu skor, jika pemain mendapatkan skor tertinggi maka akan diminta untuk memasukkan nama dan akan disimpan. Dari menu skor pemain dapat pergi ke menu utama.

### 3.1.5 *Analisis Skoring*

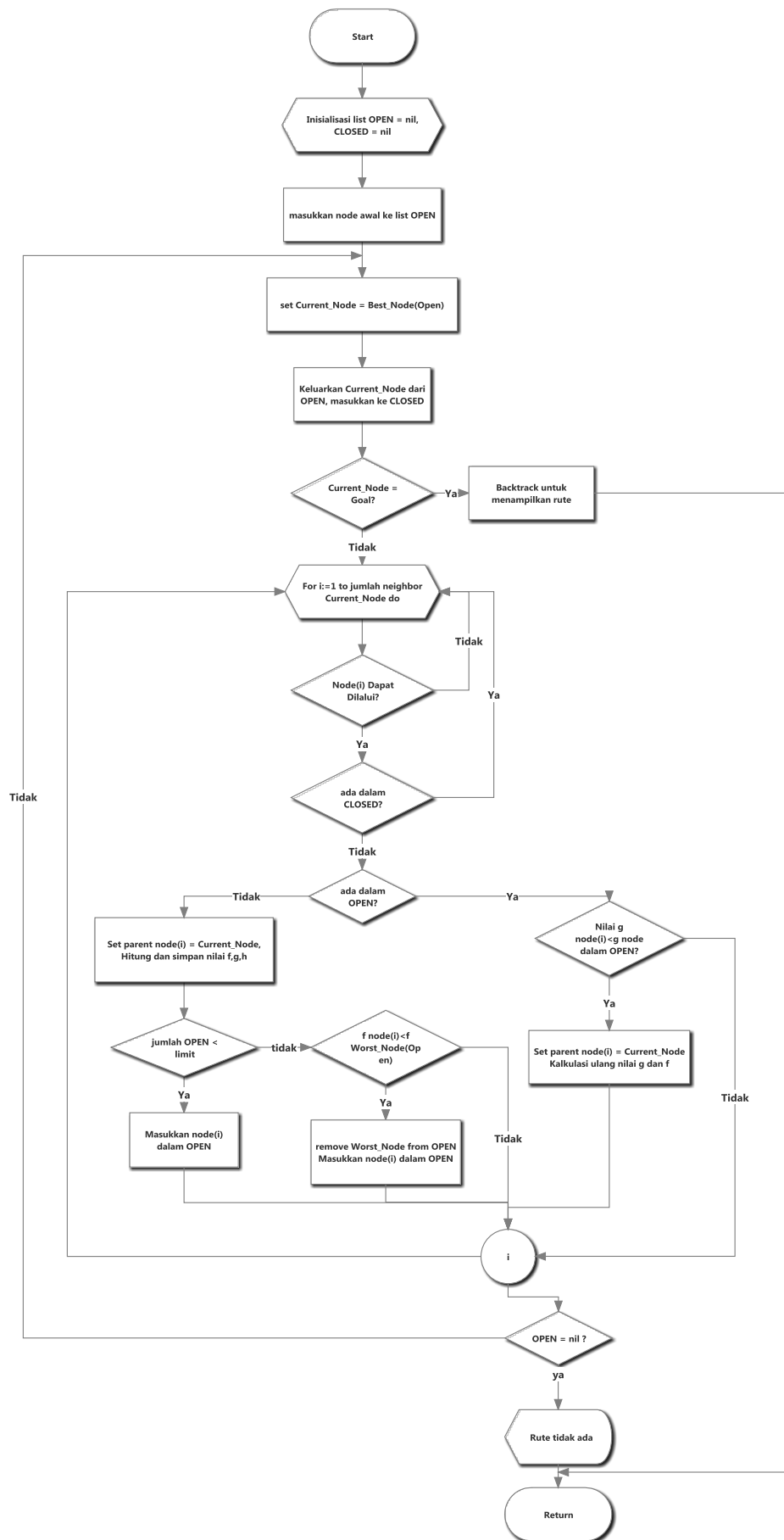
Pada setiap *level* setiap kali membunuh *monster*, *player* akan mendapatkan nilai yang sama yaitu 1 *point*, dan akan terus diakumulasikan sampai permainan selesai, jumlah ronde yang berhasil diselesaikan pemain juga akan diperhitungkan. Pada saat memberikan peringkat *highscore* jumlah ronde lebih diutamakan dibandingkan skor dari membunuh *monster*.

### 3.1.6 *Analisis Algoritma*

Dibawah ini adalah *flowchart* urutan langkah pencarian jalan dengan algoritma A\* gambar 3.1 dan SMA\* gambar 3.2.



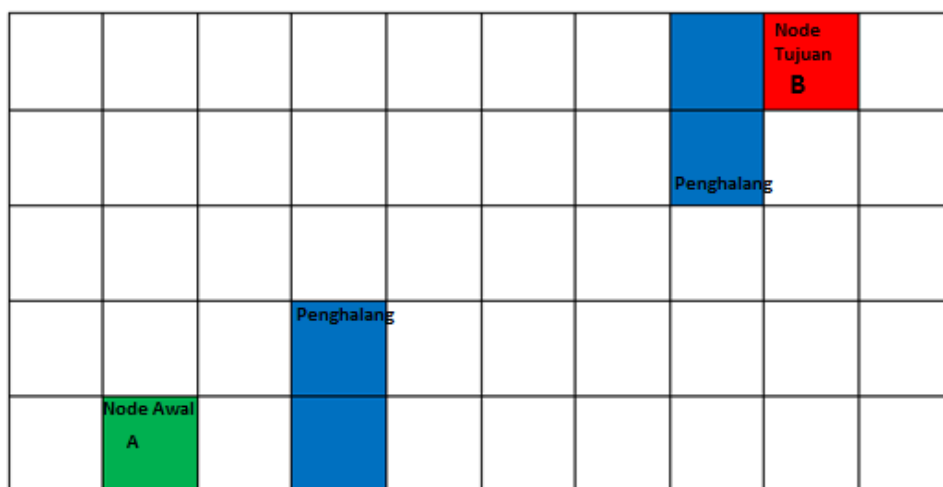
Gambar 3.1 Flowchart algoritma A\*



Gambar 3.2 Flowchart algoritma SMA\*

Dari *flowchart* pada gambar 3.1 dan gambar 3.2 dapat dilihat bahwa pada dasarnya prinsip kerja algoritma A\* dan SMA\* hampir sama, hanya saja pada SMA\* jumlah *node* di *open list* dibatasi dengan *limit* tertentu yang tetap, jika jumlah *node* di *open* melebihi dari *limit*, maka *node* dengan nilai terburuk di *open* akan dihapus dan *node* yang baru dengan nilai yang lebih baik akan dimasukkan ke dalam *open list*.

Untuk lebih memahami algoritma SMA\*, bisa dilihat dari contoh kasus. Seperti terlihat pada gambar 3.3. Diasumsikan dalam *game*, karakter ingin bergerak dari *point A* ke *point B*, kedua *point* dipisahkan oleh dinding. Untuk memudahkan setiap *node* diberi warna, warna hijau tua adalah titik mulai A, dan warna merah titik tujuan B, warna biru tua adalah dinding atau *collider*, warna biru muda adalah *node* yang berada di *closed list* dan warna hijau muda adalah *node* yang berada di *open list*, disini diasumsikan hanya tersedia 2 *node* untuk *open list*, .



Gambar 3.3 langkah 1 pencarian SMA\*

Mulai pencarian dari *point A* dan tambahkan *point A* ke *list*. *Generate suksesor* dari A, kemudian hitung  $f(s)$  nya dengan rumus:

$$f(s) = \text{MAX}(f(n), g(s) + h(s)) \quad (3.1)$$

dengan keterangan:

$f(s)$  = fungsi evaluasi *node* yg dicari

$f(n)$  = fungsi evaluasi *node parent*

$g(s)$  = biaya yang sudah dikeluarkan

$h(s)$  = estimasi biaya untuk sampai pada suatu tujuan

Untuk perhitungan nilai  $H$  digunakan fungsi *heuristic*, metode yang digunakan di dalam contoh ini adalah metode *Manhattan* yang dirumuskan dengan:

$$H = 10 * (\text{abs}(\text{currentX} - \text{targetX}) + \text{abs}(\text{currentY} - \text{targetY})) \quad (3.2)$$

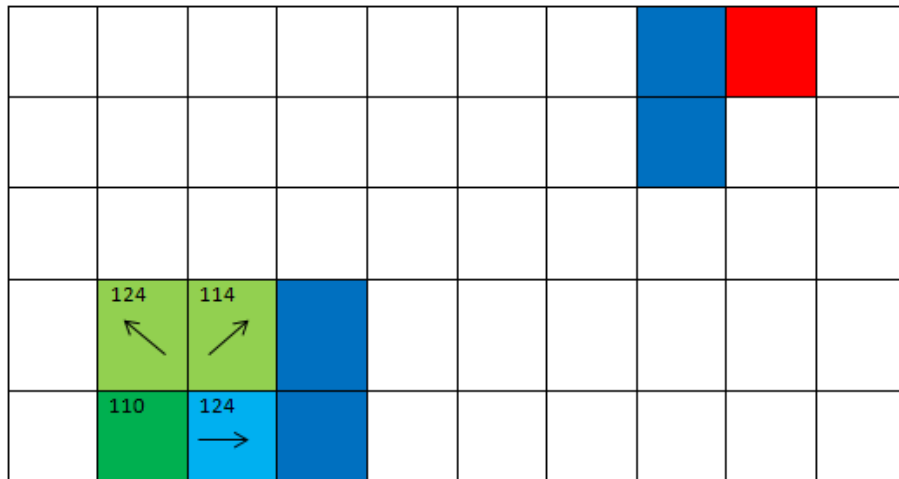
Sedangkan untuk perhitungan nilai  $g$  dilakukan dengan ketentuan, untuk pergerakan *horizontal* dan *vertical*, *cost*-nya adalah 10, sedangkan untuk pergerakan *diagonal* memakan *cost* 14.

Dengan  $f(a)=h(a)=110$ , karena  $g(n)=0$  biaya dari *initial state* ke *initial state* = 0, lalu masukkan suksesor ke dalam *open list*, disini suksesor hanya bisa digenerate 2 *node*, karena hanya tersedia 2 di *open list*. Ketika semua suksesor dari A sudah dibangkitkan ganti nilai  $f(A)$  dengan nilai *f-cost* terkecil dari nilai-nilai *f-cost* yang ada pada semua suksesornya, karena nilai *f-cost* yang terkecil adalah 110 yang sama dengan  $f(A)$  jadi tidak perlu diganti, lalu masukkan A ke *closed list* seperti terlihat pada gambar 3.4.

124	110	114							
130	110	110							

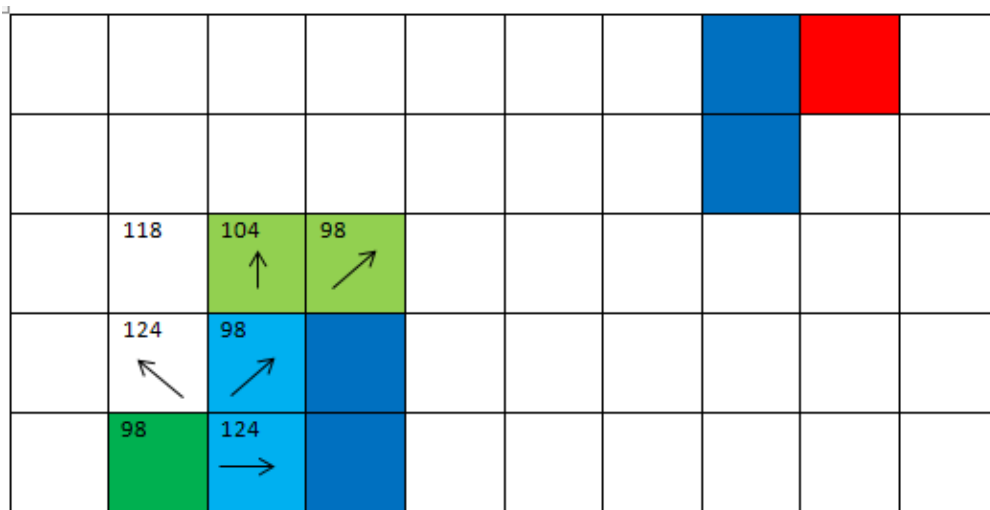
Gambar 3.4 langkah 2 pencarian SMA\*

Langkah selanjutnya adalah pilih *node* dengan *f-cost* terkecil sebagai *node* yang sedang di cek, *generate suksesor*-nya dan hitung *f-cost*-nya juga, pada gambar 3.5 *node* yang terkecil adalah dengan nilai 124.



Gambar 3.5 langkah 3 pencarian SMA\*

Sama seperti langkah sebelumnya dipilih *f-cost* yang terkecil yaitu 114, lalu *generate suksesor*-nya di dapatkan 2 *suksesor* baru, dan  $f(s)$  yang terkecil adalah 98 sehingga  $f(best)$  dari 114 diganti dengan 98, begitu juga dengan *ancestor*-nya, seperti terlihat pada gambar 3.6..



Gambar 3.6 langkah 4 pencarian SMA\*



Dipilih  $f\text{-cost}$  terkecil 98, langkah selanjutnya sama seperti diatas, untuk lebih jelasnya bisa dilihat pada gambar 3.7.

		112	98	92					
		104	92	98					
		92		112					
	92	124							

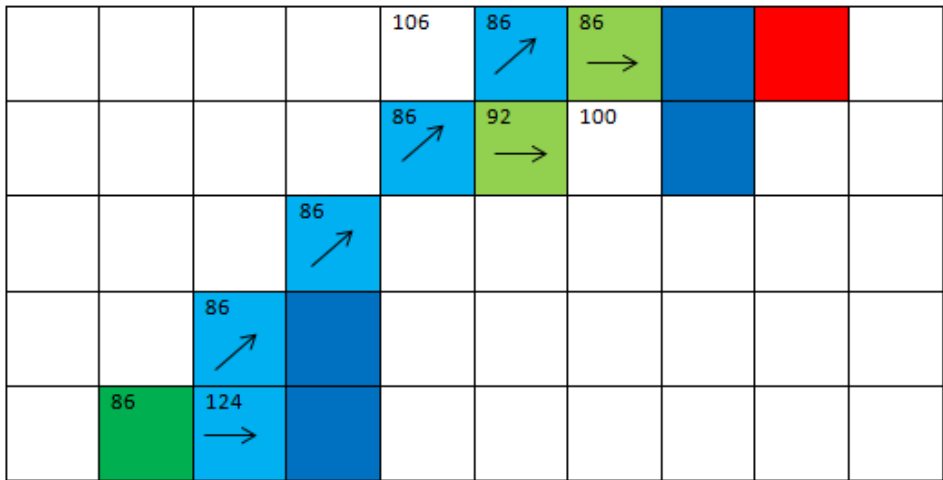
Gambar 3.7 langkah 5 pencarian SMA\*

Dipilih  $f\text{-cost}$  terkecil 92, didapatkan 2 *sukesor* baru dan ganti nilai  $f(best)$  dengan  $f(s)$  terkecil yaitu 86, seperti terlihat pada gambar 3.8.

			106	92	86				
			112	86	92				
			86	98	106				
		86							
	86	124							

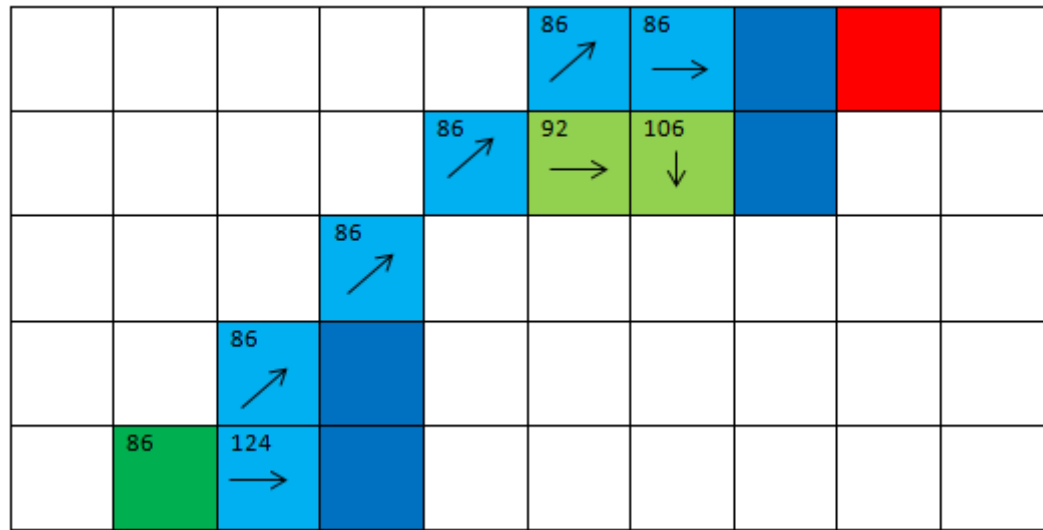
Gambar 3.8 langkah 6 pencarian SMA\*

Terlihat pada gambar 3.9 dipilih *f-cost* terkecil 86 dan didapatkan 1 *suksesor* baru lagi.



