BAB 3

ANALISIS DAN PERANCANGAN

3.1 Analisis Sistem

Pada analisis sistem akan mencakup analisis masalah, analisis *game* yang akan dikembangkan , analisis algoritma A*, analisis *accelerometer*, analisis kebutuhan non-fungsional dan analisis kebutuhan fungsional.

3.1.1 Analisis Masalah

Permainan tradisional merupakan salah satu warisan budaya, dan warisan budaya memiliki keperluan untuk dilestarikan dan dipertahankan keberadaannya. Salah satu diantaranya adalah egrang. Menurut beberapa hasil penelitian para ahli, permainan tradisional kini sudah mulai berkurang dan ditinggalkan, hal tersebut dikarenakan adanya permainan modern yang lebih efektif dan efisien yaitu *video game*. Untuk bermain *video game*, pemain tidak memerlukan tempat yang luas dan tidak perlu capek untuk memainkannya.

Namun alangkah baiknya jika permainan tradisional egrang tetap dilestarikan, selain merupakan salah satu warisan budaya dalam permainan tradisional egrang terkandung nilai budaya yaitu kerja keras, keuletan, dan sportivitas. Nilai kerja keras tercermin dari semangat para pemain yang berusaha agar dapat mengalahkan lawannya. Nilai keuletan tercermin dari proses pembuatan alat yang digunakan untuk berjalan yang memerlukan keuletan dan ketekunan agar seimbang dan mudah digunakan untuk berjalan. Dan, nilai sportivitas tercermin tidak hanya dari sikap para pemain yang tidak berbuat curang saat berlangsungnya permainan, tetapi juga mau menerima kekalahan dengan lapang dada.

3.1.1.1 Analisis *Game* Sejenis

Pada tahap ini, dilakukan analisis pada *game* sejenis sebagai perbandingan. Analisi game terhadap *game* sejenis dapat memberikan keuntungan tersendiri

terhadap pengembangan sebuah *game*, dengan menganalisis kelebihan dan kekurangan *game* sejenis. *Game* sejenis yang dianalisis adalah Egrang Bambu dan Balap Egrang.

1. *Game* Egrang Bambu

   Game egrang bambu adalah *game* bergenre balapan, *single player*. *Game* ini berbasis desktop dengan menggunakan flash. Dalam *game* ini, pemain akan memainkan balapan egrang tersebut dengan cara menekan tombol kiri, kanan, atas, dan bawah pada *keyboard* untuk menggerakan karakter.

   Egrang bambu menceritakan tentang seorang anak yang ikut perlombaan balapan egrang guna melestarikan permainan tradisional. Pemain terlebih dahulu dihadapkan pada sebuah pertanyaan mengenai egrang, jika semua jawabannya benar maka pemain akan memulai balapan. Arena balapan (*track*) berupa jalan lurus yang memiliki rintangan berupa kayu dan rumput, jika karakter tidak melewati rintangan tersebut atau menginjaknya maka kecepatan karakter akan melambat.



Gambar 3.1 Egrang bambu [11]

Kelebihan Game Egrang Bambu

Kelebihan game ini antara lain :

1) Antarmuka yang menarik
2) *Gameplay* cukup mudah untuk dimainkan

Kekurangan Game Egrang Bambu

Kekurangan game ini antara lain :

1) NPC tidak cerdas
2) Kontrol karakter cukup sulit

2. Balapan Egrang

Balapan egrang adalah game berbasis mobile J2ME, *single player*. Konsep yang diambil cukup sederhana yaitu tiga karakter berlomba di arena balapan (*track*) yang terdiri dari tiga jalan lurus untuk masing-masing karakternya. Untuk kontrol karakter pemain hanya menekan *keypad* atau tombol 5 (lima), jika pemain semakin cepat menekan tombol maka semakin cepat juga karakter bergerak dan permainan dibatasi oleh waktu yaitu selama 40 detik.



Gambar 3.2 Balapan Egrang [12]

Kelebihan Game Balap Egrang

Kelebihan game ini antara lain :

1) Antarmuka yang cukup menarik
2) *Gameplay* cukup mudah untuk dimainkan

Kekurangan Game Balap Egrang

Kekurangan game ini antara lain :

1) NPC tidak cerdas
2) Jalur musuh statis
3) Tidak ada tingkat kesulitan

3.1.2 Analisis *Game* yang Akan Dikembangkan

Game yang akan dibangun adalah Tradisional Egrang yang bertipe balapan (*racing*). *Game* ini dibangun dengan mengaplikasikan teknologi sebagai jembatan untuk memainkan egrang secara *digital*. Berikut fitur-fitur yang ada pada *game* ini

1. Sistem permainan bersifat *single player*.
2. Mengangkat tema tentang permainan tradisional egrang.
3. *Game* ini dijalankan secara *offline.*
4. Memiliki 3 (tiga) level yang berkesesuaian dengan tingat kesulitannya.
5. Kontrol karakter menggunakan accelerometer.
6. Aplikasi *game* bersifat bermain dan belajar yaitu setiap menyelesaikan *level*, pemain mendapatkan informasi tentang sejarah egrang, cara membuat egrang, dan nama-nama panggilan selain egrang di beberapa daerah di Indonesia.
7. NPC menggunakan algoritma *A Star*. Pada lawan digunakan untuk menentukan jalur terpendek menuju garis *finish*.
8. Grafis 2D (2 Dimensi).

3.1.2.1 *Story Line*

Game ini bercerita tentang seorang anak bernama Eman yang minder dan sering dikucilkan oleh teman-temannya. Kebetulan ketika itu di kampungnya sedang ramai dengan permainan egrang, karena melihat anak-anak seusianya

bermain egrang tentu saja timbul sebuah inisiatif dan keinginan untuk menunjukkan bahwa dia juga bisa melawan dengan cara mengikuti perlombaan balapan egrang. Tentu saja perlombaan itu juga diikuti oleh anak-anak yang mengucilkannya. Lalu mulailah Eman membuat egrang yang dibantu oleh ayahnya, setelah egrangnya jadi anak ini mulai melawan satu persatu (balapan) dengan anak-anak yang lain, walaupun bersusah payah untuk memenangkan setiap balapan tersebut namun Eman tidak patah semangat dan sampai akhirnya Eman menang sehingga mereka tidak meremehkan lagi .

3.1.2.2 *Leveling*

Tempat-tempat atau lokasi balapan egrang yang harus dimenangkan.

1. Rt 01

   Di tempat ini pemain akan melawan Asep, rintangan masih sedikit.
2. Rt 02

   Di tempat ini pemain akan melawan Nana, rintangan bertambah banyak dan NPC semakin cepat.
3. Rt 03

   Di tempat ini pemain akan melawan Dede, banyak rintangan dan NPC semakin cepat.

3.1.2.3 *Game Play*

Karakter akan berada di samping kiri layar, begitu juga dengan NPC. Ketika tombol mulai ditekan karakter dan NPC akan langsung balapan. Pemain harus berhati-hati terhadap rintangan, pada *level* 1 hingga *level* 3 rintangan akan bertambah banyak dan lebih rapat. Apabila pemain menabrak rintangan atau NPC maka kecepatan pemain akan berkurang.

Tugas utama pemain dalam memainkan *game* ini adalah memenangkan balapan pada setiap *level*-nya. Pemain dinyatakan menang apabila sampai di garis *finish* berada diposisi pertama sehingga infromasi mengenai sejarah egrang akan tampil dan pada saat itu juga pemain dapat melanjutkan ke *level* selanjutnya. Apabila pemain tidak berhasil memenangkan perlombaan informasi mengenai

sejarah egrang tidak akan tampil tapi yang akan tampil yaitu permintaan untuk mengulang kembali perlombaan atau keluar dari area permainan. Jumlah *level* yang harus dimenangkan yaitu terdiri dari tiga *level*, dimana pada setiap *level*-nya memiliki kesulitan yang berbeda, semakin tinggi *level* maka tingkat kesulitan akan semakin sulit.

Cara bermain *game* tradisional egrang ini yaitu pemain memiringkan *device* yang sudah terinstal atau memiliki *sensor accelerometer* untuk mengontrol karakter pada game dan menekan tombol virtual yang terdapat pada pojok kiri dan kanan layar secara bergantian untuk mempercepat pergerakan karakter.

3.1.3 Analisis Algoritma A*

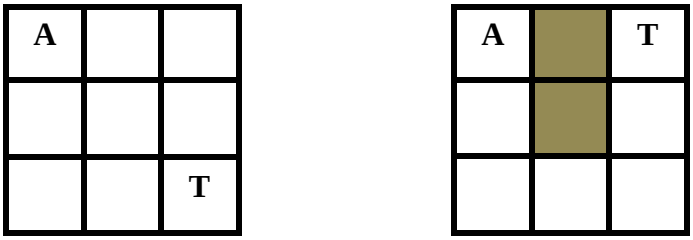
Game tradisional egrang akan berupa kotak-kotak dengan ordo X x Y. Pada ordo X x Y, dua kotak akan dipakai oleh titik awal dan titik tujuan, sisanya untuk menghasilkan jalan agar terlihat jalurnya yang pada akhirnya akan menentukan jalan terpendek ke titik tujuan. Maksimal penghalang pada tiap ordo dapat kita tentukan yaitu dengan rumus pada tabel.

