BAB 3
ANALISIS DAN PERANCANGAN

1.1 Analisis Masalah

Berdasarkan latar belakang masalah dengan mengevaluasi jurnal yang dibuat oleh Safril Rizki Waluyo, Mochamad Hariadi dan I ketut Eddy Purnama yang berjudul pencaian jalur terbaik menggunakan algoritma *particle swarm optimization* untuk optimasi lalulintas kendaraan, maka masalah yang teridentifikasi pada implementasi algoritma PSO untuk perilaku NPC menyerang secara berkelompok adalah bagaimana setiap NPC dalam suatu kelompok NPC dapat memilih NPC mana yang memiliki posisi terbaik untuk dijadikan pemimpin dan diikuti NPC lain dalam mengejar karakter *player*.

1.2 Analisis Game

Game yang akan dibangun merupakan *game* 2D dengan *genre action*-RPG dimana *game* yang dikembangkan berupa *prototype* dan berbasis *desktop*

1.2.1 Perilaku NPC

Berikut adalah rancangan perilaku setiap NPC dalam kelompok NPC :

1. Setiap NPC mengetahui posisi NPC lain yang ada disekitar
2. Setiap NPC mengetahui dimana posisinya sendiri
3. Setiap NPC memiliki satu tujuan yang sama dengan NPC lain
4. NPC pemimpin dapat memberikan informasi posisinya kepada NPC lain yang bukan pemimpin.
5. NPC lain yang bukan pemimpin mengikuti pergerakan NPC pemimpin dalam mengejar karakter *player*

1.2.2 Game Play

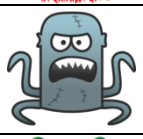
Dalam *game* ini terdapat 10 NPC yang akan mengejar dan menghalangi karakter *player* dalam menyelesaikan misi. Karakter *player* harus menghindari 10 NPC ini untuk melanjutkan misi. Pergerakan karakter *player* dapat dikontrol dengan tombol arah pada *keyboard*, karakter *player* harus mengumpulkan 12 buah kunci emas ajaib untuk dapat keluar dari stage ini. Nyawa karakter *player* akan berkurang jika NPC menabrak karakter *player*.

1.2.3 Story Line

Karakter player memerankan sosok *hero* dimana dalam satage ini *hero* terjebak di sebuah padang rumput yang didalamnya terdapat 10 *monster*, untuk keluar dari kejaran *monster*, *hero* harus mengumpulkan 12 buah kunci emas, apabila semua kunci emas sudah terkumpul akan muncul pintu ajaib dimana pintu ini akan mengeluarkan *hero* dari padang rumput ini.

1.2.4 Perancangan Karakter

Tabel 1.1 Perancangan Karakter

Jenis Karakter	Nama Karakter	Gambar
Karakter Player	Player	
NPC	NPC 1	
NPC	NPC 2	
NPC	NPC 3	
NPC	NPC 4	
NPC	NPC 5	
NPC	NPC 6	

NPC	NPC 7	
NPC	NPC 8	
NPC	NPC 9	
NPC	NPC 10	

1.3 Analisis Game Action RPG

Game *action* RPG merupakan pengembangan dari *game* RPG dimana *game* dengan *genre action* RPG ini memiliki unsur *action* sehingga membutuhkan sedikit keahlian dan kegesitan dalam memainkannya. Game dengan *genre* ini adalah salah satu *game* yang berasal dari kertas dan pena. Dalam *game action*-RPG terdapat dua elemen utama yang menjadi kunci suksesnya sebuah *game* RPG. Fitur yang pertama adalah cerita (*story*) dan fitur yang kedua adalah pembangunan karakter (*character development*), dimana karakter merupakan tokoh hayalan yang akan di perankan oleh *player*.

1.3.1 Karakter Dalam Game Action RPG

Dalam *game action*-RPG adapun karakter dibagi menjadi 2 yaitu :

1. Karakter *player* / *player character*
2. *Non player character* (NPC)

1.3.1.1 Karakter Player / Player Character

Karakter *player* adalah karakter yang dimainkan atau dikendalikan oleh *player* dimana pergerakan maupun tingkah laku dari karakter ini sepenuhnya tergantung dari bagaimana *player* mengendalikan dan memainkannya. Karakter *player* dalam *game action* RPG merupakan sosok tokoh hayalan dimana *player* dalam *game* ini akan berperan sebagai karakter tersebut yang akan menyelesaikan sebuah misi tertentu.

1.3.1.2 Non Player Character (NPC)

Bukan karakter *player* / NPC merupakan karakter yang dikendalikan sepenuhnya oleh sistem dimana tingkah laku maupun pergerakan dari NPC ini sudah dirancang sebelumnya didalam sistem. Dalam penelitian kali ini peneliti mencoba mengendalikan NPC dalam *game action*-RPG ini dengan mengimplementasikan algoritma PSO kedalam model perilaku NPC menyerang secara berkelompok.

1.3.2 Perilaku NPC Menyerang Secara Berkelompok

Model perilaku NPC menyerang secara berkelompok ini merupakan model perilaku multi NPC dimana setiap NPC dalam suatu kelompok NPC tersebut dapat saling berkomunikasi dalam menyerang karakter *player*. Perilaku menyerang secara berkelompok dalam *game* ini ditunjukkan dari bagaimana kelompok NPC tersebut mengejar karakter *player* untuk menghalangi karakter *player* tersebut dalam mencapai misi tertentu dalam *game.*

Dalam mengejar karakter *player* setiap NPC dalam kelompok NPC ini akan berkomunikasi untuk menentukan NPC mana yang memiliki posisi terbaik, NPC yang memiliki posisi terbaik ini akan menjadi pemimpin dari NPC yang lainnya, untuk kemudian di ikuti oleh NPC lainnya tersebut dalam mengejar karakter *player.*

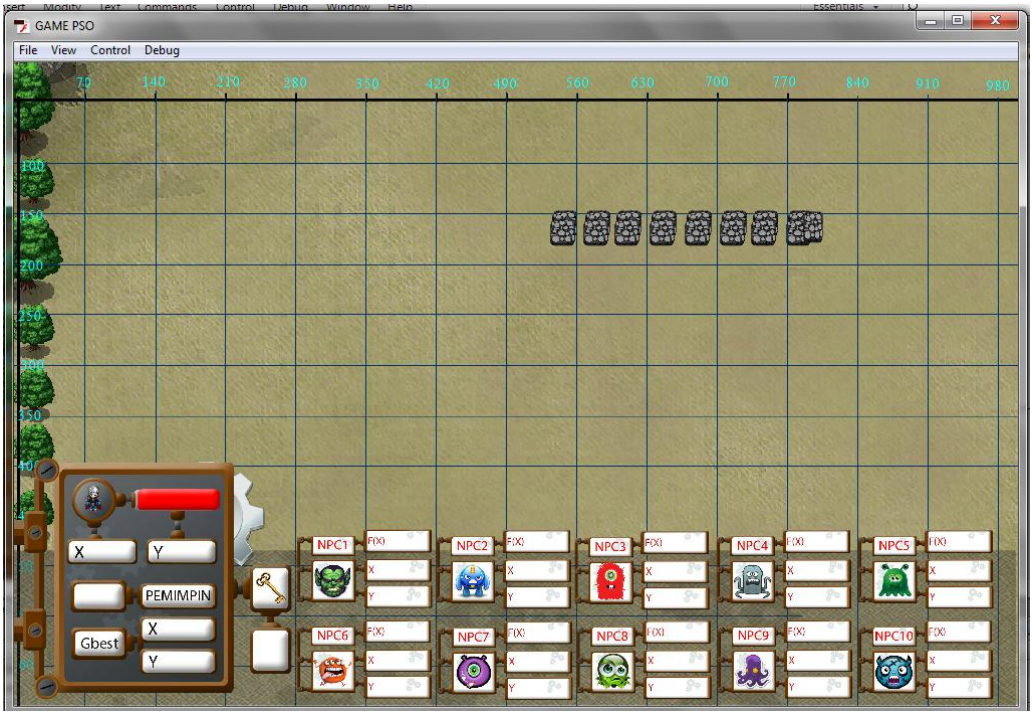
1.3.3 Story Game Action RPG

Story atau cerita dalam *game action* RPG merupakan salah satu unsur yang juga penting dalam *game* ini dimana *player* akan mengacu terhadap cerita dari *game* dalam memainkan *game action* RPG ini.

1.3.4 Arena Permainan Game Action RPG

Arena permainan dalam *game action* RPG merupakan ruang gerak dimana karakter *player* dan NPC bergerak maupun berperilaku. Pergerakan yang dapat dilakukan NPC yaitu bergerak ke kiri, bergerak ke kanan, bergerak ke atas, bergerak ke bawah, dan secara diagonal berbeda dengan karakter *player* yang hanya dapat bergerak kiri, bergerak ke kanan, bergerak ke atas, bergerak ke bawah. Pergerakan tiap karakter dalam arena permainan *game action* RPG ini mengacu terhadap sumbu x dan y, yang apabila tiap karakter bergerak ke kiri atau

ke kanan maka koordinat x pada tiap karakter akan berubah, begitu pula apabila tiap karakter bergerak ke atas atau ke bawah maka koordinat y pada tiap karakter akan berubah. Nilai dari koordinat setiap karakter bertipe integer dimana tipe data ini yang akan dieksekusi oleh algoritma dan menghasilkan sebuah nilai. Adapun pengimplementasian algoritma PSO terhadap model perilaku NPC menyerang secara berkelompok ini akan mempengaruhi pergerakan NPC, dimana pergerakan NPC ke kiri, ke kanan, ke atas, ke bawah maupun diagonal akan dipengaruhi dari hasil perubahan dari perhitungan algoritma PSO itu sendiri. Berikut arena dari *game action* RPG dengan disertakan *grid* dari sumbu x dan sumbu y yang akan di bangun :



Gambar 1.1 Arena Permainan

1.4 Analisis Masukan

Dalam algoritma PSO untuk mencapai suatu titik tujuan setiap partikel akan menentukan partikel mana yang memiliki posisi terbaik terlebih dahulu kemudian partikel lain yang tidak memiliki posisi terbaik akan memperbaharui kecepatan dan posisinya untuk mencapai posisi partikel yang memiliki posisi paling baik tersebut .untuk memperbaharui kecepatan dan posisi dari tiap partikel tersebut dibutuhkan masukan, diantaranya :

1. Nilai koordinat posisi setiap partikel
   Nilai ini diperoleh dari nilai titik koordinat setiap NPC. Nilai titik koordinat awal dari setiap NPC sudah di tentukan sebelumnya[1]

2. Nilai Fitness
   Nilai dari fitness diperoleh dengan mengevaluasi koordinat setiap partikel kedalam fungsi *fitness*[1] dimana fungsi *fitness* merupakan fungsi yang akan dioptimasi [13]

3. Nilai konstanta C_1 dan C_2
   Nilai ini merupakan konstanta faktor *cognitive* dan sosial.
   Nilai C_1 dan C_2 ini berkisar antara 0 sampai dengan 1.
   Adapun nilai C_1 dan C_2 ini ditentukan terlebih dahulu. Dimana nilai C_1 dan C_2 yang digunakan dalam tahap analisis = 1.