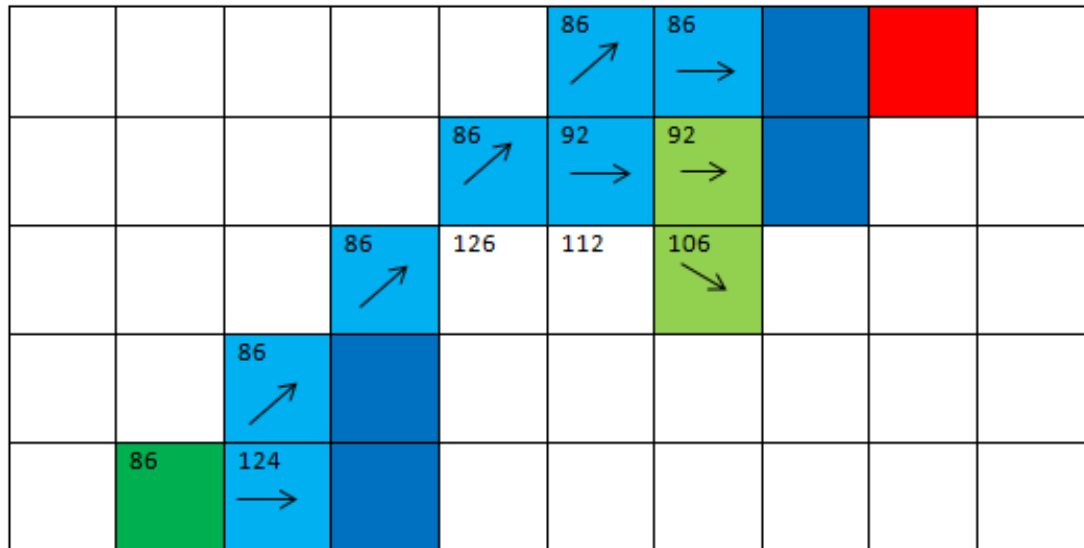
Gambar 3.9 langkah 7 pencarian SMA\*

Terlihat pada gambar 3.10 dipilih *f-cost* terkecil 86, didapatkan 1 *suksesor* baru



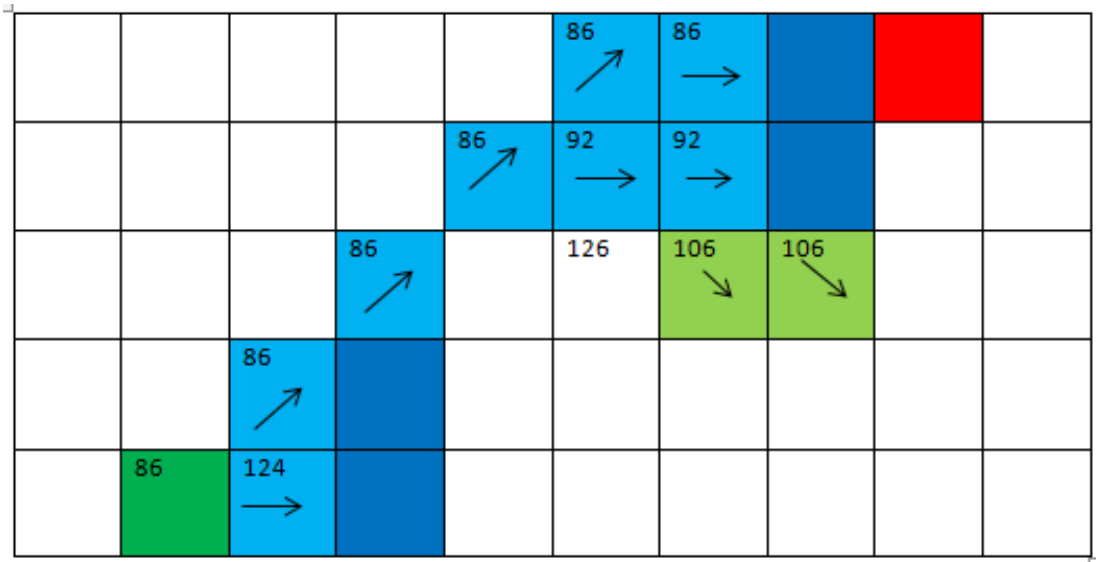
Gambar 3.10 langkah 8 pencarian SMA\*

Pada gambar 3.11 dipilih  $f$ -cost terkecil 92, didapatkan 2 *sukesor* baru, dengan penggantian *node* dengan  $f(s) = 106$  diganti dengan *node*  $f=92$  karena lebih kecil.



gambar 3.11 langkah 9 pencarian SMA\*

Terlihat pada dipilih  $f$ -cost terkecil 92, didapatkan 1 *sukesor* baru



Gambar 3.12 langkah 10 pencarian SMA\*

Dipilih  $f$ -cost terkecil 106, disini ada 2 *node* dengan  $f = 106$ , pilih *node* dengan kedalaman yang terdalam, didapatkan  $f(s)$  terkecil 100, ganti nilai  $f(n)$  dengan 100, seperti terlihat pada gambar 3.13.

					86 ↗	86 →			
				86 ↗	92 →	92 →		100 ↗	
			86 ↗			106 ↘	100 ↘	106 →	
		86 ↗							
	86 →	124 →							

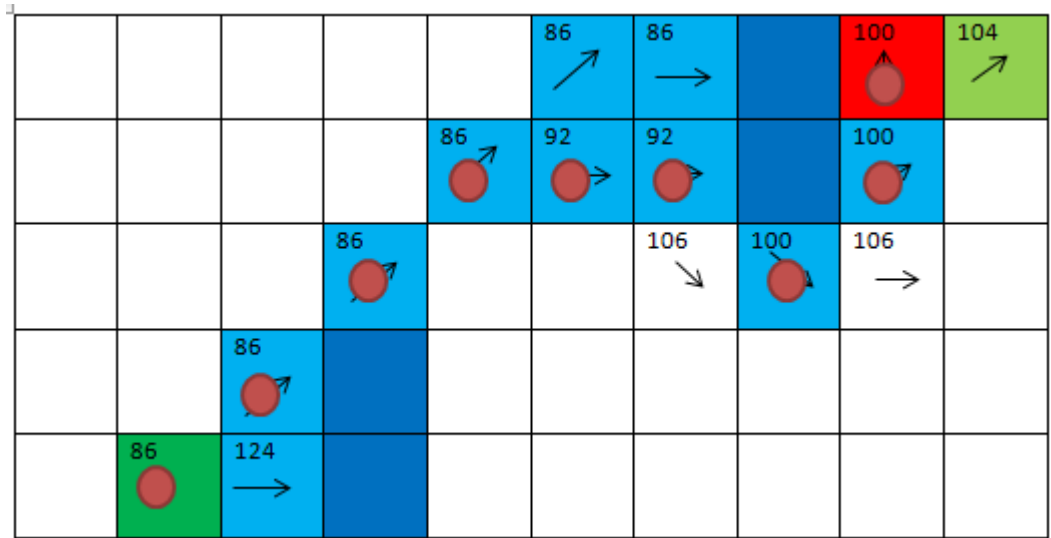
Gambar 3.13 langkah 11 pencarian SMA\*

Pada gambar 3.14 dipilih  $f$ -cost terkecil 100 dan didapatkan 2 *sukesor* baru.

					86 ↗	86 →		100 ↑	104 ↗
				86 ↗	92 →	92 →		100 ↗	
			86 ↗			106 ↘	100 ↘	106 →	
		86 ↗							
	86 →	124 →							

Gambar 3.14 langkah 12 pencarian SMA\*

*Node* selanjutnya yang terpilih merupakan tujuan, sehingga didapatkan jalur terpendek dari titik A menuju ke B, terlihat pada gambar 3.15 yang diberi tanda merah merupakan hasil dari pencarian jalur.



Gambar 3.15 langkah 13 pencarian SMA\*

### 3.2 Analisis Kebutuhan *Non-Fungsional*

Analisis kebutuhan *non-fungsional* menggambarkan kebutuhan luar sistem yang diperlukan untuk menjalankan aplikasi yang dibangun. Adapun kebutuhan *non-fungsional* pada implementasi algoritma SMA\* pada *game TPS Monster Nest* meliputi kebutuhan perangkat keras, kebutuhan perangkat lunak dan pemain sistem yang akan memakai aplikasi. Analisis kebutuhan *non-fungsional* bertujuan agar aplikasi yang dibangun dapat digunakan sesuai dengan kebutuhan.

#### 3.2.1 Analisis Perangkat Keras

Untuk menjalankan suatu aplikasi maka diperlukan perangkat keras yang dapat mendukung proses kerja dari sistem itu sendiri. Pada dasarnya *game* ini dapat dijalankan di semua perangkat *mobile android* dengan *touch screen* dengan *multitouch* tapi untuk kenyamanan sebaiknya dijalankan di tablet atau perangkat dengan ukuran layar minimal 7 *inchi*. Untuk lebih jelasnya bisa dilihat pada spesifikasi sebagai berikut :

Tabel 3.1 Analisis Perangkat Keras

No	Perangkat Keras	Spesifikasi / Keterangan
1	<i>Chipset</i>	<i>Hummingbird</i>
2	<i>CPU</i>	1 GHz <i>Cortex-A8</i>
3	<i>GPU</i>	<i>PowerVR SGX540</i>
5	<i>Type</i>	TFT <i>capacitive touchscreen</i> , 16M colors
6	<i>Size</i>	600 x 1024 <i>pixels</i> , 7.0 <i>inches</i> (~170 ppi <i>pixel density</i> )
7	<i>Multitouch</i>	<i>Yes</i>

### 3.2.2 Analisis Perangkat Lunak

Perangkat lunak atau *software* merupakan hal yang terpenting dalam mendukung kinerja sebuah sistem. Perangkat lunak digunakan dalam sebuah sistem merupakan perintah-perintah yang diberikan kepada perangkat keras agar dapat saling berinteraksi diantara keduanya. Perangkat lunak yang dibutuhkan untuk menjalankan aplikasi *game* ini adalah minimal sistem operasi *android* versi 2.3 *Gingerbeard*.

### 3.2.3 Analisis Pemain

Pada analisis *user* (pemakai) ini akan mencakup analisis beberapa parameter terhadap calon *user* dari aplikasi.

#### 1. *User Knowledge and Experience* dari target *user* yang akan menggunakan *game*.

*Game monster nest* ini bisa digunakan oleh kalangan apapun, tetapi pengetahuan dan pengalaman akan memudahkan *user* dalam bermainnya. Terutama pengetahuan dan pengalaman dalam memainkan *game action* atau *third person shooter*. Berikut ini klasifikasi *knowledge and experience* dari pemain aplikasi:

Tabel 3.2 Analisis klasifikasi *knowledge and experience*

<b><i>Educational level</i></b>	<b><i>Reading Level</i></b>	<b><i>Typing Skills</i></b>
<i>Game</i> ini bisa digunakan oleh berbagai kalangan, seperti pelajar, mahasiswa hingga masyarakat awam	<i>Game</i> ini bisa digunakan oleh berbagai <i>level</i> pendidikan dengan <i>reading level</i> tingkat menengah	<i>Game</i> ini tidak memerlukan <i>typing skills</i> yang tinggi
<b><i>Computer Literacy</i></b>	<b><i>Task Experience</i></b>	<b><i>System Experience</i></b>
<i>Game</i> ini bisa digunakan oleh pemain yang memiliki kemampuan komputer yang <i>moderate</i> (menengah)	<i>Game</i> ini bisa digunakan oleh pemain dengan pengalaman penggunaan komputer dan <i>game</i> yang <i>moderate</i> (menengah)	<i>Game</i> ini bisa digunakan oleh pengguna dengan pengalaman penggunaan komputer dan <i>game</i> yang <i>moderate</i> (menengah)
<b><i>Application Experience</i></b>	<b><i>Native Language</i></b>	<b><i>Use Of Other System</i></b>
<i>Game</i> ini bisa digunakan dalam semua sistem operasi Windows xp/vista/7	<i>Game</i> ini menggunakan satu bahasa, yakni Inggris	<i>Game</i> ini bisa di jalankan tanpa perlu memasang aplikasi lain

## 2. *Users Physical Characteristic*

Keadaan fisik seseorang mungkin akan berpengaruh pada penggunaan aplikasi *game* ini. Ada hal-hal yang harus diperhatikan juga terhadap user dari karakteristik fisiknya untuk dapat menggunakan aplikasi ini yaitu, *Color Blind*, *Handednes*, dan *Gender*.

Tabel 3.3 Analisis *Users Physical Characteristic*

Age	15 tahun keatas
Gender	Pria dan Wanita
Handedness	Kanan dan Kiri
Color Blind	<i>User</i> yang tidak bisa membedakan warna yang satu dengan yang lainnya (buta warna) masih mampu menggunakan aplikasi ini, karena tidak ada <i>indicator</i> warna-warna khusus yang membedakan antara fungsional yang satu dengan fungsional yang lainnya. Akan tetapi penggunaannya tidak akan optimal karena dalam <i>game</i> ini terdapat banyak sekali perbedaan warna yang menunjang interaksi dan ketertarikan dalam permainan.

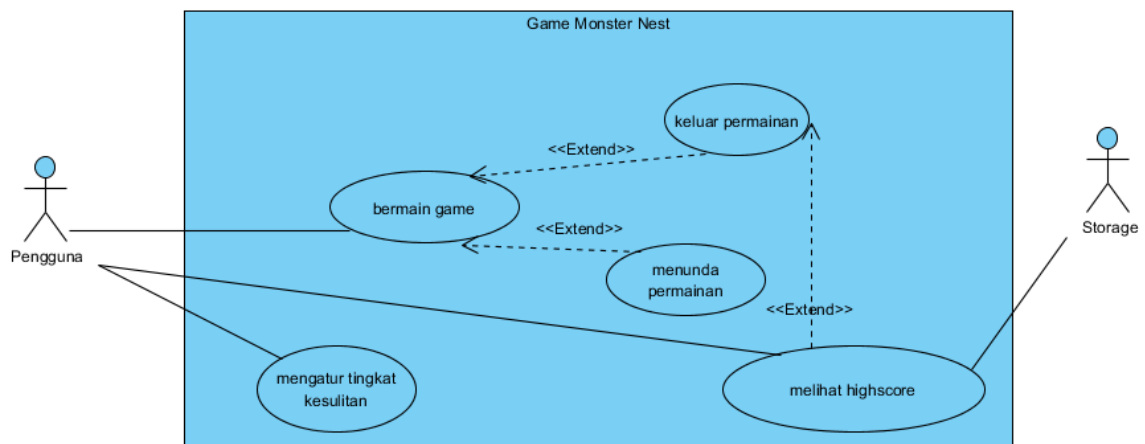
### 3.3 Analisis Kebutuhan Fungsional

Identifikasi aktor dapat dilakukan dalam analisis berorientasi objek dengan menggunakan UML yaitu menentukan aktor atau pemain sistem. Aktor dalam konteks UML menampilkan peran pemain atau sesuatu diluar sistem yang dikembangkan dapat berupa perangkat keras, *end user*, sistem yang lain dan sebagainya.



### 3.3.1 Use Case Diagram

*Use case* diagram merupakan pemodelan untuk mendeskripsikan interaksi antara satu atau lebih aktor dengan sistem yang akan dibuat. Berikut ini adalah *use case* diagram dari *game* ini:



Gambar 3.16 *Use Case* Diagram

#### 3.3.1.1 Definisi Aktor

Definisi Aktor berfungsi untuk menjelaskan aktor yang terdapat pada *use case* diagram. Definisi aktor diterangkan pada tabel 3.4 berikut.

Tabel 3.4 Definisi Aktor

No	Aktor	Deskripsi
1.	Pemain	Pemain atau orang yang memainkan <i>game</i> .

#### 3.3.1.2 Definisi *Use Case*

Definisi *use case* berfungsi untuk menjelaskan proses yang terdapat pada setiap *use case*. Definisi *use case* diterangkan pada tabel 3.5 :

Tabel 3.5 Definisi *Use Case*

No.	Use Case	Deskripsi
1	mengatur <i>level</i>	Pemain memilih " <i>Set Level</i> " dan akan ditampilkan <i>level</i> saat ini. Pemain dapat mengubah tingkat kesulitan dengan memilih menu " <i>easy</i> ", " <i>medium</i> ", " <i>hard</i> ".
2	bermain <i>game</i>	Pemain memilih " <i>Play Game</i> " dari menu utama. Tingkat permainan diperiksa dan kemudian tingkat kesulitan yang tepat ditampilkan dan <i>player</i> digerakkan sesuai dengan input dari <i>user</i> . Ada 4 darah, jika <i>player</i> terkena musuh, maka 1 darah akan berkurang. Jika darah terakhirnya habis, kemudian menuju ke <i>use case</i> keluar permainan. Jika <i>player</i> membunuh musuh, 1 poin diberikan dan ditampilkan. Pemain harus membunuh semua musuh yang ada, sebelum waktu habis untuk menyelesaikan permainan.
3	<i>pause game</i>	Pemain dapat menghentikan permainan saat bermain dan kemudian melanjutkannya lagi. Pilihan untuk keluar permainan juga ada disini.

No.	Use Case	Deskripsi
4	melihat <i>highscore</i>	Pemain memilih “ <i>Highscore</i> ” dari menu dan daftar tabel nama dan skor tertinggi akan ditampilkan. <i>Highscore</i> juga ditampilkan setelah <i>player</i> mati atau kehabisan 4 nyawanya.
5	keluar permainan	Pemain dapat keluar dari permainan selama bermain dan kemudian nilai tertinggi akan ditampilkan. Ketika semua darah telah habis, permainan akan keluar dan menampilkan nilai tertinggi. Jika <i>player</i> mendapatkan nilai yang tinggi, pilihan untuk memasukkan nama diberikan dan kemudian disimpan.