Tabel 3.1 Maksimal penghalang

Ordo	Maksimal Penghalang
X x Y	(M x N) – 2
3 x 3	(3 x 3) - 2 = 7
4 x 4	(4 x 4) - 2 = 14
5 x 5	(5 x 5) - 2 = 23
... x ...	(... x ...) - 2 = ...

Keterangan :
X : Nilai baris
Y : Nilai kolom

Dengan maksimal penghalang (X x Y) – 2, maka akan menghasilkan jalan yang tidak terlihat disebabkan jarak antara titik awal dan titik tujuan saling berdekatan, tapi dengan kondisi awal dan titik tujuan saling berjauhan maka akan menghasilkan jalan tidak akan ditemukan. Agar terlihat jalan yang dihasilkan maka diusahakan titik awal dengan titik tujuan tidak saling berdekatan dan jumlah penghalang yang akan dipasangkan kurang dari maksimal, implementasinya dapat dilihat dengan contoh ordo 4x10 pada gambar 3.4.



(i) (ii)

Gambar 3.3 Ordo 3x3 dengan dua kondisi

Keterangan :

A : Titik awal :Penghalang

T : Titik tujuan

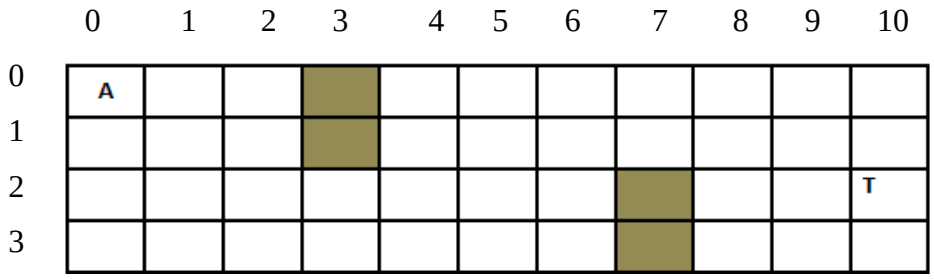
Maksimal penghalang yang akan dipasangkan pada ordo lainnya adalah sebanyak ordo tersebut dikurangi dua untuk menempatkan titik awal dan titik tujuan. Pada gambar menunjukkan suatu ruang (*map*) dengan ordo 3x3 di dalam *game* yang akan dibangun. Setiap kotak mempresentasikan simpul (*node*). Setiap kotak terhubung ke delapan kotak yang paling dekat, artinya setiap simpul (*node*) terhubung ke simpul lain yang berada di sebelah kanan, kiri, atas-kanan, bawah-kanan, bawah-kiri, dan atas-kiri dari simpul tersebut. Kotak warna cokelat diimplementasikan sebagai penghalang, yaitu kotak yang tidak dapat dilalui oleh titik awal.

Karena titik A tidak terhubung langsung ke titik T, maka kita perlu melewati simpul-simpul tertentu yang pada akhirnya akan mengantarkan ke titik T dengan jarak sependek mungkin.

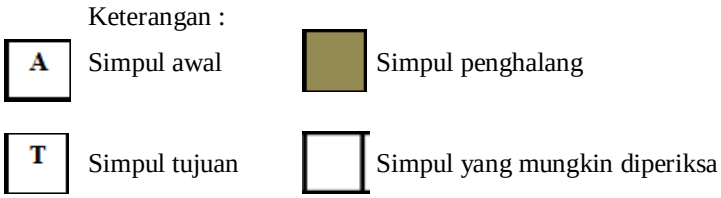
3.1.3.1 Analisis Pemecahan Masalah dengan Algoritma A* (*A Star)*

Sudah dijelaskan sebelumnya pada analisis lintasan bahwa ordo dapat disesuaikan dengan rentang X x Y, salah satu contoh perhitungan pada *game* tradisional egrang ini akan dijelasakan dengan ordo minimal yaitu 4x10 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. Contoh perhitungannya seperti pada gambar 3.4 yang diberi nilai bobot yang terkecil yaitu 1 dan bobot selanjutnya ditambahkan 1.

Contoh kasus pada lintasan dengan ordo 4 x 10 yang terlihat pada gambar 3.4, akan dihitung pencariannya dengan algoritma A*.



Gambar 3.4 Kondisi lintasan dengan ordo 4 x 10



Posisi simpul awal = Ax : 0, Ay : 0

Posisi simpul tujuan = goal x : 10, goal y : 2

Langkah ke satu

n (1,1) : 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

n (1,0) : 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)) 57

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

n (0,1) : 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)))

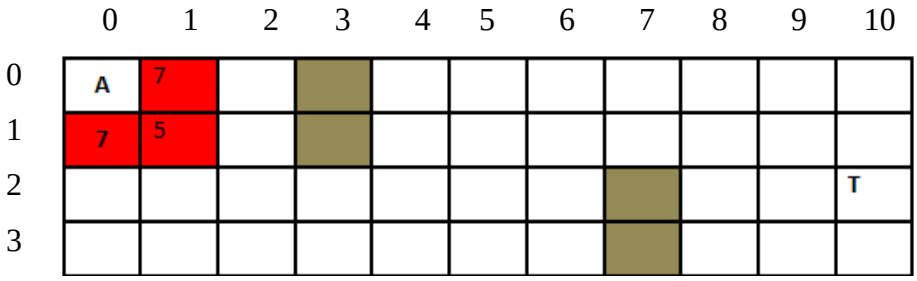
= -3 + 9

= 6 58

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

= 1 + 6

= 7



Gambar 3.5 Langkah pertama pencarian *bestnode*

Keterangan :

Simpul yang diperiksa

A Simpul awal

T Simpul tujuan

Pada gambar 3.5 terdapat tiga simpul yang mungkin menjadi *best node* 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) dengan biaya terkecil yaitu 5.

Langkah ke dua

n (2,2) : 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(0,1) = (-0) + (0-(2*(-0)))

= 0 + 0

= 0

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

= 2 + 0

= 2

n (2,1) : g (2,1) = 2

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

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

= -1 + 3

= 2

f (2,1) = g (2,1) + h (2,1)

= 2 + 2

= 4

n (1,2) : g (1,2) = 2

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

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

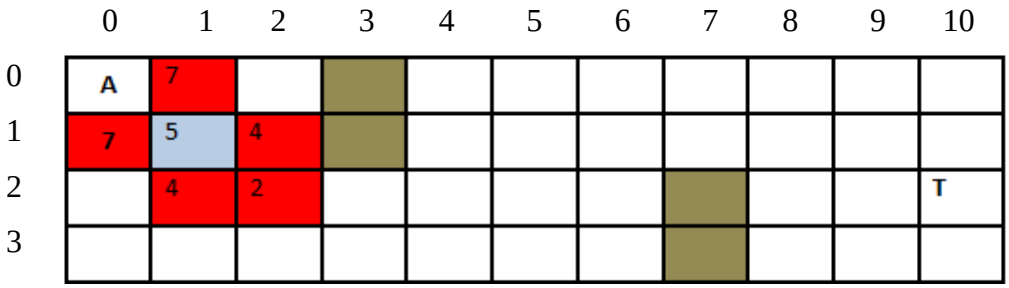
= -1 + 3

= 2

f (1,2) = g (1,2) + h (1,2)

= 2 + 2

= 4



Gambar 3.6 Langkah kedua pencarian *bestnode*

 Simpul yang diperiksa  Simpul awal

 Simpul tujuan  Simpul yang dipilih

Dari ke tiga simpul yang mungkin maka dipilihlah simpul (2,2) dengan biaya terkecil yaitu 2.

Langkah ke tiga

n (3,3) : g (3,3) = 3

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

h_orthogonal(3,3) = (abs(3 - 2) + abs(3 - 2))

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

= 2

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

h_diagonal(3,3) = min(abs(3 - 2)+abs(3 - 2))

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

= min 2

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

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

= -2 + 6

= 4

f (3,3) = g (3,3) + h (3,3)

= 3 + 4

= 7

n (3,2) : g (3,2) = 3

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

h_orthogonal(3,2) = (abs(3 - 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(3,2) = min(abs(3 - 2)+abs(2 - 2))

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

= min 1

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

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

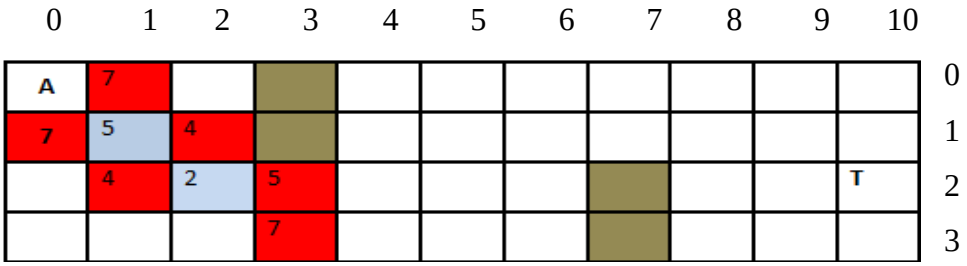
= -1 + 3

= 2

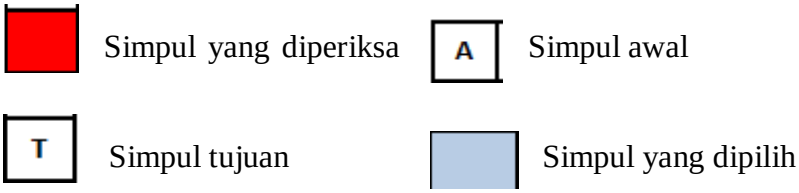
f (3,2) = g (3,2) + h (3,2)

= 3 + 2

= 5



Gambar 3.7 Langkah ketiga pencarian *bestnode*



Dari ke dua simpul yang mungkin maka dipilihlah simpul (3,2) dengan biaya terkecil yaitu 5.