4. Bilangan acak r_1 dan r_2
   Nilai ini merupakan nilai random.
   Nilai ini diperoleh dari nilai random antara 0 sampai dengan 1 [1]

5. Nilai Pbest
   Nilai Pbest atau *personal best*, nilai ini adalah nilai Posisi terbaik yang dimiliki setiap partikel.
   Nilai ini diperoleh dari nilai titik koordinat setiap partikel [1]

6. Nilai Gbest
   Nilai Gbest atau *global best*, nilai ini adalah posisi partikel yang memiliki posisi paling baik dari partikel lainnya.
   Nilai Gbest diperoleh dari koordinat partikel yang paling baik dimana partikel yang memiliki koordinat paling baik ini merupakan partikel yang memiliki *fitness* paling optimal, *fitness* dikatakan optimal tergantung kasus yang dimiliki minimasi atau maksimasi [1]

1.5 Analisis Algoritma

Dalam penelitian kali ini algoritma PSO akan diimplementasikan terhadap perilaku NPC menyerang secara berkelompok, dimana algoritma PSO digunakan untuk menentukan posisi NPC terbaik untuk dijadikan pemimpin dan NPC lain yang bukan merupakan pemimpin dapat mengikuti NPC yang memiliki posisi terbaik tersebut dalam mengejar karakter *player*. Adapun dalam penelitian kali ini nilai *fitness* yang akan dioptimasi adalah jarak antara NPC dengan *player* dimana NPC yang memiliki jarak paling pendek dengan *player* maka NPC tersebut merupakan NPC pemimpin yang akan diikuti NPC lain dalam mengejar karakter *player*. Berikut fungsi menghitung jarak antar player dengan NPC yang nantinya akan dioptimasi :

$$F(X) = \sqrt{(player.x - NPC.x)^2 + (player.y - NPC.y)^2} \quad (3.1)$$

Dimana :

F(X) adalah *fitness*

$player.x$ adalah koordinat X *player*

$player.y$ adalah koordinat Y *player*

$NPC.x$ = koordinat X NPC

$NPC.y$ = koordinat Y NPC

1.5.1 Langkah Penyelesaian Dalam PSO

adapun langkah langkah menyelesaikan masalah dalam algoritma PSO adalah sebagai berikut :

1. Inisialisasi populasi partikel

Langkah ini merupakan tahap awal dimana jumlah dan koordinat partikel / posisi partikel ditentukan terlebih dahulu.

2. Melakukan perhitungan nilai fitness dari setiap partikel

Untuk memperoleh nilai fitness dibutuhkan nilai posisi setiap partikel tersebut untuk dievaluasi oleh fungsi fitness

3. Menentukan Pbest (*personal best*)

Pbest adalah posisi terbaik setiap partikel. Posisi ini merupakan posisi yang ditempati oleh partikel saat ini.

4. Menentukan Gbest (*Global best*)

Nilai Gbest merupakan nilai posisi partikel yang memiliki posisi paling baik dari partikel lainnya.Nilai Gbest diperoleh dari koordinat partikel yang paling baik dimana partikel yang memiliki koordinat paling baik ini merupakan partikel yang memiliki fitness paling optimal,fitness dikatakan optimal tergantung kasus yang dimiliki minimasi atau maksimasi.

5. Menghitung nilai *velocity* (V)

Untuk mendapatkan nilai V terbaru dapat digunakan persamaan (2.1) untuk perbaharui kecepatan

6. Menghitung nilai koordinat terbaru

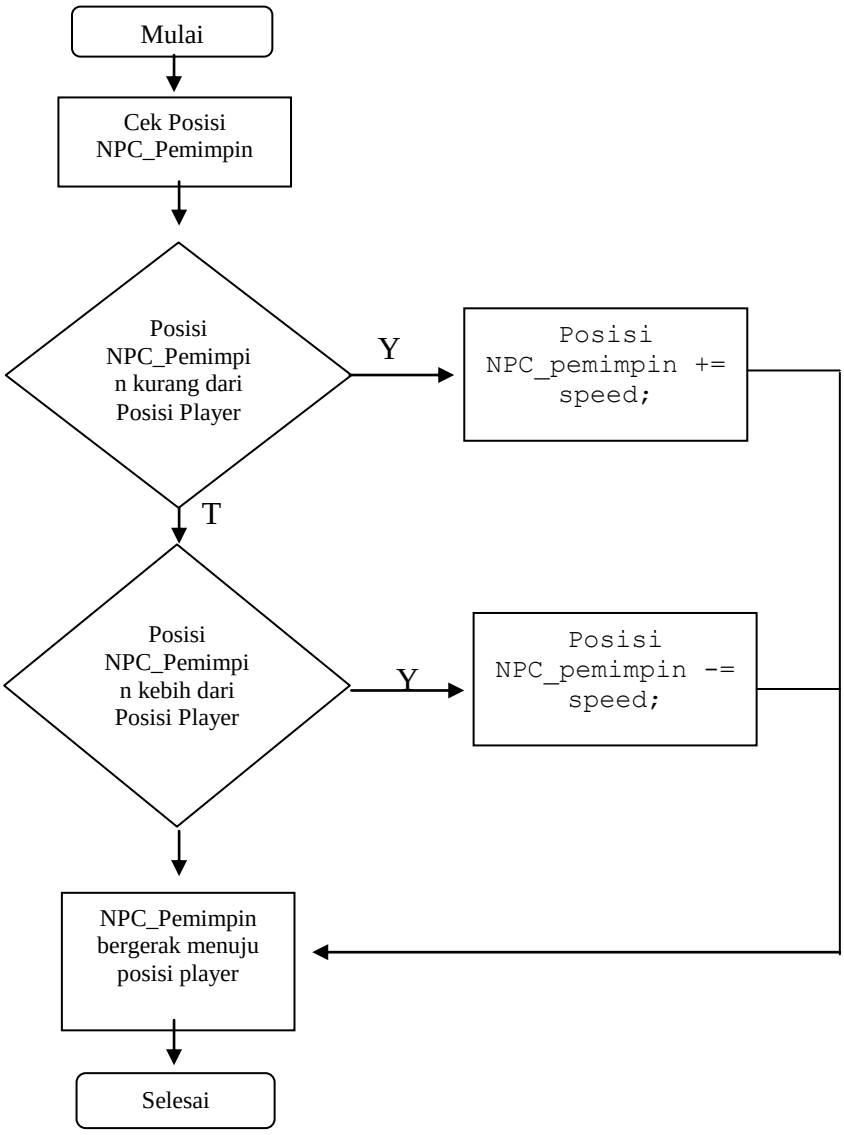
Memasukan hasil perhitungan nilai V dari setiap partikel kedalam persamaan (2.2)

7. Mengecek posisi partikel apakah sudah mencapai titik tujuan atau belum

Nilai dikatakan konveregen jika posisi partikel mencapai posisi tujuan. Jika partikel belum mencapai titik tujuan maka langkah 2 diulangi dengan mengevaluasi nilai partikel saat ini kedalam fungsi tujuan.

1.5.2 Algoritma PSO Dalam Game

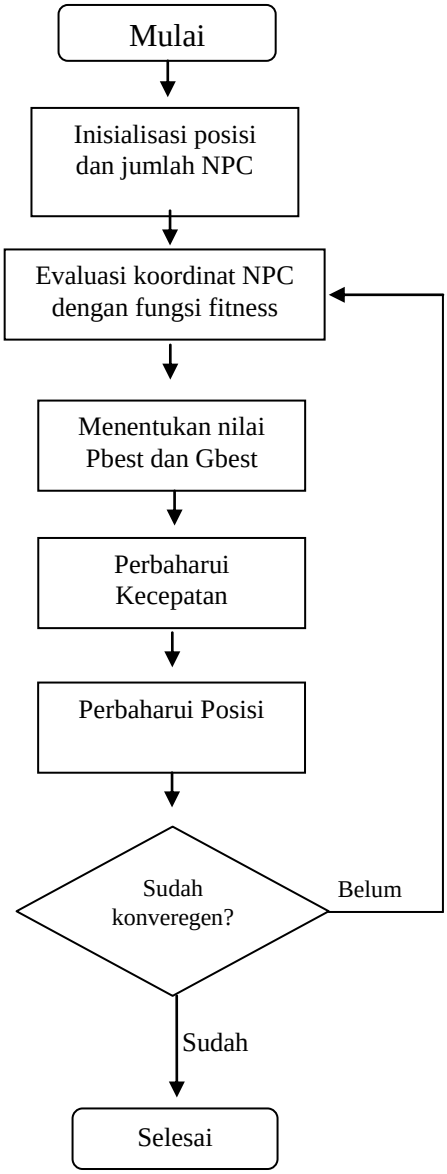
Dalam *game* yang akan dibangun penggunaan algoritma PSO hanya untuk pemilihan NPC mana yang menjadi pemimpin dan NPC yang bukan pemimpin bergerak menuju NPC pemimpin. Maka dari itu dalam *game* yang akan dibangun dibutuhkan algoritma lain untuk menggerakkan NPC pemimpin agar dapat mencapai posisi karakter *player*. Berikut gambar flowchart algoritma yang digunakan NPC pemimpin untuk mengejar karakter player :



Gambar 1.2 Flowchart Algoritma untuk NPC pemimpin

1.5.3 Alur Proses Penyelesaian Dalam PSO

Berikut merupakan alur proses algoritma PSO yang diterapkan dalam game :



Gambar 1.3 Alur Proses Penyelesaian Dalam PSO

1.6 Analisis Kasus

Berikut contoh masalah yang akan diselesaikan menggunakan algoritma PSO

Diketahui:

Kasus minimasi

Fungsi *fitness*:

$$F(X) = \sqrt{(player.x - NPC.x)^2 + (player.y - NPC.y)^2}$$

Titik koordinat *player* = (500,380)

$C_1 = C_2 = 1$ // Nilai C_1 dan C_2 sudah ditentukan (nilai antara 0 sampai dengan 1)

r_1 , r_2 = nilai random // nilai berkisar antara 0 sampai dengan 1

Banyak partikel = 5

Nilai titik Koordinat awal NPC :