### 3.3.1.3 Skenario Use Case

Skenario *use case* merupakan bagian pada *use case* yang menunjukkan proses apa saja yang terjadi pada setiap bagian di dalam *use case*, dimana pemain memberikan perintah pada setiap bagian dan respon apa yang diberikan oleh sistem kepada pemain setelah pemain memberikan perintah pada setiap bagian-bagian *use case*. Berikut ini beberapa skenario *use case* berdasarkan *use case* yang ada yaitu :

Tabel 3.6 Mengatur tingkat kesulitan

Identifikasi	
Nomor	1
Nama	Mengatur tingkat kesulitan
Tujuan	Mengubah tingkat kesulitan dalam <i>game</i>
Deskripsi	<i>User interface</i> tempat untuk menentukan tingkat kesulitan dalam permainan. Kesulitan disesuaikan dengan memodifikasi kecepatan dan jumlah <i>monster</i> (musuh).
Aktor	Pemain
Skenario Utama	
Kondisi Awal	Program <i>game Monster Nest</i> sudah berjalan
Aksi Aktor	Reaksi Sistem
1. Pemain memilih “ <i>set level</i> ” pada <i>main menu</i>	
	2. Aplikasi mengambil tingkat kesulitan jika sebelumnya sudah tersimpan dalam <i>preferences</i>
	3. Menampilkan menu pilihan untuk mengubah tingkat kesulitan dan tingkat kesulitan yang sudah dipilih sebelumnya
4. Pemain memilih tingkat kesulitan yang diinginkan	
	5. Aplikasi mengubah tingkat kesulitan sesuai dengan pilihan pemain
Kondisi Akhir	Tingkat kesulitan berubah sesuai dengan pilihan dari pemain

Tabel 3.7 Bermain *Game*

Identifikasi							
Nomor	2						
Nama	Bermain <i>game</i>						
Tujuan	Pemain bermain <i>game</i> dengan menggerakkan karakter						
Deskripsi	Pemain menggerakkan karakter di dalam <i>game</i> untuk membunuh semua <i>monster</i> yang ada di arena.						
Aktor	Pemain						
Skenario Utama							
Kondisi Awal	Pemain telah memilih menu <i>play game</i> dan <i>game</i> siap dimainkan.						
<table border="1"> <thead> <tr> <th>Aksi Aktor</th><th>Reaksi Sistem</th></tr> </thead> <tbody> <tr> <td>1. Pemain menekan tombol yang disediakan untuk menggerakkan karakter di dalam <i>game</i></td><td></td></tr> <tr> <td></td><td>2. Karakter bergerak sesuai dengan tombol yang ditekan oleh pemain</td></tr> </tbody> </table>		Aksi Aktor	Reaksi Sistem	1. Pemain menekan tombol yang disediakan untuk menggerakkan karakter di dalam <i>game</i>			2. Karakter bergerak sesuai dengan tombol yang ditekan oleh pemain
Aksi Aktor	Reaksi Sistem						
1. Pemain menekan tombol yang disediakan untuk menggerakkan karakter di dalam <i>game</i>							
	2. Karakter bergerak sesuai dengan tombol yang ditekan oleh pemain						
Kondisi Akhir	Karakter bergerak sesuai dengan tombol yang ditekan oleh pemain sampai permainan berakhir						

Tabel 3.8 Menunda Permainan

Identifikasi		
Nomor	3	
Nama	Menunda permainan	
Tujuan	Menghentikan permainan untuk sementara	
Deskripsi	Pemain dapat menghentikan permainan dan kemudian melanjutkan permainan ketika pemain menekan tombol resume.	
Aktor	Pemain	
Skenario Utama		
Kondisi Awal	Permainan sudah berjalan dan sedang dimainkan	
Aksi Aktor		Reaksi Sistem
1. Pemain menekan tombol <i>pause</i>		
		2. Menampilkan menu yang menanyakan pemain untuk melanjutkan atau mengakhiri permainan
3. Pemain memilih menu yang diinginkan		
		4. Jilka pemain memilih <i>resume</i> maka permainan akan dikembalikan seperti sebelum tombol <i>pause</i> ditekan.
Kondisi Akhir	Permainan kembali seperti semula atau keluar dari permainan	

Tabel 3.9 Melihat *Highscore*

Identifikasi		
Nomor	4	
Nama	Melihat <i>highscore</i>	
Tujuan	Melihat skor tertinggi dari permainan yang sudah dilakukan sebelumnya	
Deskripsi	menampilkan 8 top skor tertinggi dari permainan sebelumnya. Kondisi ini muncul dengan memilih " <i>View Highscore</i> " dari menu utama. Hal ini juga terjadi secara otomatis setelah permainan berakhir.	
Aktor	Pemain	
Skenario Utama		
Kondisi Awal	<i>Game Monster Nest</i> sudah berjalan	
Aksi Aktor		Reaksi Sistem
1. Pemain memilih menu <i>View Highscore</i>		
		2. Aplikasi membaca data <i>Highscore</i> dari <i>preferences</i> yang tersimpan
		3. Menampilkan 8 skor tertinggi dengan nama pemainnya.
Kondisi Akhir	Skor tertinggi ditampilkan	

Tabel 3.10 Keluar Permainan

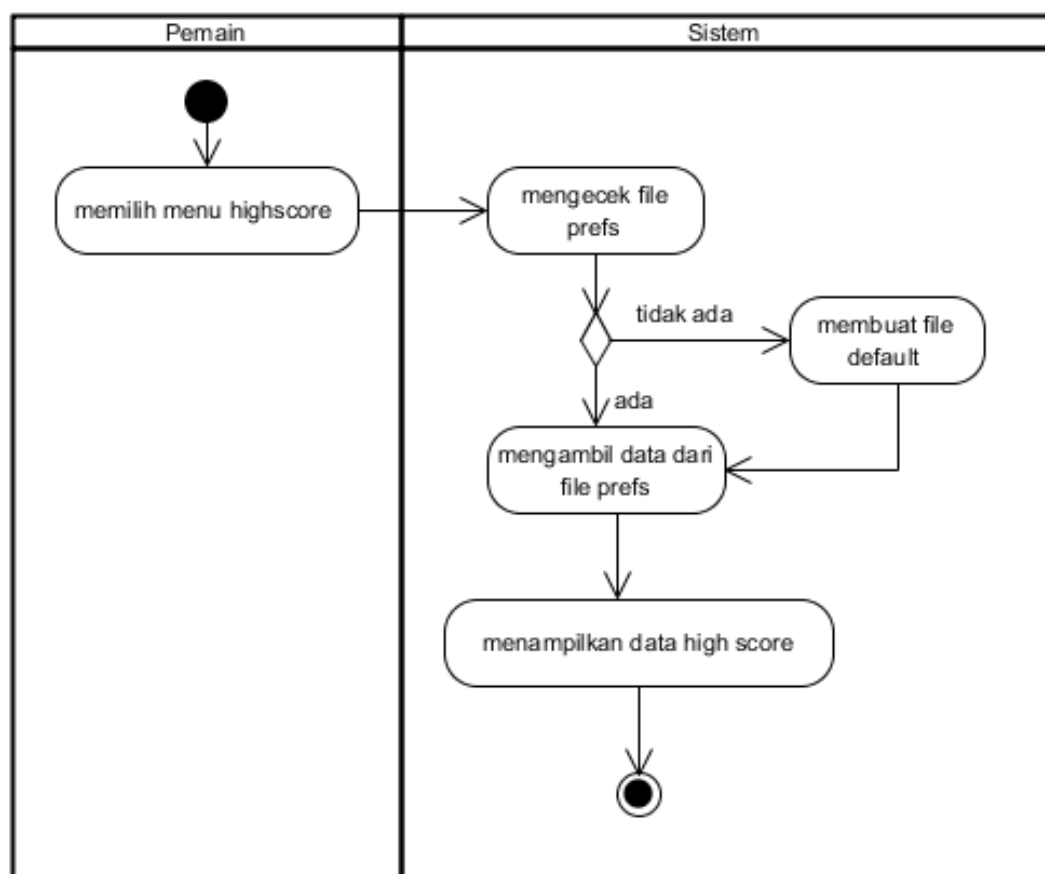
Identifikasi															
Nomor	5														
Nama	Keluar permainan														
Tujuan	Keluar dari permainan														
Deskripsi	Ketika permainan berakhir, program harus tahu mana langkah-langkah yang harus dilakukan berdasarkan pada jenis keluar yang dibuat oleh pemain														
Aktor	Pemain														
Skenario Utama															
Kondisi Awal	Pemain telah menekan tombol keluar, <i>player</i> kehilangan semua darahnya. Kita tahu mana dari skenario ini berlaku dan permainan selesai sesuai.														
<table border="1"> <thead> <tr> <th>Aksi Aktor</th><th>Reaksi Sistem</th></tr> </thead> <tbody> <tr> <td></td><td>1. Menampilkan <i>layer game over</i></td></tr> <tr> <td></td><td>2. Membandingkan skor yang baru dengan yang lama</td></tr> <tr> <td>3. Memasukkan Nama</td><td></td></tr> <tr> <td></td><td>4. Meng-<i>update</i> data <i>highscore</i> yang terbaru</td></tr> <tr> <td></td><td>5. Menampilkan menu utama</td></tr> <tr> <td>Kondisi Akhir</td><td>Menu utama ditampilkan dan <i>highscore</i> yang baru tersimpan</td></tr> </tbody> </table>		Aksi Aktor	Reaksi Sistem		1. Menampilkan <i>layer game over</i>		2. Membandingkan skor yang baru dengan yang lama	3. Memasukkan Nama			4. Meng- <i>update</i> data <i>highscore</i> yang terbaru		5. Menampilkan menu utama	Kondisi Akhir	Menu utama ditampilkan dan <i>highscore</i> yang baru tersimpan
Aksi Aktor	Reaksi Sistem														
	1. Menampilkan <i>layer game over</i>														
	2. Membandingkan skor yang baru dengan yang lama														
3. Memasukkan Nama															
	4. Meng- <i>update</i> data <i>highscore</i> yang terbaru														
	5. Menampilkan menu utama														
Kondisi Akhir	Menu utama ditampilkan dan <i>highscore</i> yang baru tersimpan														
Kondisi Akhir															
Menu utama ditampilkan dan <i>highscore</i> yang baru tersimpan															



### 3.3.2 Activity Diagram

Activity diagram menggambarkan berbagai aliran aktivitas dalam sistem yang sedang dirancang. Berikut ini beberapa *activity diagram* yang terdapat pada *game* ini yaitu :

#### A. Activity proses tampilan *highscore*

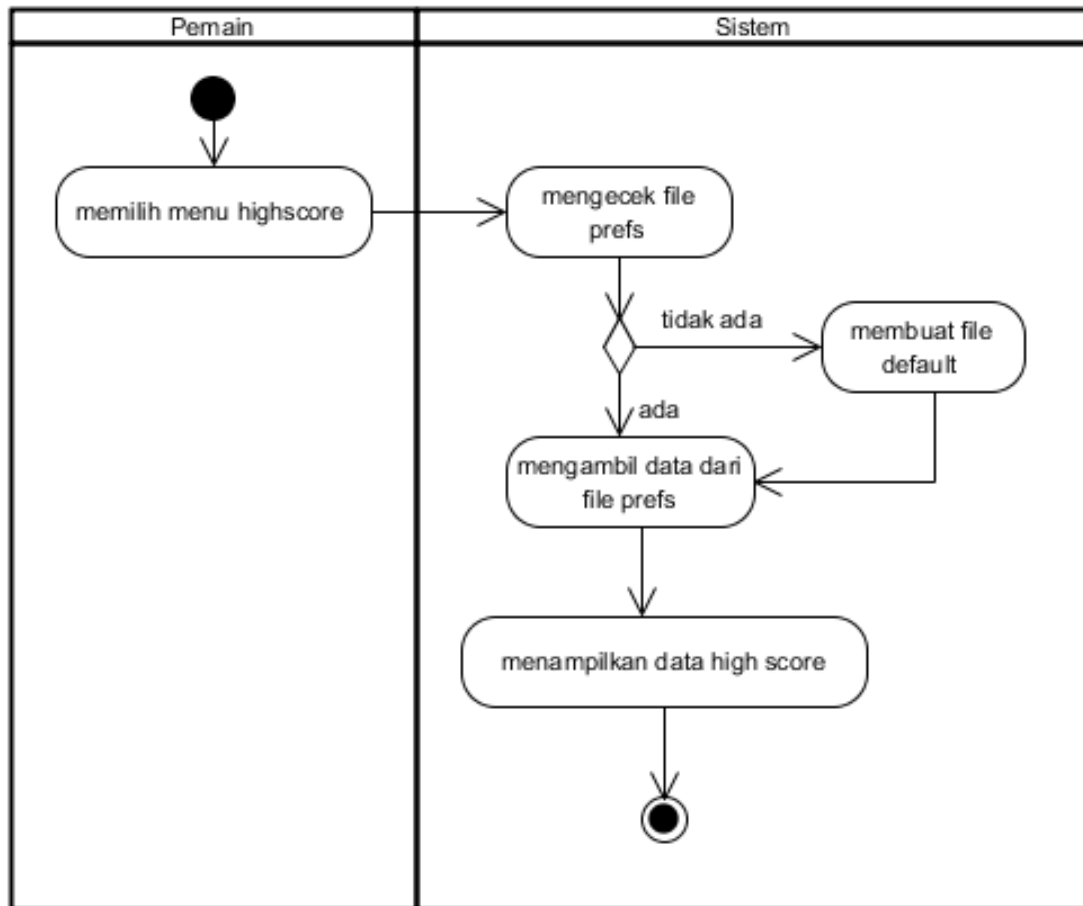


Gambar 3.17 Activity diagram proses tampilan *highscore*

Tabel 3.6 Penjelasan tampil high Score

Deskripsi	Menampilkan pilihan <i>highscore</i> , dimana pemain dapat melihat <i>history</i> berupa <i>highscore</i>
Normal Flow	A. Pemain memilih pilihan Tampilan <i>highscore</i> . B. Pemain keluar dari tampilan <i>highscore</i> .

B. Activity diagram memilih *level*

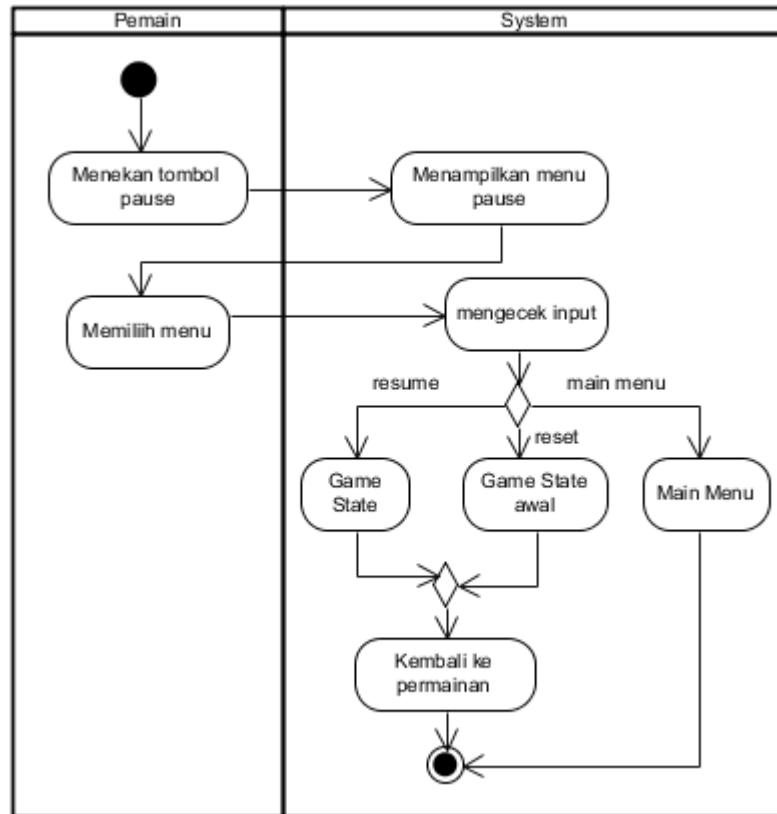


Gambar 3.18 Activity diagram proses memilih *level*

Tabel 3.7 Penjelasan proses memilih *level*

Deskripsi	Proses memilih tingkat kesulitan dalam permainan
Normal <i>Flow</i>	<ol style="list-style-type: none"> <li>1. Pemain memilih pilihan <i>set level</i>.</li> <li>2. Pemain memilih <i>level</i> yang ditampilkan sistem.</li> <li>3. Sistem akan menyimpan <i>level</i> yang dipilih kemudian kembali ke menu utama</li> </ol>

C. Activity diagram menunda permainan

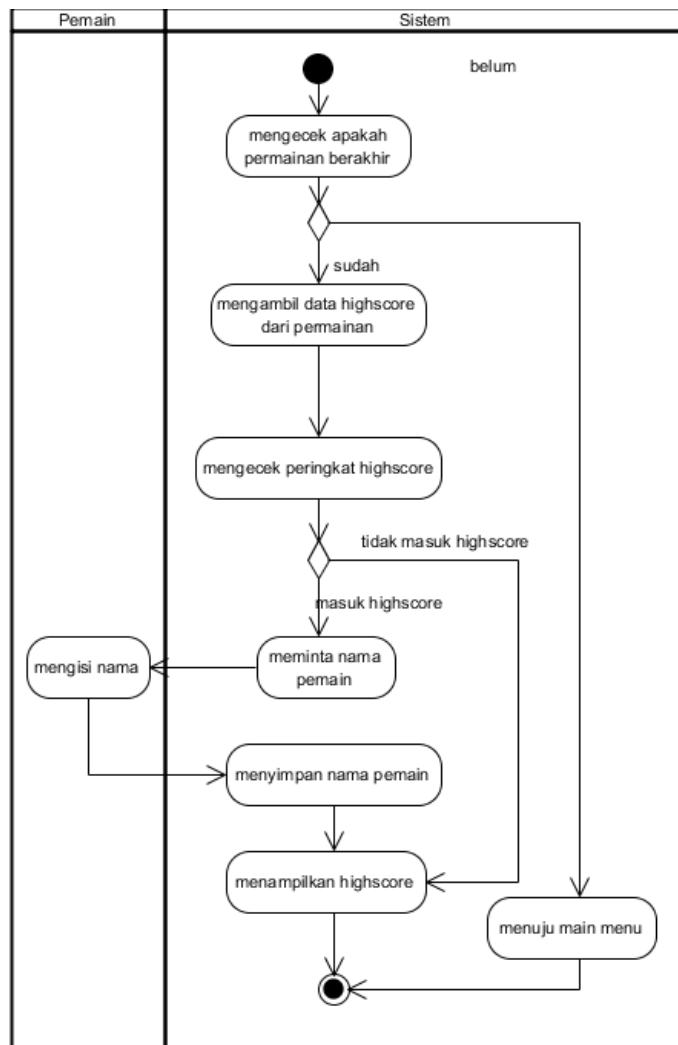


Gambar 3.19 Activity diagram menunda permainan

Tabel 3.8 Penjelasan menunda permainan

Deskripsi	Menjelaskan proses <i>pause</i> dalam permainan
Normal Flow	<ol style="list-style-type: none"> <li>1. Pemain menekan tombol <i>pause</i> yang ditampilkan pada layar <i>touch screen</i>.</li> <li>2. Pemain masuk ke dalam menu <i>pause</i>.</li> <li>3. Jika memilih <i>main menu</i> pemain mengakhiri permainan dan masuk <i>main menu</i></li> <li>4. Jika memilih <i>resume</i> dan <i>reset</i> pemain kembali ke permainan</li> </ol>