Langkah ke empat

n (4,2) : g (4,2) = 4

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

h_orthogonal(4,2) = (abs(4 - 2) + abs(2 - 2))

= (abs(2) + abs(0))

= 2

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

h_diagonal(4,2) = min(abs(4 - 2)+abs(2 - 2))

= min(abs(2)+abs(0))

= min 2

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

h(4,2) = (-2) + (2-(2*(-2)))

= -2 + 6

= 4

f (4,2) = g (4,2) + h (4,2)

= 4 + 4

= 8

n (4,3) : g (4,3) = 4

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

h_orthogonal(4,3) = (abs(4 - 2) + abs(3 - 2))

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

= 3

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

h_diagonal(4,3) = min(abs(4 - 2)+abs(3 - 2))

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

= min 3

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

h(4,3) = (-3) + (3-(2*(-3)))

= -3 + 9

= 6

f (4,3) = g (4,3) + h (4,3)

= 4 + 6

= 10

n (4,1) : g (4,1) = 4

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

h_orthogonal(4,1) = (abs(4 - 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(4,1) = min(abs(4 - 2)+abs(1 - 2))

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

= min 3

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

h(4,1) = (-3) + (3-(2*(-3)))

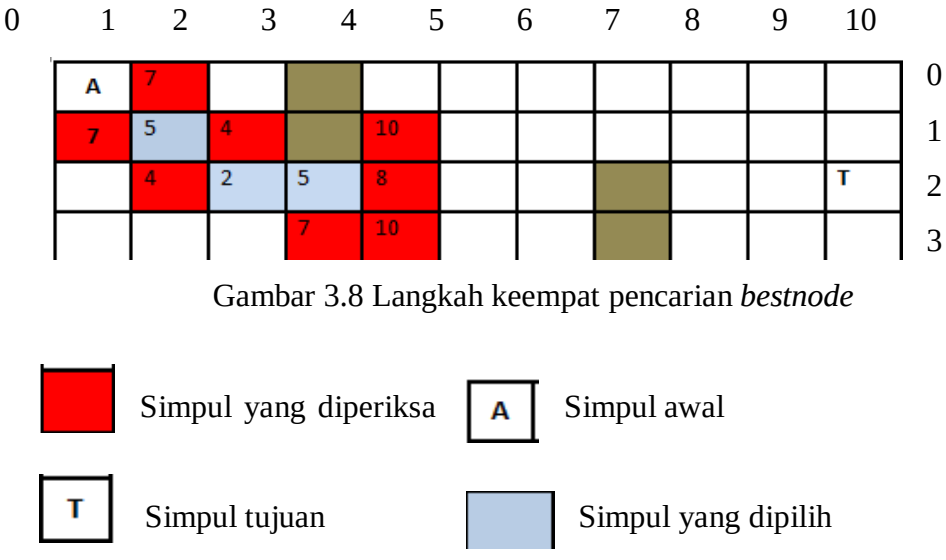
= -3 + 9

= 6

f (4,1) = g (4,1) + h (4,1)

= 4 + 6

= 10



Gambar 3.8 Langkah keempat pencarian *bestnode*

Dari ke tiga simpul yang mungkin maka dipilihlah simpul (4,2) dengan biaya terkecil yaitu 8.

Langkah ke lima

n (5,2) : g (5,2) = 5

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

h_orthogonal(5,2) = (abs(5 - 2) + abs(2 - 2))

= (abs(3) + abs(0))

= 3

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

h_diagonal(5,2) = min(abs(5 - 2)+abs(2 - 2))

= min(abs(3)+abs(0))

= min 3

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

h(5,2) = (-3) + (3-(2*(-3)))

= -3 + 9

= 6

f (5,2) = g (5,2) + h (5,2)

= 5 + 6

= 11

n (5,3) : g (5,3) = 5

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

h_orthogonal(5,3) = (abs(5 - 2) + abs(3 - 2))

= (abs(3) + abs(1))

= 4

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

h_diagonal(5,3) = min(abs(5 - 2)+abs(3 - 2))

= min(abs(3)+abs(1))

= min 4

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

h(5,3) = (-4) + (4-(2*(-4)))

= -4 + 12

= 8

f (5,3) = g (5,3) + h (5,3)

= 5 + 8

= 13

n (5,1) : g (5,1) = 5

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

h_orthogonal(5,1) = (abs(5 - 2) + abs(1 - 2))

= (abs(3) + abs(-1))

= 4

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

h_diagonal(5,1) = min(abs(5 - 2)+abs(1 - 2))

= min(abs(3)+abs(-1))

= min 4

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

h(5,1) = (-4) + (4-(2*(-4)))

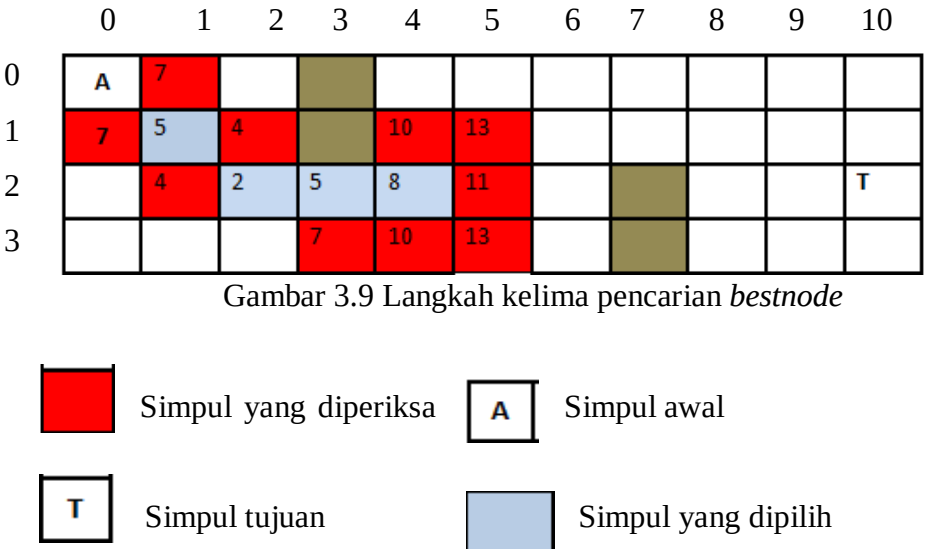
= -4 + 12

= 8

f (5,1) = g (5,1) + h (5,1)

= 5 + 8

= 13



Gambar 3.9 Langkah kelima pencarian *bestnode*

Dari ke tiga simpul yang mungkin maka dipilihlah simpul (5,2) dengan biaya terkecil yaitu 11.

Langkah ke enam

n (6,2) : g (6,2) = 6

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

h_orthogonal(6,2) = (abs(6 - 2) + abs(2 - 2))

= (abs(4) + abs(0))

= 4

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

h_diagonal(6,2) = min(abs(6 - 2)+abs(2 - 2))

= min(abs(4)+abs(0))

= min 4

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

h(6,2) = (-4) + (4-(2*(-4)))

= -4 + 12

= 8

f (6,2) = g (6,2) + h (6,2)

= 6 + 8

= 14

n (6,3) : g (6,3) = 6

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

h_orthogonal(6,3) = (abs(6 - 2) + abs(3 - 2))

= (abs(4) + abs(1))

= 5

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

h_diagonal(6,3) = min(abs(6 - 2)+abs(3 - 2))

= min(abs(4)+abs(1))

= min 5

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

h(6,3) = (-5) + (5-(2*(-5)))

= -5 + 15

= 10

f (6,3) = g (6,3) + h (6,3)

= 6 + 10

= 16

n (6,1) : g (6,1) = 6

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

h_orthogonal(6,1) = (abs(6 - 2) + abs(1 - 2))

= (abs(4) + abs(-1))

= 5

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

h_diagonal(6,1) = min(abs(6 - 2)+abs(1 - 2))

= min(abs(4)+abs(-1))

= min 5

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

h(6,1) = (-5) + (5-(2*(-5)))