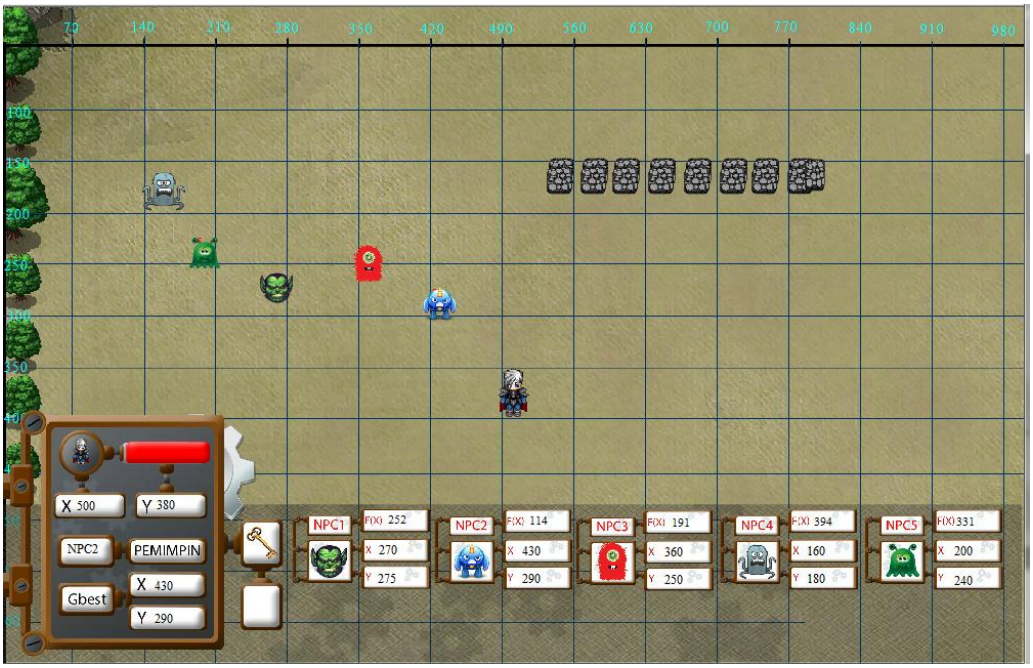
Koordinat NPC1 : X_1= (270,275)

Koordinat NPC2 : X_2 = (430,290)

Koordinat NPC 3 : X_3 = (360,250)

Koordinat NPC 4 : X_4= (160,180)

Koordinat NPC 5 : X_5= (200,240)



Gambar 1.4 Posisi awal NPC dan Player

Nilai Velocity setiap partikel saat ini

Velocity NPC 1 : $V_1 = 0$

Velocity NPC 2 : $V_2 = 0$

Velocity NPC 3 : $V_3 = 0$

Velocity NPC 4 : $V_4 = 0$

Velocity NPC 5 : $V_5 = 0$

1.6.1 Penyelesaian

ITERASI KE 0

Nilai random pada iterasi ke 0 :

$r_1 = 0,2$ $r_2 = 0,5$

Langkah pertama

Nilai koordinat NPC saat ini :

Koordinat NPC1 pada iterasi ke 0 : $X_1 = (270,275)$

Koordinat NPC2 pada iterasi ke 0 : $X_2 = (430,290)$

Koordinat NPC3 pada iterasi ke 0 : $X_3 = (360,250)$

Koordinat NPC4 pada iterasi ke 0 : $X_4 = (160,180)$

Koordinat NPC5 pada iterasi ke 0 : $X_5 = (200,240)$

Langkah kedua

Evaluasi posisi NPC dengan menggunakan fungsi *fitness* :

$$F(X) = \sqrt{(player.x - NPC.x)^2 + (player.y - NPC.y)^2}$$

Perhitungan nilai fitness tiap NPC :

Fitness NPC1 : $F(X)_1 = \sqrt{(500-270)^2 + (380-275)^2} = 252$

Fitness NPC2 : $F(X)_2 = \sqrt{(500-430)^2 + (380-290)^2} = 114$

Fitness NPC3 : $F(X)_3 = \sqrt{(500-360)^2 + (380-250)^2} = 191$

Fitness NPC4 : $F(X)_4 = \sqrt{(500-160)^2 + (380-180)^2} = 394$

Fitness NPC5 : $F(X)_5 = \sqrt{(500-200)^2 + (380-240)^2} = 331$

Langkah ketiga

Nilai Pbest = nilai koordinat partikel saat ini

Pbest NPC 1 : $Pbest_1 = (270,275)$

Pbest NPC 2 : $Pbest_2 = (430,290)$

Pbest NPC 3 : $Pbest_3 = (360,250)$

Pbest NPC 4 : $Pbest_4 = (160,180)$

Pbest NPC 5 : $Pbest_5 = (200,240)$

Langkah keempat

Nilai Gbest merupakan nilai titik koordinat NPC yang memiliki nilai *fitness* paling optimal dimana dalam kasus ini akan meminimalkan jarak antara NPC

dengan *player*. NPC dengan jarak paling sedikit merupakan NPC yang paling optimal dan dijadikan pemimpin. Adapun nilai *fitness* yang di dapat :

Fitness NPC1 : $F(X)_1 = 252$

Fitness NPC2 : $F(X)_2 = 114$

Fitness NPC3 : $F(X)_3 = 191$

Fitness NPC4 : $F(X)_4 = 394$

Fitness NPC5 : $F(X)_5 = 331$

Dengan melihat nilai *fitness* setiap NPC diatas, NPC yang memiliki nilai *fitness* paling rendah adalah NPC ke 2.

Maka didapat nilai Gbest = koordinat NPC ke 2 = (430,290)

Langkah kelima

NPC lain yang bukan pemimpin memperbaharui kecepatan dengan persamaan sebagai berikut :

$$V_i\,(t+1) = V_i(t)+C_1.r_1(Pbest_i - X_i(t)) + C_2.r_2(Gbest_i - X_i(t))$$

Perhitungan nilai perbaharui kecepatan tiap NPC :

NPC_1 : $Vx_1\,(1) = 0 + 0.2(270 - 270) + 0.5(430 - 270) = 80$

$Vy_1\,(1) = 0 + 0.2(275-275) + 0.5(290-275) = 7.5$

NPC_3 : $Vx_3\,(1) = 0 + 0.2(360- 360) + 0.5(430 - 360) = 35$

$Vy_3\,(1) = 0 + 0.2(250-250) + 0.5(290-250) = 20$

NPC_4 : $Vx_4\,(1) = 0 + 0.2(160-160) + 0.5(430-160) = 135$

$Vy_4\,(1) = 0 + 0.2((180-180) + 0.5(290-180)) = 55$

NPC_5 : $Vx_5\,(1) = 0 + 0.2(200-200) + 0.5(430-200) = 115$

$Vy_5\,(1) = 0 + 0.2((240-240) + 0.5(390-240) = 25$

Langkah keenam

NPC lain yang bukan pemimpin memperbaharui posisi dengan persamaan sebagai berikut:

$$X_i\,(t+1) = X_i\,(t) + V_i\,(t+1)$$

Perhitungan perbaharui kecepatan tiap NPC :

NPC 1 : $X_1(1) = 270 + 80 = 350$

$Y_1(1) = 275 + 7.5 = 282.5$

NPC3 : $X_3(1) = 360 + 35 = 395$

$Y_3(1) = 250 + 20 = 270$

NPC4 : $X_4(1) = 160 + 135 = 295$

$Y_4(1) = 180 + 25 = 205$

NPC 5 : $X_5(1) = 200 + 115 = 315$

$Y_5(1) = 240 + 25 = 265$

Hasil :

Nilai titik koordinat yang dihasilkan :

Titik awal tiap NPC :	Koordinat baru yang dihasilkan NPC :
$X_1(0) = (270,275)$	$X_1(1) = (350 , 282.5)$
$X_3(0) = (360,250)$	$X_3(1) = (395 , 270)$
$X_4(0) = (160,180)$	$X_4(1) = (295 , 205)$
$X_5(0) = (200,240)$	$X_5(1) = (315 , 265)$

Langkah ketujuh

Mengecek posisi NPC apakah sudah konveregen atau belum

Dengan melihat hasil perhitungan pada iterasi ke 0 diatas posisi setiap NPC yang bukan pemimpin belum mencapai titik koordinat NPC pemimpin maka dilakukan kembali perhitungan langkah ke 2 pada iterasi selanjutnya dengan mengevaluasi nilai titik koordinat setiap NPC saat ini kedalam dungsi tujuan.

ITERASI KE 1

Nilai random pada iterasi ke 1 :

$r_1 = 0{,}4$ $r_2 = 0{,}7$

Langkah pertama

Nilai koordinat NPC saat ini :

Koordinat NPC 1 pada iterasi ke 1: $X_1 = (350 , 282.5)$

Koordinat NPC 2 pada iterasi ke 1: $X_2 = (430,290)$

Koordinat NPC 3 pada iterasi ke 1: $X_3 = (395 , 270)$

Koordinat NPC 4 pada iterasi ke 1: $X_4 = (295 , 205)$

Koordinat NPC 5 pada iterasi ke 1: $X_5 = (315 , 265)$

Langkah kedua

Evaluasi posisi NPC dengan menggunakan fungsi fitness :

$$F(X) = \sqrt{(player.x - NPC.x)^2 + (player.y - NPC.y)^2}$$

Perhitungan nilai fitness tiap NPC :

Fitness NPC1 : $F(X)_1 = \sqrt{(500-350)^2 + (380-282.5)^2} = 178$

Fitness NPC2 : $F(X)_2 = \sqrt{(500-430)^2 + (380-290)^2} = 114$

Fitness NPC3 : $F(X)_3 = \sqrt{(500-395)^2 + (380-270)^2} = 152$

Fitness NPC4 : $F(X)_4 = \sqrt{(500-295)^2 + (380-205)^2} = 269$

Fitness NPC5 : $F(X)_5 = \sqrt{(500-315)^2 + (380-265)^2} = 217$

Langkah ketiga

Nilai Pbest = nilai koordinat partikel saat ini

Pbest NPC 1 : $Pbest_1 = (350 , 282.5)$

Pbest NPC 2 : $Pbest_2 = (430 , 290)$

Pbest NPC 3 : $Pbest_3 = (395 , 270)$

Pbest NPC 4 : $Pbest_4 = (295 , 205)$

Pbest NPC 5 : $Pbest_5 = (315 , 265)$

Langkah keempat

Nilai Gbest merupakan nilai titik koordinat NPC yang memiliki nilai *fitness* paling optimal dimana dalam kasus ini akan meminimalkan jarak antara NPC dengan *player*. NPC dengan jarak paling sedikit merupakan NPC yang paling optimal dan dijadikan pemimpin. Adapun nilai *fitness* yang di dapat :

Fitness NPC1 : $F(X)_1 = 178$

Fitness NPC2 : $F(X)_2 = 114$

Fitness NPC3 : $F(X)_3 = 152$

Fitness NPC4 : $F(X)_4 = 269$

Fitness NPC5 : $F(X)_5 = 217$

Dengan melihat nilai *fitness* setiap NPC diatas, NPC yang memiliki nilai *fitness* paling rendah adalah NPC ke 2.

