BAB 3

ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Sistem

Pada bagian analisis sistem ini, akan dilakukan berbagai macam analisis yang berhubungan dengan *game* yang akan dibangun, mulai dari analisis terhadap *game* sejenis serta analisis berbagai kebutuhan baik fungsional maupun non-fungsional.

3.1.1 Analisis *Game* Sejenis pada *Labirinth Prison*

Pada bagian analisis sistem ini, akan dilakukan berbagai macam analisis yang berhubungan dengan *game* yang akan dibangun, mulai dari analisis terhadap *game* sejenis serta analisis berbagai kebutuhan baik fungsional maupun non-fungsional.

3.1.1.1 Pengenalan *Game Labirinth Prison*

Permainan *game Labirinth Prison* adalah *game* berjenis *arcade* yang ada www.freearcade.com. *Game* ini adalah *game* 2D (dua dimensi) yang dibangun menggunakan flash. Sesuai dengan namanya *game* ini memiliki arena permainan yang berbentuk seperti labirin dengan banyak penghalang yang tidak dapat dilalui oleh pemain ataupun musuh. Misi utama pemain adalah harus mendapatkan semua koin untuk kemudian mendapatkan kunci untuk lolos dari penjara.



Gambar 3.1 *Screenshoot* tampilan permainan *game Labirinth Prison*

3.1.1.2 *Gameplay*

Pada *game Labirinth Prison*, ini pemain harus mengumpulkan semua koin yang tersebar di arena permainan. Apabila berhasil mengumpulkan semua koin maka akan mendapatkan kunci untuk melangkah ke level selanjutnya. Diarena permainan terdapat musuh yang akan melakukan patroli. Musuh akan mengejar pemain yang mendekati daerahnya.

3.1.1.3 Elemen *Game*

Elemen permainan yang terdapat dalam permainan *game Labirinth Prison* diantaranya adalah sebagai berikut:

1. Arena Permainan

   Pada permainan *game Labirinth Prison* arena permainannya adalah disebuah penjara yang dipenuhi banyak rintangan dan berbentuk seperti layaknya

sebuah labirin. Diarena permainan inilah entitas pemain harus meloloskan diri dari kejaran musuh.

2. Pemain

   Pemain merupakan entitas utama yang akan dikendalikan oleh pengguna aplikasi dalam arena permainan. Pemain harus mengumpulkan semua koin yang terdapat dalam arena permainan, apabila dapat mengumpulkan semua koin maka akan keluar kunci yang digunakan untuk membuka pintu dan dapat melangkah ke level selanjutnya. Pemain juga harus menghindar dari kejaran musuh dalam arena permainan.

3. Musuh

   Musuh dalam *game Labirinth Prison* adalah entitas yang akan mengejar pemain dalam menyelesaikan misi diarena permainan. Musuhnya terdiri dari beberapa jenis diantaranya ulat namun dalam *game* ini musuh ditiap levelnya hanya memiliki kemampuan yang sama.

4. Koin

   Koin adalah entitas yang harus dikumpulkan oleh pemain untuk mendapatkan kunci, sehingga pemain dapat menuju ke level berikutnya.

5. *Item Trap*

   Item Trap adalah entitas yang digunakan oleh pemain untuk menjebak musuh yang berada dalam arena permainan. Apabila terkena jebakan darah/*life point* dari musuh akan berkurang.

3.1.1.4 Alur pada Permainan *Game Labirinth Prison*

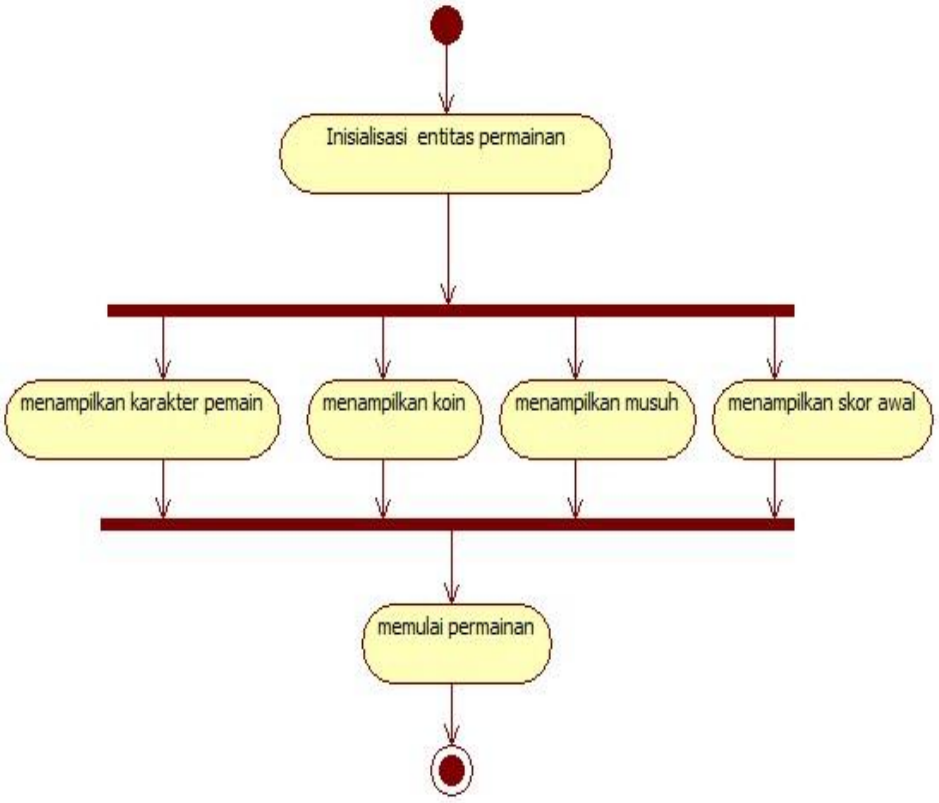
Alur dalam permainan *game Labirinth Prison* disusun oleh beberapa hal sebagai berikut :

A. Alur Sistem

Alur Sistem pada *game Labirinth Prison* adalah sebagai berikut :

1. Sistem akan menginialisasi seluruh entitas permainan yaitu arena permainan, pemain, musuh, koin, dan skor awal permainan.
2. Penghalang yang tidak dapat dilewati pemain dan musuh berada pada posisi tertentu didalam arena permainan dan ditentukan oleh sistem secara acak.
3. Posisi musuh berada pada posisi tertentu yang ditentukan secara acak oleh sistem.
4. Koin berada setiap jalur diarena permainan.
5. Posisi pemain berada disudut kiri bawah
6. Permainan dimulai.
7. Pengguna mengontrol pergerakkan pemain dengan memasukkan perintah tertentu melalui *keyboard.*
8. Sistem akan melakukan update terhadap seluruh entitas permainan, *life point* pemain dan skor pada *game*.

Adapun alur sistem dari permainan *game Labirinth Prison* ketika permainan dimulai dapat digambarkan melalui *activity diagram* berikut ini :



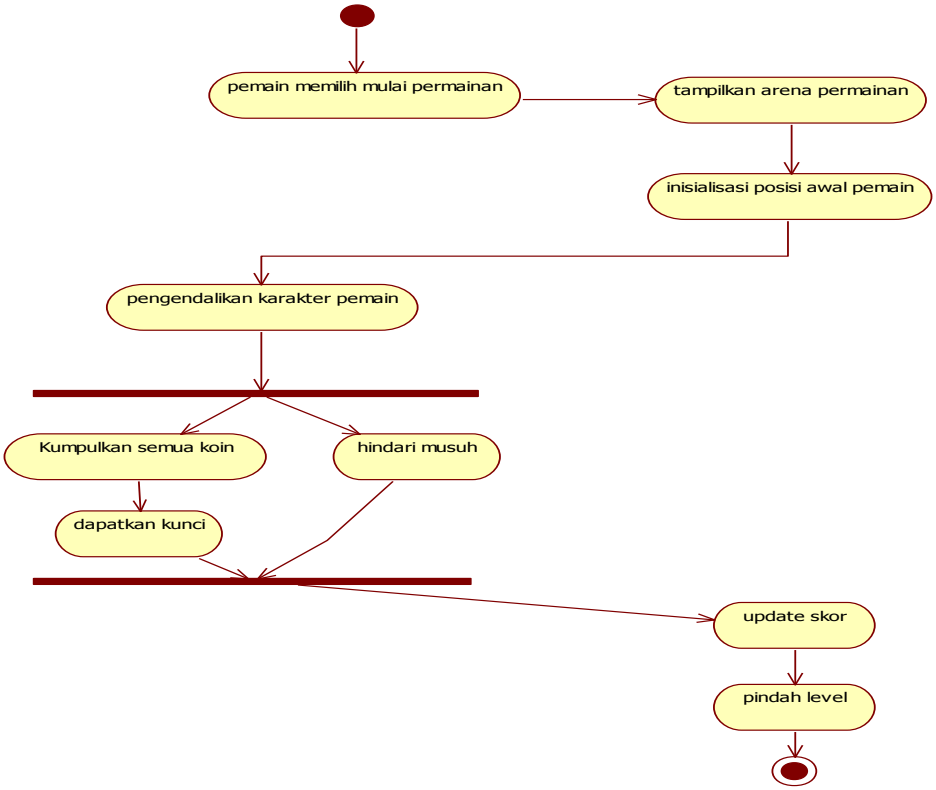
Gambar 3.2 *activity diagram* alur sistem pada *game Labirinth Prison*

B. Alur Permainan

Alur permainan pada *game Labirinth Prison* adalah sebagai berikut :

a. Pemain terjebak dalam sebuah penjara yang dipenuhi dengan musuh yang menjaga penjara tersebut.
b. Pemain harus mengumpulkan koin untuk mendapatkan kunci yang akan digunakan untuk melangkah ke level yang berikutnya.
c. Pemain harus menghindar dari musuh yang berada di arena permainan.
d. Jika pemain ditabrak oleh musuh, maka *life point*/darah akan berkurang dan apabila habis maka permainan akan berakhir.

Adapun alur permainan dari *game Labirint Prison* dapat di gambarkan melalui *activity diagram* berikut ini :



Gambar 3.3 alur permainan pada *game Labirinth Prison*

C. Misi yang harus dicapai Pemain

Pemain harus lolos dari penjara, dengan mengumpulkan semua koin didalam arena permainan. Ketika semua koin terkumpul maka pemain akan mendapatkan kunci untuk melangkah ke level berikutnya.

3.1.1.5 Objek dalam *Game Labirinth Prison*

Beberapa objek yang terlibat dalam permainan *game Labirinth Prison* adalah sebagai berikut :

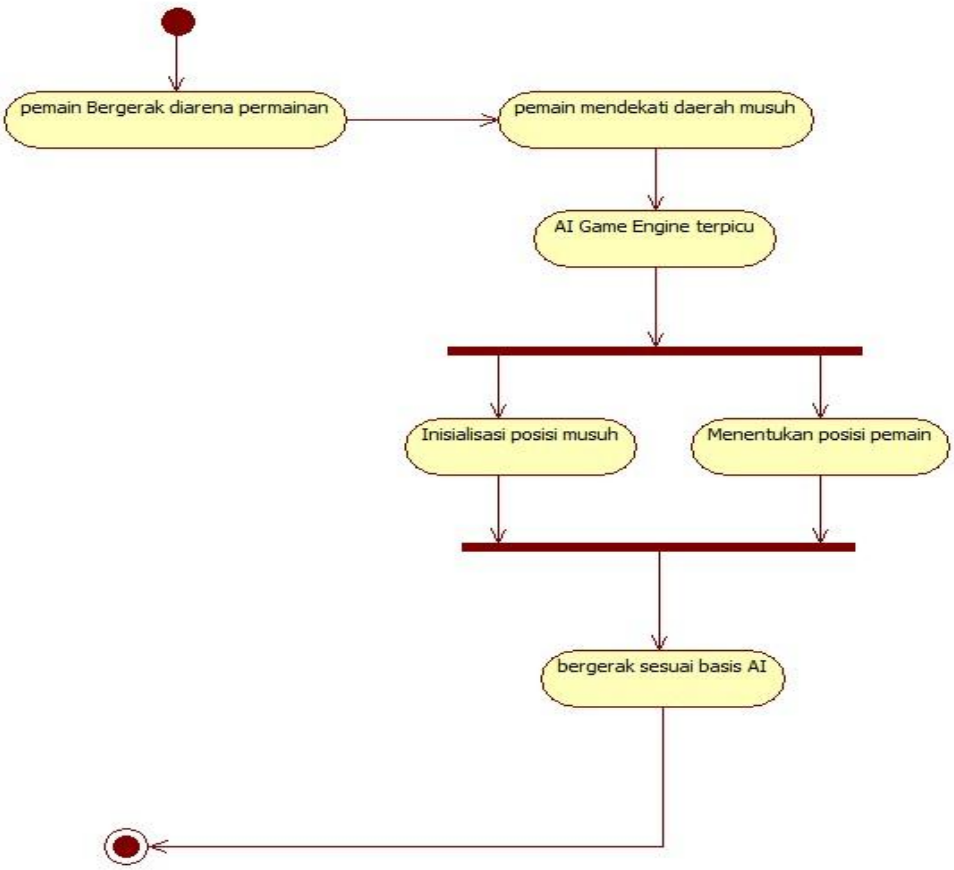
NO	NAMA	GAMBAR	PENJELASAN
1.	Pemain / *player*		Ini adalah karakter utama dalam *game* yang dikendalikan oleh pemain melalui inputan dari *keyboard*.
2.	*Pinky Slime*		Ini adalah salah karakter musuh yang berada dalam *game Labirinth Prison* . musuh ini akan bergerak secara acak diarena permainan.
3.	*Worm*		Ini adalah karakter musuh yang bernama *worm* . musuh ini akan bergerak secara acak diarena permainan.
4.	Koin		Ini adalah koin yang harus dikumpulkan oleh pemain untuk bisa mendapatkan kunci dan lolos ke level berikutnya. Yang berada di arena permainan.
5.	*Item Trap*		Ini adalah *item* yang digunakan pemain untuk menjebak musuh. Apabila musuh terkena jebakan maka *life point* dari musuh akan berkurang

Tabel 3.1 Objek yang terlibat dalam *game Labirinth Prison*.

3.1.1.6 Analisis Metode pada *game Labirinth Prison*

Pada permainan *game Labirinth Prison* objek musuh menerapkan suatu metode tertentu sehingga dapat bergerak dengan pola acak diarena permainan. Objek musuh juga ditanamkan *AI game engine* melalui pengkodean dalam tubuh

programnya yang akan terpicu apabila pemain mendekati daerah musuh. *AI game engine* akan melakukan pencarian terhadap objek pemain. Namun *AI game engine* terlihat masih kurang sempurna, hal ini dikarenakan objek musuh tidak dapat melacak posisi pemain apabila memiliki jarak yang cukup jauh sehingga apabila pemain telah berada cukup jauh dari posisi pemain, musuh akan kembali bergerak secara acak diarena permainan. Lain halnya pada permainan *game* Pacman, *AI game engine* pada game Pacman mampu melacak lokasi pemain walaupun jaraknya jauh. Sehingga semua musuh diarena permainan akan melakukan pengejaran terhadap pemain. Adapun *Activity* diagram metode pergerakan musuh adalah sebagai berikut :



Gambar 3.4 *Activity Diagram* pergerakan musuh

Pada gambar 3.4 dijelaskan bahwa *AI game engine* hanya akan terpicu apabila pemain mendekati area sekitar musuh. *AI game engine* akan menginisialisasi posisi musuh yang merupakan titik awal , posisi pemain dan setelah itu *AI game engin*e akan menentukan arah pergerakan musuh untuk melakukan pengejaran kepada pemain.

3.1.1.7 Analisis Masalah

Berdasarkan hasil observasi dari analisis *game* sejenis diketahui bahwa beberapa hal yang menyebabkan *game Labirinth Prison* kurang menarik dan diminati oleh para pengguna diantaranya :

1. Alur permainan yang monoton
2. Lemahnya penerapan AI pada objek musuh.
3. Kurangnya variatifnya entitas permainan terutama pada kemampuan musuh

Hal –hal tersebutlah yang menyebabkan permainan *Labirinth Prison* menjadi kurang menarik dan sepi peminat



Gambar 3.5 *Play Count Game Labirin Prison*

Dari gambar 3.5 diatas terlihat kalau *game labirinth Prison* kurang menarik dengan jumlah pemain yang memainkannya hanya sekitar tiga ribu lebih beda halnya yang terjadi pada *game Battle Tank* dan *Wicky Woo* yang telah dimainkan oleh pemain lebih dari enam ratus ribu kali.



Gambar 3.6 *Play Count Game Battle Tank*



Gambar 3.7 *Play Count Game Wicky Woo*

3.1.2 Analisis Perancangan *Game Death Castle*

Pada analisis ini, akan dilakukan analisis terhadap perancangan *game* yang akan dibangun. Tujuan dari analisis ini yaitu mendapatkan konsep permainan dari *game Death Castle*.

3.1.2.1 Pengenalan *Game Death Castle*

Death Castle adalah merupakan *game* berjenis *arcade* yang akan dibangun. Nama *castle* diambil karena arena permainan dari *game* ini berlokasi didalam sebuah kastil sedangkan kata *Death* digunakan karena dalam kastil tersebut terdapat berbagai macam musuh yang yang harus dihadapi oleh pemain. *Death Castle* merupakan *game* dengan sistem single player dan dibangun dengan grafis 2D(dua dimensi).

3.1.2.2 *Storyline*

Game Death Castle bercerita tentang seorang penjelajah bernama *Explorer* yang tersesat dalam sebuah kastil bernama "*Death Castle"*. Kastil ini dijaga oleh Boss monster dan banyak anak buahnya. Misi utama dari *game* ini adalah keluar dari *Death Castle.* Untuk keluar kastil ini *Explorer* harus melewati dua lantai / dua level, untuk melewati tiap lantainya *Explorer* harus menutup sejumlah peti di masing-masing lantai, namun untuk menemukan kunci tersebut tidaklah mudah dikarenakan *Explorer* harus menghadapi *Ghost* di lantai paling atas, lantai selanjutnya *Explorer* harus menghadapi *Ghost* dan *Beast*.