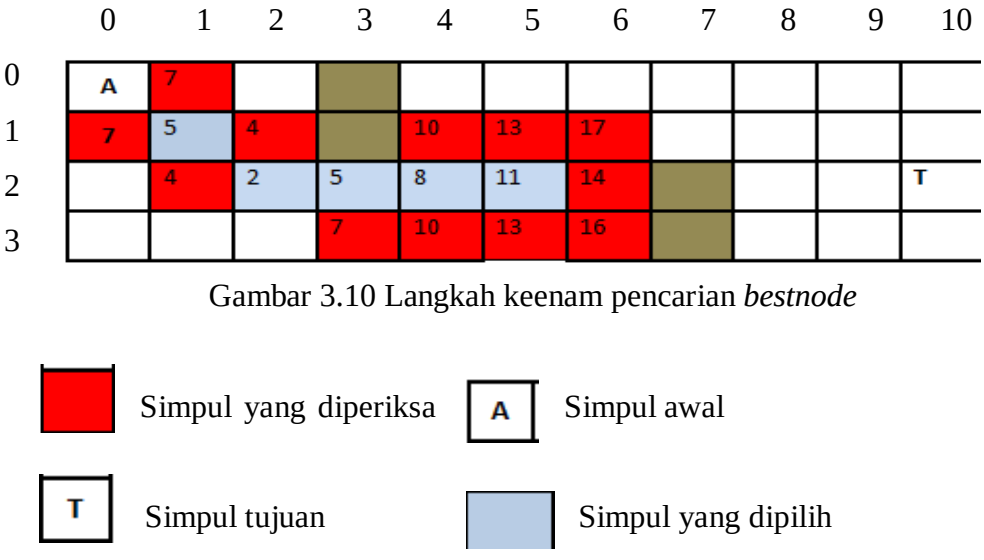
= -5 + 15

= 10

f (6,1) = g (6,1) + h (6,1)

= 6 + 10

= 16



Gambar 3.10 Langkah keenam pencarian *bestnode*

Dari ke tiga simpul yang mungkin maka dipilihlah simpul (6,2) dengan biaya terkecil yaitu 14.

Langkah ke 7

n (7,1) : g (7,1) = 7

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

h_orthogonal(7,1) = (abs(7 - 2) + abs(1 - 2))

= (abs(5) + abs(-1))

= 6

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

h_diagonal(7,1) = min(abs(7 - 2)+abs(1 - 2))

= min(abs(5)+abs(-1))

= min 6

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

h(7,1) = (-6) + (6-(2*(-6)))

= -6 + 18

= 12

f (7,1) = g (7,1) + h (7,1)

= 7 + 12

= 19

n (6,1) : g (6,1) = 7

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

h_orthogonal(6,1) = (abs(6 - 2) + abs(1 - 2))

= (abs(4) + abs(-1))

= 5

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

h_diagonal(6,1) = min(abs(6 - 2)+abs(1 - 2))

= min(abs(4)+abs(-1))

= min 5

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

h(6,1) = (-5) + (5-(2*(-5)))

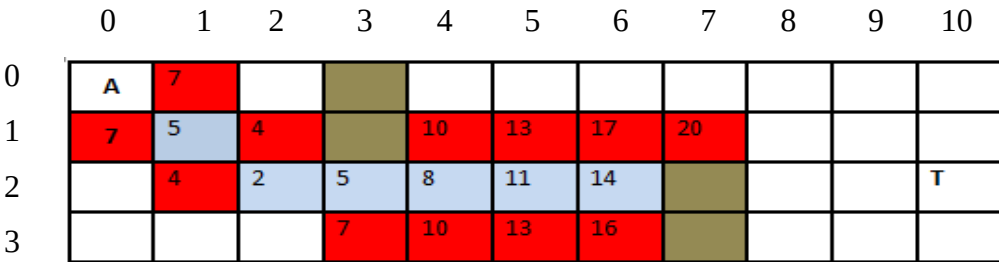
= -5 + 15

= 10

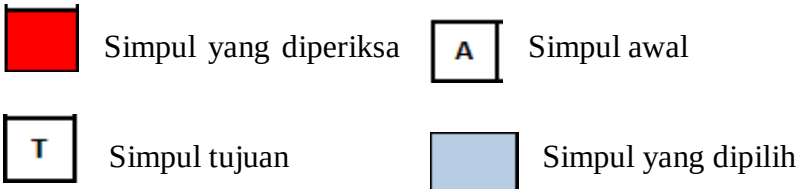
f (6,1) = g (6,1) + h (6,1)

= 7 + 10

= 17



Gambar 3.11 Langkah ketujuh pencarian *bestnode*



Dari ke dua simpul yang mungkin maka dipilihlah simpul (6,1) dengan biaya terkecil yaitu 17.

Langkah ke delapan

n (7,0) : g (7,0) = 8

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

h_orthogonal(7,0) = (abs(7 - 2) + abs(0 - 2))

= (abs(5) + abs(-2))

= 7

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

h_diagonal(7,0) = min(abs(7 - 2)+abs(0 - 2))

= min(abs(5)+abs(-2))

= min 7

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

h(7,0) = (-7) + (7-(2*(-7)))

= -7 + 21

= 14

f (7,0) = g (7,1) + h (7,1)

= 8 + 14

= 22

n (7,1) : g (7,1) = 8

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

h_orthogonal(7,1) = (abs(7 - 2) + abs(1 - 2))

= (abs(5) + abs(-1))

= 6

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

h_diagonal(7,1) = min(abs(7 - 2)+abs(1 - 2))

= min(abs(5)+abs(-1))

= min 6

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

h(7,1) = (-6) + (6-(2*(-6)))

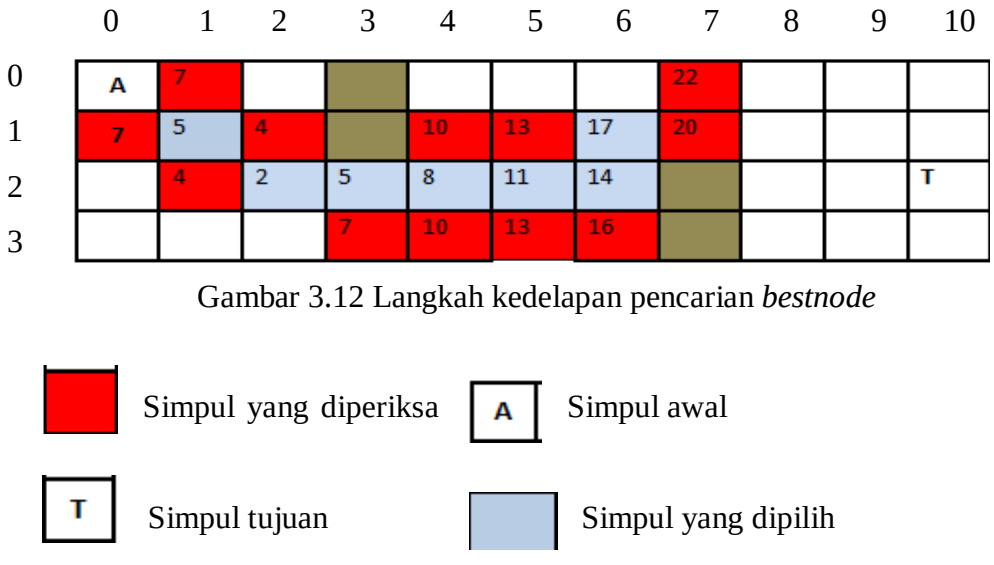
= -6 + 18

= 12

f (7,1) = g (7,1) + h (7,1)

= 8 + 12

= 20



Gambar 3.12 Langkah kedelapan pencarian *bestnode*

Dari ke dua simpul yang mungkin maka dipilihlah simpul (7,1) dengan biaya terkecil yaitu 20.

Langkah ke sembilan

n (8,0) : g (8,0) = 9

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

h_orthogonal(8,0) = (abs(8 - 2) + abs(0 - 2))

= (abs(6) + abs(-2))

= 8

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

h_diagonal(8,0) = min(abs(8 - 2)+abs(0 - 2))

= min(abs(6)+abs(-2))

= min 8

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

h(8,0) = (-8) + (8-(2*(-8)))

= -8 + 24

= 16

f (8,0) = g (7,1) + h (7,1)

= 9 + 16

= 25

n (8,1) : g (8,1) = 9

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

h_orthogonal(8,1) = (abs(8 - 2) + abs(1 - 2))

= (abs(6) + abs(-1))

= 7

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

h_diagonal(8,1) = min(abs(8 - 2)+abs(1 - 2))

= min(abs(6)+abs(-1))

= min 7

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

h(8,1) = (-7) + (7-(2*(-7)))

= -7 + 21

= 14

f (8,1) = g (8,1) + h (8,1)

= 9 + 14

= 23

n (8,2) : g (8,2) = 9

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

h_orthogonal(8,2) = (abs(8 - 2) + abs(2 - 2))

= (abs(6) + abs(0))

= 6

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

h_diagonal(8,2) = min(abs(8 - 2)+abs(2 - 2))

= min(abs(6)+abs(0))

= min 6

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

h(8,2) = (-6) + (6-(2*(-6)))