Maka didapat nilai Gbest = koordinat NPC ke 2 = (430,290)

Langkah kelima

NPC lain yang bukan pemimpin memperbaharui kecepatan dengan persamaan sebagai berikut :

$$V_i\,(t+1) = V_i(t)+C_1.r_1(Pbest_i - X_i(t)) + C_2.r_2(Gbest_i - X_i(t))$$

Perhitungan nilai perbaharui kecepatan tiap NPC :

NPC_1 : $Vx_1\,(2) = 80 + 0.4(350 - 350) + 0.7(430 - 350) = 136$

$Vy_1\,(2) = 7.5 + 0.4(282.5-282.5) + 0.7(290-282.5) = 12.75$

NPC_3 : $Vx_3\,(2) = 35 + 0.4(395-395) + 0.7(430-395) = 59.5$

$Vy_3\,(2) = 20 + 0.4(270-270) + 0.7(290-270) = 34$

NPC_4 : $Vx_4\,(2) = 135 + 0.4(295-295) + 0.7(430-295) = 229.5$

$Vy_4\,(2) = 55 + 0.4((205-205) + 0.7(290-205)) = 114.5$

NPC_5 : $Vx_5\,(2) = 115 + 0.4(315-315) + 0.7(430-315) = 195.5$

$Vy_5\,(2) = 25 + 0.4((265-265) + 0.7(390-265) = 42.5$

Langkah keenam

NPC lain yang bukan pemimpin memperbaharui posisi dengan persamaan sebagai berikut:

$$X_i\,(t+1) = X_i\,(t) + V_i\,(t+1)$$

Perhitungan perbaharui kecepatan tiap NPC :

NPC 1 : $X_1(2) = 350 + 136 = 486$

$Y_1(2) = 282.5 + 12.75 = 295.25$

NPC3 : $X_3(2) = 395 + 59.5 = 454.5$

$Y_3(2) = 270 + 34 = 304$

NPC4 : $X_4(2) = 295 + 229.5 = 524.5$

$Y_4(2) = 205 + 114.5 = 319.5$

NPC 5 : $X_5(2) = 315 + 195.5 = 510.5$

$Y_5(2) = 265 + 42.5 = 307.5$

Hasil :

Nilai titik koordinat yang dihasilkan :

Titik awal tiap NPC :	Koordinat baru yang dihasilkan NPC :
$X_1(1) = (350 , 282.5)$	$X_1(2) = (486 , 295.25)$
$X_3(1) = (395 , 270)$	$X_3(2) = (454.5 , 304)$
$X_4(1) = (295 , 205)$	$X_4(2) = (524.5 , 319.5)$
$X_5(1) = (315 , 265)$	$X_5(2) = (510.5 , 307.5)$

Langkah ketujuh

Mengecek posisi NPC apakah sudah konveregen atau belum

Dengan melihat hasil perhitungan pada iterasi ke 0 diatas posisi setiap NPC yang bukan pemimpin belum mencapai titik koordinat NPC pemimpin maka dilakukan kembali perhitungan langkah ke 2 pada iterasi selanjutnya dengan mengevaluasi nilai titik koordinat setiap NPC saat ini kedalam dungsi tujuan.

ITERASI KE 2

Nilai random pada iterasi ke 2:

$r_1 = 0,9$ $r_2 = 0,2$

Langkah pertama

Nilai koordinat NPC saat ini :

Koordinat NPC1 pada iterasi ke 2: X_1= (486 , 295.25)

Koordinat NPC2 pada iterasi ke 2: X_2 = (430,290)

Koordinat NPC 3 pada iterasi ke 2: X_3 = (454.5 , 304)

Koordinat NPC 4 pada iterasi ke 2: X_4= (524.5 , 319.5)

Koordinat NPC 5 pada iterasi kedua: X_5= (510.5 , 307.5)

Langkah kedua

Evaluasi posisi NPC dengan menggunakan fungsi *fitness* :

$$F(X) = \sqrt{(player.x - NPC.x)^2 + (player.y - NPC.y)^2}$$

Perhitungan nilai *fitness* tiap NPC :

Fitness NPC1 : $F(X)_1 = \sqrt{(500-486)^2 + (380-295.25)^2} = 85$

Fitness NPC2 : $F(X)_2 = \sqrt{(500-430)^2 + (380-290)^2} = 114$

Fitness NPC3 : $F(X)_3 = \sqrt{(500-454.5)^2 + (380-304)^2} = 88$

Fitness NPC4 : $F(X)_4 = \sqrt{(500-524.5)^2 + (380-319.5)^2} = 65$

Fitness NPC5 : $F(X)_5 = \sqrt{(500-510.5)^2 + (380-307.5)^2} = 73$

Langkah ketiga

Nilai Pbest = nilai koordinat partikel saat ini

Pbest NPC 1 : $Pbest_1$ = (486, 295.25)

Pbest NPC 2 : $Pbest_2$= (430 , 290)

Pbest NPC 3 : $Pbest_3$ = (454.5, 304)

Pbest NPC 4 : $Pbest_4$ = (524.5, 319.5)

Pbest NPC 5 : $Pbest_5$ = (510.5, 307.5)

Langkah keempat

Nilai Gbest merupakan nilai titik koordinat NPC yang memiliki nilai *fitness* paling optimal dimana dalam kasus ini akan meminimalkan jarak antara NPC dengan *player*. NPC dengan jarak paling sedikit merupakan NPC yang paling optimal dan dijadikan pemimpin. Adapun nilai *fitness* yang di dapat :

Fitness NPC1 : $F(X)_1 = 85$

Fitness NPC2 : $F(X)_2 = 114$

Fitness NPC3 : $F(X) = 88$

Fitness NPC4 : $F(X)_4 = 65$

Fitness NPC5 : $F(X)_5 = 73$

Dengan melihat nilai *fitness* setiap NPC diatas, NPC yang memiliki nilai *fitness* paling rendah adalah NPC ke 4.

Maka didapat nilai Gbest = koordinat NPC ke 4 = (524.5, 319.5)

Langkah kelima

NPC lain yang bukan pemimpin memperbaharui kecepatan dengan persamaan sebagai berikut :

$$V_i\,(t+1) = V_i(t)+C_1.r_1(Pbest_i - X_i(t)) + C_2.r_2(Gbest_i - X_i(t))$$

Perhitungan nilai perbaharui kecepatan tiap NPC :

NPC_1 : $Vx_1\,(3) = 136 + 0.4(486 - 486) + 0.7(524.5 - 486) = 143.7$

$Vy_1\,(3) = 12.75 + 0.4(282.5 - 282.5) + 0.7(319.5 - 282.5) = 17.6$

NPC_2 : $Vx_2\,(3) = 0 + 0.4(430 - 430) + 0.7(524.5 - 430) = 18.9$

$Vy_2\,(3) = 0 + 0.4(290 - 290) + 0.7(319.5 - 290) = 5.9$

NPC_3 : $Vx_3\,(3) = 229.5 + 0.4(454.5 - 454.5) + 0.7(524.5 - 454.5) = 243.5$

$Vy_3\,(3) = 114.5 + 0.4(304 - 304) + 0.7(319.5 - 304) = 117.6$

NPC_5 : $Vx_5\,(3) = 115 + 0.4(510.5 - 510.5) + 0.7(524.5 - 510.5) = 115$

$Vy_5\,(3) = 25 + 0.4(307.5 - 307.5) + 0.7(319.5 - 307.5) = 25$

Langkah keenam

NPC lain yang bukan pemimpin memperbaharui posisi dengan persamaan sebagai berikut:

$X_i (t+1) = X_i (t) + V_i (t+1)$

Perhitungan perbaharui kecepatan tiap NPC :

NPC 1 : $X_1(3) = 486 + 143.7 = 629.7$

$Y_1(3) = 295.25 + 17.6 = 312.85$

NPC2 : $X_2(3) = 430 + 18.9 = 448.9$

$Y_2(3) = 290 + 5.9 = 295.9$

NPC3 : $X_3(3) = 454.5 + 243.5 = 698$

$Y_3(3) = 304 + 117.6 = 421.6$

NPC 5 : $X_5(3) = 524.5 + 115 = 639.5$

$Y_5(3) = 319.5 + 25 = 344.5$

Hasil :

Nilai titik koordinat yang dihasilkan :

Titik awal tiap NPC :	Koordinat baru yang dihasilkan NPC :
$X_1(2) = (486 , 295.25)$	$X_1(3) = (629.7 , 312.85)$
$X_2(2) = (454.5 , 304)$	$X_2(3) = (448.9 , 295.9)$
$X_3(2) = (524.5 , 319.5)$	$X_3(3) = (698 , 421.6)$
$X_5(2) = (510.5 , 307.5)$	$X_5(3) = (639.5 , 344.5)$

Langkah ketujuh

Mengecek posisi NPC apakah sudah konveregen atau belum

Dengan melihat hasil perhitungan pada iterasi ke 0 diatas posisi setiap NPC yang bukan pemimpin belum mencapai titik koordinat NPC pemimpin maka dilakukan kembali perhitungan langkah ke 2 pada iterasi selanjutnya dengan mengevaluasi nilai titik koordinat setiap NPC saat ini kedalam dungsi tujuan.

1.6.2 Kesimpulan Hasil Penyelesaian

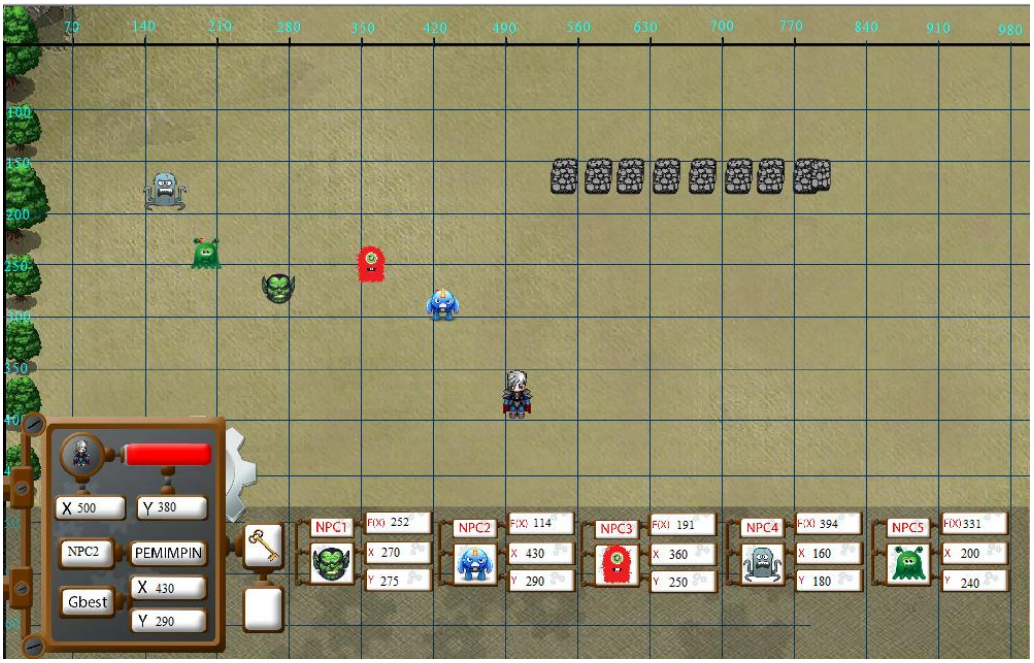
Pada perhitungan diatas menunjukkan setiap NPC dalam kelompok NPC dapat menentukan NPC pemimpin dimana NPC pemimpin merupakan NPC yang memiliki posisi paling baik dari posisi NPC lainnya, posisi NPC paling baik merupakan NPC yang memiliki jarak terpendek dengan *player*. NPC yang memiliki posisi paling baik akan dijadikan pemimpin dan NPC yang lain akan mengikuti pergerakan NPC yang menjadi pemimpin tersebut. Untuk lebih jelasnya hasil perhitungan dan pergerakan NPC pada setiap iterasi dapat dilihat dibawah :