3.1.2.3 *Gameplay*

Tugas utama pemain (pengguna) dalam memainkan aplikasi ini yaitu harus melalui 2 level, dimana tiap level memiliki tingkat kesulitan yang berbeda, semakin tinggi level maka tingkat kesulitan akan semakin sulit. Tiap level pemain harus menyelesaikan misi yaitu menutup sejumlah peti untuk melangkah ke level

berikutnya. Musuh akan menghalangi pemain untuk menyelesaikan misi. Akan tetapi, pemain dapat mengalahkan musuh dengan mengambil *item-item* yang tersedia diarena permainan misalnya tembak dan bom.

3.1.2.4 Elemen *Game*

Elemen permainan yang terdapat dalam permainan *game Death Castle* diantaranya adalah sebagai berikut:

1. Arena Permainan

   Pada permainan *game Death Castle* arena permainannya adalah disebuah kastil yang berbentuk kotak dan dipenuhi banyak rintangan atau halangan yang tidak dapat dilalui oleh pemain dan musuh. Diarena permainan inilah entitas pemain harus meloloskan diri dari kejaran musuh.

2. Pemain (*Explorer*)

   Pemain bernama *Explorer* merupakan aktor utama yang menggunakan aplikasi *game*. Ia mengontrol pergerakan untuk bergerak dari dari satu titik ke titik yang lain pada area permainan. Pemain memiliki misi yaitu menutup sejumlah peti untuk mendapatkan kunci dan menghindari kejaran musuh. Pemain dapat mengalahkan musuh dengan mengambil item-item yang tersedia di arena permainan seperti tembak dan bom.

3. Musuh (*Non-Playable Character*)

   Pada *game Death Castle* musuh terdiri dari tiga jenis yaitu *Ghost, Beast*. Musuh *Ghost* akan akan keluar dari peti untuk mengajar pemain, pemain

dapat mengalahkan *Ghost* dengan menembaknya menggunakan tembak. *Beast* juga akan mengejar pemain akan tetapi *Beast* hanya dapat dikalahkan oleh senjata peluru *cannon*.

4. Peti

   Peti akan mengeluarkan musuh *Ghost*, oleh sebab itu pemain harus menutup sejumlah peti dalam arena permainan. Apabila berhasil menutup semua peti pemain akan melangkah ke level berikutnya.

3.1.2.5 Alur pada *Game Death Castle*

Alur pada *game Death Castle* tidak jauh berbeda dengan yang terdapat dalam *game Labirinth Priso*n namun dilakukan beberapa pengembangan diantaranya :

A. Alur Sistem

Alur sistem pada *game Death Castle* adalah sebagai berikut :

1. Sistem menginisialisasi seluruh entitas yang ada didalam arena permainan seperti pemain, musuh dan item-item.
2. Peti dan penghalang dalam arena permainan ditentukan secara acak oleh sistem
3. Posisi pemain berada pada posisi kiri atas diarena permainan
4. Permainan dimulai
5. Peti akan mengeluarkan musuh *ghost*

6. Pemain akan dikontrol dan melalukan tembakan dengan memasukkan perintah melalu*i keyboard*
7. Sistem melakukan *update* terhadap posisi seluruh entitas permainan dan logika dari sistem.

B. Alur Permainan

Alur permainan pada *game Labirinth Prison* adalah sebagai berikut :

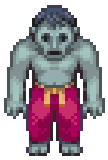
1. Pemain terkurung sebuah kastil bernama *Death Castle* yang dipenuhi oleh berbagai musuh, pemain harus menemukan jalan keluar untuk meloloskan diri dari kastil tersebut.
2. Pemain harus menghindari kejaran musuh yang berada pada arena permainan.
3. Jika musuh ditembak oleh pemain, maka musuh tersebut akan hilang dari arena permainan.
4. Pemain harus menutup sejumlah peti ditiap levelnya.
5. Pemain akan berpindah ke level selanjutnya apabila telah menutup semua peti di arena permainan

C. Misi yang harus dicapai Pemain

Pemain harus lolos dari *Death Castle*, dengan menutup semua peti didalam arena permainan. Ketika semua semua peti tertutup, maka pemain akan melangkah ke level berikutnya.

3.1.2.6 Objek yang Terlibat dalam Permainan *Game Death Castle*

Beberapa Objek yang berada dalam permainan *game Death Castle* adalah sebagai berikut :

NO	NAMA	GAMBAR	PENJELASAN
1.	Explorer		Ini adalah karakter utama dalam game yang dikendalikan oleh pemain melalui inputan dari *keyboard*.
2.	Ghost		Ini adalah karakter musuh yang bernama *ghost*, musuh ini akan keluar dari peti mati. *Ghost* akan mati apabila ditembak menggunakan peluru
3.	Beast		Ini adalah karakter musuh yang bernama *beast*, musuh ini akan muncul pada level kedua. *Beast* dapat dikalahkan dengan peluru *cannon* .
5.	Peluru		Merupakan entitas yang akan keluar dari senjata pemain ketika pemain memasukkan perintah menembak melalui perantara *keyboard*
6.	Peluru Cannon		Merupakan entitas yang akan keluar dari senjata pemain ketika pemain memasukkan perintah menembak melalui perantara *keyboard*. Peluru ini didapat apabila pemain mendapatkan *item* peluru *cannon.*

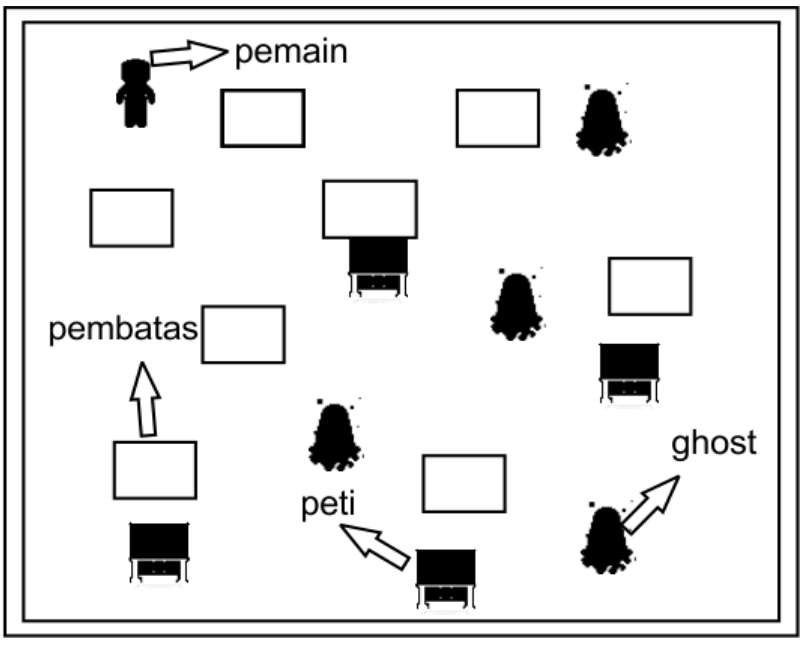
NO	NAMA	GAMBAR	PENJELASAN
7.	*Item* peluru *cannon*		*Item* ini akan mengubah peluru dari senjata pemain menjadi peluru *cannon*

Tabel 3.2 Entitas *game Death Castle*

3.1.2.7 *Story Board*

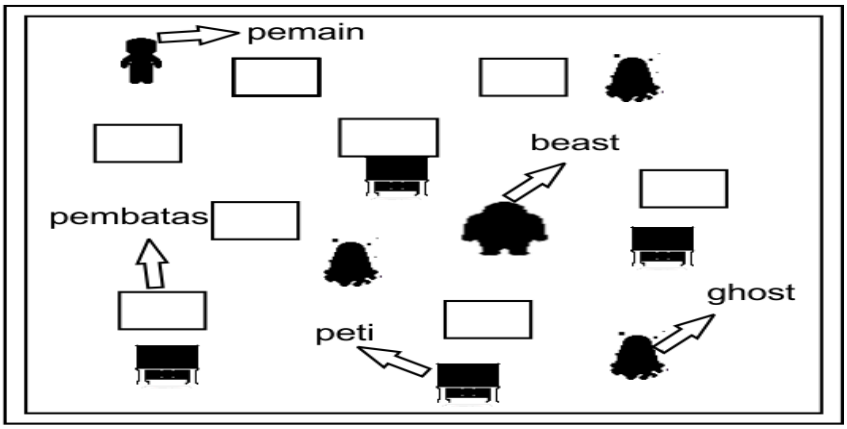
Story board adalah sketsa gambar yang disusun berurutan sesuai dengan naskah, dengan *storyboard* kita dapat menyampaikan ide cerita kita kepada orang lain dengan lebih mudah, karena kita dapat menggiring khayalan seseorang mengikuti gambar-gambar yang tersaji, sehingga menghasilkan persepsi yang sama pada ide cerita kita. Berikut merupakan *story board* dari *game Death Castle*

1. Story Board Level 1



Gambar 3.8 *Story Board level 1*

2. Story Board Level 2



Gambar 3.9 *Story Board level* 2

3.1.2.8 Analisis Metode pada *Game Death Castle*

Untuk membuat permainan *game* menjadi lebih menarik dapat dilakukan dengan meningkatkan kualitas *gameplay* bagi pengguna yang memainkannya. AI merupakan faktor yang penting dalam meningkatkan *gameplay* pada semua jenis permainan *game*, termasuk pada *game Death Castle* yang akan dibangun. Hal penting dalam menentukan keberhasilan suatu aplikasi *game* yang menerapkan AI adalah kesuksesan dalam pencarian dan pelacakan. Pada dasarnya ada dua teknik pencarian dan pelacakan yang digunakan, yaitu pencarian buta (*blind search*) dan pencarian terbimbing (*heuristic search*). Dalam pencarian terbimbing ada beberapa algoritma yang dapat digunakan salah satunya algoritma A*. Algoritma ini dapat menyelesaikan permasalahan secara optimal untuk menemukan jalan terpendek menuju suatu tujuan.

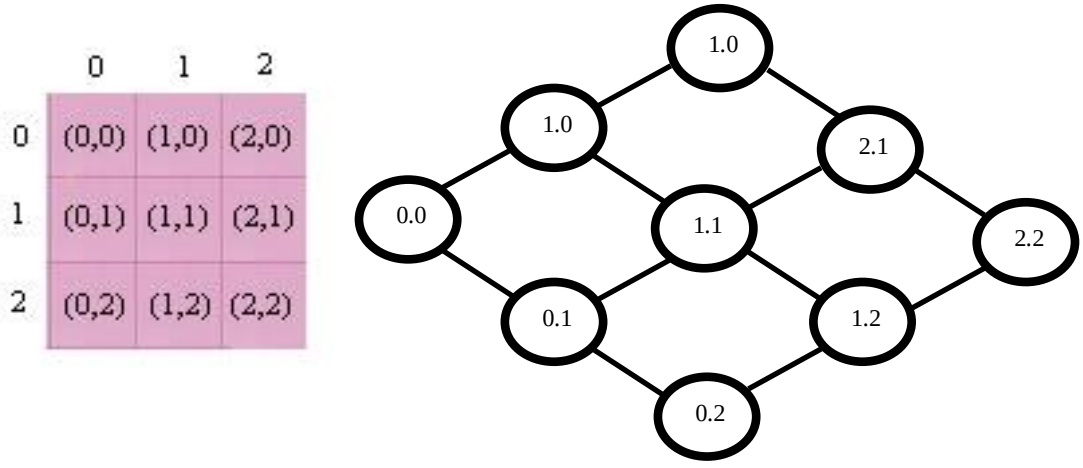
1. Analisis Penggunaan Algoritma A* (A Star) pada *Game Death Castle*

Pada *game Death Castle* arena permainan berupa kumpulan *tile-tile* yang terdiri dari orde X x Y. Tiap *tile* memiliki koordinat X dan Y atau berbentuk seperti array dua dimensi.

	0	1	2
0	(0,0)	(1,0)	(2,0)
1	(0,1)	(1,1)	(2,1)
2	(0,2)	(1,2)	(2,2)

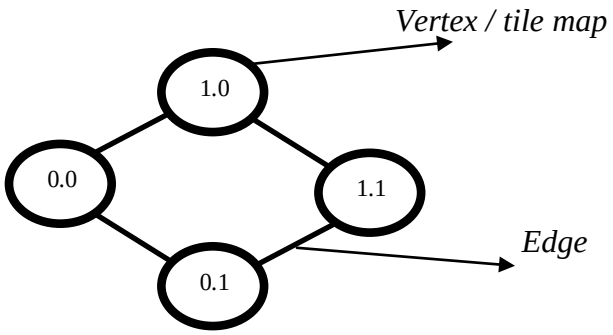
Gambar 3.10 Arena permainan pada *Game Death Castle*.

Arena permainan dalam *game Death Castle* dapat digambarkan melalui struktur data *graph*. Struktur data *graph* adalah struktur data yang berbentuk *network*/ jaringan, hubungan antar elemennya adalah *many - to – many.*



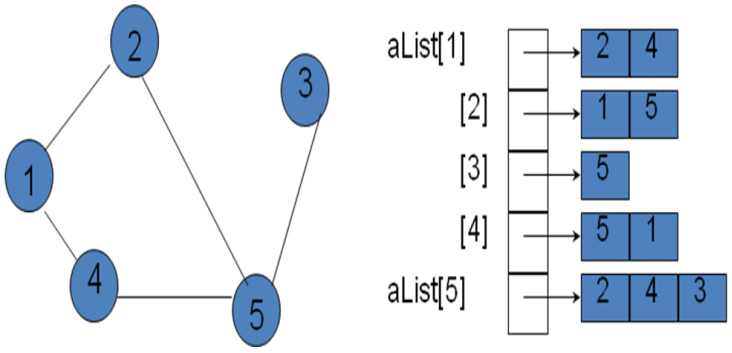
Gambar 3.11 Representasi arena permainan dalam bentuk *graph*

Tiap *tile/vertex* dalam arena permainan, simpulnya akan saling terhubung dengan sebuah *edge.*



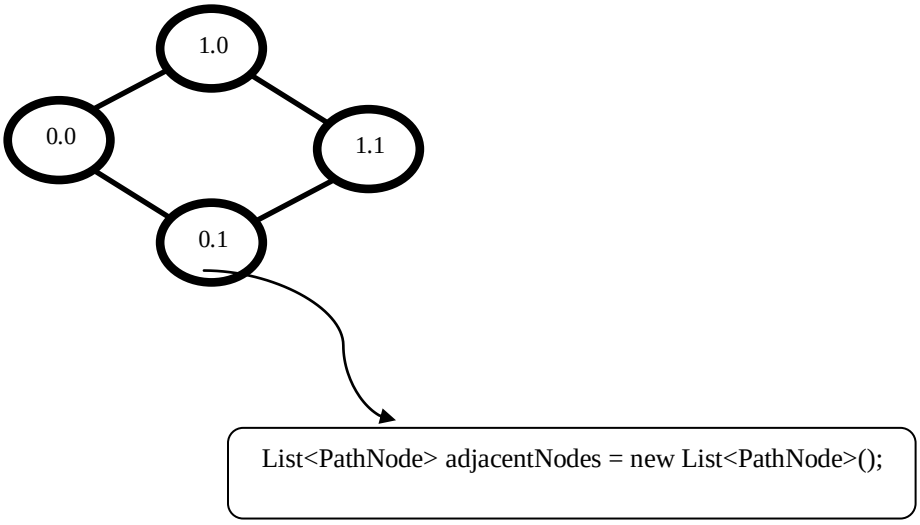
Gambar 3.12 keterhubungan tiap *vertex* pada arena permainan.

Secara umum terdapat dua macam representasi dari struktur data graf yang dapat diimplementasikan. Pertama, disebut *adjacency list*, yaitu diimplementasikan dengan menampilkan masing-masing simpul sebagai sebuah struktur data yang mengandung senarai dari semua simpul yang saling berhubungan. Yang kedua adalah representasi berupa *adjacency matrix* dimana baris dan kolom dari matriks tersebut merepresentasikan simpul awal dan simpul tujuan dan sebuah entri di dalam senarai yang menyatakan apakah terdapat sisi di antara kedua simpul tersebut.



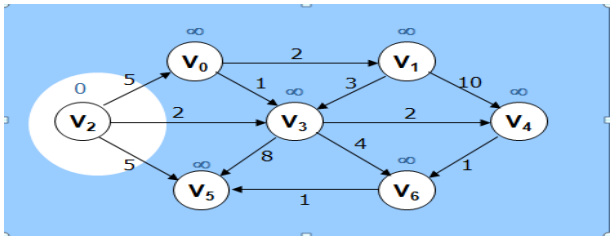
Gambar 3.13 contoh *Adjacency List* pada *Array List*

Namun dalam pembangunan aplikasi *game Death castle* akan digunakan representasi berupa *Adjacency List* yang berupa *ArrayList*. Setiap array akan menyimpan informasi *node-node* yang dapat ditelusuri.



Gambar 3.14 *ArrayList* pada arena permainan.

Dalam *graph* terdapat beberapa algoritma yang sering digunakan dalam melakukan pencarian *vertex* salah satunya adalah algoritma A*. Metode ini mengevaluasi setiap titik yang memiliki nilai/*cost* dengan melakukan fungsi *heuristic*. Nilai *cost* terkecil akan dipilih sebagai rute yang akan dipilih dalam pencarian menuju *node* tujuan. Berikut ini adalah contoh *graph* yang memiliki nilai *cost*.



Gambar 3.15 contoh *graph* yang memiliki nilai *cost*.

Algortima A* akan diterapkan pada objek musuh dalam melakukan pencarian karakter pemain di arena permainan. Namun terdapat beberapa hal yang perlu didefinisikan terlebih dahulu. Adapun istilah-istilah yang akan dibahas yaitu *open list, closed list,* nilai f, g dan n.