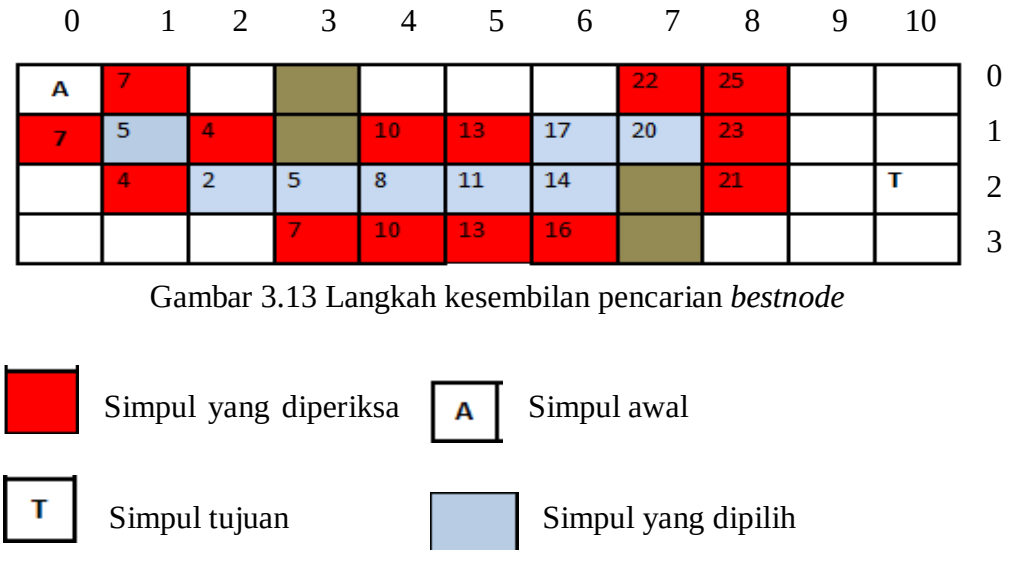
= -6 + 18

= 12

f (8,2) = g (8,2) + h (8,2)

= 9 + 12

= 21



Gambar 3.13 Langkah kesembilan pencarian *bestnode*

Dari ke dua simpul yang mungkin maka dipilihlah simpul (8,2) dengan biaya terkecil yaitu 21.

Langkah ke sepuluh

n (9,1) : g (9,1) = 10

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

h_orthogonal(9,1) = (abs(9 - 2) + abs(1 - 2))

= (abs(7) + abs(-1))

= 8

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

h_diagonal(9,1) = min(abs(9 - 2)+abs(1 - 2))

= min(abs(7)+abs(-1))

= min 8

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

h(9,1) = (-8) + (8-(2*(-8)))

= -8 + 24

= 16

f (9,1) = g (9,1) + h (9,1)

= 10 + 16

= 26

n (9,2) : g (9,2) = 10

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

h_orthogonal(9,2) = (abs(9 - 2) + abs(2 - 2))

= (abs(7) + abs(0))

= 7

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

h_diagonal(9,2) = min(abs(9 - 2)+abs(2 - 2))

= min(abs(7)+abs(0))

= min 7

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

h(9,2) = (-7) + (7-(2*(-7)))

= -7 + 21

= 14

f (9,2) = g (9,2) + h (9,2)

= 10 + 14

= 24

n (9,3) : g (9,3) = 10

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

h_orthogonal(9,3) = (abs(9 - 2) + abs(3 - 2))

= (abs(7) + abs(1))

= 8

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

h_diagonal(9,3) = min(abs(9 - 2)+abs(3 - 2))

= min(abs(7)+abs(1))

= min 8

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

h(9,3) = (-8) + (8-(2*(-8)))

= -8 + 24

= 16

f (9,3) = g (9,3) + h (9,3)

= 10 + 16

= 26

n (8,3) : g (8,3) = 10

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

h_orthogonal(8,3) = (abs(8 - 2) + abs(3 - 2))

= (abs(6) + abs(1))

= 7

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

h_diagonal(8,3) = min(abs(8 - 2)+abs(3 - 2))

= min(abs(6)+abs(1))

= min 7

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

h(8,3) = (-7) + (7-(2*(-7)))

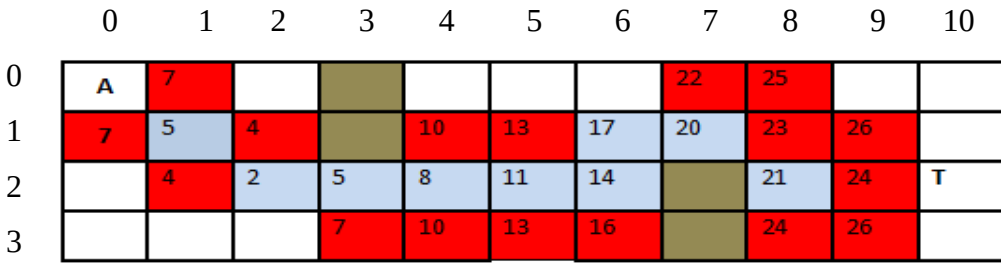
= -7 + 21

= 14

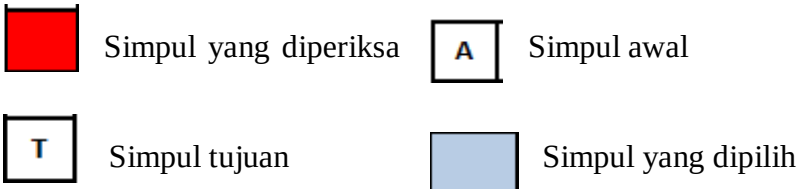
f (8,3) = g (8,3) + h (8,3)

= 10 + 14

= 24



Gambar 3.14 Langkah kesepuluh pencarian *bestnode*



Dari ke empat simpul yang mungkin maka dipilihlah simpul (9,2) dengan biaya terkecil yaitu 24.

Langkah ke sebelas

n (10,1) : g (10,1) = 11

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

h_orthogonal(10,1) = (abs(10 - 2) + abs(1 - 2))

= (abs(8) + abs(-1))

= 9

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

h_diagonal(10,1) = min(abs(10 - 2)+abs(1 - 2))

= min(abs(8)+abs(-1))

= min 9

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

h(10,1) = (-9) + (9-(2*(-9)))

= -9 + 27

= 18

f (10,1) = g (10,1) + h (10,1)

= 11 + 18

= 29

n (10,2) : g (10,2) = 11

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

h_orthogonal(10,2) = (abs(10 - 2) + abs(2 - 2))

= (abs(8) + abs(0))

= 8

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

h_diagonal(10,2) = min(abs(10 - 2)+abs(2 - 2))

= min(abs(8)+abs(0))

= min 8

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

h(10,2) = (-8) + (8-(2*(-8)))

= -8 + 24

= 16

f (10,2) = g (10,2) + h (10,2)

= 11 + 16

= 27

n (10,3) : g (10,3) = 11

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

h_orthogonal(10,3) = (abs(10 - 2) + abs(3 - 2))

= (abs(8) + abs(1))

= 9

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

h_diagonal(10,3) = min(abs(10 - 2)+abs(3 - 2))

= min(abs(8)+abs(1))

= min 9

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

h(10,3) = (-9) + (9-(2*(-9)))

= -9 + 27

= 18

f (10,3) = g (10,3) + h (10,3)

= 11 + 18

= 29

Dari ke tiga simpul yang mungkin menjadi *bestnode* maka dipilihlah simpul (10,2) dengan biaya terkecil yaitu 27, dan dikenali sebagai simpul tujuan yaitu (10,2) berarti solusi telah ditemukan.

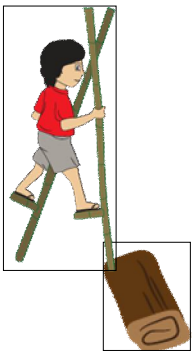
	0	1	2	3	4	5	6	7	8	9	10
0	A	7						22	25		
1	7	5	4		10	13	17	20	23	26	29
2		4	2	5	8	11	14		21	24	T
3				7	10	13	16		24	26	29

Gambar 3.15 Hasil pencarian jalan dengan Algoritma A*

3.1.4 *Collision Detection* Karakter terhadap Rintangan atau NPC

Collision atau tubrukan adalah metode yang selalu dipakai di dalam pengembangan *game*. Hampir semua *game* memiliki *collision*. Namun, apabila terlalu banyak terjadi *collision* dengan tingkat pengecekan yang kurang tepat dapat menyebabkan performa *game* menjadi menurun.

Pada *game* Tradisional Egrang *collision* diimplementasikan terhadap karakter dan rintangan atau NPC. Untuk melakukan pendeteksian *collision* pada karakter terhadap rintangan atau NPC dengan menggunakan *method line & shape collision* yaitu pengecekan didasarkan pada *bound* (kotak) luar objek. Proses ini memiliki kelebihan tidak memakan banyak memori. Contoh *collision* antara karkater dengan rintangan dapat dilihat pada gambar 3.16.



Gambar 3.16 *Collision* antara karakter dengan rintangan

Cara melakukan deteksi tubrukannya yaitu dengan menambahkan *method collidesWith* (*entity*) di dalam *method onUpdate*. Contoh pendeteksian *collision* antara karakter dengan rintangan, dan karakter dengan NPC.

Sebelumnya *player* (karakter) dan kayu (rintangan) terlebih dahulu dideklarasikan, lalu diinisialisasi pada *methode onLoadScene*. Nilai *true* akan dikembalikan apabila kedua *entity* tersebut saling bersinggungan (tubrukan). Misalnya apabila *player* bertubrukan dengan kayu maka kayu akan terbalik dan jika tidak bertubrukan maka kayu akan tetap seperti semula. Untuk melakukan hal tersebut tambahkan *method setFlippedVertical* dengan nilai (*true*) di dalam *method collidesWith*.