HASIL ITERASI KE 0

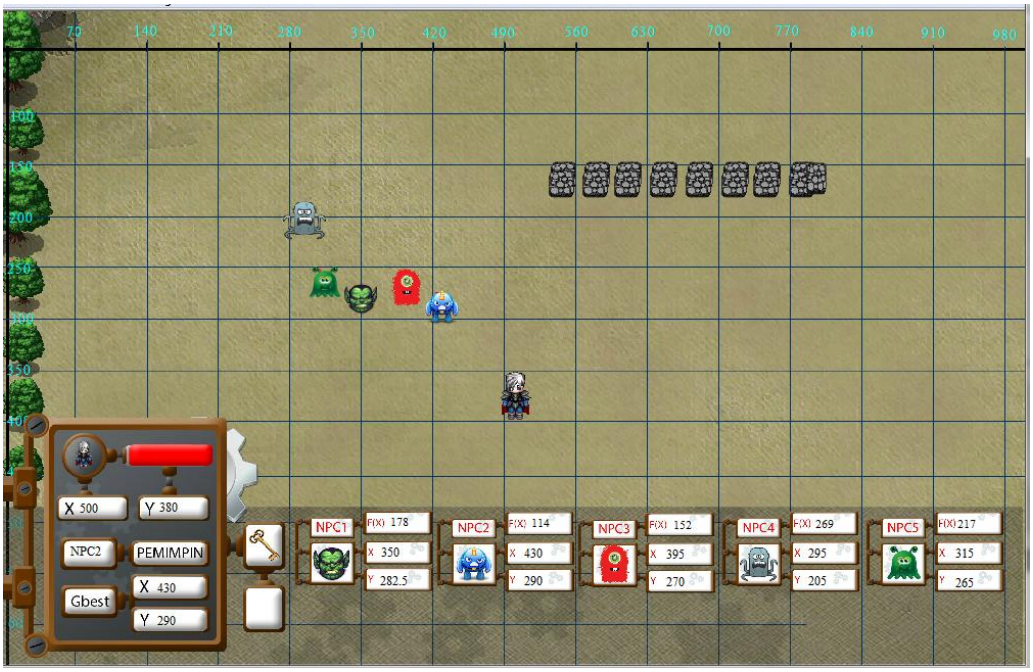
Pada iterasi ke 0 NPC 2 merupakan pemimpin, NPC lain yang bukan pemimpin bergerak mendekati NPC pemimpin. Berikut hasil perhitungan dari iterasi ke 0 yang menujukkan NPC lain yang bukan pemimpin bergerak mendekati NPC pemimpin :

Titik awal :	Koordinat iterasi selanjutnya (1) :
$X_1(0) = (270,275)$	$X_1(1) = (350 , 282.5)$
$X_3(0) = (360,250)$	$X_3(1) = (395 , 270)$
$X_4(0) = (160,180)$	$X_4(1) = (295 , 205)$
$X_5(0) = (200,240)$	$X_5(1) = (315 , 265)$

Untuk pergerakan ditunjukkan pada gambar berikut :



Gambar 1.5 Posisi NPC Pada Iterasi ke 0



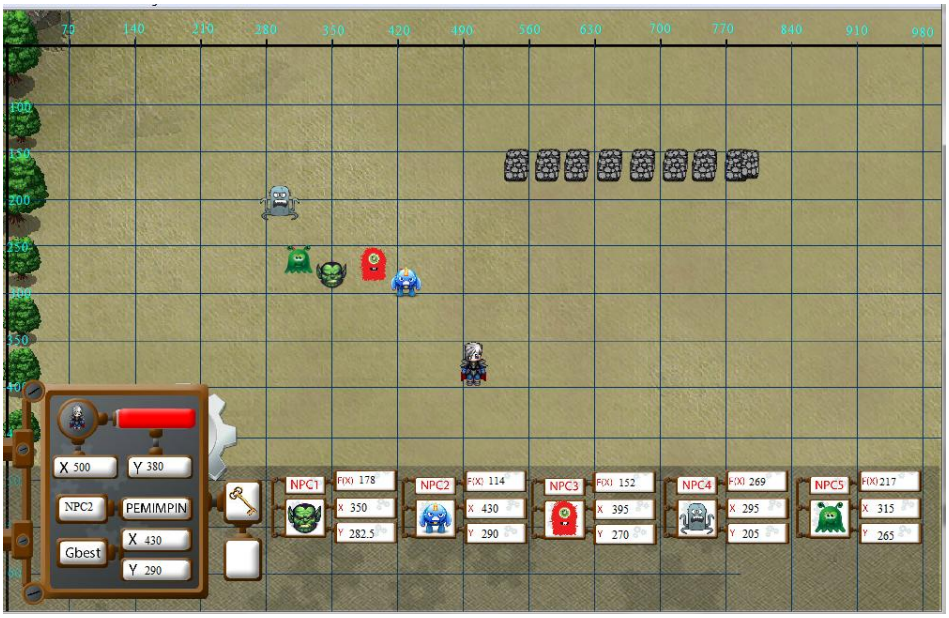
Gambar 1.6 Posisi NPC Pada Iterasi ke 1

HASIL ITERASI KE 1

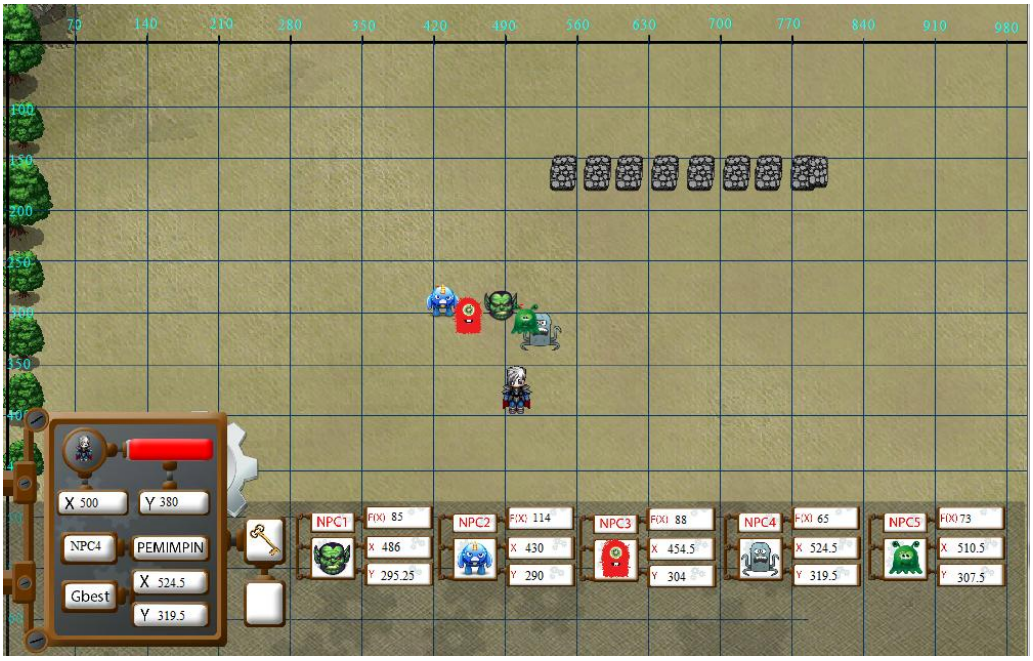
Pada iterasi ke 1 setelah melakukan evaluasi posisi setiap NPC maka diperoleh pemimpin yaitu NPC 2, NPC lain yang bukan pemimpin bergerak mendekati NPC pemimpin. Berikut hasil perhitngan dari iterasi ke 1 yang menujukkan NPC lain yang bukan pemimpin bergerak mendekati NPC pemimpin:

Titik awal :	Koordinat baru yang dihasilkan :
$X_1(1) = (350 , 282.5)$	$X_1(2) = (486 , 295.25)$
$X_3(1) = (395 , 270)$	$X_3(2) = (454.5 , 304)$
$X_4(1) = (295 , 205)$	$X_4(2) = (524.5 , 319.5)$
$X_5(1) = (315 , 265)$	$X_5(2) = (510.5 , 307.5)$

Untuk pergerakan NPC ditunjukkan pada gambar berikut :



Gambar 1.7 Posisi NPC Pada Iterasi ke 1



Gambar 1.8 Posisi NPC Pada Iterasi ke 2

HASIL ITERASI KE 2

Pada iterasi ke 2 setelah melakukan evaluasi posisi NPC maka diperoleh pemimpin baru yaitu NPC 4, NPC lain yang bukan pemimpin bergerak mendekati NPC pemimpin. berikut hasil perhitungan yang dihasilkan pada iterasi ke 2:

Titik awal : Koordinat baru yang dihasilkan :

$X_1(2) = (486 , 295.25)$ $X_1(3) = (629.7 , 312.85)$

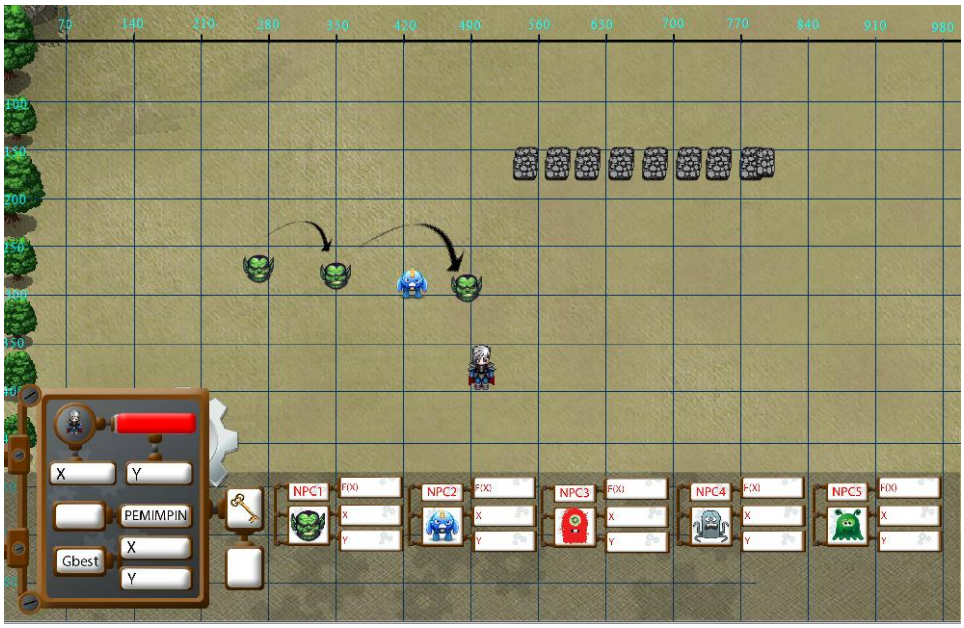
$X_2(2) = (454.5 , 304)$ $X_2(3) = (448.9 , 295.9)$