## D. Activity diagram keluar permainan

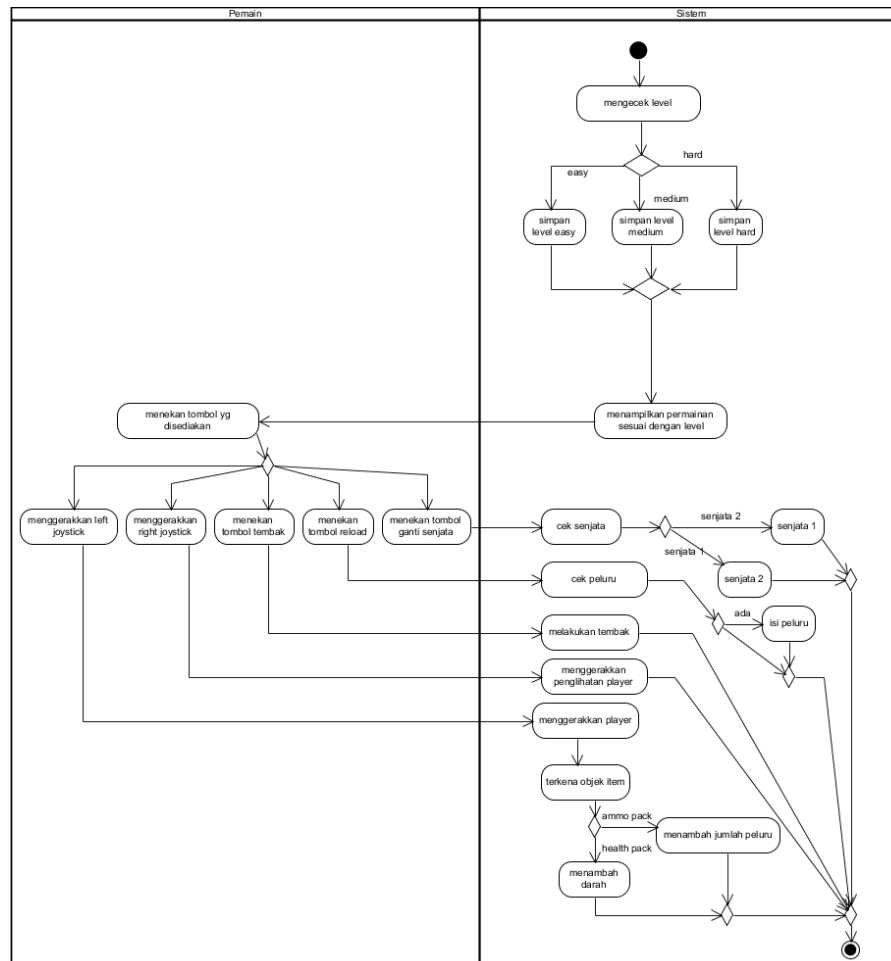


Gambar 3.20 Activity diagram keluar permainan

Tabel 3.9 Penjelasan keluar permainan

Deskripsi	Menjelaskan proses keluar dari permainan
Normal Flow	<ol style="list-style-type: none"> <li>1. Sistem mengecek apakah permainan berakhir</li> <li>2. Jika skor pemain masuk ke dalam <i>highscore</i> maka tambahkan skor</li> <li>3. Pemain mengisi nama</li> <li>4. Daftar <i>highscore</i> ditampilkan</li> </ol>

## E. Activity diagram bermain



Gambar 3.21 Activity diagram bermain

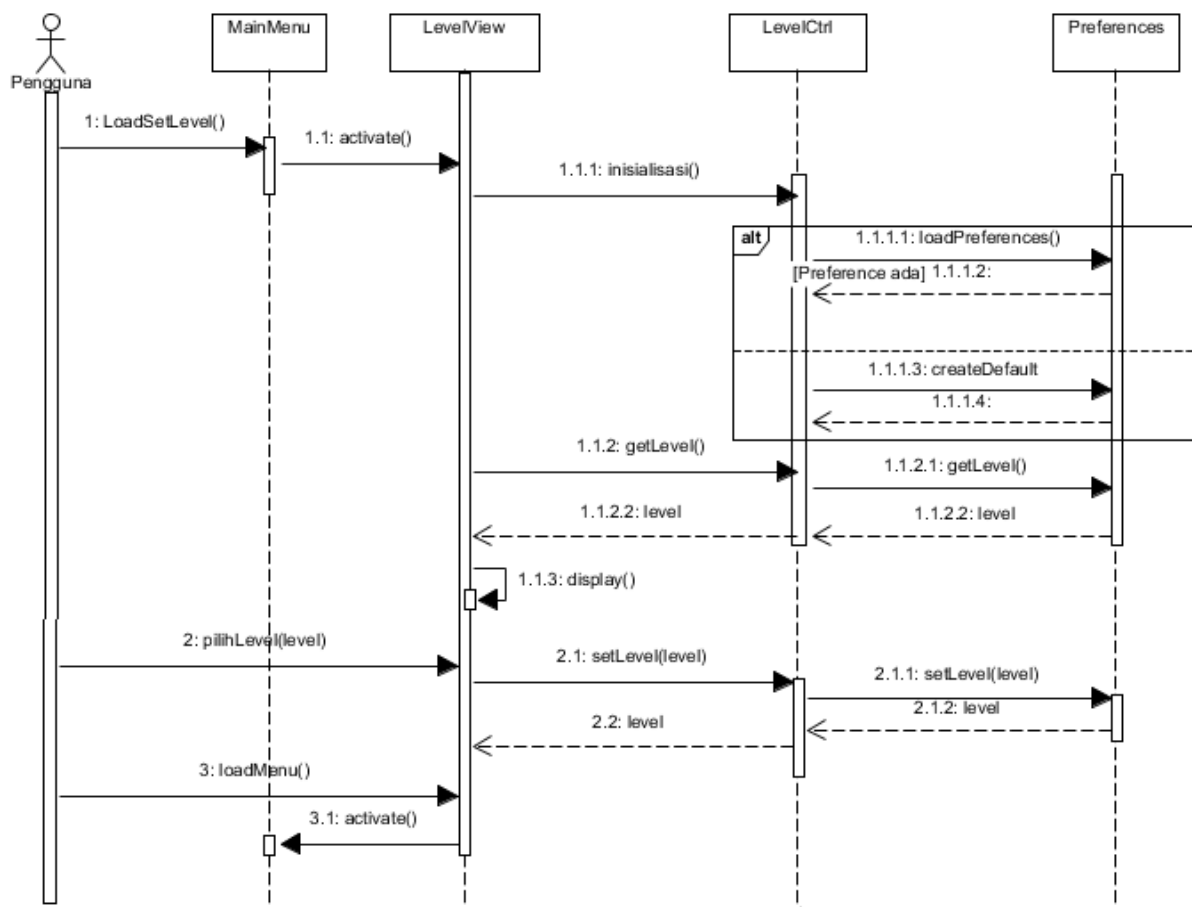
Tabel 3.10 Penjelasan bermain

Deskripsi	Menjelaskan proses keluar dari permazian
Normal Flow	<ol style="list-style-type: none"> <li>1. Sistem mengecek <i>level</i> permainan yang dipilih sebelumnya</li> <li>2. Sistem menampilkan permainan</li> <li>3. Pemain menekan tombol yang disediakan</li> <li>4. Sistem merespon berdasarkan tombol yang dipilih oleh pemain</li> </ol>

### 3.3.3 Sequential Diagram

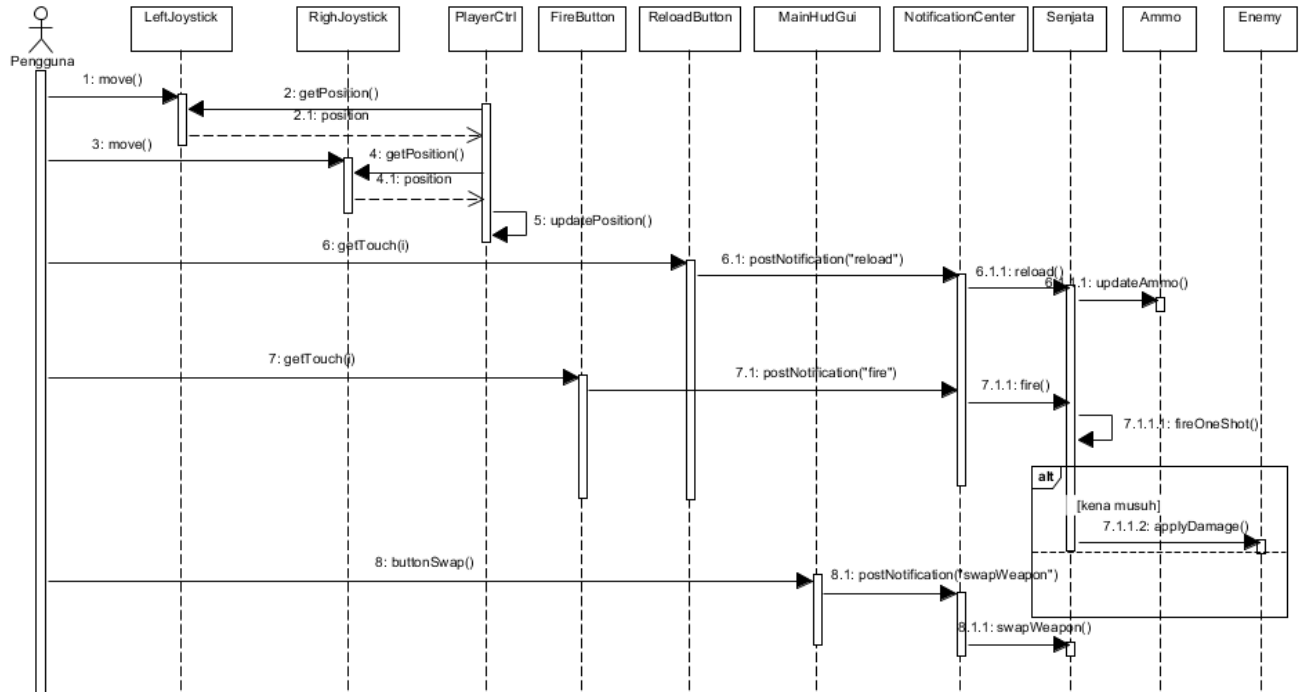
*Sequential* diagram menggambarkan interaksi antar objek di dalam dan disekitar sistem berupa *message* yang digambarkan terhadap waktu. Berikut ini beberapa *sequential* diagram yang terdapat pada *game* tersebut yaitu :

#### A. Sequential Diagram Mengatur Tingkat Kesulitan



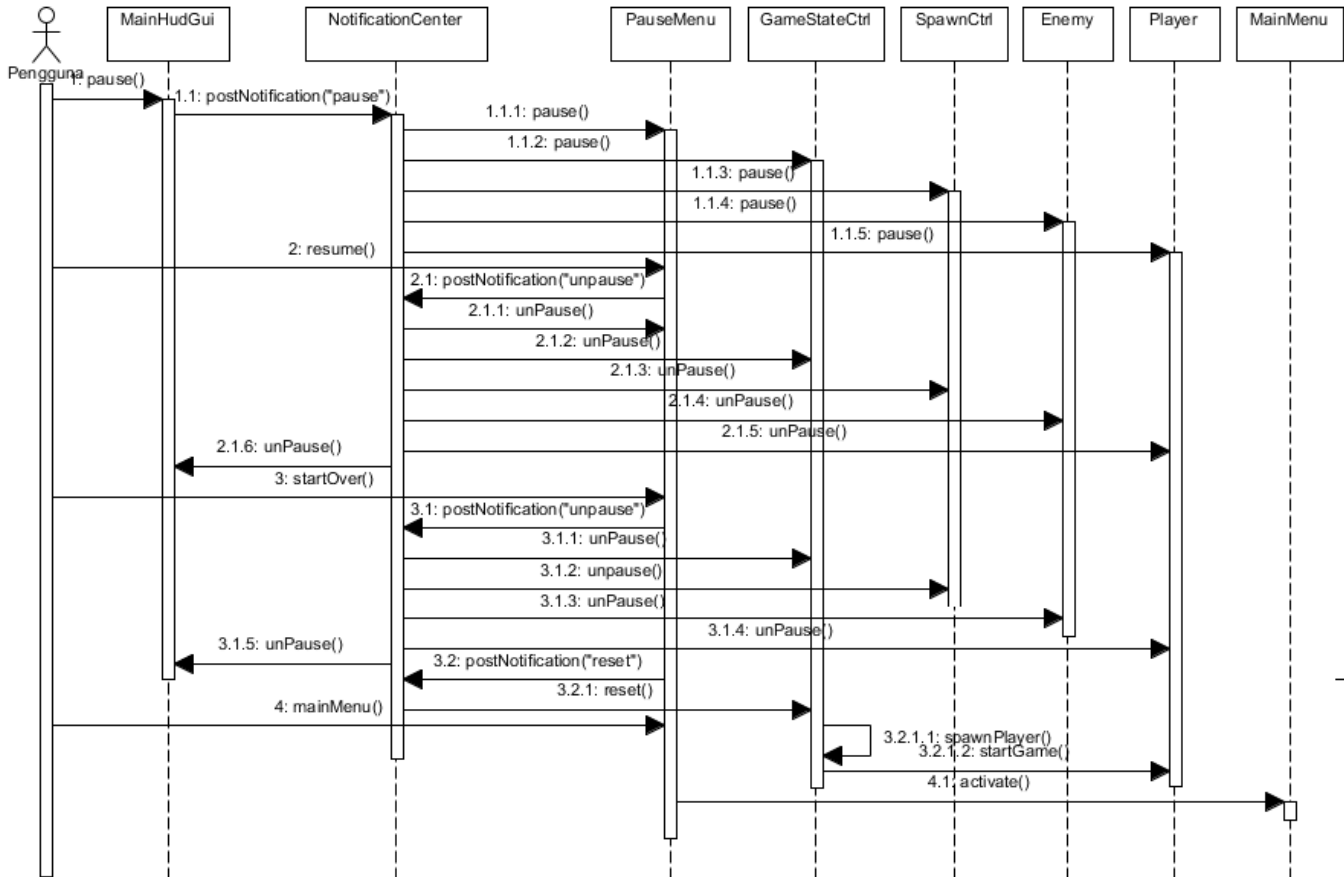
Gambar 3.22 *Sequential* Diagram Mengatur Tingkat Kesulitan

### B. Sequential Diagram Bermain Game



Gambar 3.23 Sequential Diagram Bermain Game

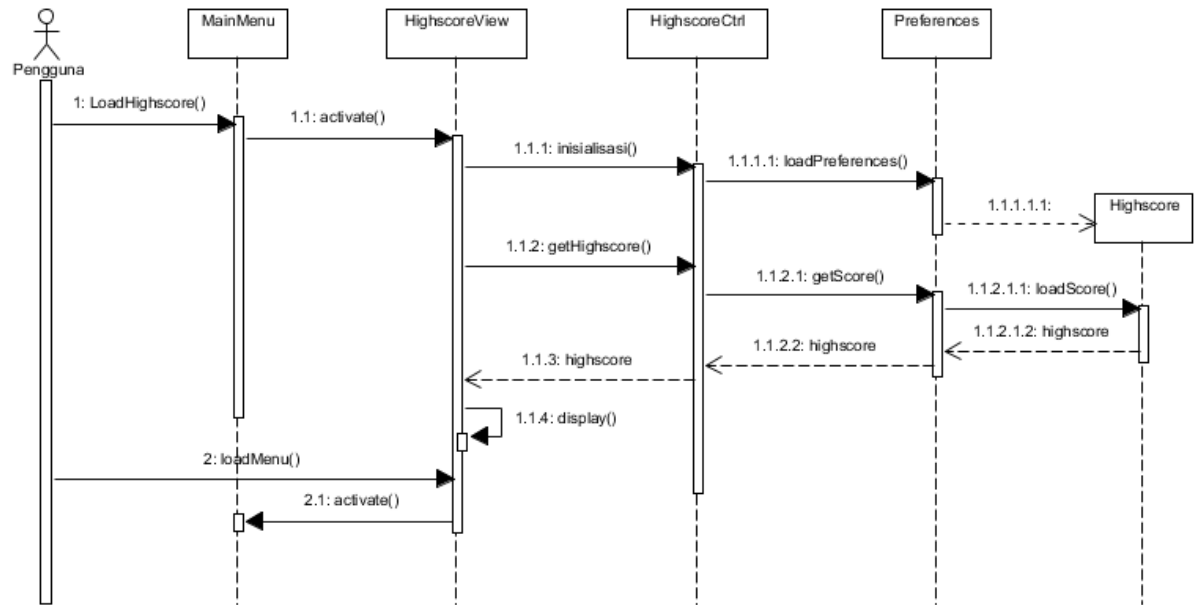
### C. *Sequential* Diagram Menunda Permainan