1. *OPEN LIST* adalah *list* yang menyimpan kemungkinan *path* yang akan diperiksa. *OPEN LIST* dibuat terurut berdasarkan nilai f. *OPEN LIST* digunakan untuk menentukan secara selektif (berdasarkan nilai f) jalan yang dikira lebih dekat menuju pada path tujuan. *OPEN* berisi simpul-simpul yang masih memiliki peluang untuk terpilih sebagai simpul terbaik (*best node*).
2. CLOSED adalah senarai (*list*) untuk menyimpan simpul-simpul yang sudah pernah dibangkitkan dan sudah pernah terpilih sebagai simpul terbaik (*best node*) atau senarai yang menyimpan jalan yang sudah diperiksa dari *open list*. Artinya, *CLOSED* berisi simpul-simpul yang tidak mungkin terpilih sebagai simpul terbaik (peluang untuk terpilih sudah tertutup). Kedua list (*OPEN LIST* dan *CLOSED LIST*) ini bertujuan juga untuk menghindari penelusuran berkali-kali jalan (rute) yang memang sudah diidentifikasi agar tidak masuk kembali ke dalam *OPEN LIST*.

3. Nilai F adalah *cost* perkiraan suatu *path* yang teridentifikasi. Nilai F merupakan hasil dari f(n).
4. Nilai G hasil dari fungsi g(n), adalah banyaknya langkah yang diperlukan untuk menuju ke *path* sekarang.
5. Setiap simpul (*node*) harus memiliki informasi nilai h(n), yaitu estimasi harga simpul tersebut dihitung dari simpul tujuan yang hasilnya menjadi nilai H[16].

Fungsi f sebagai estimasi fungsi evaluasi terhadap *node* n, dapat dituliskan :

$$f(n) = g(n) + h(n)$$

dengan :

f(n) = fungsi evaluasi (jumlah g(n) dengan h(n))

g(n) = biaya (*cost*) yang dikeluarkan dari keadaan awal sampai keadaan n

h(n) = estimasi biaya untuk sampai pada suatu tujuan mulai dari n [17]

Karena objek musuh dapat bergerak secara diagonal di arena permainan, maka digunakan fungsi *heuristic Non-Manhattan Distance,* dengan fungsi *heuristic* yang digunakan adalah sebagai berikut:

```
h_diagonal(n) = min(abs(n.x-goal.x) + abs(n.y-goal.y))
h_orthogonal(n) = (abs(n.x-goal.x) + abs(n.y-goal.y))
```

2. Ilustrasi Perhitungan Algoritma A*

Arena permainan pada *game Death Castle* terdiri dari orde X x Y . dalam analisis ini akan dijelaskan dengan ordo minimal yaitu 3x3 dikarenakan

perhitungan pada ordo berapa pun akan sama. Pada penentuan bobot setiap node akan diberikan nilai sesuai dengan jarak terdekat ke tujuan, misal *node* yang terjauh dari tujuan maka diberi bobot yang kecil sedangkan *node* yang terdekat dengan tujuan diberi bobot yang lebih besar. Pada kasus pencarian objek musuh di arena permainan akan terdapat dua kondisi. Kondisi pertama, ketika pencarian yang dilakukan oleh musuh tidak terdapat penghalang sedangkan kondisi kedua ketika pencarian yang dilakukan terdapat penghalang. Oleh karena itu, pada analisis ini akan dilakukan dua jenis perhitungan menggunakan Algoritma A* yaitu tanpa penghalang dan terdapat penghalang.

Pada kondisi tanpa penghalang, simpul awal atau posisi musuh dicontohkan berada pada koordinat (0,0) sedangkan simpul tujuan atau posisi karakter pemain berada pada koordinat(2,2) diarena permainan.

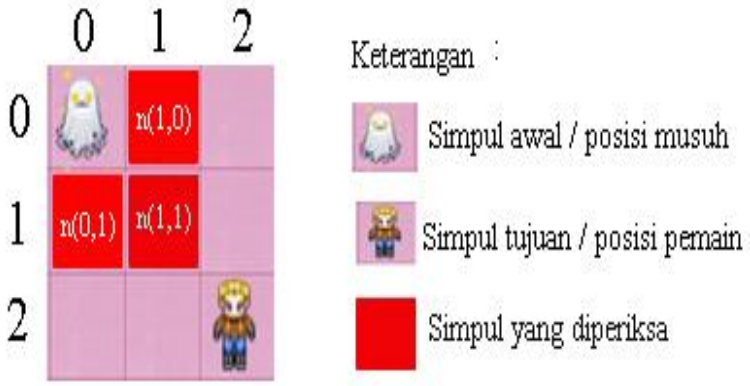


Gambar 3.16 Contoh kondisi (i) tanpa penghalang dengan pencarian A*

Langkah selanjutnya akan dilakukan pemeriksaan terhadap simpul-simpul yang bertetangga yang memiliki nilai *cost* / biaya terkecil sebagai *Best Node* sebagai rute yang akan dipilih musuh, dengan melakukan langkah perhitungan sebagai berikut :

Langkah Pertama

Pada langkah pertama akan diperiksa simpul pada koordinat n(1,1), koordinat n(1,0) dan koordinat n(0,1)



Gambar 3.17 penelusuran pada koordinat n(1,1), koordinat n(0,1) dan (1,0)

a. Koordinat n(1,1)

Nilai n.x = 1

Nilai n.y = 1

Simpul tujuan(2,2) sehingga goal.x = 2, goal.y = 2

g (1,1) = 1

h_orthogonal(n) = (abs(n.x-goal.x) + abs(n.y-goal.y))

h_orthogonal(1,1) = (abs(1 - 2) + abs(1 - 2))

= (abs(-1) + abs(-1))

= 2

h_diagonal(n) = min(abs(n.x-goal.x) + abs(n.y-goal.y))

h_diagonal(1,1) = min(abs(1 - 2)+abs(1 - 2))

= min(abs(-1)+abs(-1))

= min 2

h(n) = h_diagonal(n) + (h_orthogonal (n) – (2 * h_diagonal(n)))

h(1,1) = (-2) + (2-(2*(-2)))

= -2 + 6

= 4

f (1,1) = g (1,1) + h (1,1)

= 1 + 4

= 5

b. Koordinat n(1,0)

Nilai n.x = 1

Nilai n.y = 0

Simpul tujuan(2,2) sehingga goal.x = 2, goal.y = 2

g (1,0) = 1

h_orthogonal(n) = (abs(n.x-goal.x) + abs(n.y-goal.y))

h_orthogonal(1,0) = (abs(1 - 2) + abs(0 - 2))

= (abs(-1) + abs(-2))

= 3

h_diagonal(n) = min(abs(n.x-goal.x), abs(n.y-goal.y))

h_diagonal(1,0) = min(abs(1 - 2)+abs(0 - 2))

= min(abs(-1)+abs(-2))

= min 3

h(n) = h_diagonal(n) + (h_orthogonal (n) – (2 * h_diagonal(n)))

h(1,0) = (- 3) + (3-(2*(-3)))

= -3 + 9

= 6

f (1,0) = g (1,0) + h (1,0)

= 1 + 6

= 7

c. Koordinat n(0,1)

Nilai n.x = 0

Nilai n.y = 1

Simpul tujuan(2,2) sehingga goal.x = 2, goal.y = 2

g(n) = g (0,1) = 1

h_orthogonal(n) = (abs(n.x-goal.x) + abs(n.y-goal.y))

h_orthogonal(0,1) = (abs(0 - 2) + abs(1 - 2))

= (abs(-2) + abs(-1))

= 3

h_diagonal(n) = min(abs(n.x-goal.x) + abs(n.y-goal.y))

h_diagonal(0,1) = min(abs(0 - 2)+abs(1 - 2))

= min(abs(-2)+abs(-1))

= min 3

h(n) = h_diagonal(n) + (h_orthogonal (n) – (2 * h_diagonal(n)))

h(0,1) = (-3) + (3-(2*(-3)))

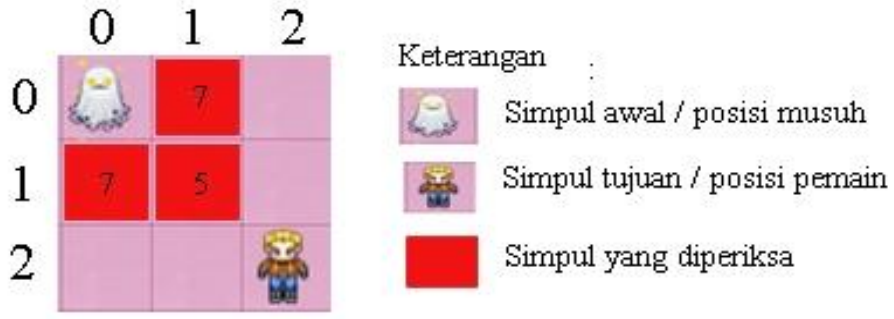
= -3 + 9

= 6

f (0,1) = g (0,1) + h (0,1)

= 1 + 6

= 7



Gambar 3.18 Langkah pertama pencarian *BestNode* pada kondisi (i)

Dari hasil perhitungan diatas seperti terlihat pada gambar 3.17 terdapat tiga simpul yang mungkin menjadi *BestNode* yaitu (1,0) dengan f(n)=7, (1,1) dengan f(n)=5 dan (0,1) dengan f(n)=7. Dari ke tiga simpul yang mungkin maka dipilihlah simpul (1,1) sebagai *Best Node* karena memiliki biaya terkecil yaitu 5.



Gambar 3.19 Simpul yang terpilih sebagai Best Node dilangkah pertama

Langkah kedua

Setelah mendapatkan *BestNode* dilangkah pertama, dilakukan penelusuran lanjutan untuk mendapatkan *BestNode* selanjutnya dengan melakukan penelusuran simpul – simpul yang memiliki nilai *cost* terkecil. Seperti pada langkah pertama dilakukan lagi penelusuran pada koordinat n(2,2), koordinat n(2,1) dan koordinat n(1,2).



Gambar 3.20 penelusuran langkah kedua pada koordinat n(2,2), koordinat n(2,1) dan (1,2)

a. Koordinat (2,2)

Nilai n.x = 2

Nilai n.y = 2

Simpul tujuan(2,2) sehingga goal.x = 2, goal.y = 2

g(n) = g (2,2) = 2

h_orthogonal(n) = (abs(n.x-goal.x) + abs(n.y-goal.y))

h_orthogonal(2,2) = (abs(2 - 2) + abs(2 - 2))

= (abs(0) + abs(0))

= 0

h_diagonal(n) = min(abs(n.x-goal.x) + abs(n.y-goal.y))

h_diagonal(2,2) = min(abs(2 - 2)+abs(2 - 2))

= min(abs(0)+abs(0))

= min 0

h(n) = h_diagonal(n) + (h_orthogonal (n) – (2 * h_diagonal(n)))

h(2,2) = (-0) + (0-(2*(-0)))

= 0 + 0

= 0

f (2,2) = g (2,2) + h (2,2)

= 2 + 0

= 2

b. Koordinat (2,1)

Nilai n.x = 2

Nilai n.y = 1

Simpul tujuan(2,2) sehingga goal.x = 2, goal.y = 2

h_orthogonal(n) = (abs(n.x-goal.x) + abs(n.y-goal.y))

h_orthogonal(2,2) = (abs(2 - 2) + abs(1 - 2))

= (abs(0) + abs(-1))

= 1

h_diagonal(n) = min(abs(n.x-goal.x) + abs(n.y-goal.y))

h_diagonal(2,2) = min(abs(2 - 2)+abs(1 - 2))

= min(abs(0)+abs(-1))

= min 1

h(n) = h_diagonal(n) + (h_orthogonal (n) – (2 * h_diagonal(n)))

h(2,2) = (-1) + (1-(2*(-1)))

= -1 + 3

= 2

f (2,2) = g (2,2) + h (2,2)

= 2 + 2

= 4

c. Koordinat (1,2)

Nilai n.x = 1

Nilai n.y = 2

Simpul tujuan(2,2) sehingga goal.x = 2, goal.y = 2

h_orthogonal(n) = (abs(n.x-goal.x) + abs(n.y-goal.y))

h_orthogonal(2,2) = (abs(1 - 2) + abs(2 - 2))

= (abs(-1) + abs(0))

= 1

h_diagonal(n) = min(abs(n.x-goal.x) + abs(n.y-goal.y))

h_diagonal(2,2) = min(abs(1 - 2)+abs(2 - 2))

= min(abs(-1)+abs(0))

= min 1

h(n) = h_diagonal(n) + (h_orthogonal (n) – (2 * h_diagonal(n)))

h(2,2) = (-1) + (1-(2*(-1)))

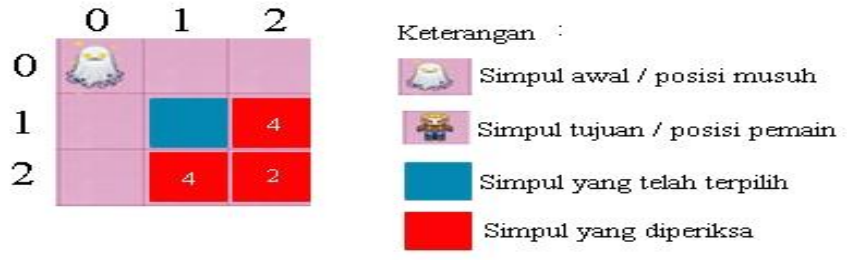
= -1 + 3

= 2

f (2,2) = g (2,2) + h (2,2)

= 2 + 2

= 4



Gambar 3.21 Langkah kedua pencarian *BestNode* pada kondisi (i)

Pada gambar 3.16 terdapat tiga simpul yang mungkin menjadi *BestNode* yaitu koordinat n(2,2) dengan f(n)=2, koordinat n(2,1) dengan f(n)=4 dan koordinat n(1,2) dengan f(n)=4. Koordinat n(2,2) dipilih sebagai *Best node* karena memiliki biaya *cost* terkecil dan koordinat n(2,2) dikenali sebagai simpul tujuan yang berarti solusi untuk menemukan rute terpendek menuju posisi pemain.



Gambar 3.22 Hasil pencarian jalan dengan Algoritma A*

Dari semua perhitungan yang telah dilakukan di pilihlah biaya/*cost* terkecil pada setiap langkahnya sehingga akan menghasilkan jalur terpendek yang akan dilalui musuh untuk melakukan pencarian posisi pemain seperti terlihat pada gambar 3.15.

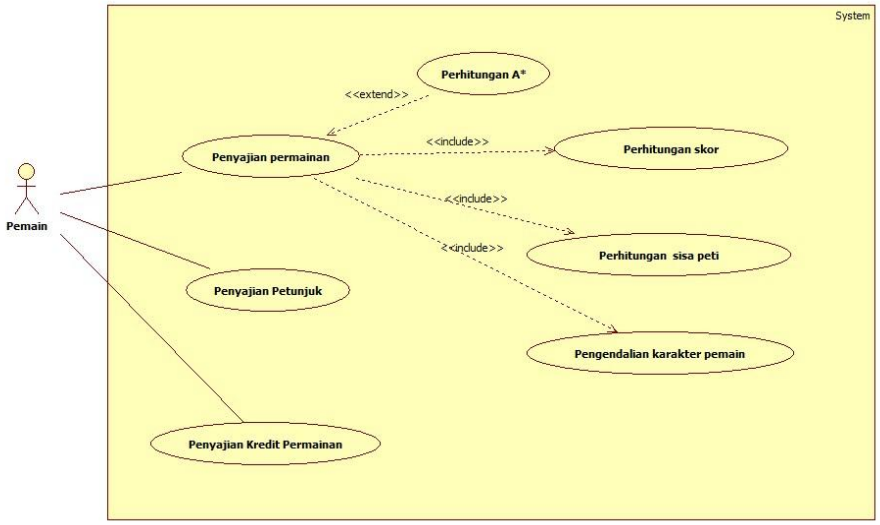
3.1.3 Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional dilakukan untuk memberikan gambaran aliran data yang ada pada program aplikasi yang akan dibangun.

3.1.3.1 Diagram *Use Case*

Use case adalah abstraksi dari interaksi antara sistem dan dan aktor. *Use case* mendeskripsikan bagaimana sistem akan terlihat di mata *user*.

Secara umum, interaksi antara pengguna dengan aplikasi permainan *Death Castle* digambarkan pada diagram *use case* dibawah ini :



Gambar 3.23 Diagram *Use Case Game Death Castle*.

3.1.3.2 Definisi *Use Case*

Definisi *use case* berfungsi intuk menjelaskan fungsi *use case* yang terdapat pada *use case* diagram. Definisi *use case* diterangkan pada table berikut :

Table 3.3 Definisi *Use case*

No	*Use case*	Deskripsi
1	Penyajian Permainan	Proses untuk memulai permainan
2	Penyajian Petunjuk	Proses untuk menampilkan informasi petunjuk permainan
3	Penyajian Kredit	Proses untuk menampilkan informasi tentang *game Death Castle*
4	Perhitungan skor	Proses pada permainan yang akan menghitung skor dari pemain
5	Perhitungan sisa peti	Proses pada permainan yang akan menginformasikan kepada pemain sisa peti yang harus di tutup.
6	Perhitungan A*	Proses pada permainan yang akan menginformasikan perhitungan Algoritma A* pada objek musuh.