$X_3(2) = (524.5 , 319.5)$ $X_3(3) = (698 , 421.6)$

$X_5(2) = (510.5 , 307.5)$ $X_5(3) = (639.5 , 344.5)$

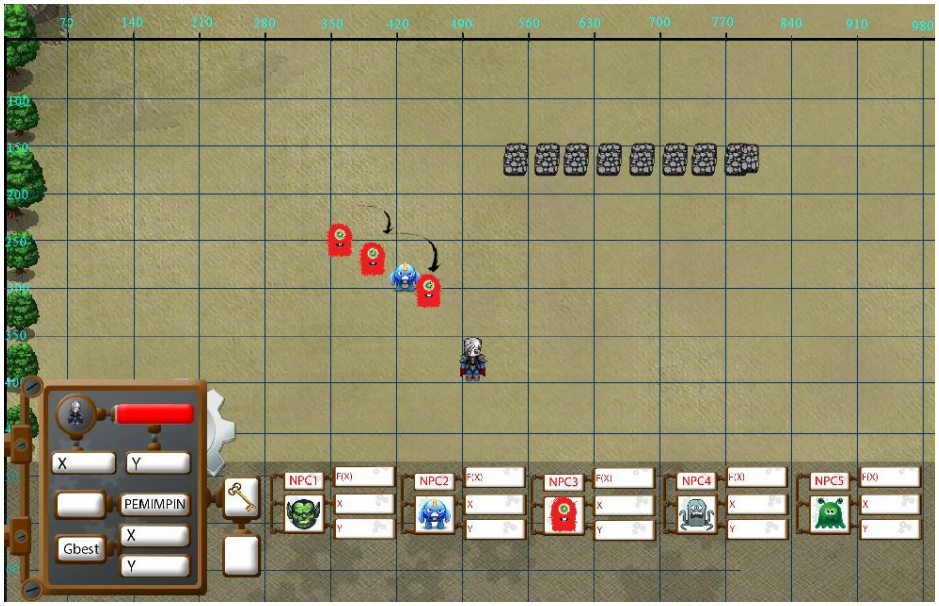
Untuk lebih detailnya gambar dibawah akan memperlihatkan pergerakan setiap NPC :

Pergerakan NPC 1 :



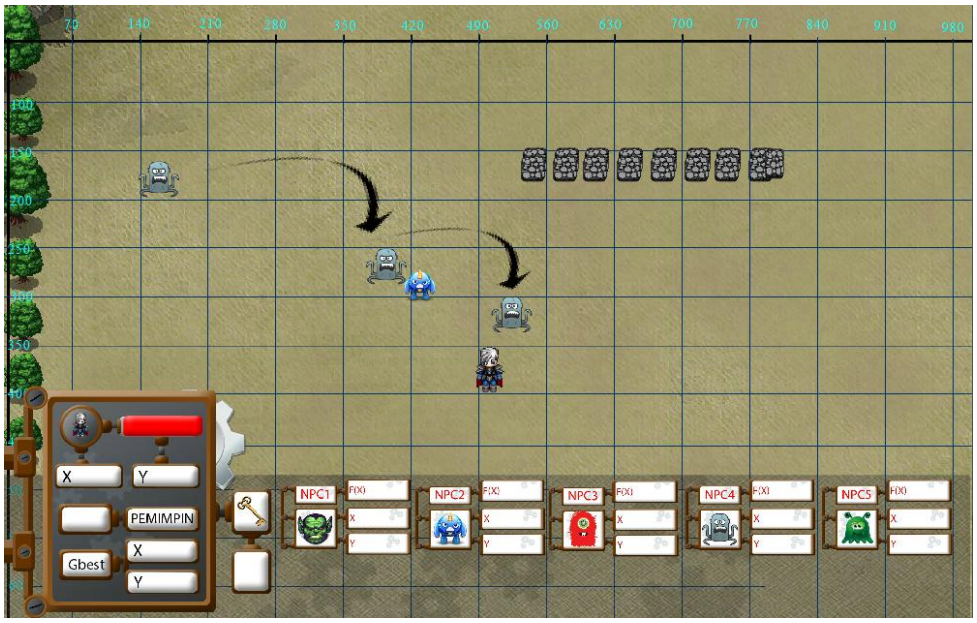
Gambar 1.9 Pergerakan NPC 1

Pergerakan NPC 3 :



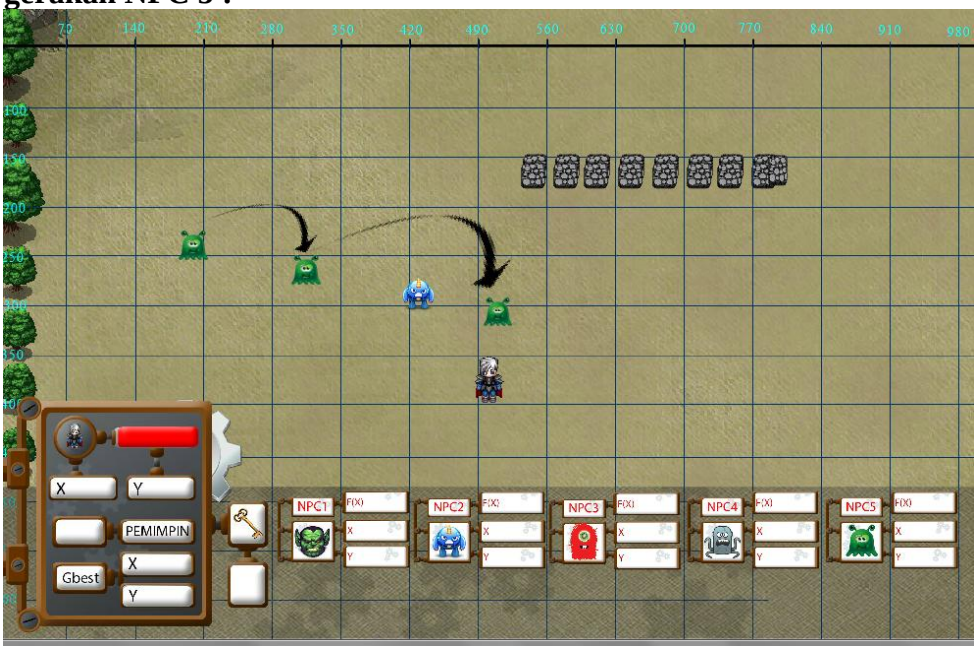
Gambar 1.10 Pergerakan NPC 3

Pergerakan NPC 4 :



Gambar 1.11 Pergerakan NPC 3

Pergerakan NPC 5 :



Gambar 1.12 Pergerakan NPC 5

1.7 Analisis Kebutuhan Non-Fungsional

Analisis kebutuhan non fungsional dilakukan untuk mengetahui spesifikasi kebutuhan untuk sistem. Spesifikasi kebutuhan melibatkan analisis perangkat keras/hardware,analisis perangkat lunak/software.

1.7.1 Analisis Kebutuhan Perangkat Lunak

Perangkat lunak digunakan dalam sebuah sistem merupakan perintah-perintah yang diberikan kepada perangkat keras agar bisa dapat berinteraksi diantara keduanya. Adapun perangkat lunak yang dibutuhkan dalam pembangunan prototype *game* sebagai sarana implementasi algoritma PSO untuk perilaku NPC menyerang secara berkelompok pada game *action*-RPG ini adalah sebagai berikut :

1. Sistem Operasi Windows 7
2. Adobe Flash CS6
3. Adobe Photoshop CS5

1.7.2 Analisis Kebutuhan Perangkat Keras

Komputer terdiri dari perangkat keras dan perangkat lunak yang saling berinteraksi. Perangkat lunak memberikan instruksi-instruksi kepada perangkat keras untuk melakukan suatu tugas tertentu, sehingga dapat menjalankan suatu sistem di dalamnya. Pada Implementasi algoritma PSO untuk perilaku NPC menyerang secara berkelompok ini, perangkat yang digunakan adalah sebagai berikut :

Spesifikasi *minimum* perangkat keras yang dibutuhkan.

a. Prosesor dengan kecepatan 1.8 Ghz.

b. RAM 4 Gb.

c. Hardisk 320 Gb.

d. VGA *Card* 512 Mb.

e. Monitor.

f. *Mouse* dan *Keyboard*.

g. *Speaker*.

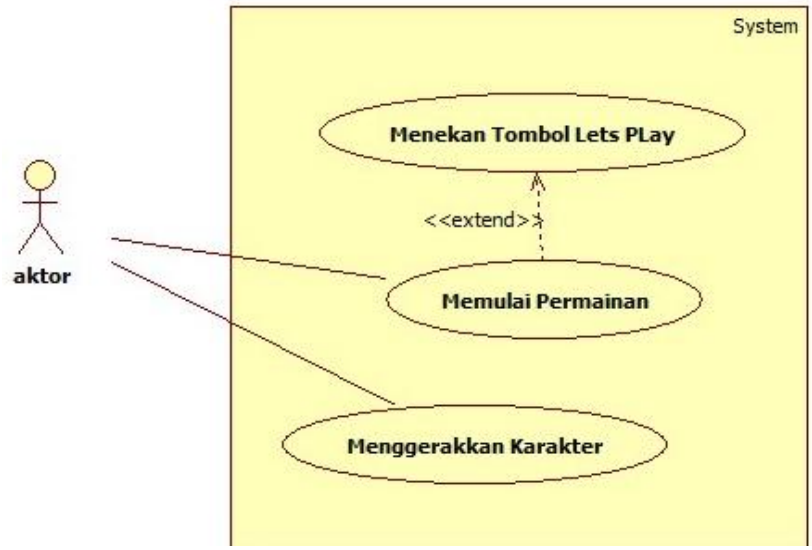
Maka dengan spesifikasi tersebut *game* dapat dijalankan dengan baik dan tidak akan terjadi masalah yang dapat menghambat berjalannya sebuah *game*.

1.8 Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional menggambarkan proses kegiatan yang akan diterapkan dalam sebuah sistem dan menjelaskan kebutuhan yang diperlukan sistem agar sistem dapat berjalan dengan baik sesuai kebutuhan. Pada bagian ini akan dibahas tentang *use case diagram*, *activity diagram*, *class diagram*, dan *sequence diagram*.

1.8.1 Use Case Diagram

Pada sub bab berikut ini dijelaskan bagaimana *Use Case* yang terjadi dalam pembangunan prototype *game* ini yang dimodelkan dalam sekumpulan *Use Case* dan *Actor* dan bagaimana hubungan-hubungannya yang akan dipetakan dalam diagram *Use Case*, dilengkapi dengan skenario. Adapun gambarnya adalah sebagai berikut :



Gambar 1.13 Use Case Diagram

Use case terdiri dari tiga bagian yaitu definisi *Actor*, definisi *use case*, dan skenario *use case*.