3.1.5 NPC menghalangi Karakter

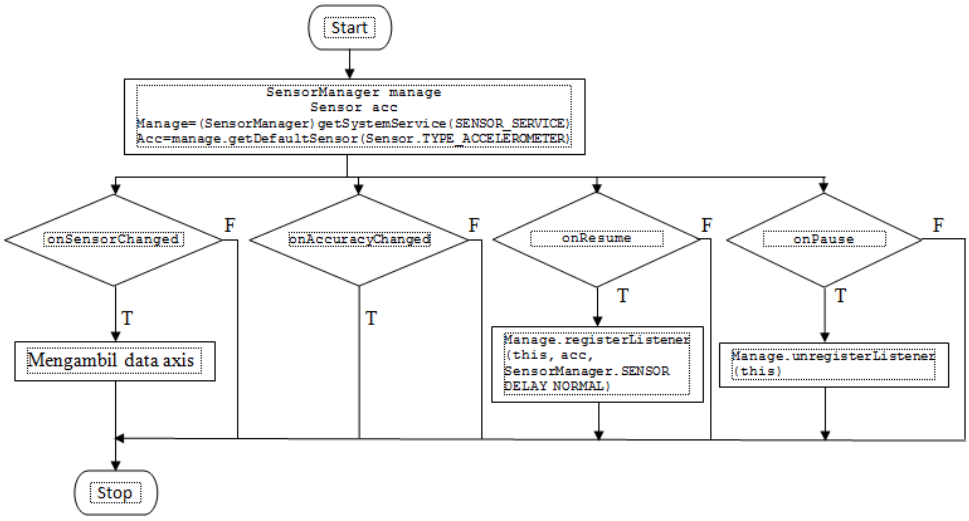
Agar NPC dapat menghalangi atau mendahului karakter, pada *method onUpdate* NPC memerlukan data dari posisi karakter berada. Sehingga NPC dapat memperhitungkan apakah perlu menambah kecepatan atau tidak. Misalkan *enemy* tertinggal di belakang karakter, agar *enemy* dapat mengejar karakter maka diperlukan koordinat posisi *player* berada berupa data x dan data y.

Setelah data koordinat posisi *player* sudah diterima oleh *enemy* dan posisinya berada di depannya, maka *enemy* akan menambahkan kecepatan hingga posisi *player* berada di belakang *enemy*. Akantetapi apabila posisinya di depan *player* makan kecepatannya akan tetap. Sebelumnya *player* dan enemy terlebih dahulu dideklarasikan, lalu diinisialisasi pada *methode onLoadScene*.

3.1.6 Analisis *Accelerometer*

Untuk melakukan pendeteksian perubahan axis dengan sensor *accelerometer*, diperlukan tiga komponen yaitu *SensorManager*, Sensor, dan *SensorEventListener*. Untuk menggunakan suatu sensor, objek *SensorManager* harus dihubungkan terlebih dahulu dengan sistem android yang mengatur bagian sensor. Selain itu, sensor juga harus ditentukan ingin menggunakan sensor *accelerometer*. Untuk bisa mendeteksi sensor, suatu class harus didaftarkan

terlebih dahulu. Setelah didaftarkan, setiap ada perubahan axis, android akan memanggil fungsi *onSensorChanged*. Pada fungsi tersebut proses pengambilan nilai axis dari posisi telepon genggam dilakukan. Ketika sudah tidak menggunakan sensor lagi, terjadi proses penghapusan *class* yang bersangkutan dari daftar. *OnAccuracyChanged* tidak digunakan karena tidak melakukan perubahan *accuracy* pada sensor *accelerometer.*



Gambar 3.17 Pendeteksian axis

Dalam penerapannya, dibutuhkan perhitungan untuk mendefinisikan kecepatan dan percepatan dalam permainan dan *AccelerometerHandler class* untuk penanganan *accelerometer*nya.

1. Perhitungan kecepatan dan Percepatan

Nilai-nilai *accelerometer* terkadang akan mendapatkan nilai yang lebih dari jangkauan besaran *accelerometer*. Hal ini disebabkan karena ketidak akuratan nilai-nilai yang dilaporkan oleh *listener*, sehingga harus melakukan perhitungan nilai-nilainya agar dapat sesuai. Berikut adalah perkiraan besaran percepatan yang akan digunakan dalam pembangunan *game* tradisional egrang :

Tabel 3.2 Perkiraan Besaran Percepatan

No	**Besaran kecepatan (*velocity*)**
1	Ketika sepenuhnya miring ke kiri maka sumbu y akan melaporkan percepatan sebesar -10 m/s2 dan ketika sepenuhnya miring ke kanan maka sumbu y akan melaporkan percepatan sebesar 10 m/s2.
2	Ketika sepenuhnya miring ke atas maka sumbu x akan melaporkan percepatan sebesar -10 m/s2 dan ketika sepenuhnya miring ke bawah maka sumbu x akan melaporkan percepatan sebesar 10 m/s2.
3	Kecepatan untuk gerakan horizontal karakter pemain didefinisikan sebesar 0-20 m/s2.

Semua perhitungan digunakan untuk menormalkan pembacaan *accelerometer* dengan membagi nilai *absolute* maksimum yaitu 10 dan kemudian dikalikan dengan kecepatan maksimum *horizontal*. Dengan demikian, maka akan didapat percepatan sebesar 20 m/s di sumbu-y.

2. Pembacaan Sensor *Accelerometer*

Untuk menggerakkan karakter menggunakan *sensor accelerometer* pada *device*, yaitu inisialisasi variabel pada *class* egrang. Setelah mendeklarasikan *variable*, untuk mendeteksi apakah *device* memiliki sensor *accelerometer*. Pendeteksian dilakukan pada *constructor class egrang* dengan menggunakan *method sensorManager*.

Setelah mendeteksi sensor, selanjutnya yaitu mendapatkan nilai axis dari posisi *device*. Untuk mendapatkan nilai axis dari posisi *device* dengan menambahkan *method onSensorChanged* (*SensorEvent event*). Langkah terakhir yaitu menambahkan *method updateSpritePosition* untuk menghitung posisi karakter ketika *device* dimiringkan, namun dengan batasan-batasan pada nilai axis. Hal ini dimaksudkan agar karakter tidak melewati batas layar pada *device*. Apabila proses perhitungan sudah dilakukan maka proses akan dilakukan dengan pemanggilan perintah *player.setPosition* (sX, sY), dimana nilai sX dan sY adalah nilai dari posisi karakter ketika *device* dimiringkan.

3.1.7 Analisis Kebutuhan Non Fungsional

Analisis kebutuhan non fungsional adalah sebuah langkah dimana seorang pembangun perangkat lunak (*software develover)* menganalisis sumber daya manusia yang akan menggunakan perangkat lunak yang dibangun, perangkat keras dan perangkat dimiliki sesuai dengan kebutuhan, sehingga dapat ditentukan kompabilitas apliasi yang dibangun terhadap sumber daya yang ada.

Setelah melakukan analisis non fungsional, maka dilanjutkan ke langkah berikutnya yaitu menentukan kebutuhan fungsional sistem yang akan dibangun untuk disesuaikan dengan fakta yang ada. Apabila kebutuhan tidak terpenuhi maka sistem yang dibangun tidak akan berjalan dengan baik sesuai yang diharapkan.

Analisis kebutuhan non fungsional yang dilakukan dibagi dalam tiga tahap, yaitu :

1. Analisis perangkat keras (*hardware*).
2. Analisis perangkat lunak (*software*).
3. Analisis pengguna (*user)*.

3.1.7.1 Analisis Kebutuhan Perangkat Keras (*Hardware*)

Berdasarkan analisis spesifikasi perangkat keras minimum yang dibutuhkan untuk membangun aplikasi ini adalah :

1) Processor minimal 2 Ghz
2) Hardisk minimal 40 GB
3) RAM minimal 512 MB
4) Grapic Card minimal 64 MB
5) Mouse & Keyboard

Sedangkan spesifikasi perangkat keras untuk pengguna pada *smartphone* Android minimum yang dibutuhkan adalah :

1) Processor minimal 800 mhz
2) RAM minimal 128 MB
3) Menggunakan layar dengan dimensi 320x480 pixel
4) Adanya perangkat sensor *accelerometer.*

Spesifikasi minimal perangkat keras yang tercantum pada umumnya perangkat *smartphone* yang beredar dipasaran sekarang ini telah memenuhi persyaratan minimum untuk menjalankan aplikasi sehingga tidak memerlukan penambahan.

3.1.7.2 Analisis Perangkat Lunak (*Software*)

Perangkat lunak digunakan dalam sebuah sistem merupakan perintah-perintah yang diberikan kepada perangkat keras agar bisa saling berinteraksi di antara keduanya. Perangkat lunak yang dibutuhkan untuk membangun aplikasi ini adalah sebgai berikut :

1) Sistem Operasi Windows 7
2) JDK versi 5 atau 6.
3) *Java Runtime Edition,* sebagai *platform* untuk menjalankan sistem.
4) *Eclipse Juno*.
5) ADT (*Android Development Tools*).
6) AVD (*Android Virtual Device*).
7) *Andengine* sebagai *library*.