7	Pengendalikan karakter pemain	Proses yang dilakukan oleh pemain untuk menggerakkan karakter pemain diarena permainan.

3.1.3.3 Skenario *Use Case*

Skenario pada diagram *use case* berisi mengenai alur kejadian '*flow of event' use case* utama. Didalamnya menjelaskan urutan interaksi antara aktor dengan *use case* tersebut dari awal sampai akhir. Adapun skenario *use case* yang terdapat pada aplikasi *game Death Castle* adalah sebagai berikut :

Penyajian Permainan

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain, dengan *use case* mulai permainan yang dijelaskan pada tabel berikut ini :

Tabel 3.4. Skenario *Use Case* Penyajian Permainan

Identifikasi	
Nomor *use case*	1
Nama	Mulai Permainan
Deskripsi	Proses yang dilakukan pemain untuk memulai permainan
Trigger	☐ Internal ☒ Eksternal
Aktor	Pemain
Skenario Normal	
Aksi Aktor	**Reaksi Sistem**
a. Klik Mulai Permainan	
	b. Tampilkan layar permainan
	c. *Inisialisasi* semua konten yang dibutuhkan permainan

	d. *Inisialisasi* informasi awal permainan
	e. *Inisialisasi* posisi Peti
	f. Menempatkan posisi karakter pemain di sebelah kiri atas arena permainan
g. Mengendalikan karakter Pemain	
	h. Musuh keluar dari peti
	i. Jalankan *AI Game Engine*
	j. Musuh mengejar pemain
k. Menutup peti	
	l. Menampilkan informasi sisa peti yang harus di tutup
	o. Apabila sisa peti sama dengan 0, maka akan menampilkan layar di level berikutnya.

2. Perhitungan Sisa Peti

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain, dengan *use case* hitung skor yang dijelaskan pada tabel berikut ini :

Tabel 3.5. Skenario *Use Case* Perhitungan sisa peti

Identifikasi	
Nomor *use case*	3
Nama *use case*	Perhitungan Sisa Peti
Deskripsi	Proses yang akan menginformasikan sisa peti yang harus ditutup oleh pemain
Trigger	☐ Internal ☒ Eksternal
Aktor	Pemain
Skenario Utama	

Aksi Aktor	Reaksi Sistem
a. Pemain menutup peti	
	b. Menghitung sisa peti yang berada diarena permainan dan menampilkan informasi sisa peti

3. Perhitungan Skor

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain, dengan *use case* hitung skor yang dijelaskan pada tabel berikut ini :

Tabel 3.6. Skenario *Use Case* Hitung Skor

Identifikasi	
Nomor *use case*	2
Nama *use case*	Hitung Skor
Deskripsi	Proses yang akan menginformasikan nilai skor pemain
Trigger	☐ Internal ☒ Eksternal
Aktor	Pemain
Skenario Utama	
Aksi Aktor	**Reaksi Sistem**
a. Pemain menembak musuh	
	b. Objek musuh terkena tembakan dan akan hilang dari arena permainan
	c. Skor pemain akan bertambah 10
Skenario Alternatif	
a. Pemain menembak musuh	
	b. Objek musuh tidak terkena tembakan dan akan tetap mengejar pemain
	c. Skor pemain tidak akan bertambah.

4. Perhitungan A*

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain, dengan *use case* perhitungan A* yang dijelaskan pada tabel berikut ini :

Tabel 3.7. Skenario *Use Case* Perhitungan A*

Identifikasi	
Nomor *use case*	4
Nama *use case*	Perhitungan
Deskripsi	Proses yang akan menginformasikan perhitungan algoritma A*
Trigger	☐ Internal ☒ Eksternal
Aktor	Pemain
Skenario Utama	
Aksi Aktor	**Reaksi Sistem**
a. Pemain menekan tombol lihat perhitungan A*	
	b. Mengambil nilai posisi koordinat musuh sebagai *node* awal
	c. Mengambil nilai posisi koordinat pemain sebagai node tujuan
	d. Melakukan perhitungan heuristic dengan rumus f = g + h untuk menentukan rute terdekat ke *node* tujuan.
	e. Menampilkan nilai koordinat node awal dan node tujuan serta menampilkan nilai f pada layar perhitungan algoritma A*

5. Penyajian Petunjuk

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain, dengan seluruh *use case* Petunjuk yang dijelaskan pada tabel berikut ini :

Tabel 3.8. Skenario *Use Case* Lihat Petunjuk

Identifikasi	
Nomor *use case*	4
Nama *use case*	Lihat Petunjuk
Deskripsi	Proses yang dilakukan pemain untuk melihat petunjuk tentang permainan *game*
Trigger	☐ Internal ☒ Eksternal
Aktor	Pemain
Skenario Utama	
Aksi Aktor	**Reaksi Sistem**
a. Klik Petunjuk	
	b. Masuk ke submenu Petunjuk
	c. Menampilkan Layar tentang info cara memainkan *game*

6. Penyajian Kredit

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain, dengan seluruh *use case* Kredit yang dijelaskan pada tabel berikut ini :

Tabel 3.9. Skenario *Use Case* Lihat Kredit

Identifikasi	
Nomor *use case*	5
Nama *use case*	Lihat Kredit
Deskripsi	Proses yang dilakukan pemain untuk melihat informasi kredit *game*
Trigger	☐ Internal ☒ Eksternal

Aktor	Pemain
Skenario Utama	
Aksi Aktor	**Reaksi Sistem**
a. Klik Kredit	
	b. Masuk ke submenu Kredit
	c. Menampilkan Layar tentang info cara kredit *game*

7. Mengendalikan Karakter Pemain

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain, dengan seluruh *use case* mengendalikan karakter pemain yang dijelaskan pada tabel berikut ini :

Tabel 3.10. Skenario *Use Case* Mengendalikan Karakter pemain

Identifikasi	
Nomor *use case*	7
Nama *use case*	Mengendalikan karakter pemain
Deskripsi	Proses yang dilakukan pemain untuk mengendalikan karakter pemain diarena permainan
Trigger	☐ Internal ☒ Eksternal
Aktor	Pemain
Skenario Utama	
Aksi Aktor	**Reaksi Sistem**
a. Menekan tombol A di *keyboard*	
	b. mengerakkan karakter pemain kearah kiri
b. Menekan tombol W di *keyboard*	
	c. mengerakkan karakter pemain kearah atas

b. Menekan tombol S di *keyboard*	
	c. mengerakkan karakter pemain kearah bawah
b. Menekan tombol D di *keyboard*	
	c. mengerakkan karakter pemain kearah kanan
Skenario Alternatif	
a. pemain menekan tombol selain tombol AWSD di *keyboard*	
	b.karakter pemain tidak akan bergerak

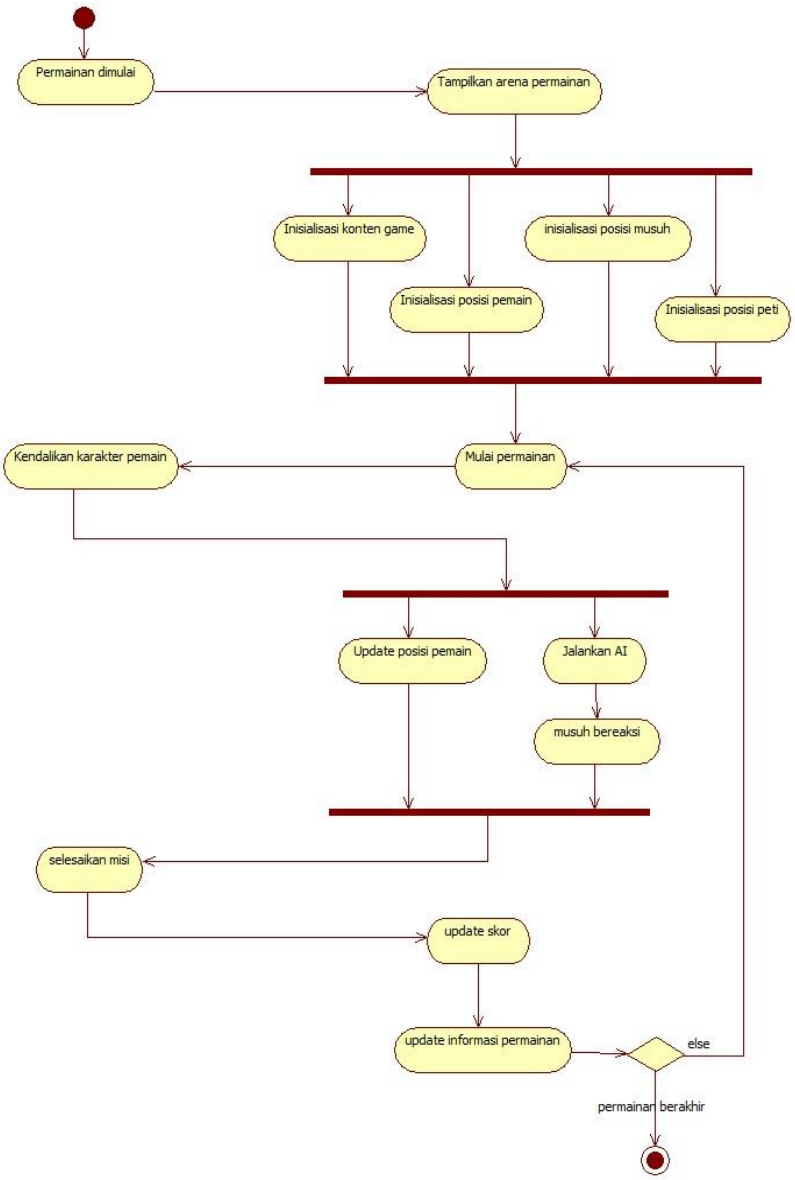
3.1.3.4 *Activity diagram*

Activity Diagram berisi penjelasan lebih lengkap mengenai urutan mengenai urutan suatu proses yang terjadi antara pelaku dan sistem. Aktivitas yang pertama selalu dimulai dari pelaku, kemudian akan mendapat respon dari sistem yang nantinya akan menampilkan akhir dari proses tersebut.

1. Penyajian Permainan

Proses ini terjadi saat pemain memilih menu mulai permainan pada halaman utama permainan *game Death Castle*. Sistem akan menampilkan arena permainan, kemudian

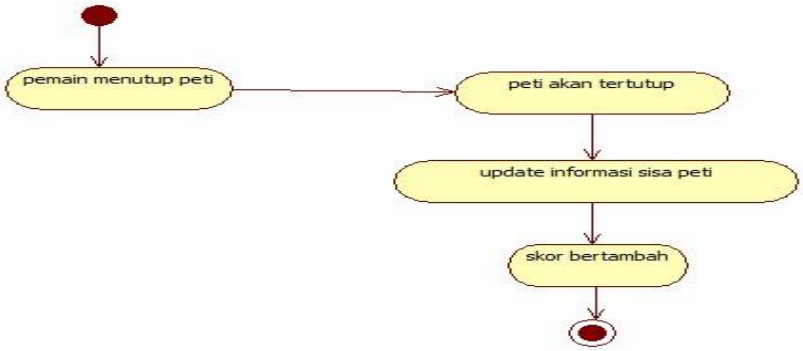
sistem akan menginisialisasi entitas yang ada dalam permainan, menginisialisasi posisi (pemain, musuh, peti dan penghalang).



Gambar 3.24 Diagram *Activity* Penyajian Permainan.

2. Perhitungan Skor

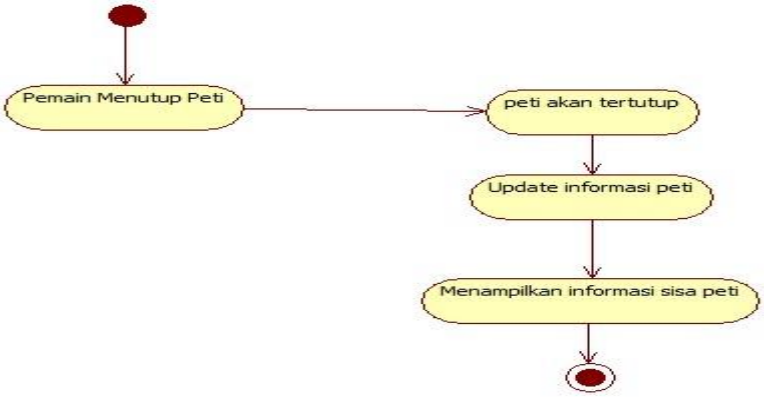
Proses ini akan memberikan informasi kepada pemain tentang skor dari permainan. Skor permainan akan bertambah sepuluh jika pemain berhasil menembak musuh.



Gambar 3.25 Diagram *Activity* hitung skor.

3. Perhitungan Sisa Peti

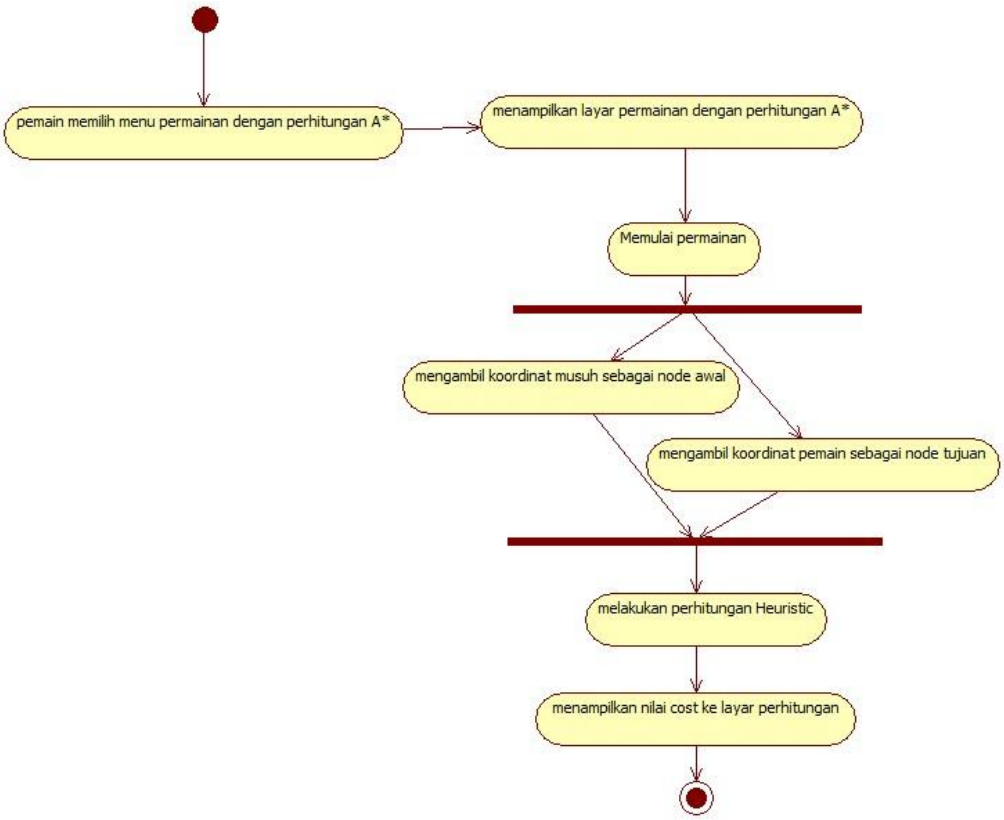
Proses ini akan memberikan informasi kepada pemain tentang skor dari permainan. Skor permainan akan bertambah sepuluh jika pemain berhasil menembak musuh.



Gambar 3.26 Diagram *Activity* hitung sisa peti.

4. Perhitungan A*

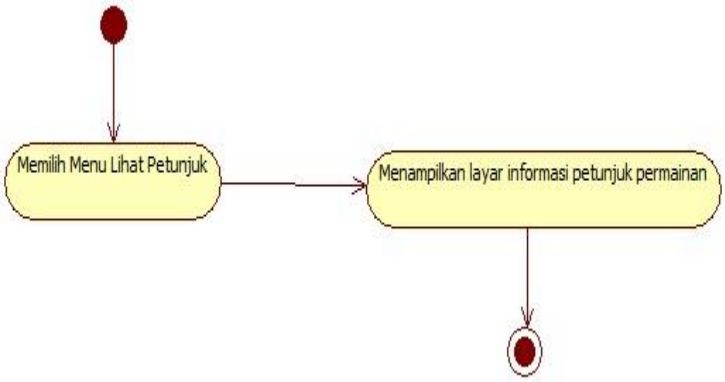
Proses ini akan memberikan informasi kepada pemain hasil perhitungan algoritma A* pada objek musuh dalam melakukan penelusuran posisi pemain diarena permainan.



Gambar 3.27 Diagram *Activity* Perhitungan A*

5. Penyajian Petunjuk

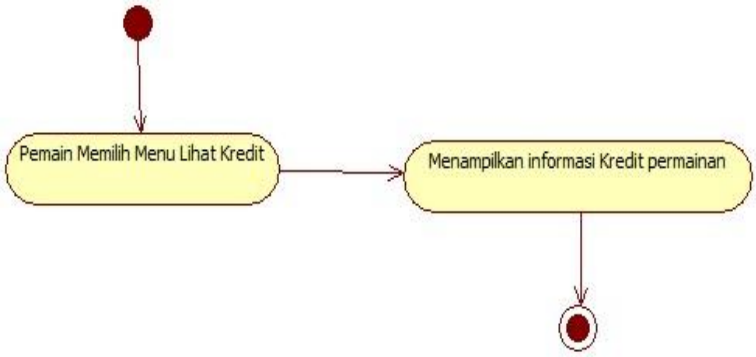
Proses ini terjadi saat pemain memilih menu lihat petunjuk pada halaman utama. Sistem akan menampilkan layar lihat petunjuk untuk memberikan informasi kepada pemain tentang petunjuk memainkan *game*.



Gambar 3.28 Diagram *Activity* Lihat Petunjuk.