Gambar 3.24 *Sequential* Diagram Menunda Permainan

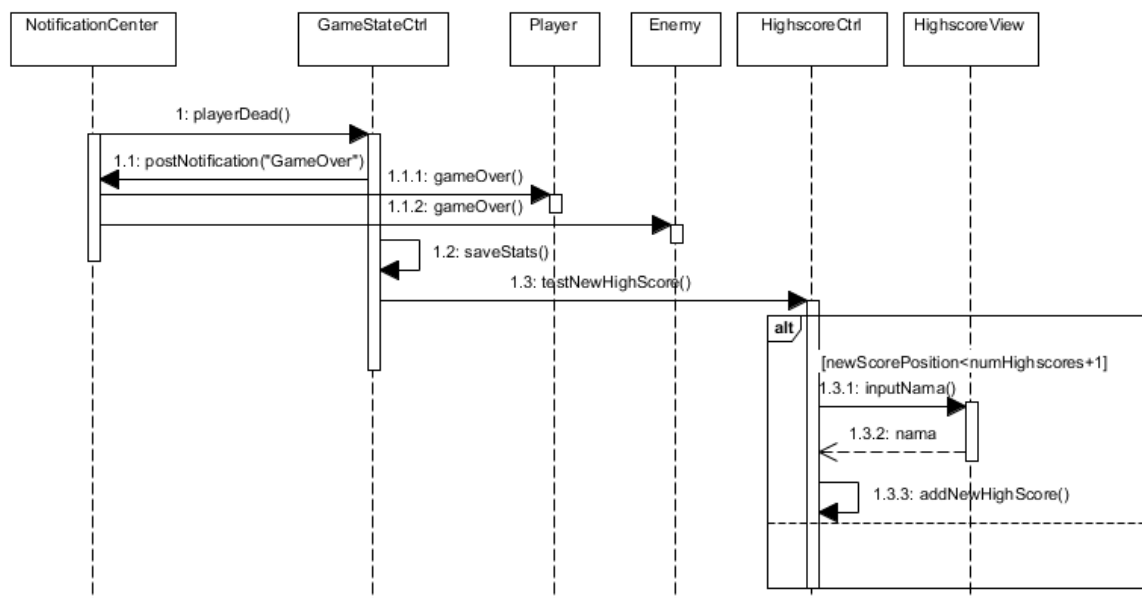


#### D. Sequential Diagram Melihat *Highscore*



Gambar 3.25 Sequential Diagram Melihat *Highscore*

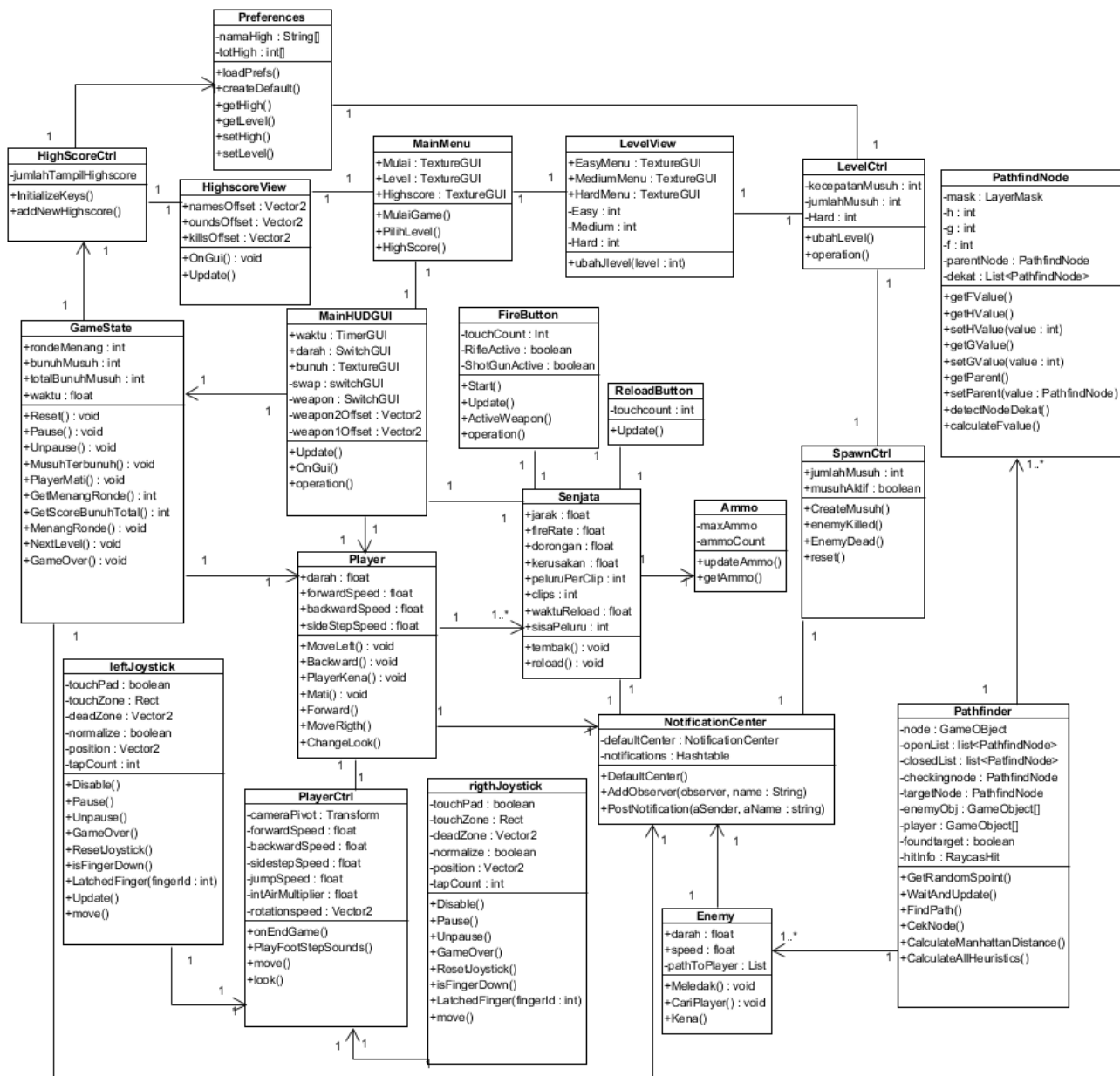
#### E. Sequential Diagram Keluar Permainan



Gambar 3.26 Sequential Diagram Keluar Permainan

### 3.3.4 Class Diagram

Class diagram menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang dibuat untuk membangun sistem. Berikut adalah class diagram pada game *Monster Nest* :



Gambar 3.27 Class Diagram

### 3.4 Perancangan Sistem

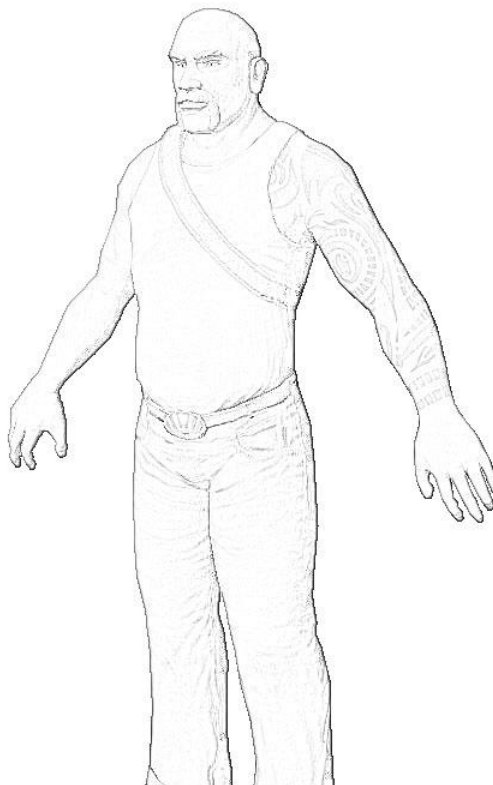
Perancangan sistem merupakan metodologi pengembangan suatu perangkat lunak yang dilakukan setelah melalui tahapan analisis. Dalam tahap ini digambarkan rancangan sistem yang akan dibangun sebelum dilakukan pengkodean ke dalam suatu bahasa pemrograman.

#### 3.4.1 Perancangan Komponen Permainan

##### 3.4.1.1 Karakter

###### A. Karakter Utama

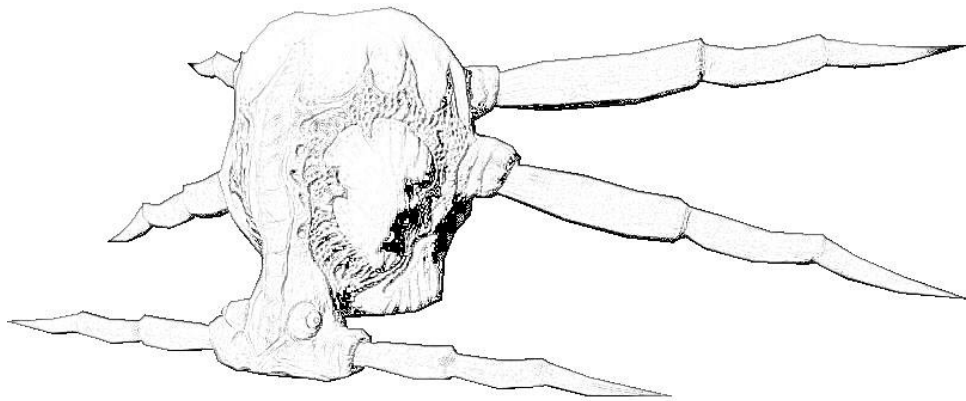
Karakter ini memiliki badan besar kekar, kepala botak dan sikap buruk. Di tangan kanannya ada perban berdarah dan kaosnya yang kotor dan penuh keringat, menunjukkan dia sudah berjuang untuk membunuh *monster*. Dia memiliki tali di punggungnya yang memungkinkan dia untuk berganti senjata ketika ia membutuhkan senjata lebih. Dia memakai celana jeans dan sepatu bot tua yang tertutup kotoran dari lantai terowongan bawah tanah.



Gambar 3.28 Karakter utama

### *B. Monster*

Makhluk kecil ini memenuhi terowongan kereta bawah tanah tua. Memiliki tinggi selutut orang dewasa, berwarna merah, berkaki enam, makhluk ini cepat dan agresif dan dapat membunuh dengan cairan asam korosif berwarna biru. Ada banyak tempat makhluk ini bersembunyi dan mereka menyerang dengan melemparkan cairan beracun dalam tubuhnya. Mereka dapat dibunuh dengan senjata api dan ketika ditembak, asam dalam tubuh mereka akan menyebabkan mereka menguap.



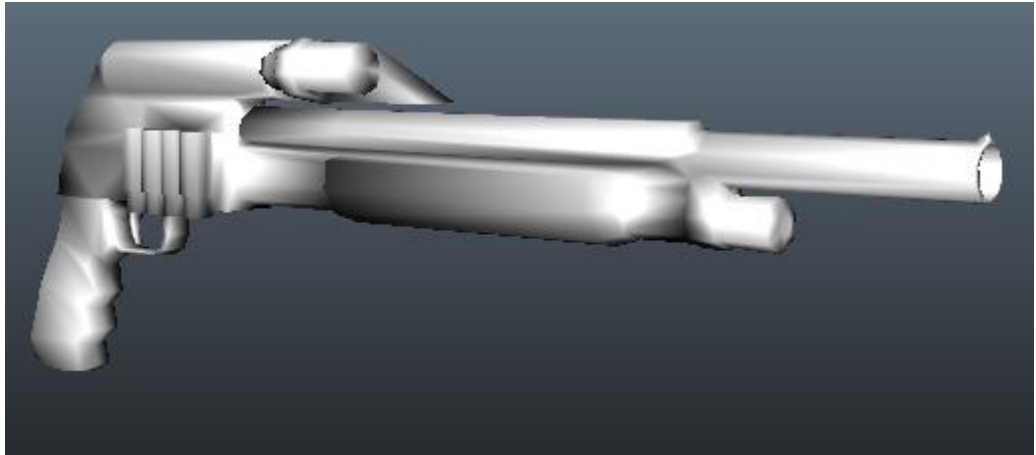
Gambar 3.29 *Monster*

### *C. Senjata*

Ada dua macam senjata yang dapat digunakan, yang pertama adalah *assault riffle* yang dapat dilihat pada gambar 3.30 dan yang kedua adalah *shotgun* yang dapat dilihat pada gambar 3.30.



Gambar 3.30 *Assault riffle*

Gambar 3.31 *Shotgun*

### Perbandingan Senjata

Perbandingan dari dua senjata yang ada dijelaskan pada tabel 3.11 dibawah ini.

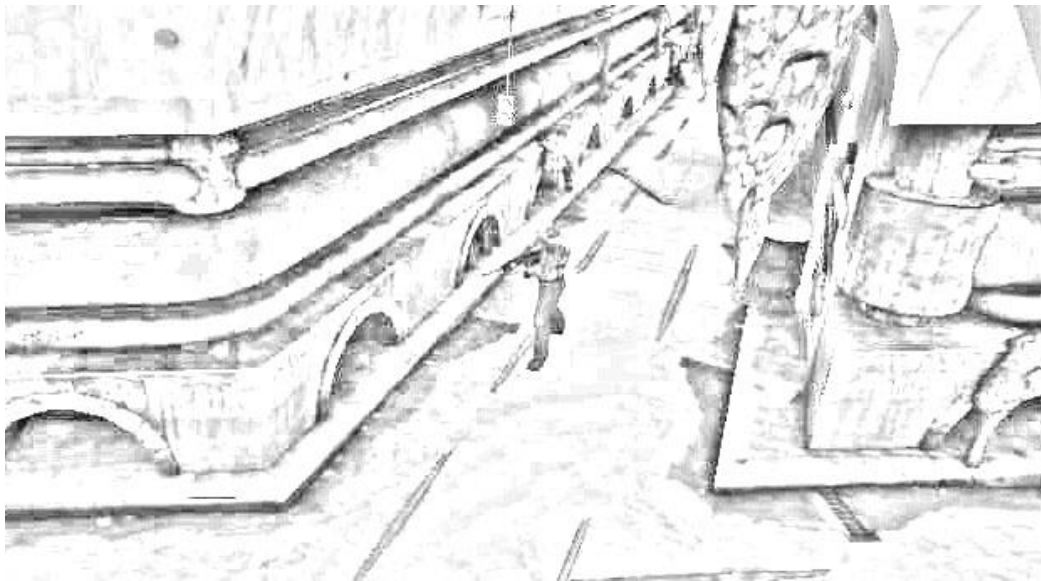
Tabel 3.11 Perbandingan senjata

<i>Properties</i>	<i>Assault Rifle</i>	<i>Shotgun</i>
<i>Damage</i>	10	30
<i>Range</i>	100	100
<i>Reload Time</i>	0.5 detik	2 detik
<i>Bullet</i>	15	15
<i>Clip</i>	10	20

#### 3.4.1.2 Storyboard

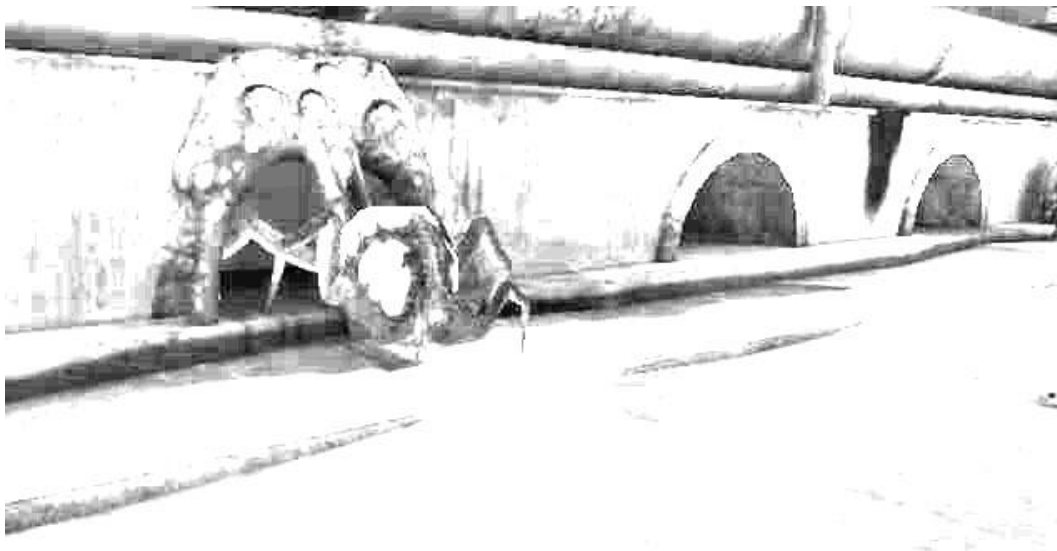
*Storyboard* adalah sketsa gambar yang disusun berurutan sesuai dengan naskah, dengan menggunakan *storyboard* dapat menyampaikan ide cerita kepada orang lain dengan lebih mudah karena dapat dengan *storyboard* dapat mengimajinasikan khayalan yang sesuai dengan gambar-gambar yang ada sehingga dapat menghasilkan persepsi yang sama pada ide cerita yang dibuat. Berikut beberapa *storyboard* dari Aplikasi *Game TPS monster nest* yaitu :

Pada gambar 3.32 terlihat *player* berdiri di sebuah, terowongan kereta api tua yang sudah menjadi sarang *monster*



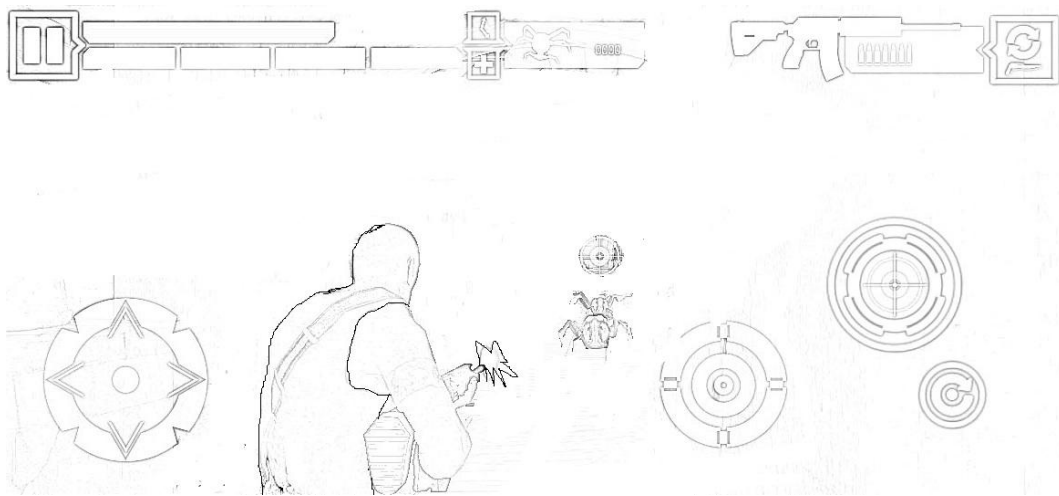
Gambar 3.32 *Storyboard* posisi awal *player*

Satu - persatu *monster* akan keluar dari tempat persembunyiannya dan siap untuk menyerang *player* yang telah mengganggu sarang mereka, seperti terlihat pada gambar 3.33.