Game ini dirancang menggunakan *Java Eclipse* dengan menggunakan standar untuk aplikasi GUI pada *level* tinggi dan *level* rendah. Jadi aplikasi ini dapat digunakan pada *smartphone* berbasis android dengan minimal versi 2.3 atau biasa dikenal dengan *Gingerbread.* Oleh karena itu dalam pengujianya menggunakan *emulator* android.

3.1.7.3 Analisis Pengguna (*User*)

Pada dasarnya *game* ini diperuntukkan untuk semua kalangan. Namun untuk spesifikasi pengguna atau kriteria minimal yang harus dimiliki oleh pengguna adalah sebagai berikut :

1) Pengetahuan dan Pengalaman Pengguna

Tabel 3.3 Pengetahuan dan Pengalaman Pengguna

Kemampuan	Level
Pengalaman	Menengah – Mahir
Tingkat Edukasi	SD, SMP, SMA/SMK (sederajat), umum
Level Pembaca	Kelas 3 ke atas
Bahasa Asli	Indonesia

2) Psikologi Pengguna

Tabel 3.4 Psikologi Pengguna

Psikologi	Level
Sikap	Positif
Motivasi	Menengah
Kesabaran	Cukup Sabar
Harapan	User tertarik dengan tradisional egrang
Level Stress	Tidak Stress
Kognitif	Kongkrit

3) Fisik Pengguna

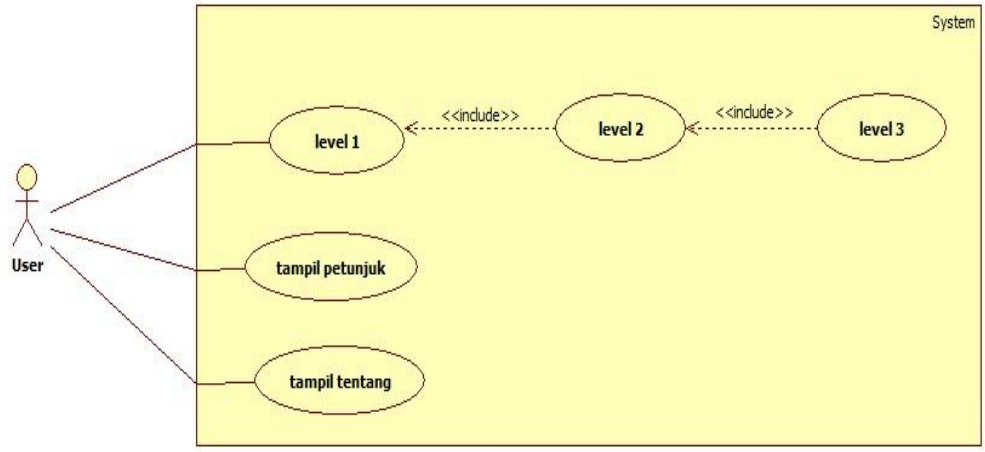
Tabel 3.5 Fisik Pengguna

Fisik	Level
Umur	Anak-anak (8 tahun) ke atas
Gender	Laki-laki dan perempuan
Pegangan Tangan	Tangan kiri dan kanan
Ketidakmampuan	Cacat tangan

3.1.8 Analisis Kebutuhan Fungsional

3.1.8.1 Use Case Diagram

Pemodelan yang digunakan dalam menggambarkan kebutuhan fungsional pada aplikasi yang dibangun yaitu dengan pemodelan Use Case Diagram. Use Case Diagram pada *game* balapan tradisional egrang dapat dilihat pada gambar 3.18.



Gambar 3.18 Use Case Diagram balapan tradisional egrang

3.1.8.2 Identifikasi Aktor

Identifikasi aktor adalah aktor yang berperan dalam menjalankan sistem, aktor yang berperan dapat dilihat pada tabel dibawah ini :

Tabel 3.6 Identifikasi Aktor

No.	Aktor	Deskripsi
A-01	User	Merupakan aktor yang berperan dalam memainkan *game* tradisional egrang.

Gambar 3.18 menjelaskan bahwa aktor sebagai pengguna aplikasi (*frontend)* yang berinteraksi dengan perangkat mobile atau *smartphone* berbasis android.

3.1.8.3 Identifikasi *Use Case*

Identifikasi *Use Case* adalah *Use Case* yang bekerja pada sistem adalah sebagai berikut :

Tabel 3.7 Identifikasi Use Case

No	**Nama use case**	**Deskripsi**
UC-01	Level 1	Proses permainan pada level 1
UC-02	Level 2	Proses permainan pada level 2
UC-03	Level 3	Proses permainan pada level 3
UC-04	Tampil petunjuk	Proses untuk mengetahui petunjuk atau cara bermain *game* tradisional egrang.
UC-05	Tampil tentang	Proses yang menampilkan mengenai pembuat aplikasi

3.1.8.4 Skenario *Use Case*

Skenario *Use Case* adalah alur cerita atau proses-proses yang terjadi pada sistem antara aktor dengan *Use Case*. Skenario *Use Case* ditunjukan pada tabel dibawah ini :

Tabel 3.8 Skenario *Use Case Level* 1

Identifikasi		
Nama Use Case	*Level* 1	
Aktor	Pengguna	
Tujuan	Memulai permainan pada *level* 1	
Deskripsi	Memulai permainan di *level* 1	
Skenario Utama (UC01-A01)		
Kondisi awal	Karakter berada di samping kiri layar	
Aksi Aktor		**Reaksi Sistem**
		1. Karakter diam
		2. Tanda untuk memulai perlombaan

3. Pemain memiringkan *device* atau *accelerometer*	
	4. Karakter bergerak sesuai dengan *accelerometer*
Skenario Alternatif-Autentifikasi Gagal	
Aksi Aktor	**Reaksi Sistem**
	1. Karakter diam
2. Pemain memiringkan *device* atau *accelerometer*	
	3. *Device* tidak memiliki *accelerometer*
Kondisi akhir	Karakter bergerak sesuai dengan arah atau kemiringan *accelerometer*

Tabel 3.9 Skenario *Use Case Level* 2

Identifikasi	
Nama Use Case	*Level* 2
Aktor	Pengguna
Tujuan	Memulai permainan pada *level* 2
Deskripsi	Memulai permainan di *level* 2
Skenario Utama (UC02-A01)	
Kondisi awal	Karakter berada di samping kiri layar
Aksi Aktor	**Reaksi Sistem**
Memilih *level* 2	
	1. Jika *level* 1 belum berhasil, maka pemain tidak dapat melanjutkan ke *level* 2
	2. Jika *level* 1 berhasil dimenangkan, maka pemain dapat melanjutkan ke *level* 2

	3. Karakter diam
	4. Tanda untuk memulai perlombaan
5. Pemain memiringkan *device* atau *accelerometer*	
	6. Karakter bergerak sesuai dengan *accelerometer*
Skenario Alternatif-Autentifikasi Gagal	
Aksi Aktor	**Reaksi Sistem**
	4. Karakter diam
5. Pemain memiringkan *device* atau *accelerometer*	
	6. *Device* tidak memiliki *accelerometer*
Kondisi akhir	Karakter bergerak sesuai dengan arah atau kemiringan *accelerometer*

Tabel 3.10 Skenario *Use Case Level* 3

Identifikasi	
Nama Use Case	*Level* 3
Aktor	Pengguna
Tujuan	Memulai permainan pada *level* 3
Deskripsi	Memulai permainan di *level* 3
Skenario Utama (UC03-A01)	
Kondisi awal	Karakter berada di samping kiri layar
Aksi Aktor	**Reaksi Sistem**
Memilih *level* 2	
	1. Jika *level* 2 belum berhasil, maka pemain tidak dapat melanjutkan ke *level* 3
	2. Jika *level* 2 berhasil dimenangkan,

	maka pemain dapat melanjutkan ke *level* 3
	3. Karakter diam
	4. Tanda untuk memulai perlombaan
5. Pemain memiringkan *device* atau *accelerometer*	
	6. Karakter bergerak sesuai dengan *accelerometer*
Skenario Alternatif-Autentifikasi Gagal	
Aksi Aktor	**Reaksi Sistem**
	1. Karakter diam
2. Pemain memiringkan *device* atau *accelerometer*	
	3. *Device* tidak memiliki *accelerometer*
Kondisi akhir	Karakter bergerak sesuai dengan arah atau kemiringan *accelerometer*