6. Penyajian Kredit

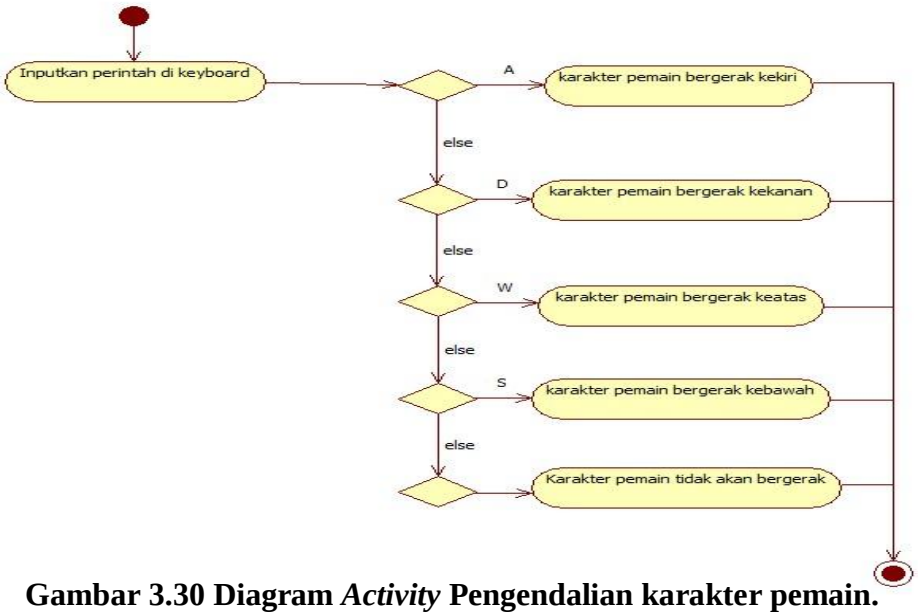
Proses ini terjadi saat pemain memilih menu lihat kredit pada halaman utama. Sistem akan menampilkan layar lihat kredit untuk memberikan informasi kepada pemain tentang pembuat *game*.



Gambar 3.29 Diagram *Activity* Lihat kredit.

7. Mengendalikan karakter pemain

Proses ini terjadi saat pemain menekan sejumlah tombol di *keyboard* untuk mengendalikan karakter pemain yang berada di arena permainan.

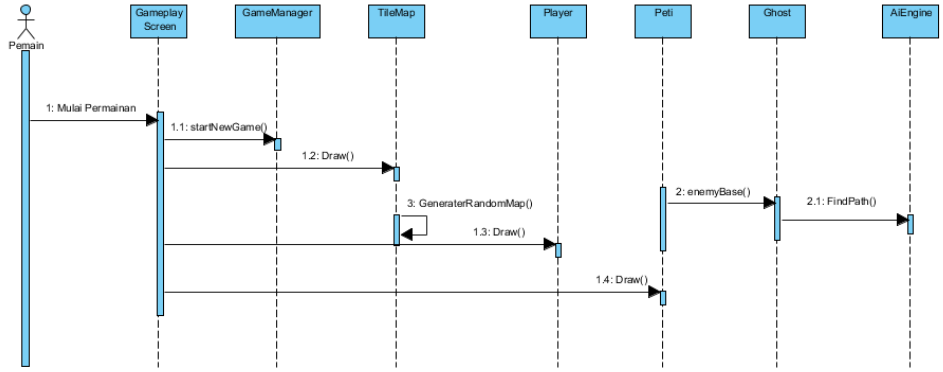


Gambar 3.30 Diagram *Activity* Pengendalian karakter pemain.

3.1.3.5 *Sequence Diagram*

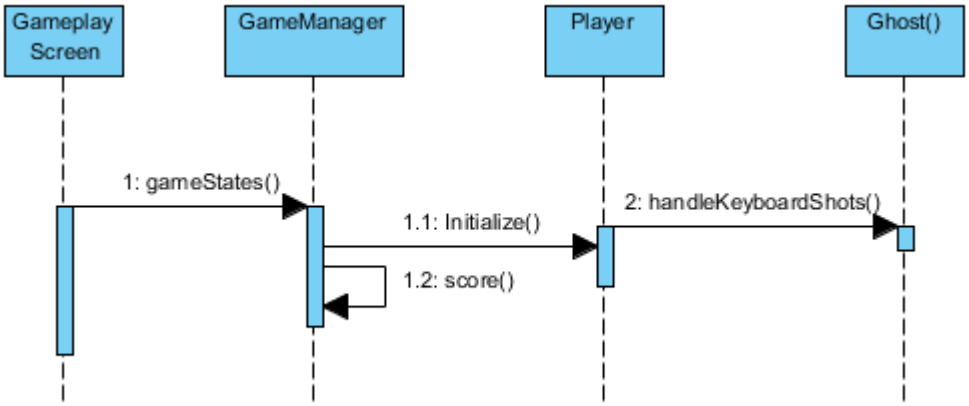
Sequence diagram mengambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup dan *message* yang dikirimkan dan diterima antarobjek.

1. **Penyajian Permainan**



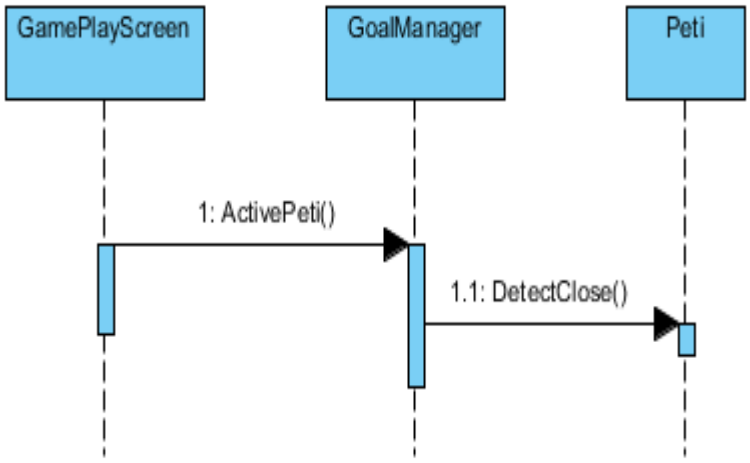
Gambar 3.31 *Sequence Diagram* Penyajian Permainan

2. **Perhitungan Skor**



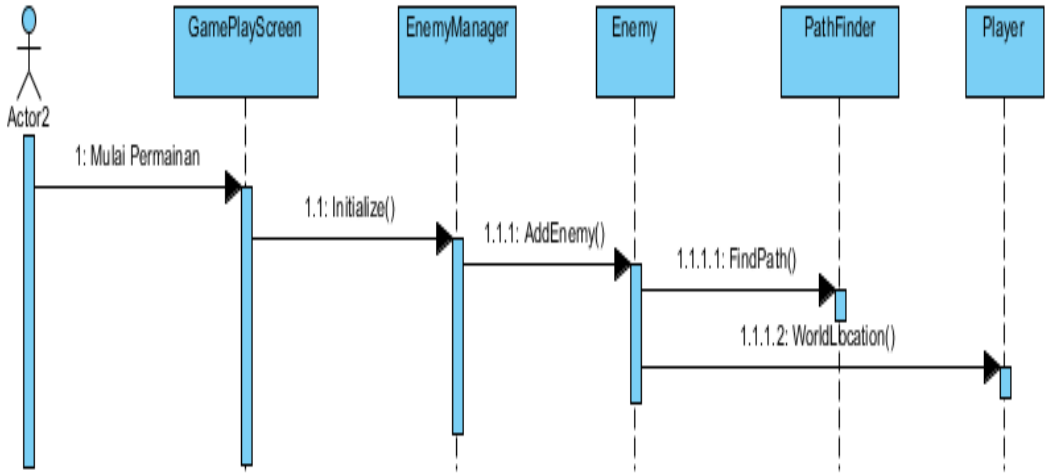
Gambar 3.32 *Sequence Diagram* Hitung skor

3. Perhitungan Sisa Peti



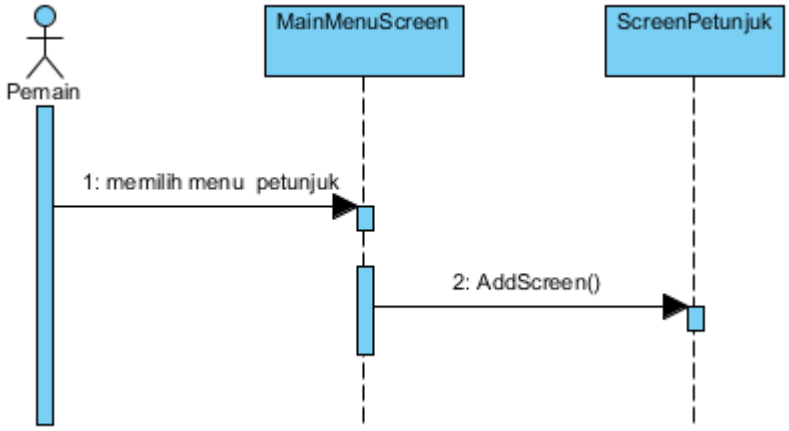
Gambar 3.33 *Sequence Diagram* Hitung sisa peti

4. Perhitungan Algoritma A*



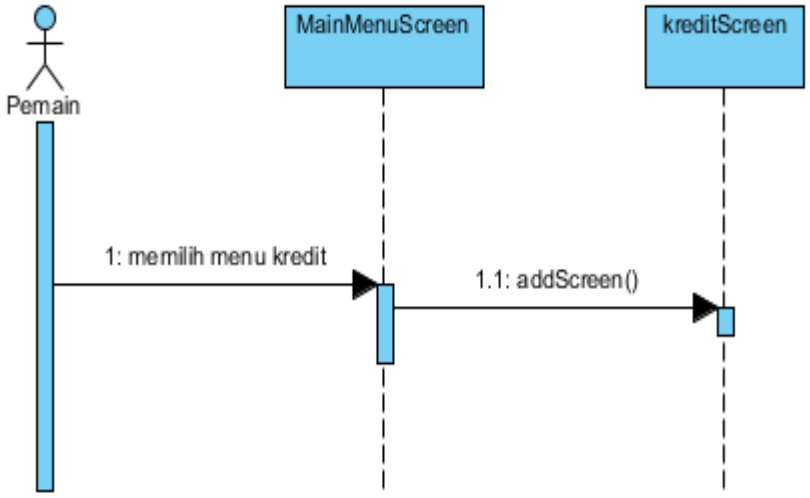
Gambar 3.34 *Sequence Diagram* Perhitungan A*

5. Penyajian Petunjuk



Gambar 3.35 *Sequence Diagram* Penyajian Petunjuk

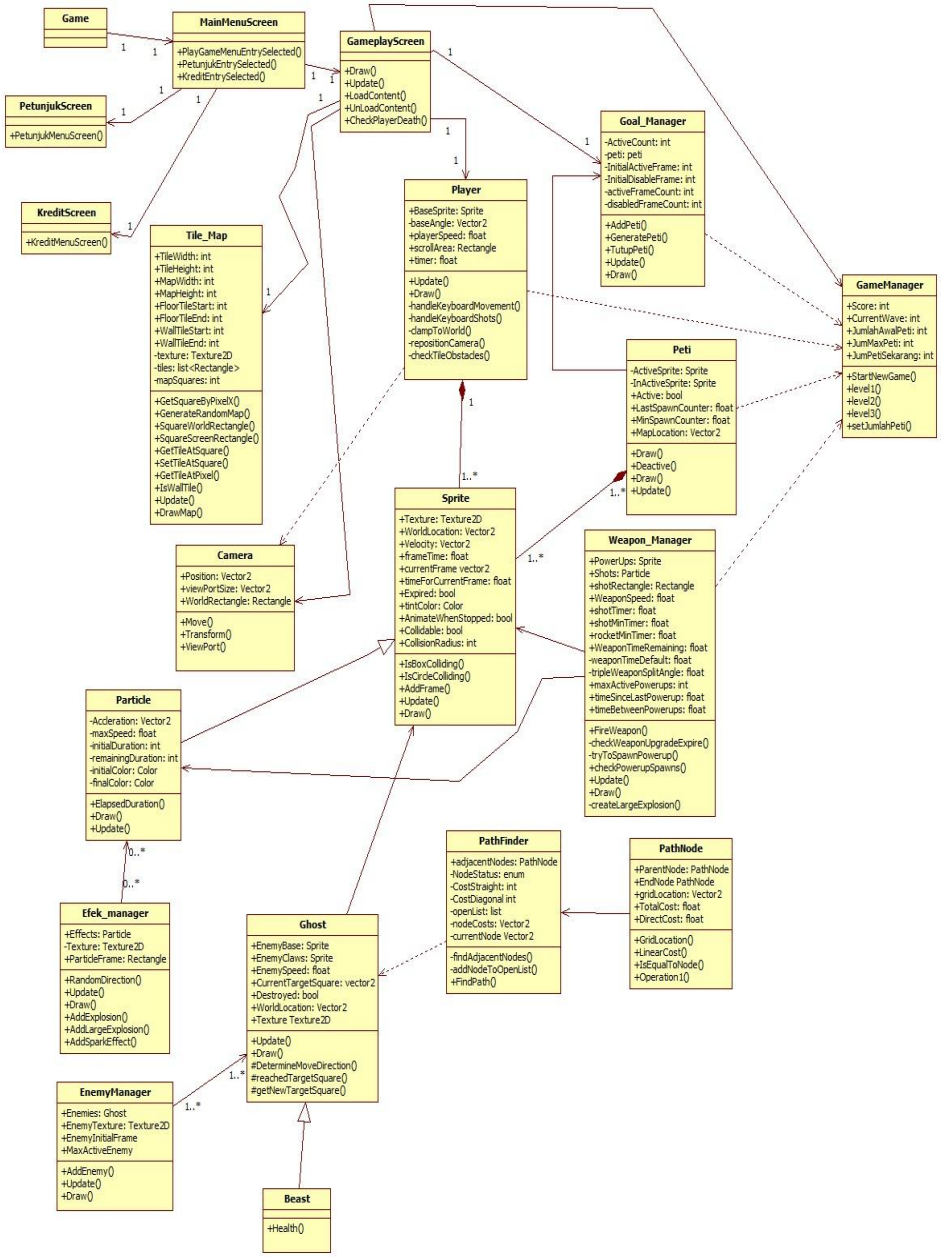
6. Penyajian Kredit



Gambar 3.36 *Sequence Diagram* Penyajian Kredit

3.1.3.6 Class diagram

Adapun *Class Diagram* pada aplikasi *game Death Castle* adalah sebagai berikut :



Gambar 3.37 *Class Diagram*

3.1.4 Analisis dan Kebutuhan Non-Fungsional

3.1.4.1 Analisis Kebutuhan Perangkat Lunak (Pengembang)

Perangkat lunak digunakan dalam sebuah sistem merupakan perintah-perintah yang diberikan kepada perangkat keras agar bisa saling berinteraksi diantara keduanya. Perangkat lunak yang dibutuhkan untuk membangun aplikasi *game Death Castle* ini adalah sebagai berikut :

1. Sistem operasi Windows 7 Profesional
2. NET Framework 3.5
3. Menggunakan IDE Microsoft visual studio 2010 Express Edition
4. Framework XNA

3.1.4.2 Analisis Kebutuhan Perangkat Keras (Pengembang)

Komputer terdiri dari perangkat keras dan perangkat lunak yang saling berinteraksi. Perangkat lunak memberikan instruksi-instruksi kepada perangkat keras untuk melakukan tugas tertentu. Adapun perangkat keras yang digunakan dalam membangun aplikasi *game Death Castle* adalah sebagai berikut :

1. *Processor* amd phenom x3
2. Ram 4 GB 800 Mhz
3. *Hardisk* 160 GB
4. *Display LCD Monitor* dengan resolusi 1360 x 768
5. *Keyboard*
6. *Mouse*

3.1.4.3 Analisis Pengguna

3.1.4.4 Analisis Perangkat Lunak (Pengguna)

Berdasarkan hasil penelitian, spesifikasi minimal bagi pengguna yang dapat menggunakan aplikasi ini adalah sebagai berikut

1. Peran : Pemain
2. Usia : Lebih dari 5 tahun
3. Tingkat Keterampilan : Dapat menggunakan komputer
4. Pengalaman : Pernah memainkan aplikasi *game*

3.1.4.5 Analisis Perangkat Lunak (Pengguna)

Pada sisi pengguna, perangkat lunak yang dibutuhkan untuk menjalankan aplikasi ini adalah sebagai berikut :

1. Sistem operasi windows 7
2. NET Framework 3.5

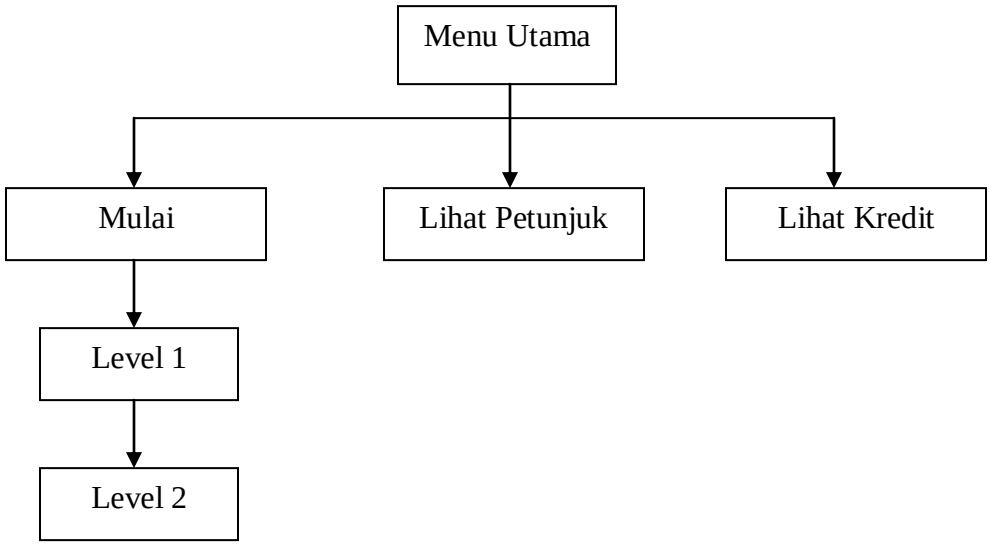
3.2 Perancangan sistem

3.2.1 Perancangan Perangkat Lunak

Perancangan perangkat lunak adalah kegiatan untuk merancang aplikasi yang akan kita bangun, tahapannya dimulai dari perancangan arsitektur menu, lalu berlanjut ke perancangan antar muka dan jaringan semantik.