Gambar 3.33 *Storyboard* *monster* keluar dari sarang

*Player* harus membunuh semua *monster* yang ada di hadapannya untuk bertahan hidup, agar bisa keluar dari sarang tersebut hidup-hidup, tampak pada gambar 3.34 *player* sedang menembaki *monster*.



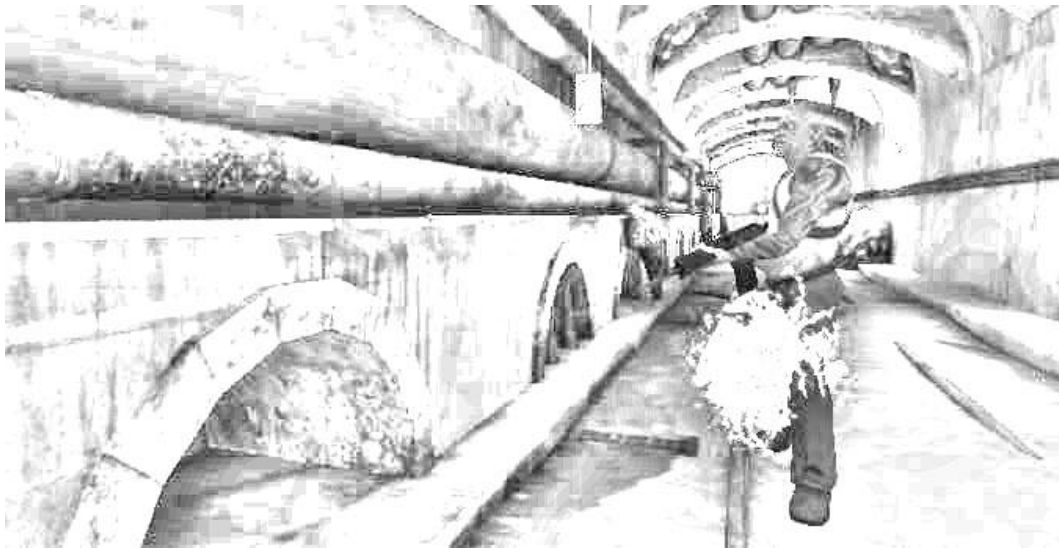
Gambar 3.34 *Storyboard monster nest player menembak*

Ketika sudah dekat dengan *player monster* akan melompat dan meledakkan dirinya, seperti terlihat pada gambar 3.35.



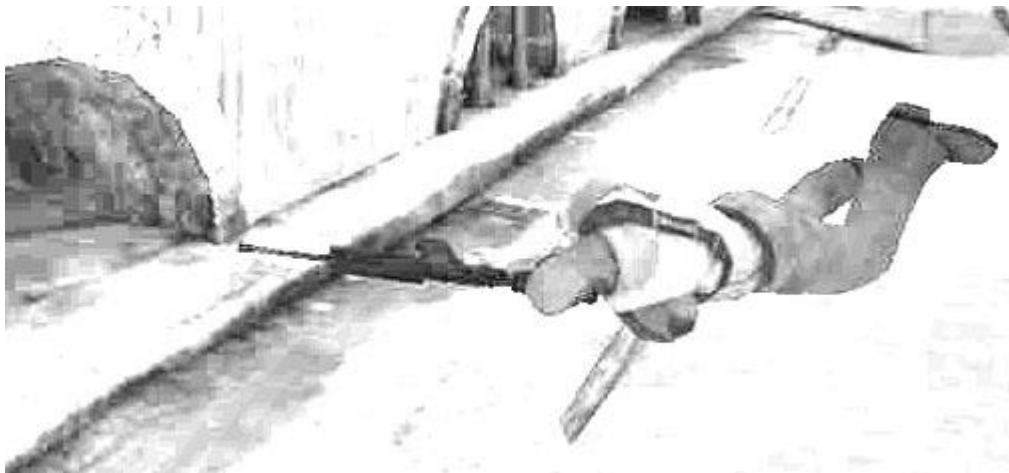
Gambar 3.35 *Storyboard monster nest diserang musuh*

Pada gambar 3.36 terlihat ketika *player* sedang diserang musuh dan musuh meledakkan dirinya dan mengeluarkan racun yang bisa membunuh *player*.



Gambar 3.36 *Storyboard monster nest* terkena serangan musuh

Ketika *player* telah kehabisan nyawanya, maka *player* pun akan mati dan tergeletak di lantai, seperti terlihat pada gambar 3.37



Gambar 3.37 *Storyboard monster nest* player mati

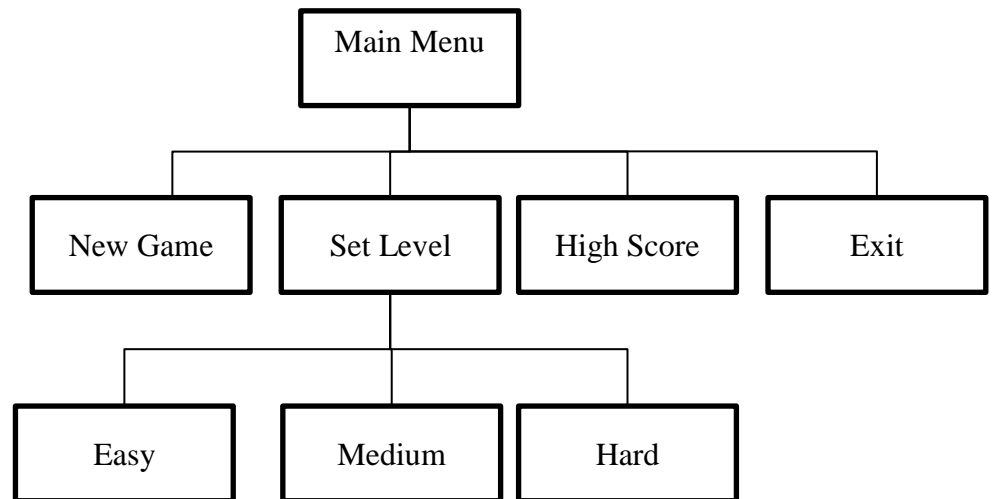
### 3.4.2 Perancangan Arsitektur

Setelah melakukan perancangan data pada sistem yang dibangun, maka dilakukanlah perancangan arsitektur. Perancangan arsitektur yang telah dibuat meliputi beberapa perancangan di antaranya struktur menu, perancangan antarmuka,



### 3.4.2.1 Perancangan Struktur Menu

Perancangan struktur menu dirancang untuk menyediakan fungsi-fungsi yang akan digunakan dalam *game Monster Nest*.



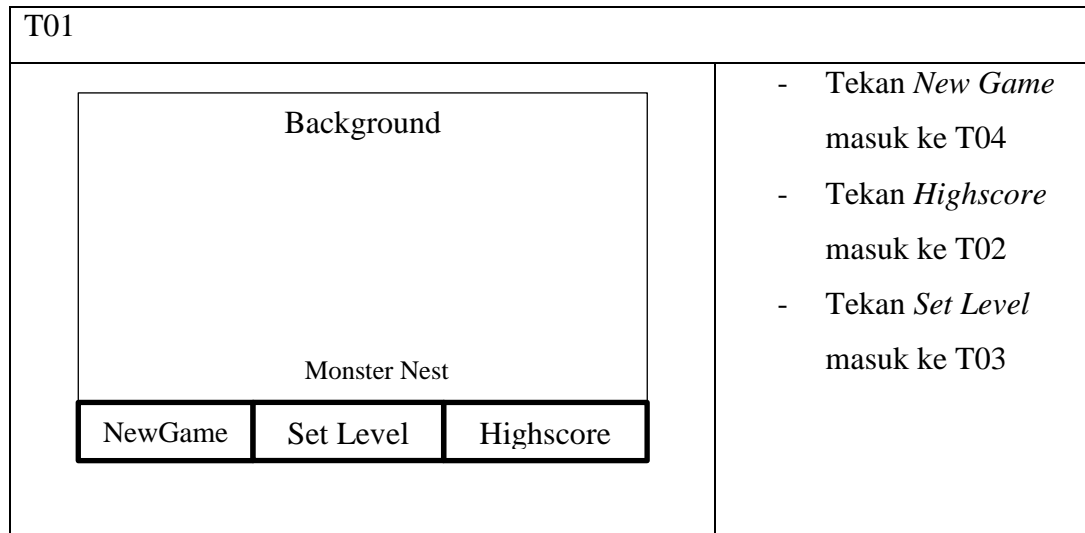
Gambar 3.38 Struktur menu

### 3.4.3 Perancangan Antarmuka

Perancangan antarmuka dibutuhkan untuk mewakili keadaan sebenarnya dari aplikasi yang akan dibangun. Berikut ini beberapa perancangan antarmuka dari aplikasi yang akan dibangun yaitu :

### 1. Menu Utama

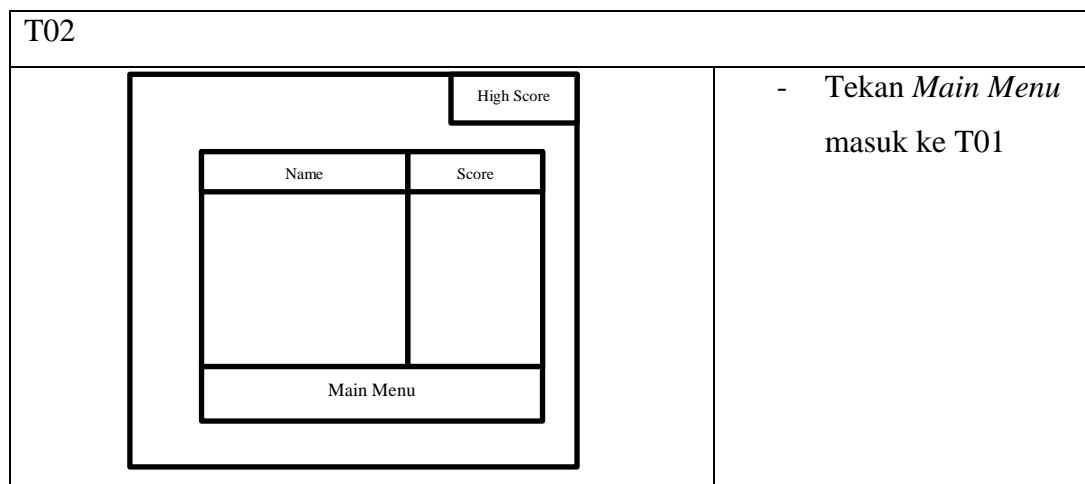
Berikut dapat dilihat perancangan antarmuka menu utama dari pembangunan aplikasi game *monster nest*.



Gambar 3.39 Perancangan Menu Utama

### 2. *HighScore*

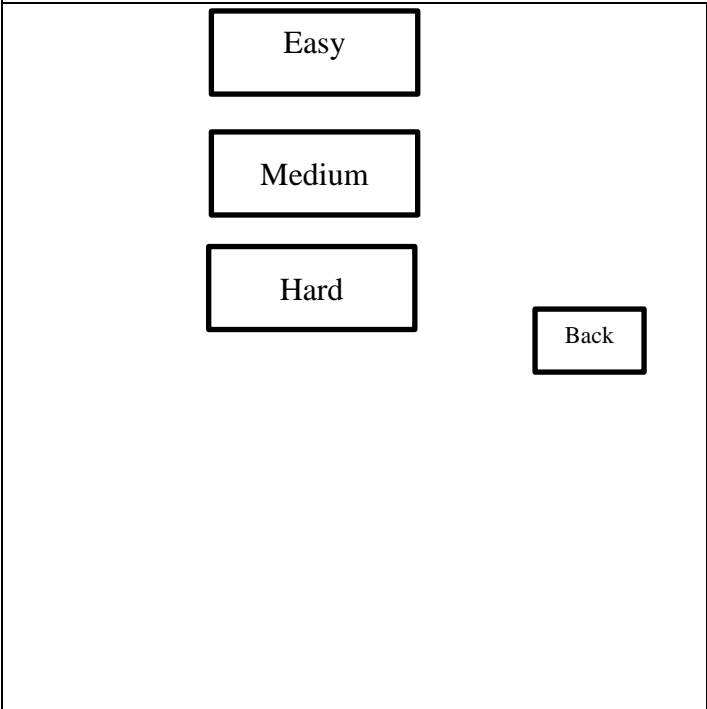
Berikut dapat dilihat perancangan antarmuka *highscore* dari pembangunan aplikasi game *monster nest*.



Gambar 3.40 Perancangan *Highscore*

### 3. Pilih *Level*

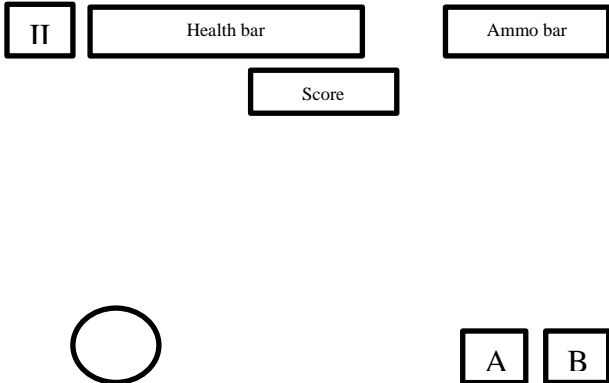
Berikut dapat dilihat perancangan antarmuka pilih *level* dari pembangunan aplikasi *game monster nest*.

T03	
	<ul style="list-style-type: none"> <li>- Tekan <i>Easy</i> masuk ke T01 dengan setting <i>level</i> mudah</li> <li>- Tekan <i>Medium</i> masuk ke T01 dengan setting <i>level</i> sedang</li> <li>- Tekan <i>Hard</i> masuk ke T01 setting <i>level</i> sulit</li> <li>- Tekan <i>Back</i> masuk ke T01 tanpa mengubah <i>level</i></li> </ul>

Gambar 3.41 Perancangan Pilih *Level*

#### 4. Permainan

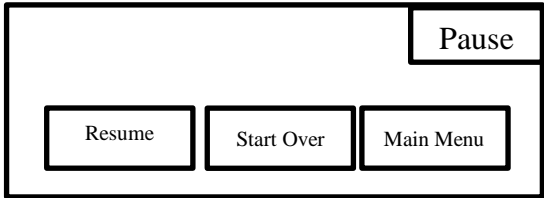
Berikut dapat dilihat perancangan antarmuka permainan dari pembangunan aplikasi game *monster nest*.

T04	
	<ul style="list-style-type: none"> <li>- Tekan II masuk ke T05</li> <li>- Gerakkan tombol o untuk menggerakkan pemain</li> <li>- Tekan tombol a untuk menembak</li> <li>- Tekan tombol b untuk mengisi amunisi</li> <li>- Apabila permainan selesai akan menuju ke T02</li> </ul>

Gambar 3.42 Perancangan permainan

## 5. Menu *Pause*

Berikut dapat dilihat perancangan antarmuka menu *pause* dari pembangunan aplikasi game *monster nest*.

T05	
	<ul style="list-style-type: none"> <li>- Tekan <i>Resume</i> masuk ke T04 untuk melanjutkan permainan</li> <li>- Tekan <i>StartOver</i> masuk ke T04 dengan permainan baru</li> <li>- Tekan <i>MainMenu</i> masuk ke T01</li> </ul>

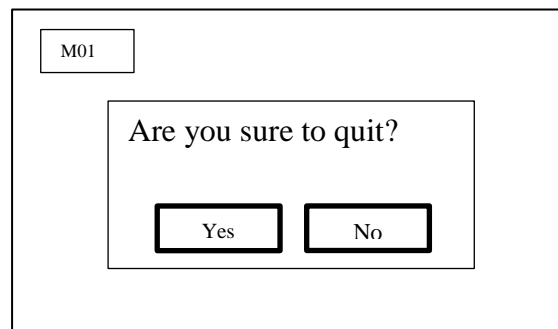
Gambar 3.43 Perancangan Menu *Pause*

### 3.4.3.1 Perancangan Pesan

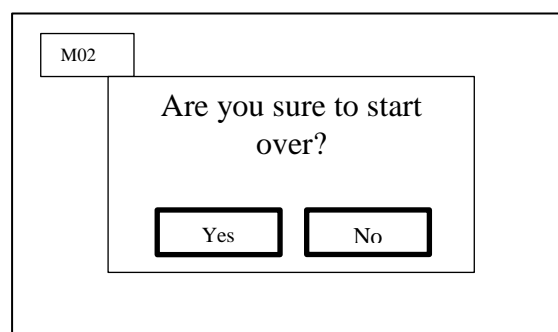
Berikut ini merupakan *form* perancangan pesan.

Tabel 3.12 Tabel Perancangan Pesan

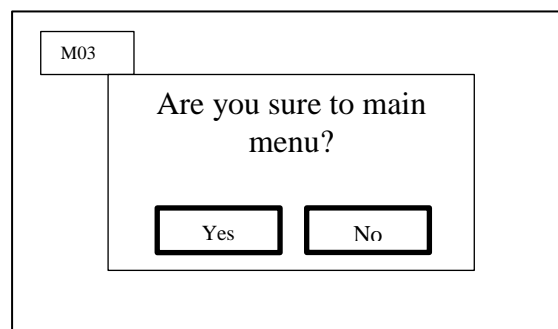
No	No Pesan	Isi Pesan	No Form
1	M01	<i>Are you sure to quit?</i>	T01
2	M02	<i>Are you sure to start over?</i>	T05
3	M03	<i>Are you sure to main menu?</i>	T05



Gambar 3.44 Perancangan Pesan M01



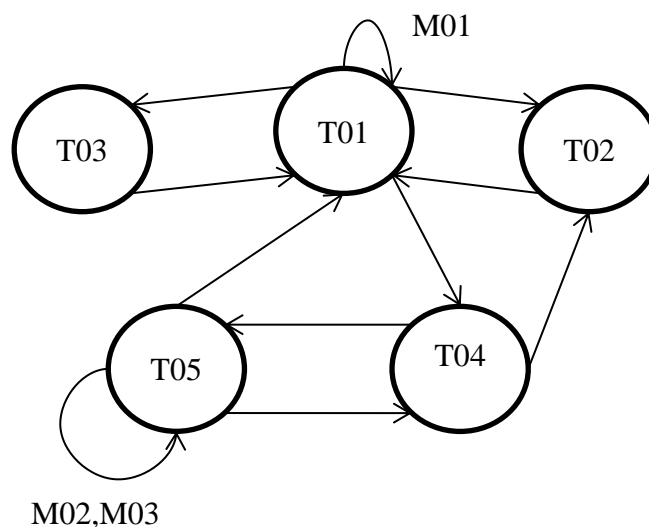
Gambar 3.45 Perancangan Pesan M02



Gambar 3.46 Perancangan Pesan M03

### 3.4.3.2 Jaringan Semantik

Struktur aplikasi ini dibuat secara modular, yaitu program dipecah menjadi modul-modul kecil yang mudah dibuat, mudah dites, dan mudah dimodifikasi. Dalam pembuatan *game* ini menggunakan alat bantu bagan struktural yang disebut juga jaringan semantik.