1. **Definisi *Actor***

Actor yaitu pihak yang mengakses *use case.* Hanya ada satu *actor* yang berperan yaitu *player /* pemain yang akan memainkan *game.*

2. Definisi *Use Case*

Use case berfungsi untuk mewakili apa yang sistem lakukan. Definisi *use case* bisa dilihat pada tabel 2 :

Tabel 1.2 Definisi *Use Case*

No	*Use Case*	Deskripsi
1	Memulai Permainan	Merupakan proses untuk memulai permainan dimana actor masuk pada halaman awal game.
2	Menekan Tombol *Lets Play*	Merupakan proses dimana aktor enekan tombol *Lets Play* untuk masuk kehalaman arena permainan.
3	Menggerakkan karakter	Proses yang dilakukan oleh aktor untuk menggerakkan karakter *player* diarena permainan

1. Skenario *Use Case*

Skenario *use case* merupakan hasil instansiasi dari setiap *use case*. Berikut adalah skenario jalannya masing-masing *use case game* yang telah didefinisikan sebelumnya :

1. Skenario *use case* Memulai Permainan

Tabel 1.3 Skenario *use case* Memulai Permainan

Identifikasi	
No	1
Nama *Use Case*	Memulai permainan
Aktor	*Player*
Kondisi awal	Pemain belum memulai permainan
Skenario Utama	
Aksi Aktor	**Reaksi Sistem**
1. Menekan tombol *control + enter* pada *keyboard*	
	2.Sistem menampilkan halaman awal *game*
Kondisi akhir	Aktor berada pada halaman awal *game*

2. Skenario *use case* Menekan Tombol *Lets Play*

Tabel 1.4 Skenario *use case* Menekan Tombol *Lets Play*

Identifikasi	
No	2
Nama *Use Case*	Menekan Tombol *Lets Play*
Aktor	*Player*
Kondisi awal	Aktor berada pada halaman awal *game*
Skenario Utama	

Aksi Aktor	Reaksi Sistem
1.Menekan Tombol *Lets Play*	
	2.Sistem menampilkan halaman arena permainan
Kondisi akhir	Aktor berada pada halaman arena permainan

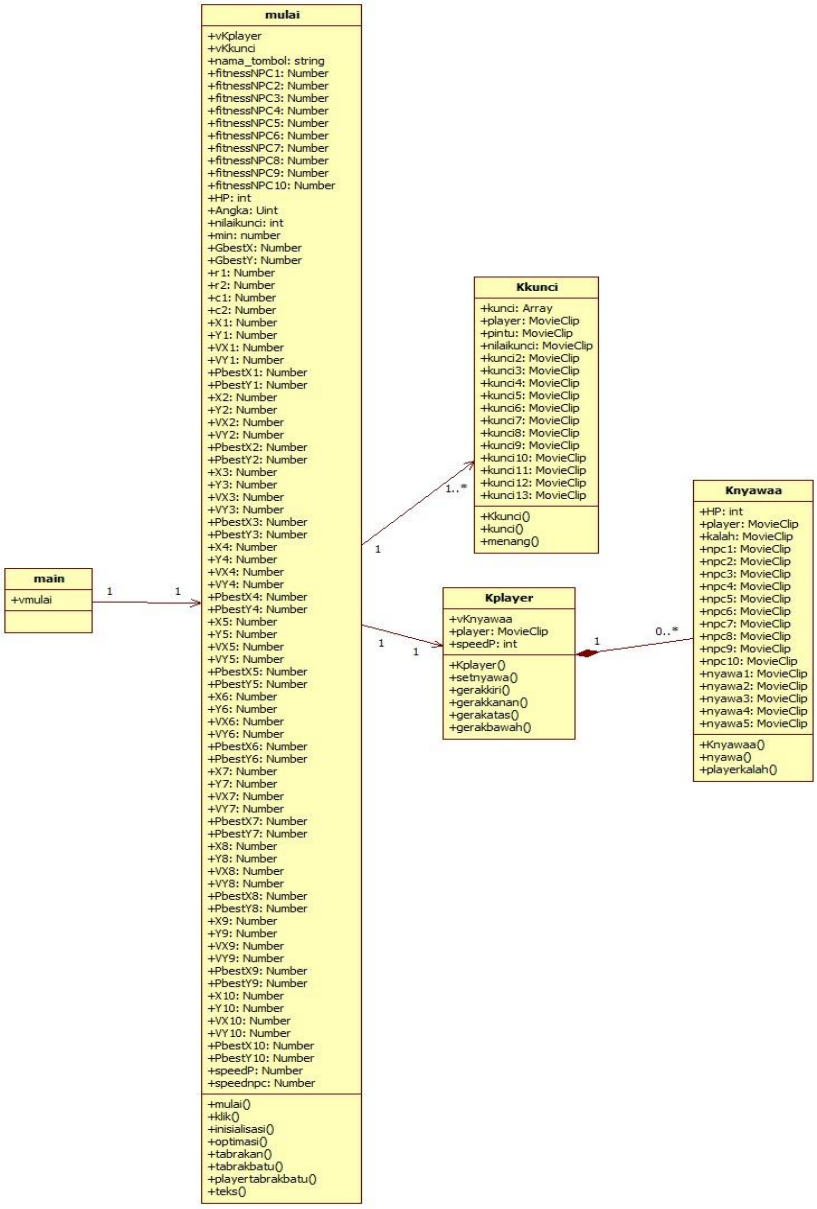
3. Skenario *use case* Menggerakkan karakter

Tabel 1.5 Skenario *use case* Menggerakkan karakter

Identifikasi	
No	2
Nama *Use Case*	Menggerakkan karakter
Aktor	*Player*
Kondisi awal	Pemain sudah berada dalam arena permainan dan masih pada posisi awal (belum bergerak)
Skenario Utama	
Aksi Aktor	**Reaksi Sistem**
1. Menekan tombol navigasi kiri kanan atas atau bawah pada *keyboard*	
	2.Menggerakkan karakter player ke kiri, ke kanan, ke atas, atau ke bawah
Kondisi akhir	Karakter player bergerak

1.8.1 Class Diagram

Class diagram menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang dibuat untuk membangun sistem. Class Diagram dapat dilihat pada gambar berikut :



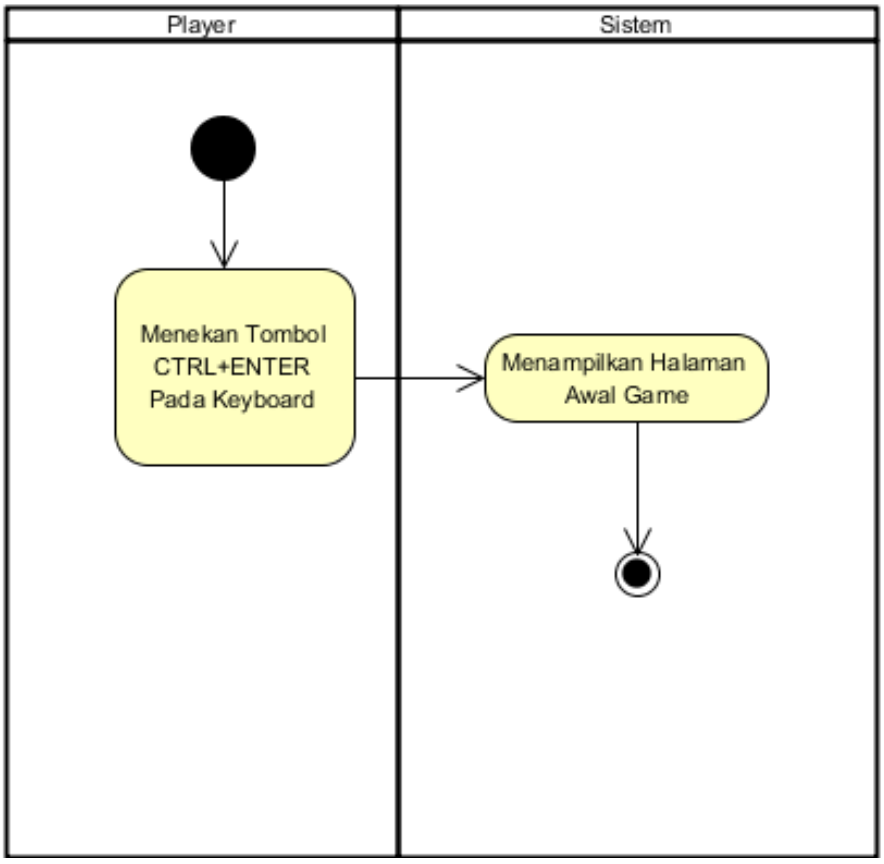
Gambar 1.14 Class Diagram

1.8.2 Activity Diagram

Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. Berikut diagram aktifitas :

Berikut ini adalah diagram yang menunjukan alur aksi pada aktivitas Memulai permainan dapat dilihat pada gambar 3.15 :

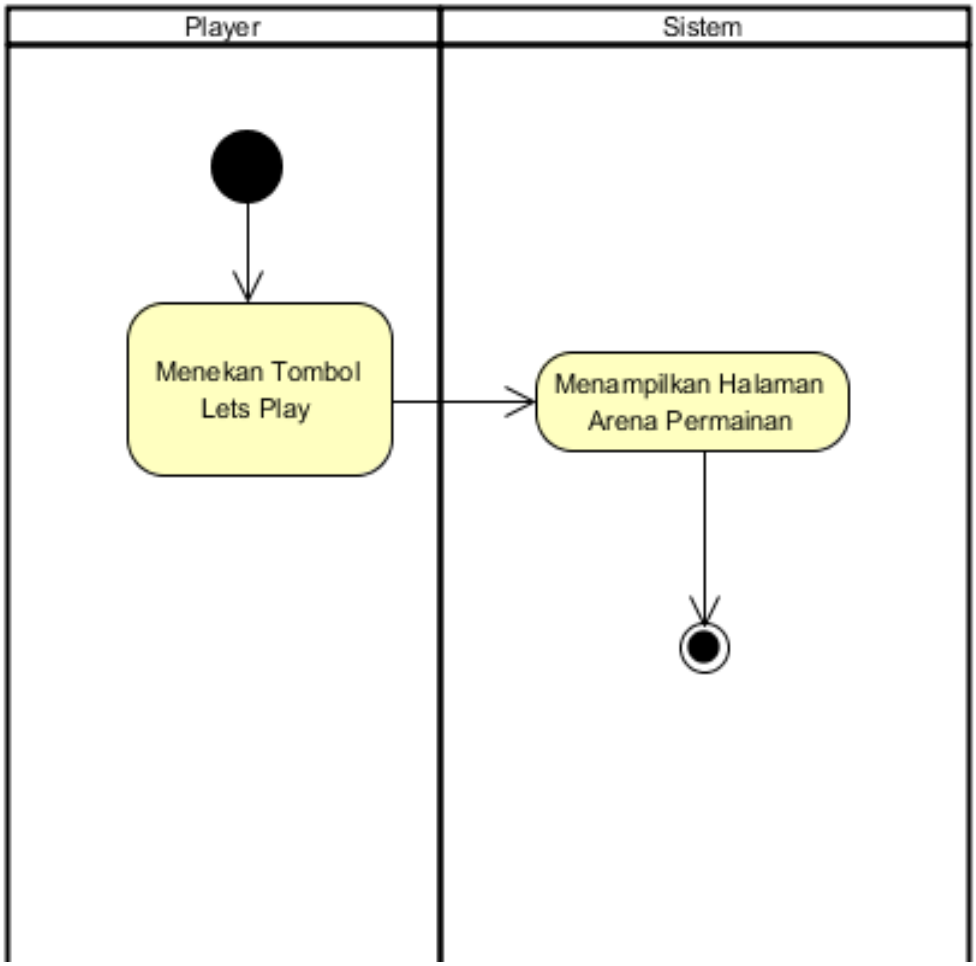
1. *Activity Diagram* Memulai Permainan



Gambar 1.15 *Activity Diagram* Memulai Permainan

Berikut ini adalah diagram yang menunjukan alur aksi pada aktivitas Menekan Tombol *Lets Play* dapat dilihat pada gambar 3.16