3.2.2 Perancangan Arsitektur Menu

Untuk mempermudah pembuatan sistem diperlukan perancangan struktur menu program yang akan dibangun. Perancangan struktur menu program ini membantu dalam merancang bagian-bagian dari sistem yang sebenarnya dan untuk mengetahui bagian mana yang terlebih dahulu nantinya yang akan diakses setelah program tersebut selesai. Gambar 3.38 di bawah ini menunjukkan perancangan arsitektur menu dari sistem yang akan dibangun



Gambar 3.38 Arsitektur Menu *Death Castle*

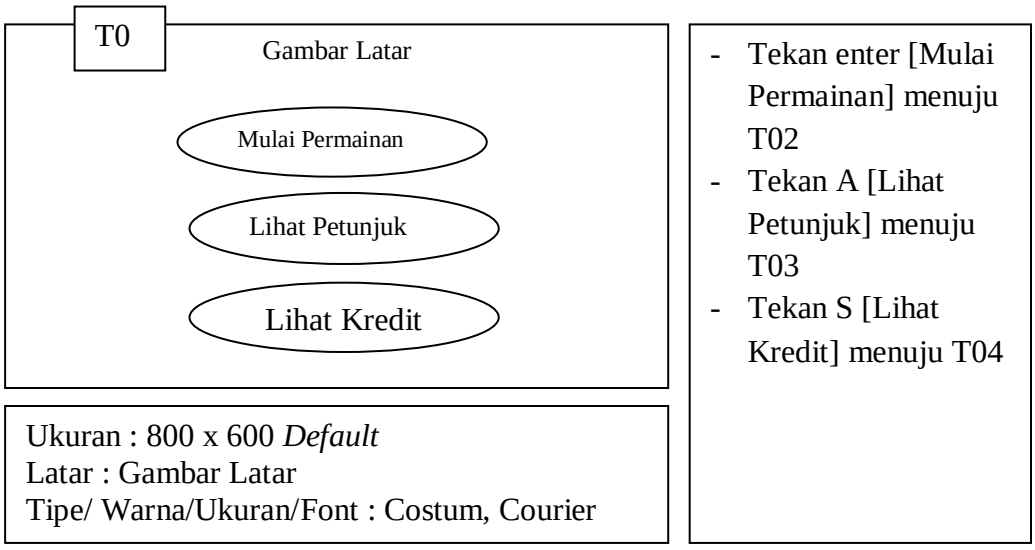
1. Mulai Permainan : merupakan submenu untuk memulai permainan baru
2. Lihat Petunjuk : merupakan submenu yang menampilkan informasi seputar petunjuk permainan seperti penekanan tombol tertentu pada *keyboard* beserta reaksinya

3. Lihat Kredit : merupakan submenu yang menampilkan informasi mengenai pengembang aplikasi beserta informasi lain seputar *game Death Castle*
4. Level 1 : merupakan submenu untuk menampilkan layar permainan pada level 1
5. Level2 : merupakan submenu untuk menampilkan layar permainan pada level 2

3.2.3 Perancangan Antar Muka

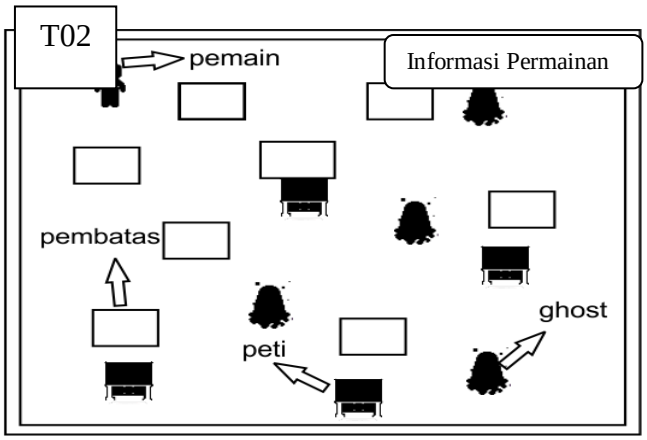
User interface atau antar muka pengguna adalah merupakan aspek dari sistem komputer atau program yang dapat dilihat, didengar atau di persepsikan oleh pengguna

1. Antar Muka Menu Utama



Gambar 3.39 Antar Muka Menu Utama

2. Antar Muka Permainan

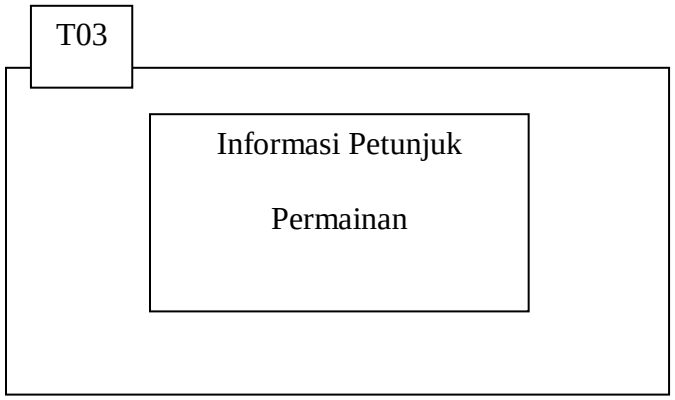


- Informasi permainan merupakan info skor permainan dan jumlah sisa peti yang harus di tutup pemain
- Tekan ESC menuju jeda (*paused*)

Ukuran : 800 x 600 *Default*
Latar : Gambar Latar
Tipe/ Warna/Ukuran/Font : Costum, Courier

Gambar 3.40 Antar Muka Mulai Permainan

3. Antar Muka Lihat Petunjuk

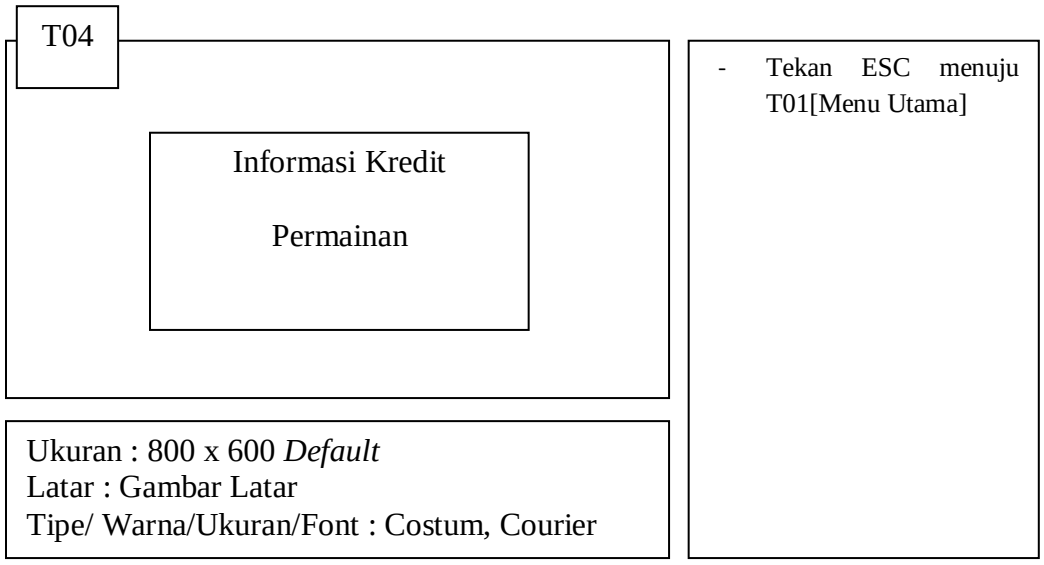


- Tekan ESC menuju T01[Menu Utama]

Ukuran : 800 x 600 *Default*
Latar : Gambar Latar
Tipe/ Warna/Ukuran/Font : Costum, Courier

Gambar 3.41 Antar Muka Menu Petunjuk

4. Antar Muka Lihat Kredit

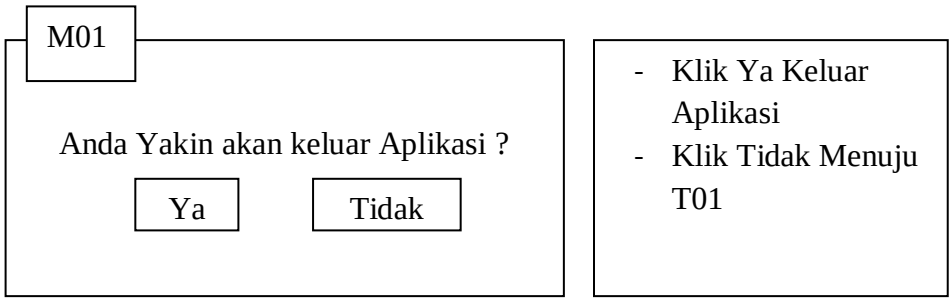


Gambar 3.42 Antar Muka Lihat Kredit

3.2.4 Perancangan Pesan

Pesan yang dirancang dalam konteks aplikasi yang dikembangkan merupakan pesan peringatan yang keluar jika pengguna melakukan aktivitas tertentu yang akan memicu *event* tertentu.

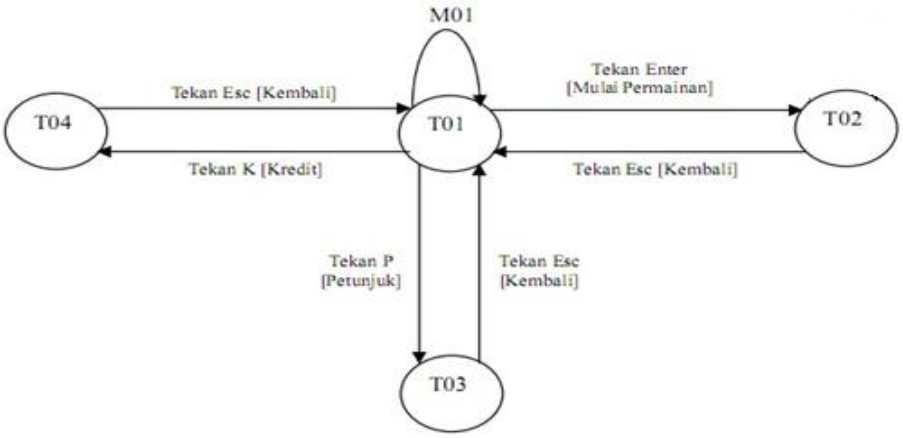
1. Pesan Peringatan Keluar Aplikasi



Gambar 3.43 Pesan Keluar permainan.

3.2.5 Jaringan Semantik

Tahap selanjutnya adalah menggambarkan hubungan antar menu pada aplikasi *game Death Castle*. Penggambaran tersebut diwujudkan melalui jaringan semantik.



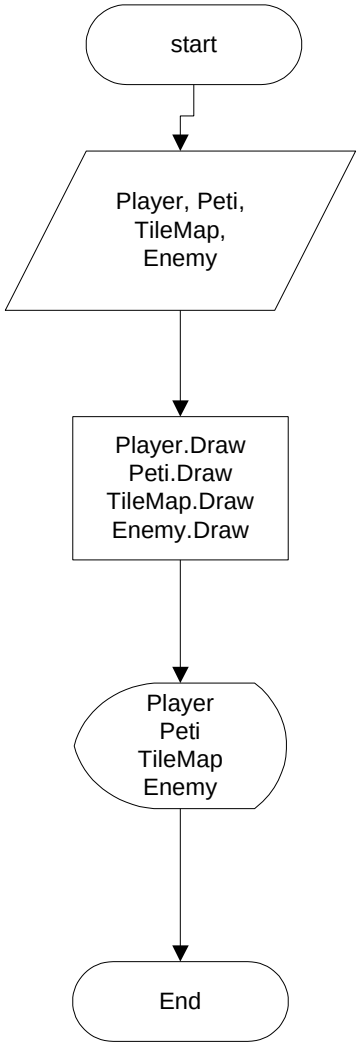
Gambar 3.44 Jaringan Semantik *Death Castle*

3.2.6 Perancangan Method

Perancangan komponen method merupakan perancangan yang dibuat setelah perancangan arsitektur menu, antar muka, dan jaringan semantik. Perancangan ini berfungsi untuk mendeskripsikan method-method yang berada di dalam aplikasi. Adapun method-method yang terdapat dalam *game Death Castle* yang akan dibangun adalah sebagai berikut :

1. Method StartNewGame

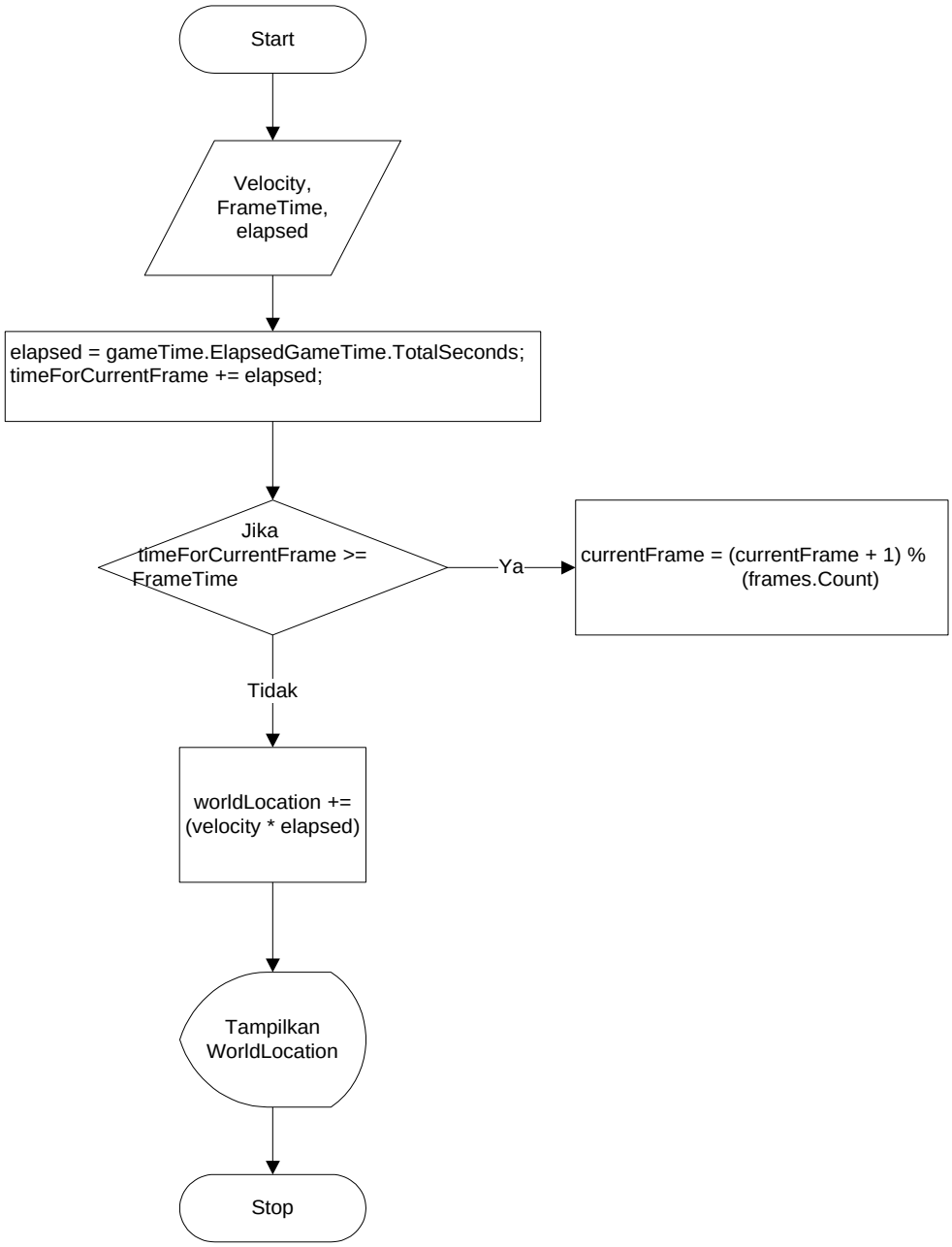
Method ini akan pertama kali dipanggil ketika menu mulai permainan dipilih. Method ini akan menginisialisasi posisi awal dari pemain dan jumlah peti.



Gambar 3.45 *Flowchart* Method StartNewGame

2. Method Update

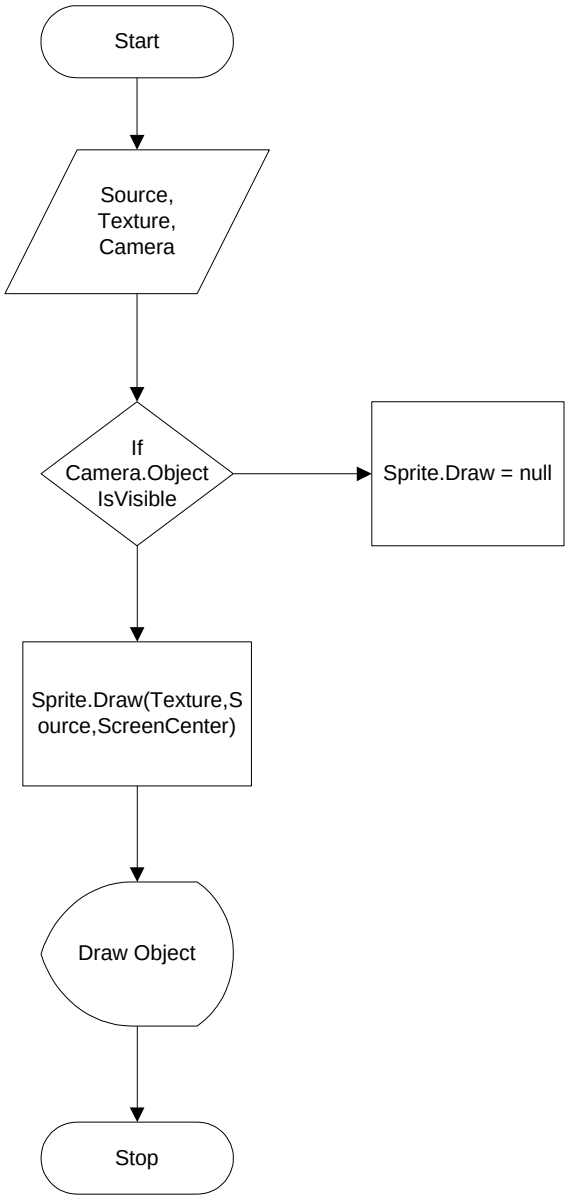
Method ini digunakan untuk melakukan perubahan *state* terhadap objek pemain, musuh dan peti.