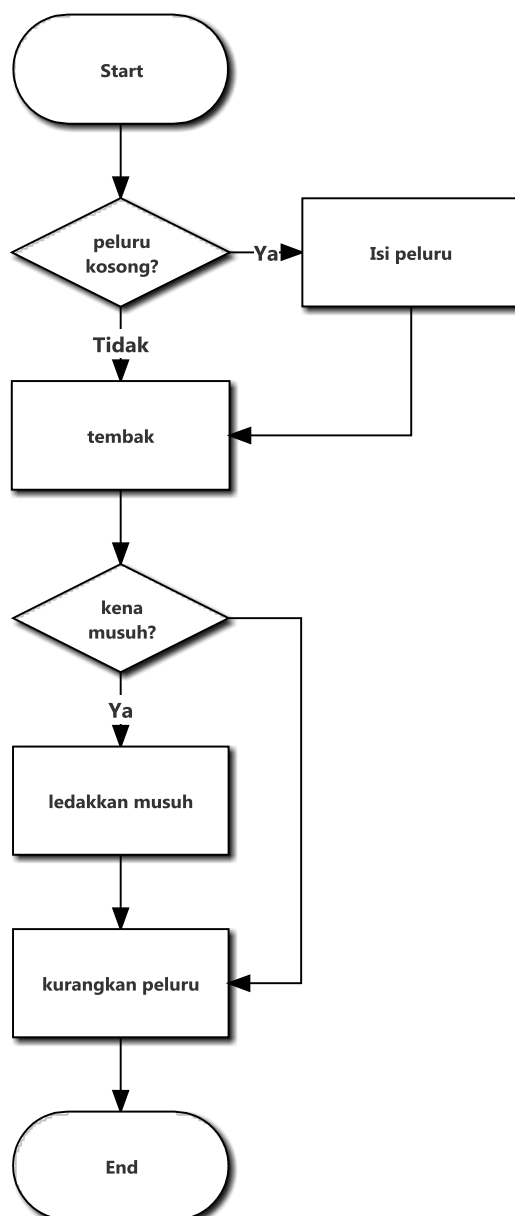
Gambar 3.47 Jaringan Semantik Pemain *Game Monster Nest*

### 3.4.4 Perancangan Method

Perancangan *method* merupakan perncangan yang berfungsi untuk mendeskripsikan *method - method* yang berada di dalam aplikasi. *Method* dapat dipanggil dengan menyertakan variabel, baik hanya satu variabel, banyak variabel atau bahkan tidak ada sama sekali. Adapun *method - method* yang terdapat dalam *game monster nest* yang akan dibangun adalah sebagai berikut :

#### A. Senjata.Tembak()

*Method* ini digunakan untuk membuat *player* menembak. Untuk lebih jelasnya dapat dilihat pada gambar 3.48:

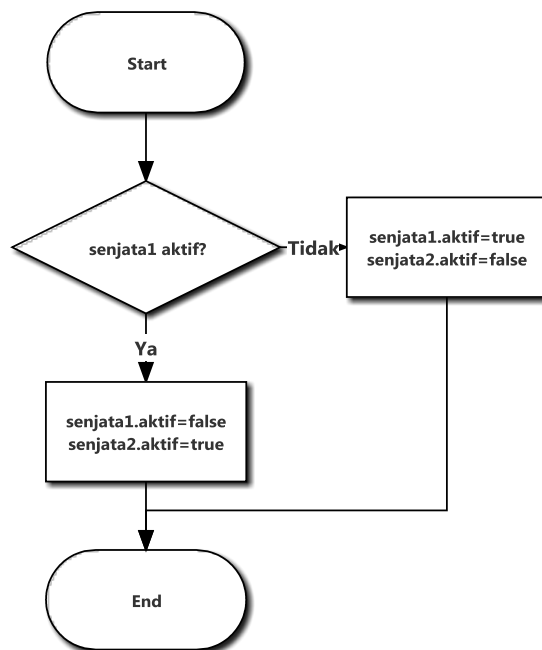


Gambar 3.48 *method* tembak



### B. *Player*.GantiSenjata()

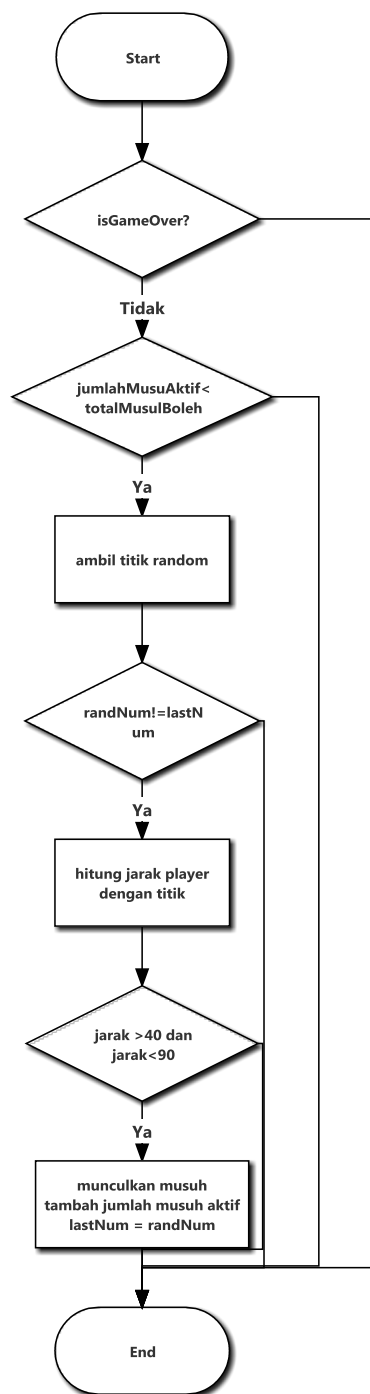
*Method* ini digunakan untuk membuat *player* melakukan ganti senjata. Untuk lebih jelasnya dapat dilihat pada gambar 3.49:



Gambar 3.49 *method* ganti senjata

### C. SpawnController.Update()

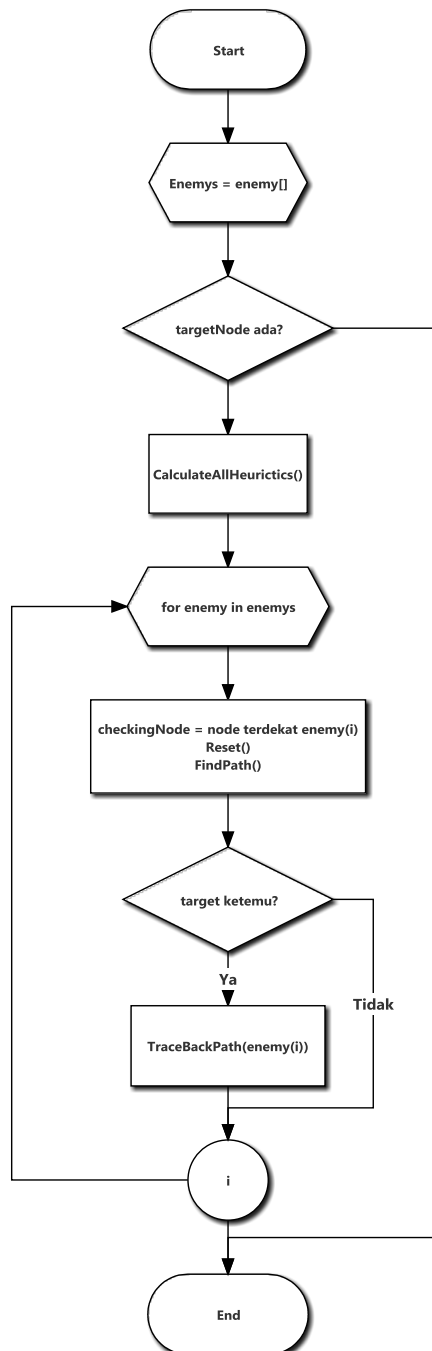
*Method* ini digunakan untuk memunculkan musuh kedalam arena permainan. Untuk lebih jelasnya dapat dilihat pada gambar 3.50:



Gambar 3.50 *method* SpawnController.Update()

#### D. *Pathfinder.Update()*

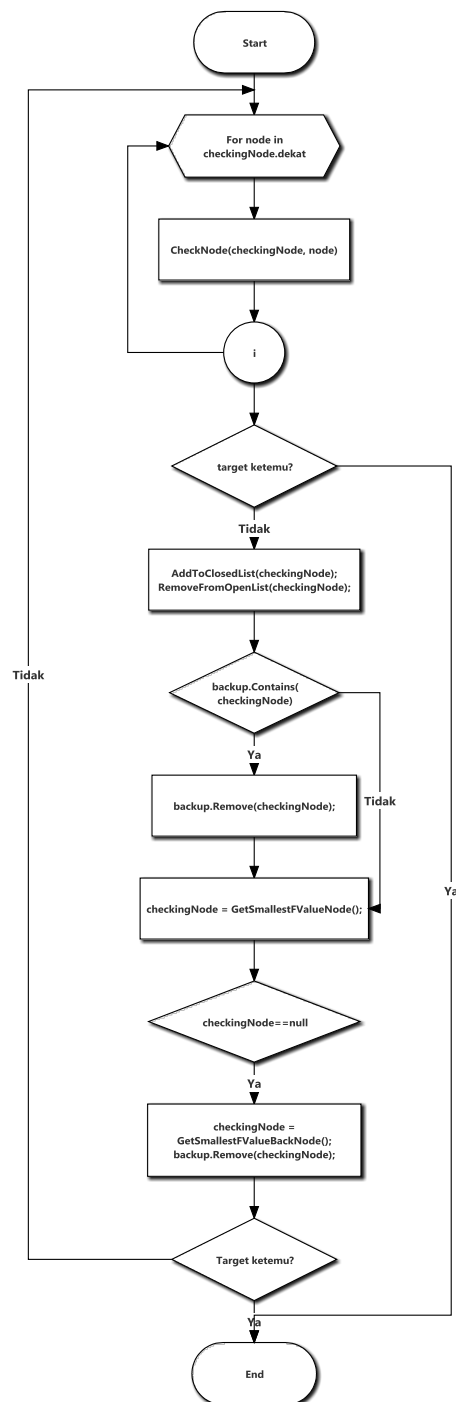
*Method* ini digunakan untuk memberikan *list node* ke musuh, yang digunakan sebagai jalan oleh musuh. Untuk lebih jelasnya dapat dilihat pada gambar 3.51:



Gambar 3.51 method *Pathfinder.Update()*

### E. Pathfinder.Findpath()

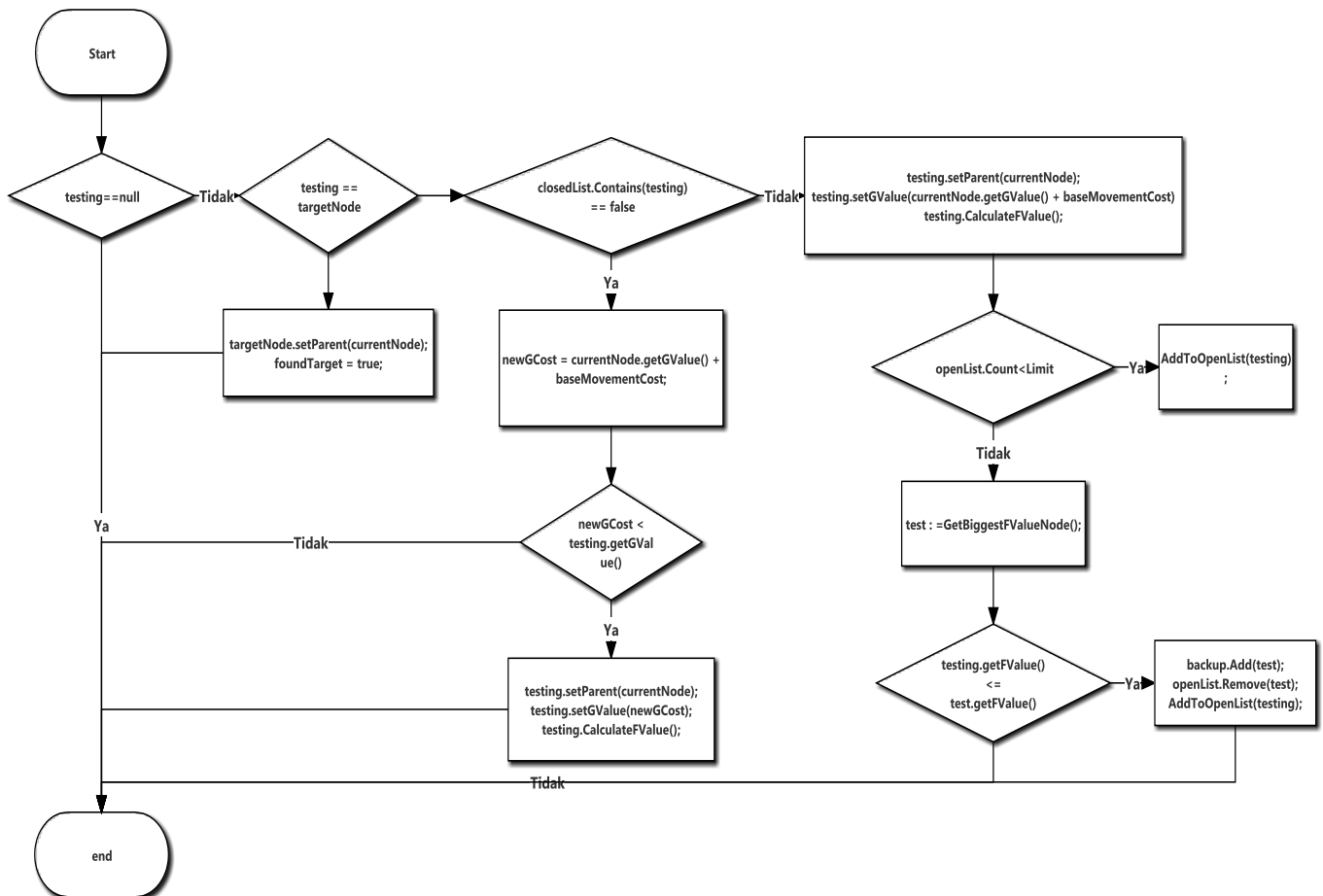
*Method* ini digunakan untuk melakukan pencarian jalan terpendek menuju *player*. Untuk lebih jelasnya dapat dilihat pada gambar 3.52:



Gambar 3.52 *method* Pathfinder.Findpath()

# F. Pathfinder.Checknode(currentNode,testing)

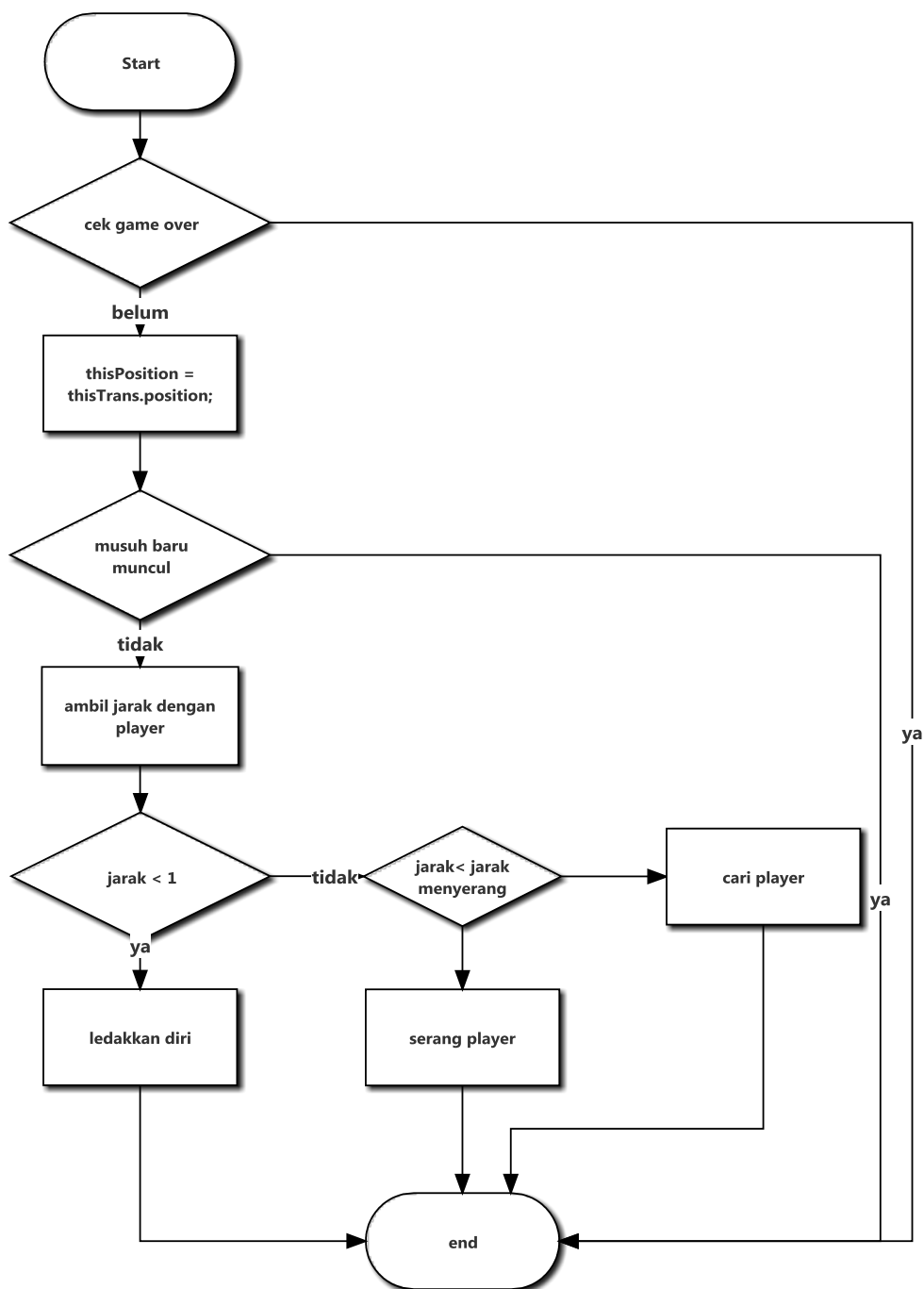
*Method* ini digunakan untuk mengecek *node* pada pencarian jalan. Untuk lebih jelasnya dapat dilihat pada gambar 3.53:



Gambar 3.53 *method checknode*

## G. Enemy.gerak()

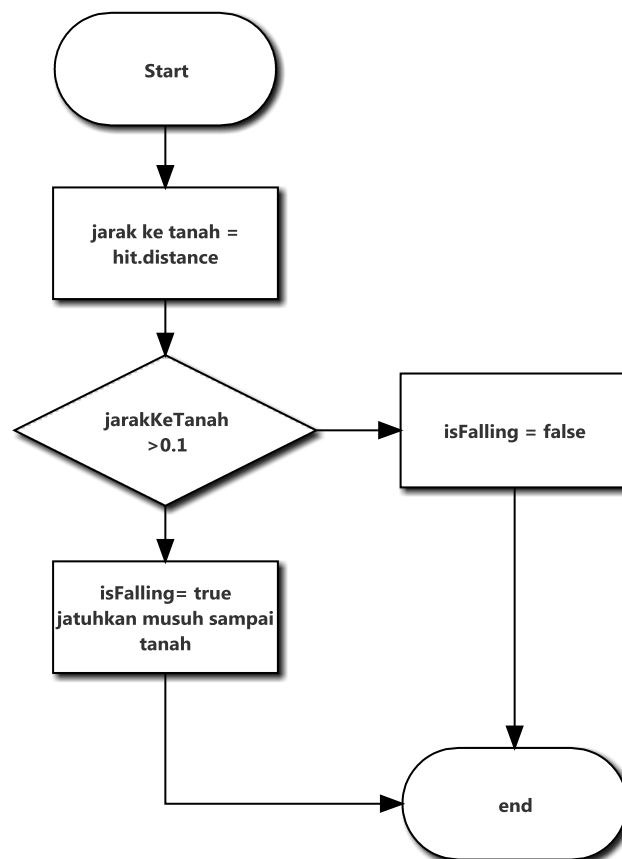
*Method* ini digunakan untuk mengatur gerakan pada musuh. Untuk lebih jelasnya dapat dilihat pada gambar 3.54 method gerak musuh :



Gambar 3.54 *method* gerak musuh

H. Enemy.fall()

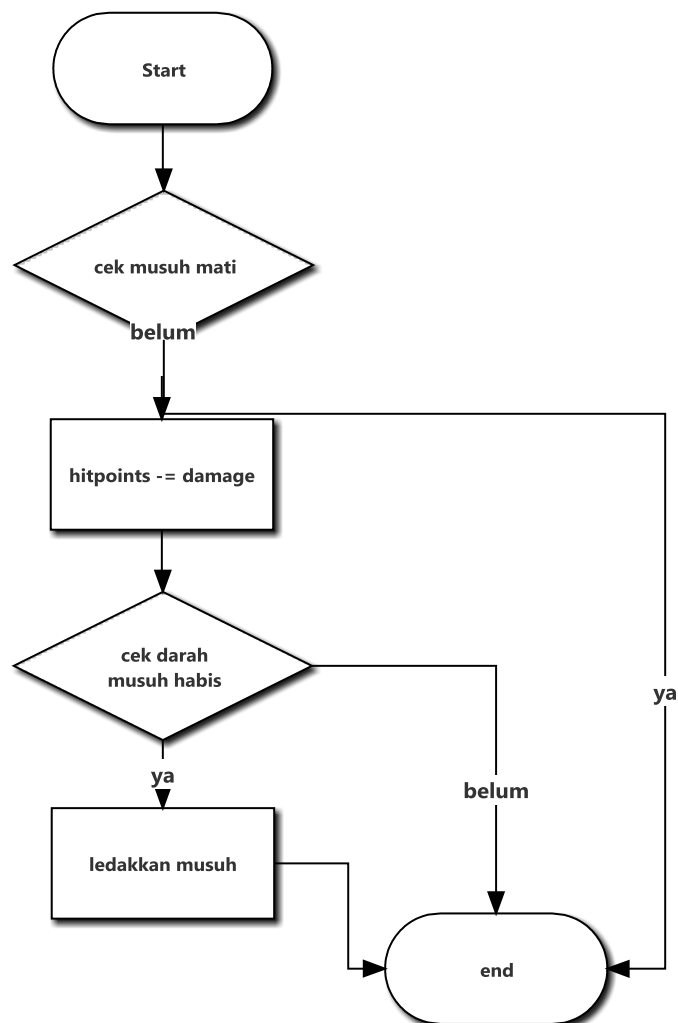
*Method* ini digunakan untuk mengatur pada saat musuh jatuh. Untuk lebih jelasnya dapat dilihat pada gambar 3.55 *method* musuh jatuh:



Gambar 3.55 *method* musuh jatuh

# I. Enemy.applyDamage(damage)

*Method* ini digunakan untuk mengatur pada saat musuh jatuh. Untuk lebih jelasnya dapat dilihat pada gambar 3.56:

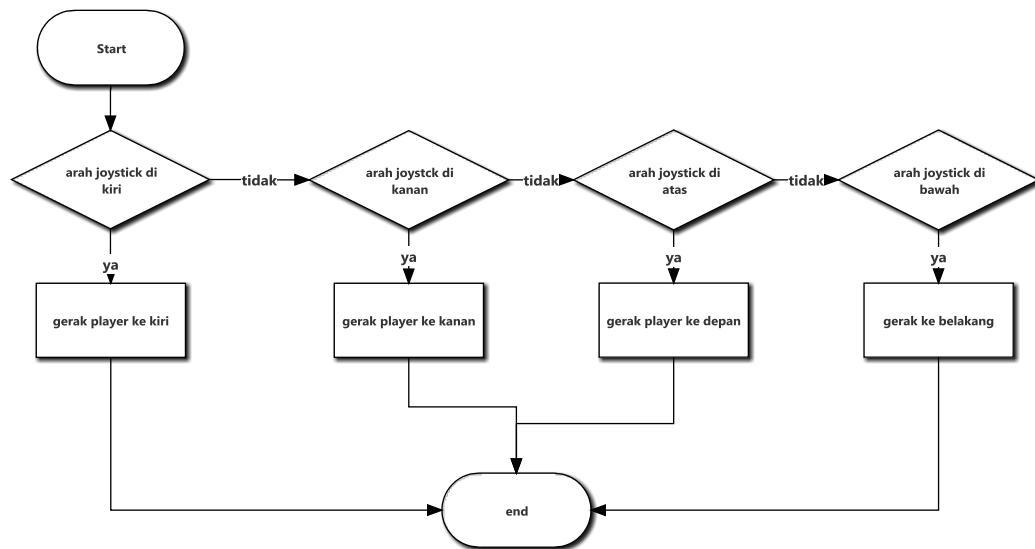


Gambar 3.56 *method* musuh *apply damage*



#### J. Player.gerak()

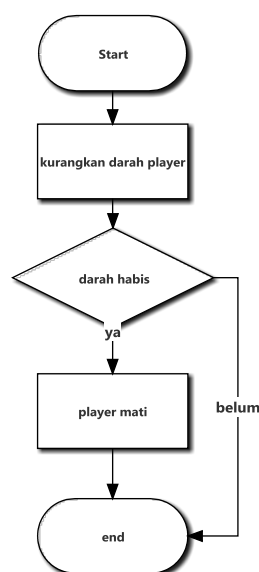
*Method* ini digunakan untuk mengatur pergerakan pemain. Untuk lebih jelasnya dapat dilihat pada gambar 3.57:



Gambar 3.57 *method* player gerak

#### K. Player.kena()

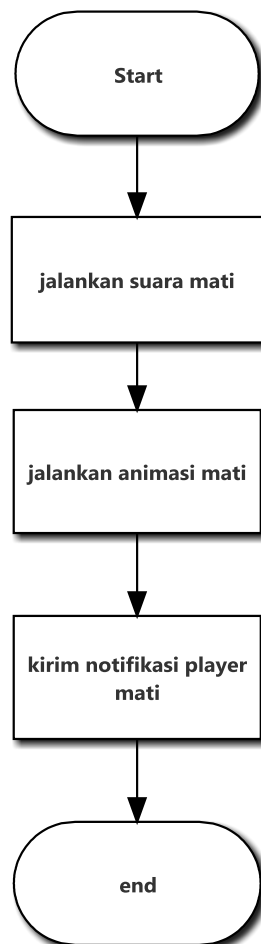
*Method* ini digunakan pada saat player terkena serangan musuh. Untuk lebih jelasnya dapat dilihat pada gambar 3.58:



Gambar 3.58 *method* player kena

#### L. Player.mati()

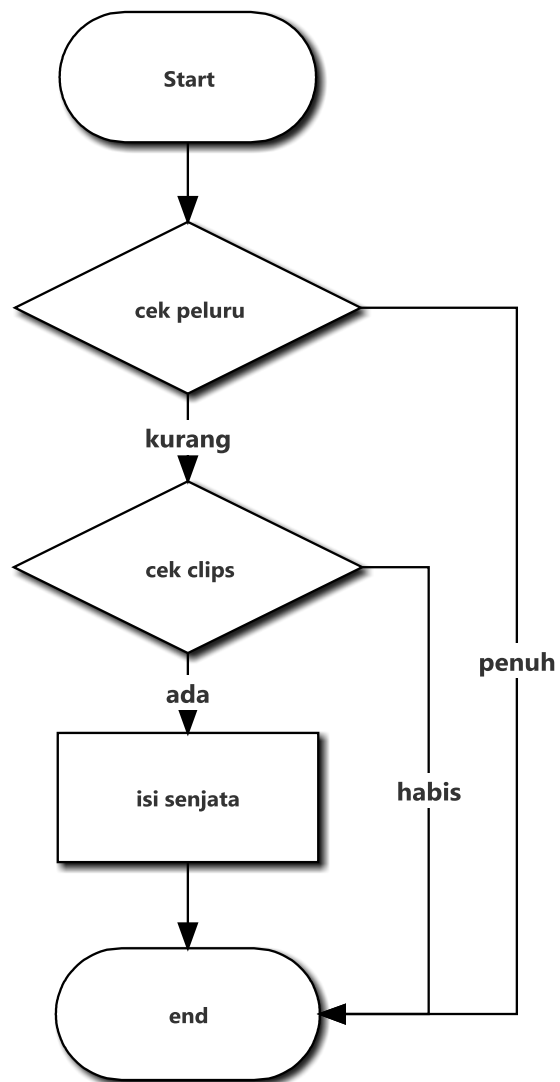
*Method* ini digunakan pada saat *player* mati. Untuk lebih jelasnya dapat dilihat pada gambar 3.59:



Gambar 3.59 *method player mati*

M.Senjata.reload()

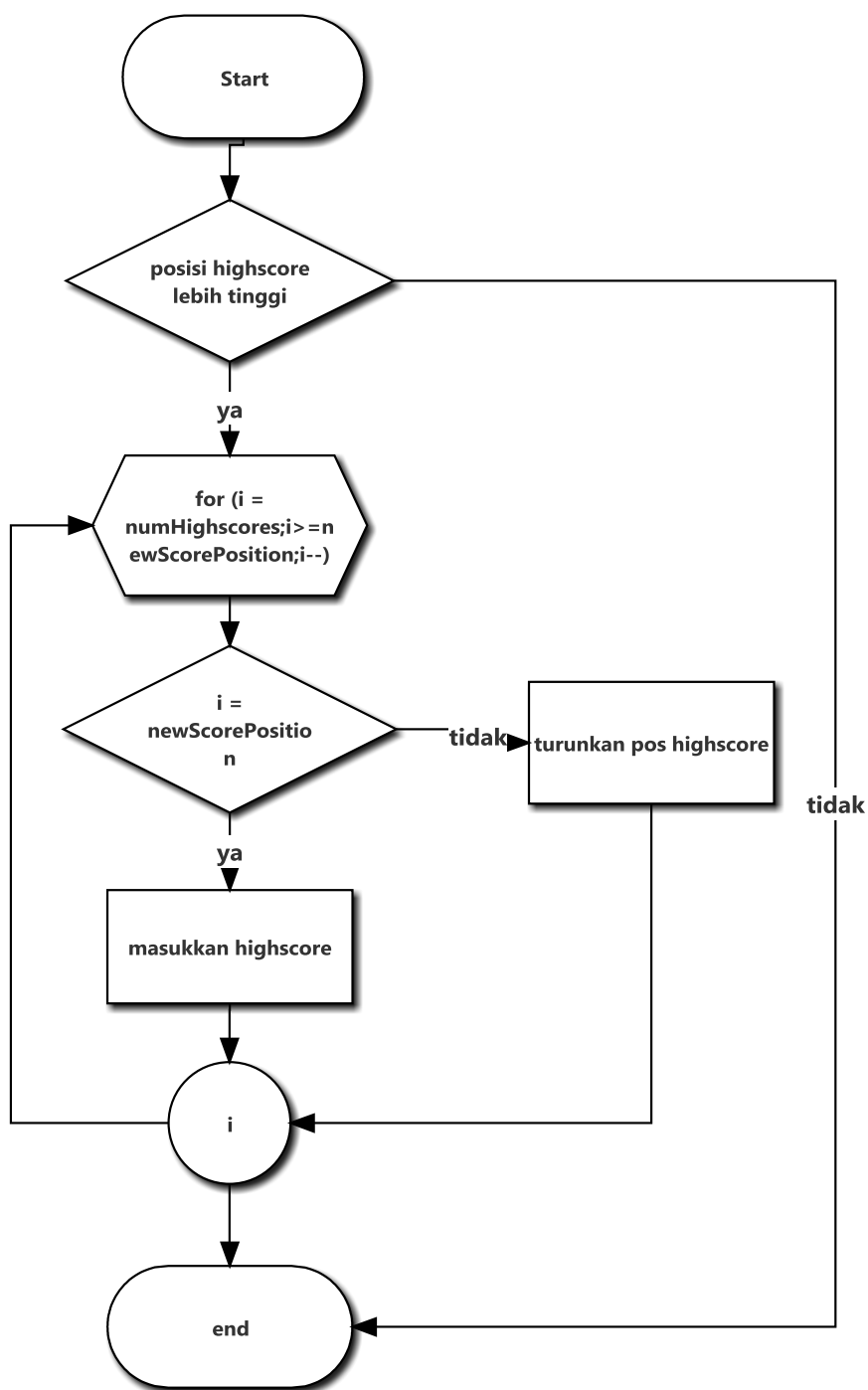
*Method* ini digunakan untuk mengisi senjata yang sedang digunakan. Untuk lebih jelasnya dapat dilihat pada gambar 3.60:



Gambar 3.60 *method senjata reload*

N. Highscore.addNew()

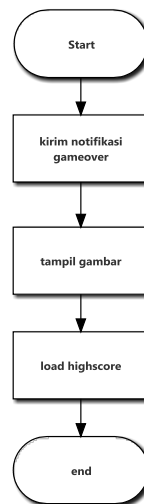
*Method* ini digunakan jika player mendapatkan skor yang termasuk dalam delapan tertinggi. Untuk lebih jelasnya dapat dilihat pada gambar 3.61:



Gambar 3.61 *method* tambah *highscore*

#### O. GameState.gameOver()

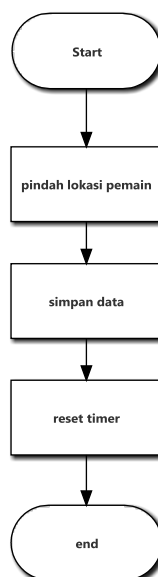
*Method* ini digunakan pada saat permainan akan berakhir. Untuk lebih jelasnya dapat dilihat pada gambar 3.62:



Gambar 3.62 *method gamestate gameover*

#### P. GameState.nextLevel()

*Method* ini digunakan pada saat akan pindah ke *ronde* berikutnya. Untuk lebih jelasnya dapat dilihat pada gambar 3.63:

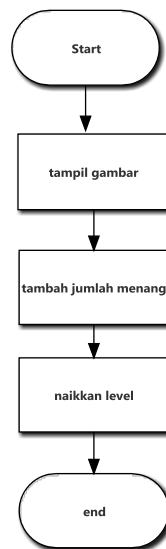


Gambar 3.63 *method gamestate next level*

#### Q. Gamestate.wonRound()

*Method* ini digunakan pada ketika pemain memenangkan sebuah *ronde*.

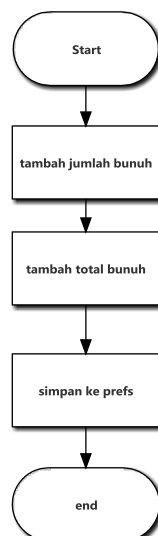
Untuk lebih jelasnya dapat dilihat pada gambar 3.64:



Gambar 3.64 *method gamestate won round*

#### R. Gamestate.enemyKilled()

*Method* ini digunakan pada saat musuh terbunuh oleh *player*. Untuk lebih jelasnya dapat dilihat pada gambar 3.65:



Gambar 3.65 *method gamestate musuh terbunuh*

#### S. Gamestate.spawnPlayer()

Method ini digunakan untuk memindahkan *player* ke lokasi yang berbeda. Untuk lebih jelasnya dapat dilihat pada gambar 3.66:



Gambar 3.66 *method spawn player*