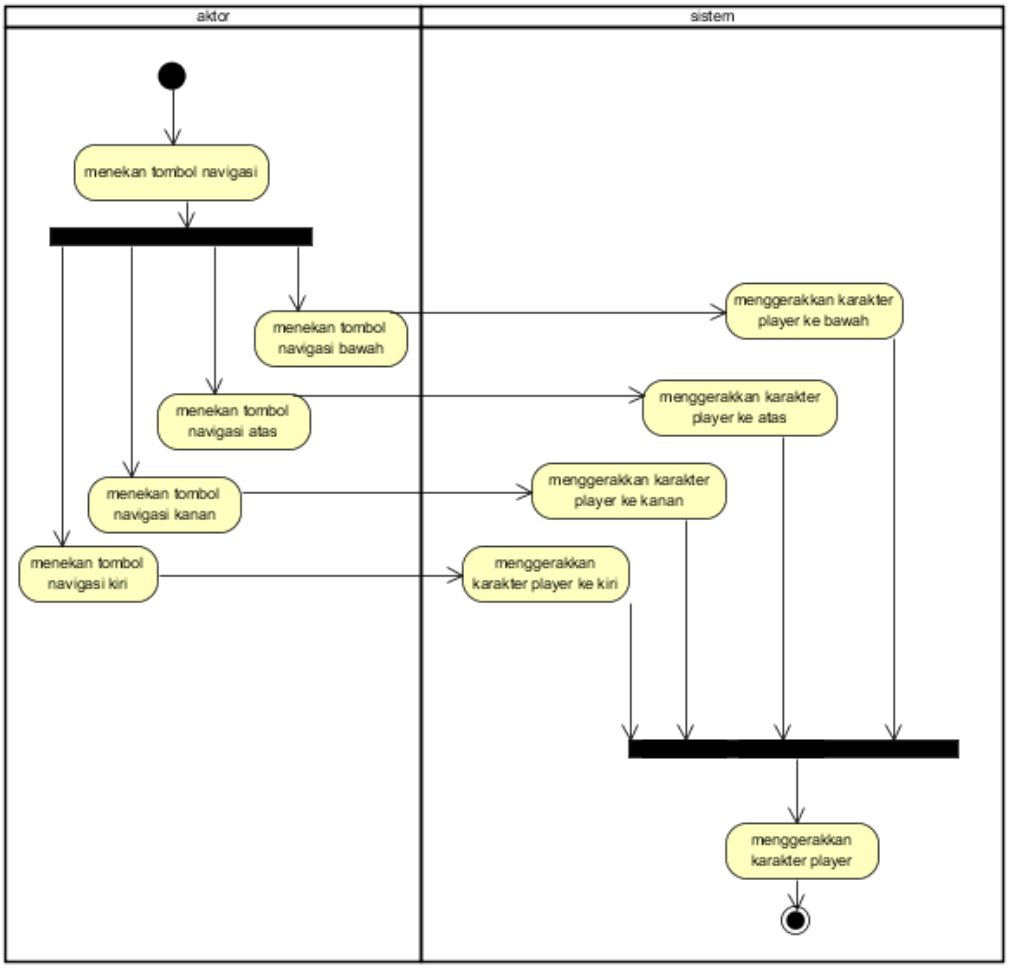
2. *Activity Diagram* Menekan Tombol *Lets Play*



Gambar 1.16 Activity Diagram Menekan Tombol Lets Play

Berikut ini adalah diagram yang menunjukan alur aksi pada aktivitas menggerakkan karakter dapat dilihat pada gambar 3.17 :

3. *Activity Diagram* Menggerakkan karakter

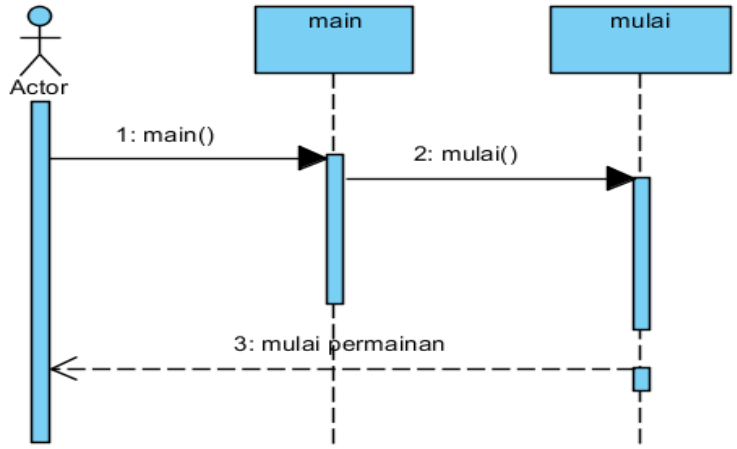


Gambar 1.17 Activity Diagram Menggerakkan karakter

1.8.3 Sequence Diagram

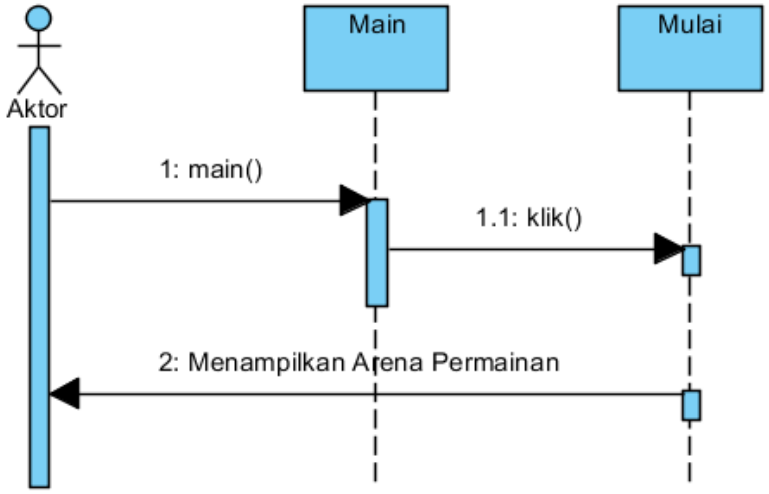
Sequence Diagram menggambarkan interaksi antar masing-masing objek pada setiap *use case* dalam urutan waktu. Interaksi ini berupa pengiriman serangkaian data antar objek-objek yang saling berinteraksi. Sequence diagram dapat dilihat pada gambar berikut.

1. *Sequence Diagram* Memulai Permainan



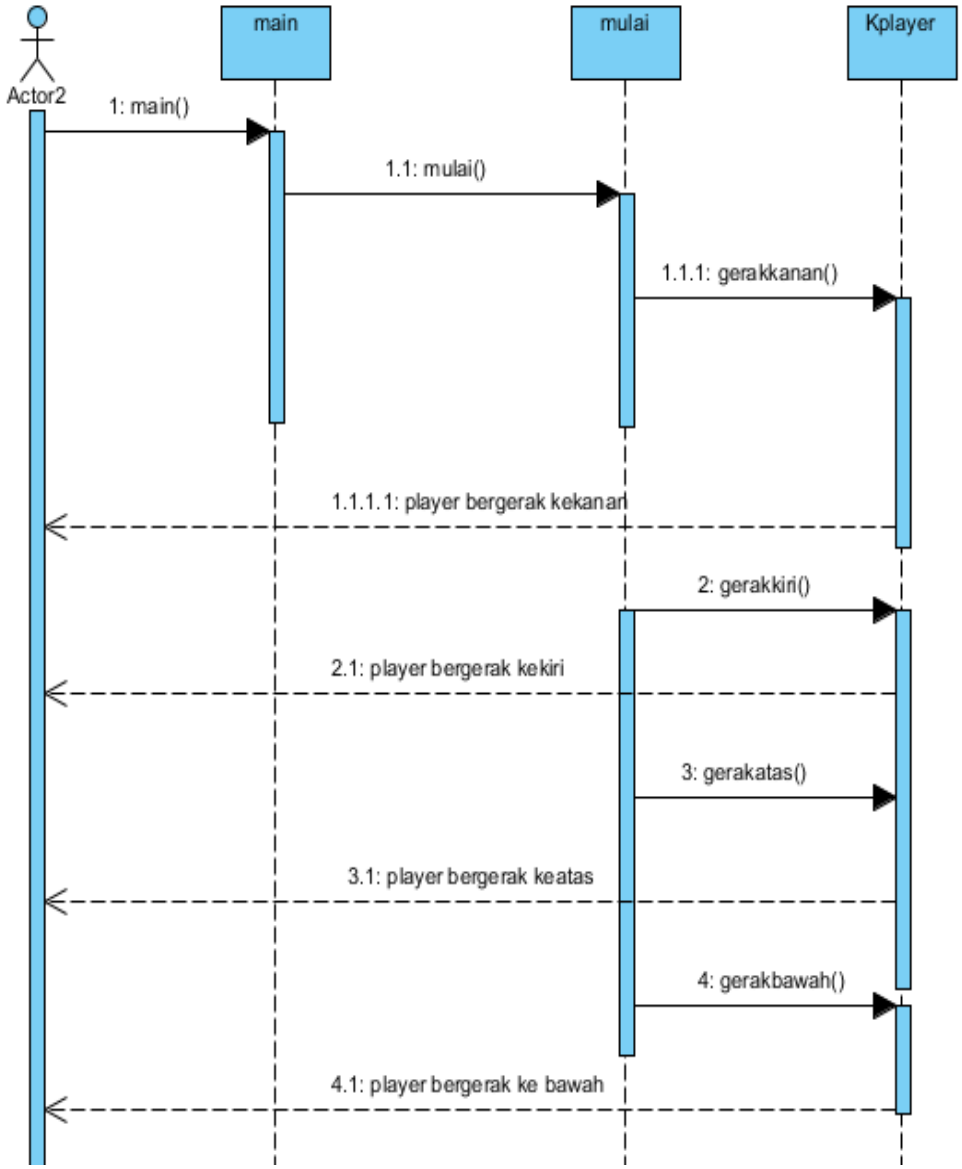
Gambar 1.18 *Sequence Diagram* Memulai Permainan

2. *Sequence Diagram* Menekan Tombol *Lets Play*



Gambar 1.19 Sequence Diagram Menekan Tombol Lets Play

3. *Sequence Diagram* Menggerakkan karakter



Gambar 1.20 Sequence Diagram Menggerakkan karakter