Gambar 3.46 *Flowchart* Method Update

3. Method Draw

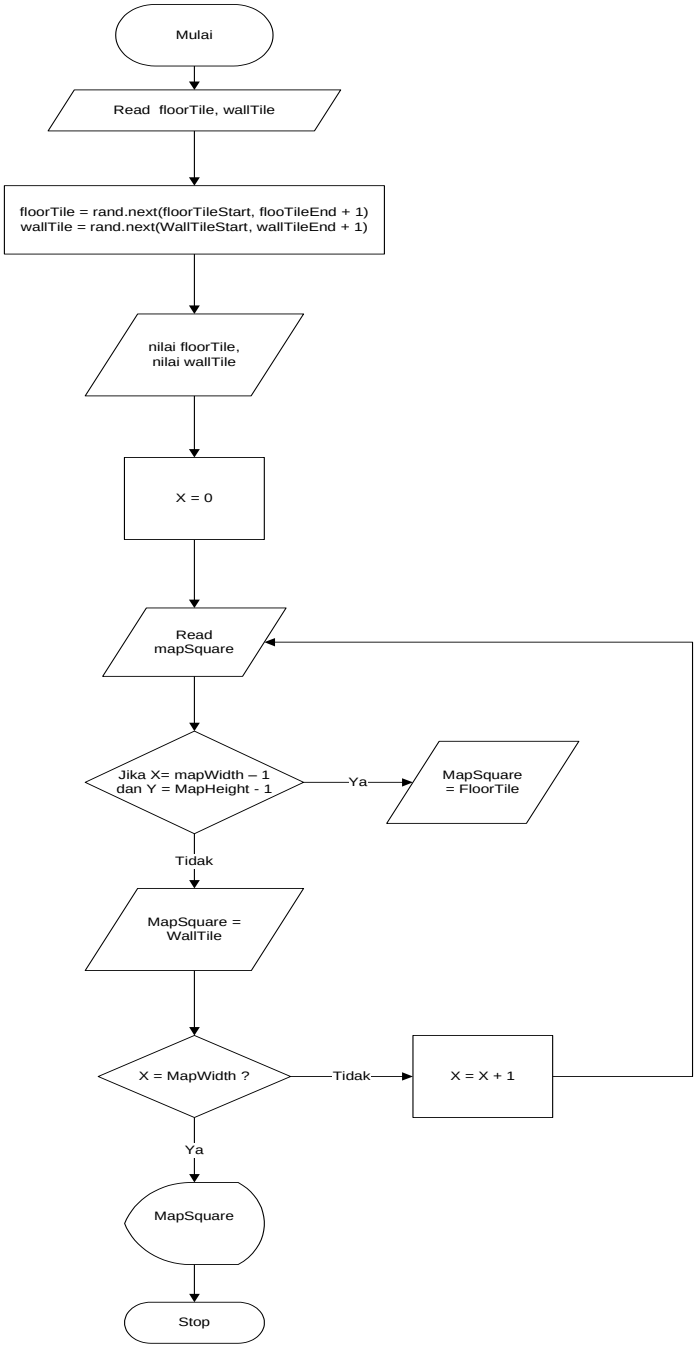
Method Draw berfungsi untuk menampilkan objek di dalam permainan. method ini akan digunakan pada saat menampilkan arena permainan, karakter pemain, musuh dan peti.



Gambar 3.47 *Flowchart* Method Draw

4. Method GenerateTileMap

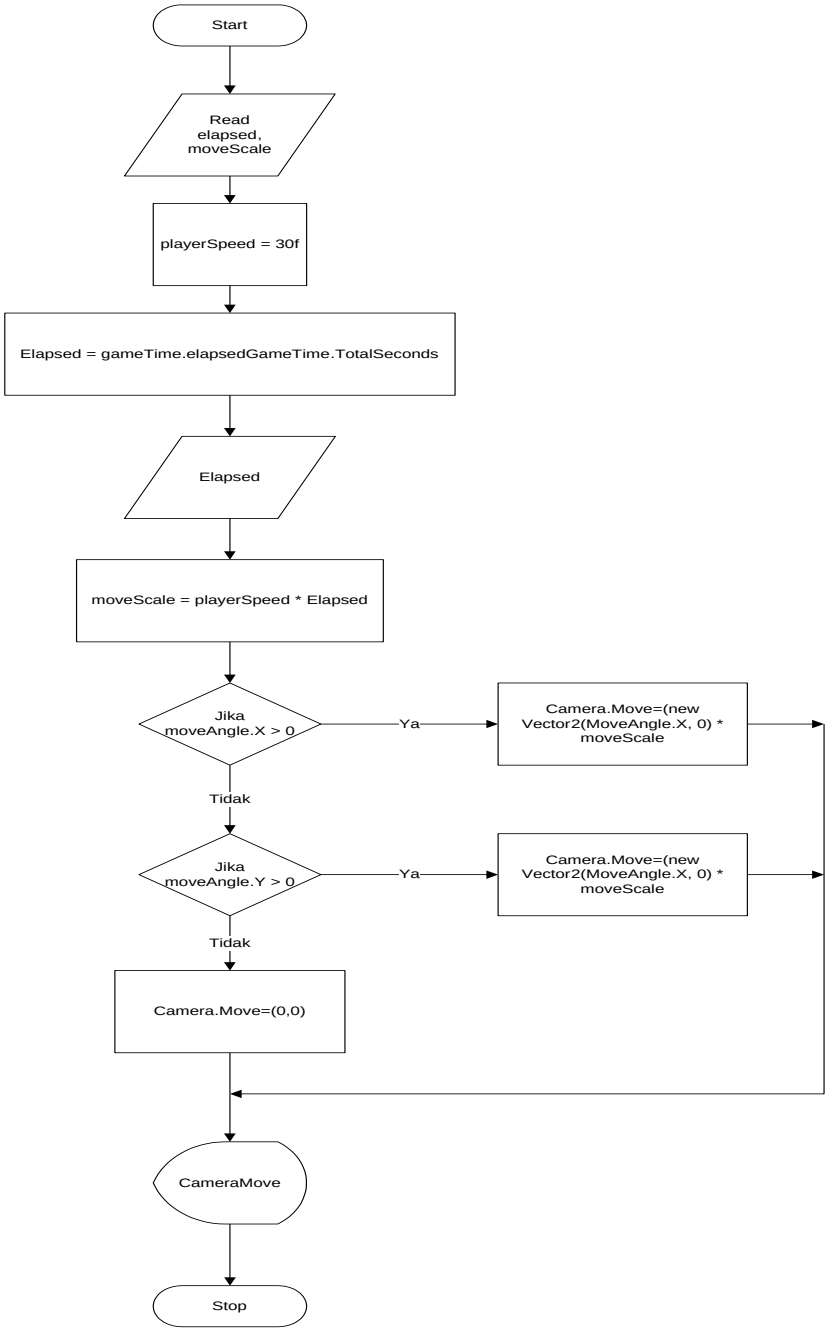
Method ini akan menampilkan pembatas secara acak di arena permainan dengan melakukan *random.*



Gambar 3.48 *Flowchart* Method GenerateTileMap

5. Method RepositionCamera

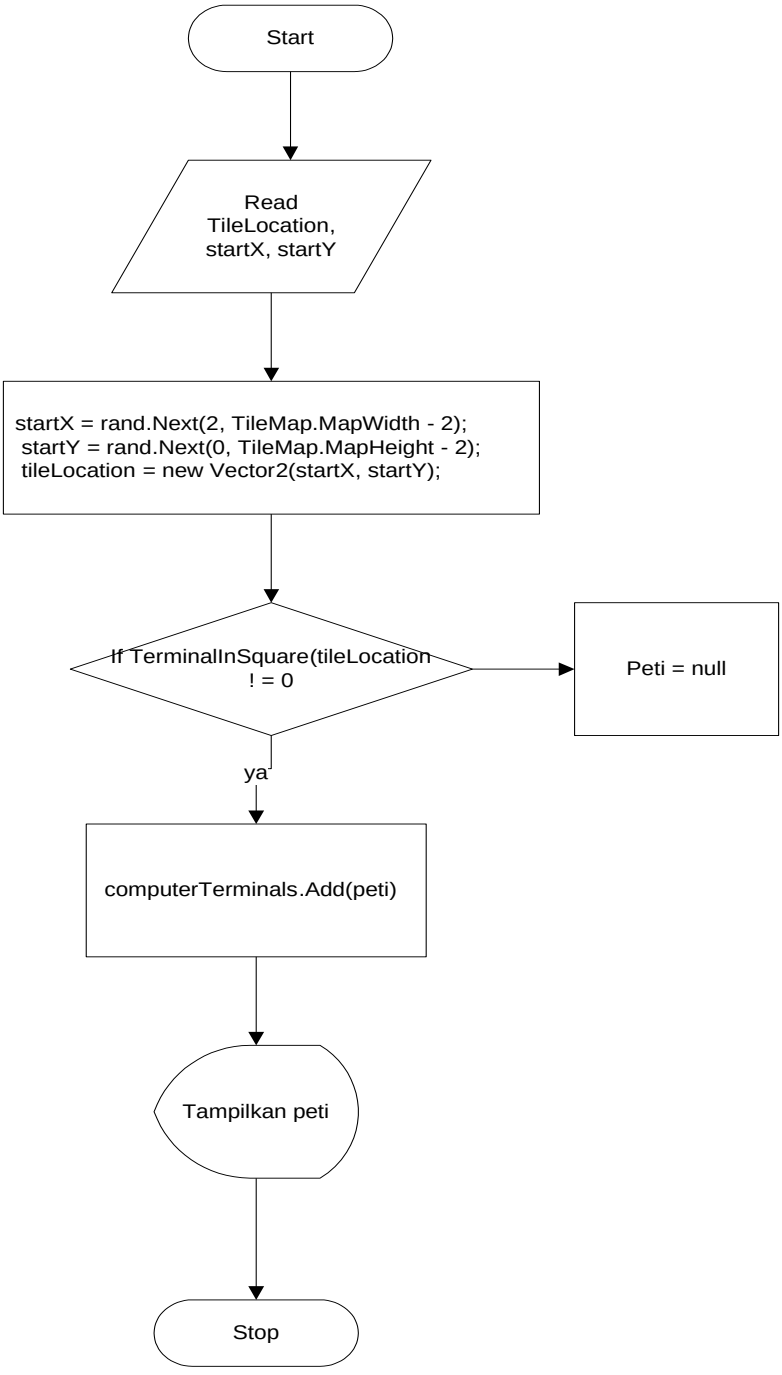
Method ini berfungsi untuk melakukan perubahan posisi kamera sesuai dengan arah pergerakan pemain.



Gambar 3.49 *Flowchart* Method RepositionCamera

6. Method AddPeti

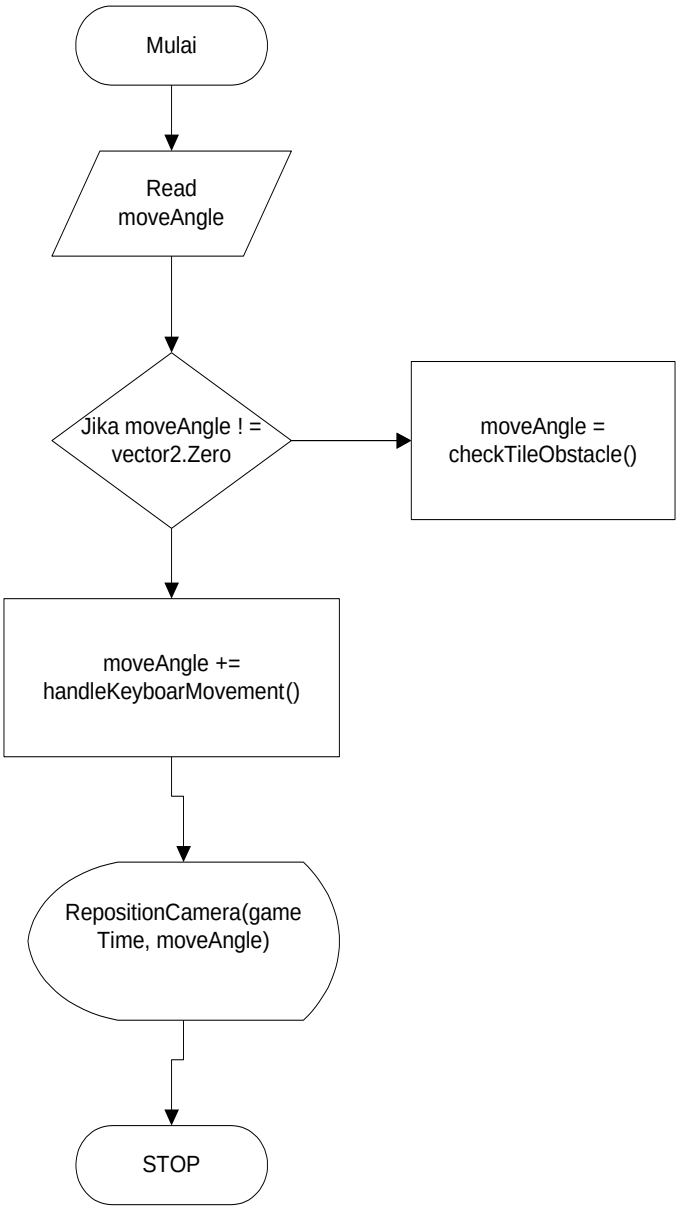
Method ini berfungsi melakukan penambahan serta penggambaran objek peti di arena permainan .



Gambar 3.50 *Flowchart* Method AddPeti

7. Method HandleInput

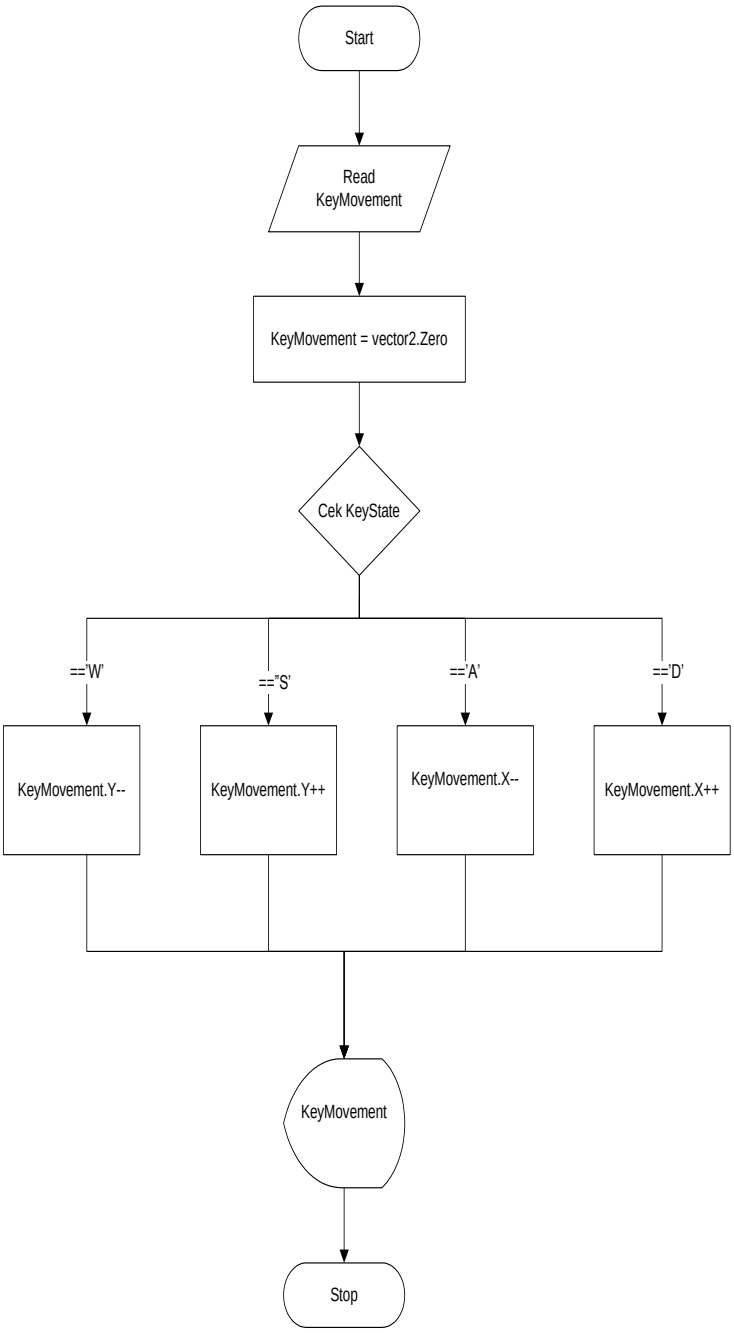
Method ini berfungsi melakukan penanganan inputan dari pemain . method ini juga akan akan mengubah posisi kamera mengikuti arah pergerakan pemain.



Gambar 3.51 *Flowchart* Method HandleInput

8. Method HandleKeyboardMovement

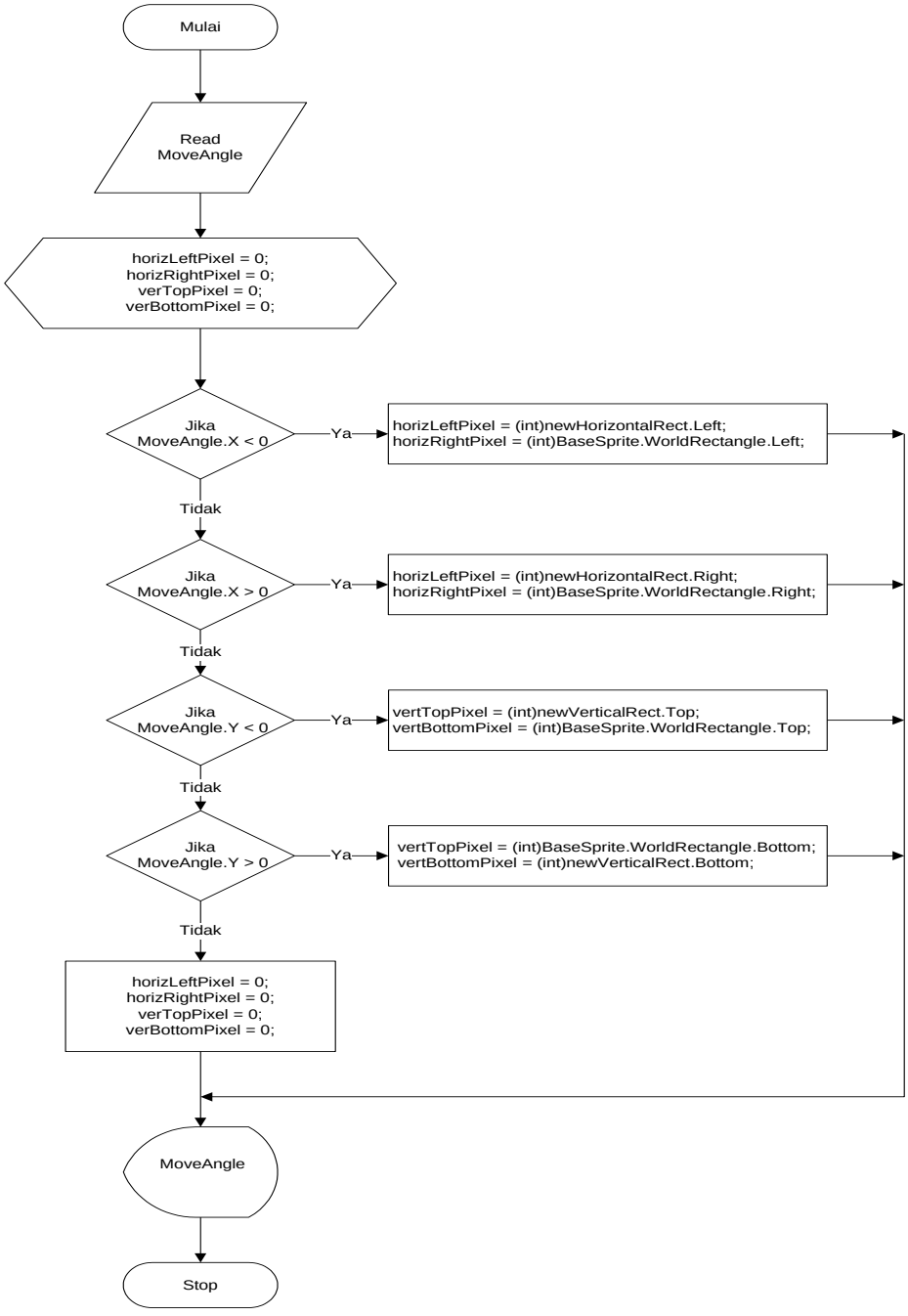
Method ini akan menangani inputan *keyboard* dari pengguna, sehingga karakter pemain dapat bergerak sesuai dengan keinginan dari pengguna di arena permainan.



Gambar 3.52 *Flowchart* Method HandleKeyboardMovement

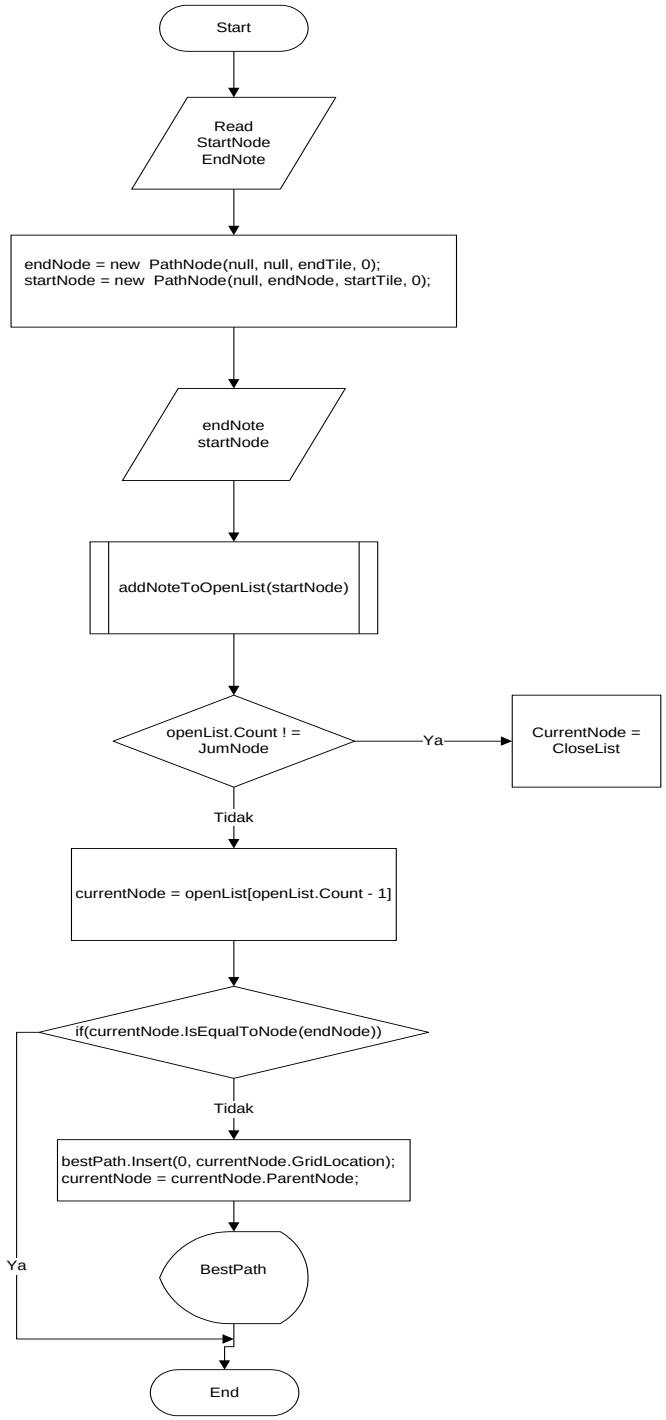
9. Method CheckTileObstacles

Method ini akan melakukan pengecekan terhadap *tile*/pembatas di arena permainan sehingga pemain dan musuh tidak dapat melewati *tile*.



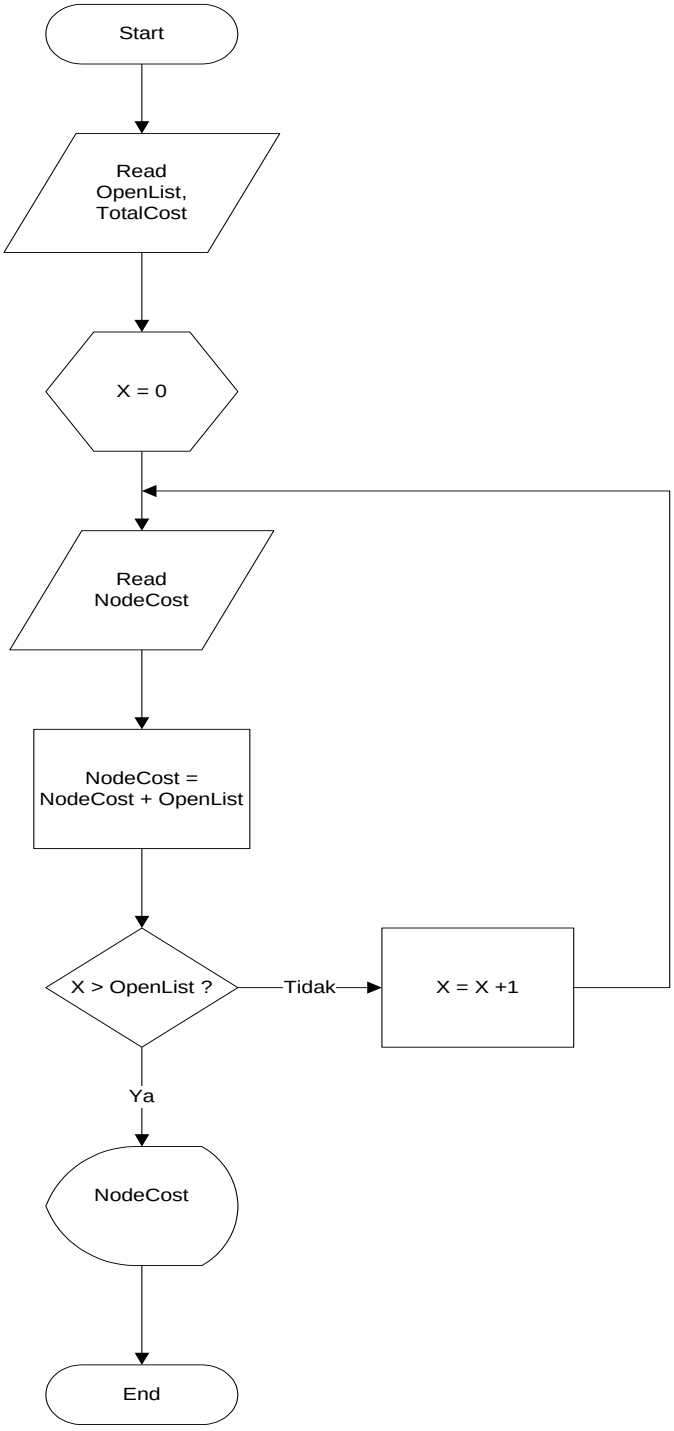
Gambar 3.53 *Flowchart* Method CheckTileObstacles

10. Method FindPath



Gambar 3.54 *Flowchart* Method FindPath

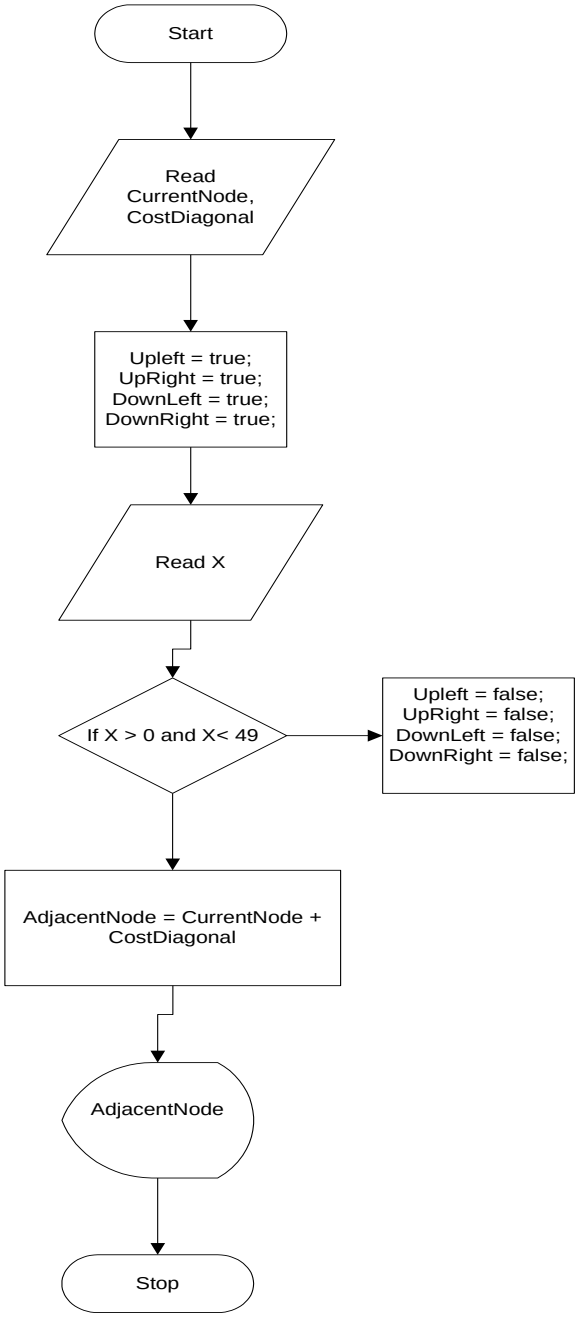
11. Method AddOpenList



Gambar 3.55 *Flowchart* Method AddOpenList

12. MethodAdjacentNode

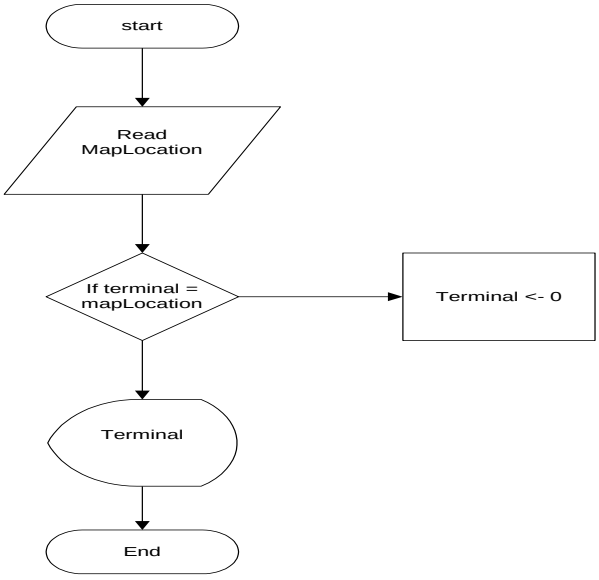
Method ini akan menghitung *node cost* yang harus ditempuh oleh musuh dalam melakukan pengejaran terhadap pemain.



Gambar 3.56 *Flowchart* Method AdjacentNode

13. Method PetiSquare

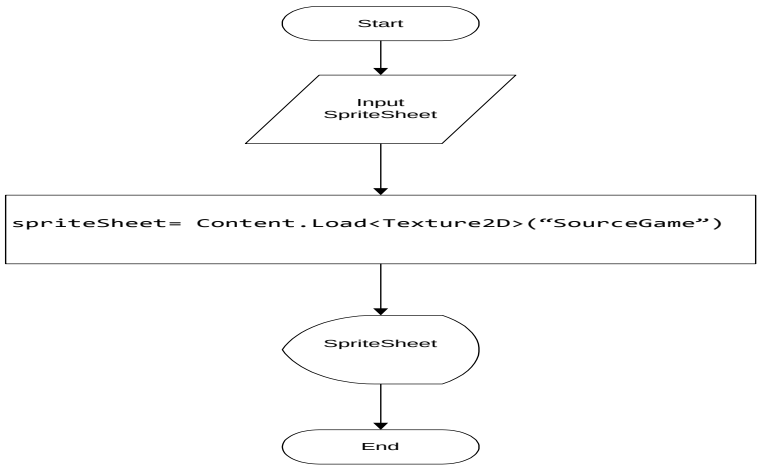
Method ini berfungsi meletakkan peti di arena permainan .



Gambar 3.57 *Flowchart* Method PetiSquare

14. Method LoadContent

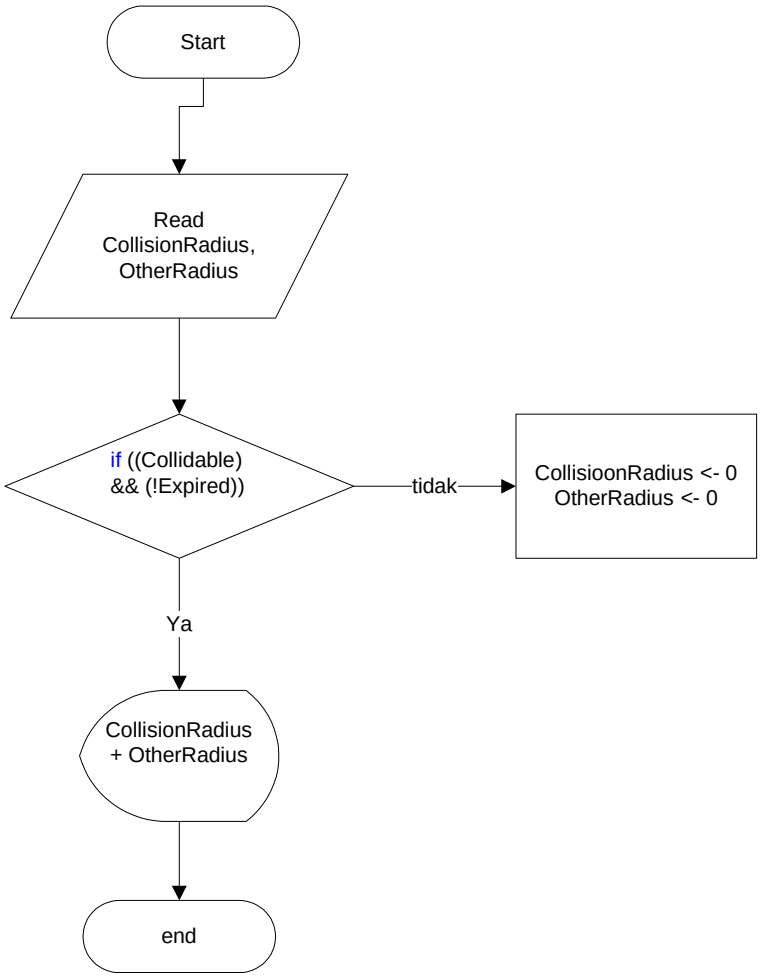
Method ini akan mengambil *source* yang dibutuhkan dalam *game Death Castle*, seperti gambar dari karakter pemain, musuh dan arena permainan.



Gambar 3.58 *Flowchart* Method LoadContent

15. Method CheckCollision

Method ini akan melakukan pengecekan terhadap suatu objek apabila ada kedua objek yang saling bertabrakan.



Gambar 3.59 *Flowchart* Method CheckCollision