Tabel 3.11 Skenario *Use Case* Petunjuk

Identifikasi	
Nama Use Case	Petunjuk
Aktor	Pengguna
Tujuan	Melihat informasi mengenai petunjuk bermain
Deskripsi	Proses untuk melihat informasi mengenai pentunjuk bermain
Skenario Utama (UC04-A01)	
Kondisi awal	Pengguna berada di menu utama aplikasi *game* tradisional egrang
Aksi Aktor	**Reaksi Sistem**
Memilih menu petunjuk	Aplikasi merespon dengan cara menampilkan tahapan-tahapan petunjuk

	bermain
Kondisi akhir	Aplikasi menampilkan tahapan petunjuk bermain

Tabel 3.12 Skenario *Use Case* Tentang

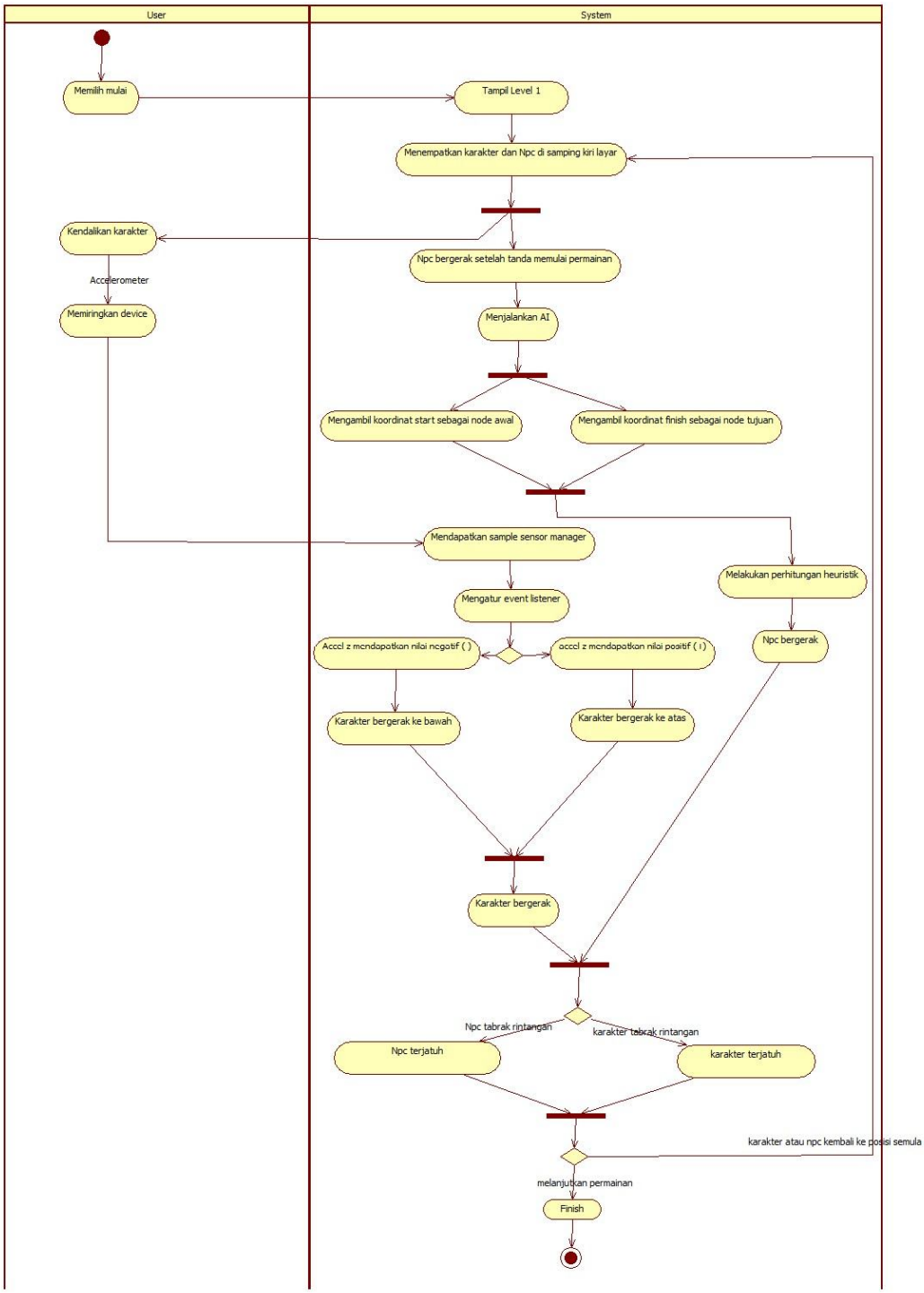
Identifikasi		
Nama Use Case	Tentang	
Aktor	Pengguna	
Tujuan	Menampilkan keterangan tentang aplikasi dan pembuat	
Deskripsi	Untuk menampilkan keterangan tentang aplikasi dan pembuat	
Skenario Utama (UC05-A01)		
Kondisi awal	Pengguna berada di menu utama aplikasi *game* tradisional egrang	
Aksi Aktor		**Reaksi Sistem**
Memilih menu Tentang		Aplikasi merespon dengan cara menampilkan keterangan tentang aplikasi dan pembuat
Kondisi akhir	Aplikasi menampilkan keterangan tentang aplikasi dan pembuat	

3.1.8.5 *Activity Diagram*

Activity diagram adalah teknik untuk mendeskripsikan logika procedural dan alur kerja program. *Activity diagram* mempunyai peran seperti halnya *flowchart* akan tetapi perbedaannya dengan *flowchart* adalah activity diagram bisa mendukung perliaku pararel sedangkan *flowchart* tidak bisa.

1. *Activity Diagram level* 1

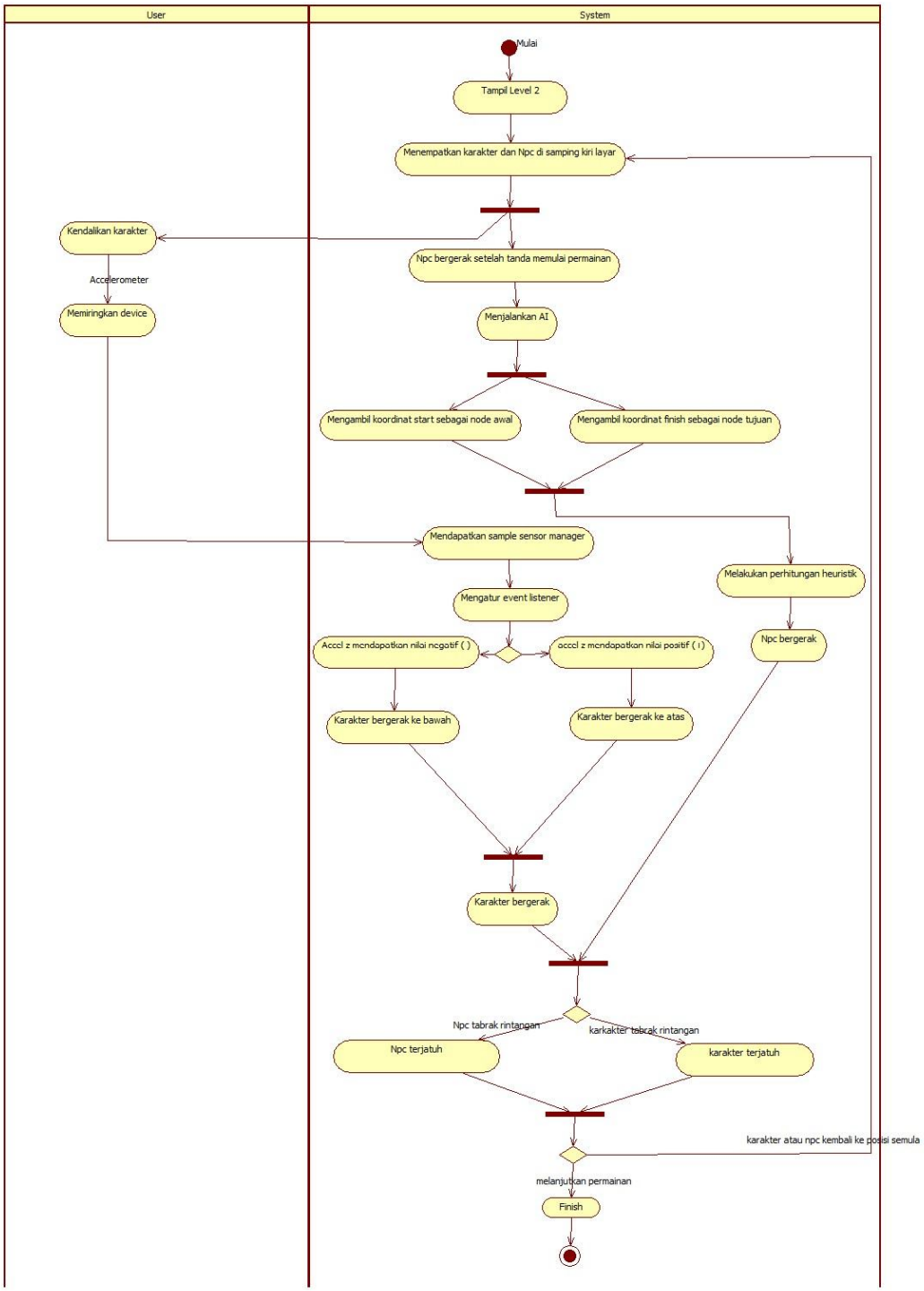
   Aktivitas ini terjadi saat pemain memainkan permainan pada *level* 1. Diagram *activity* Permainan *level* 1 dapat dilihat pada gambar 3.19



Gambar 3.19 *Activity Diagram level 1*

2. *Activity Diagram level* 2

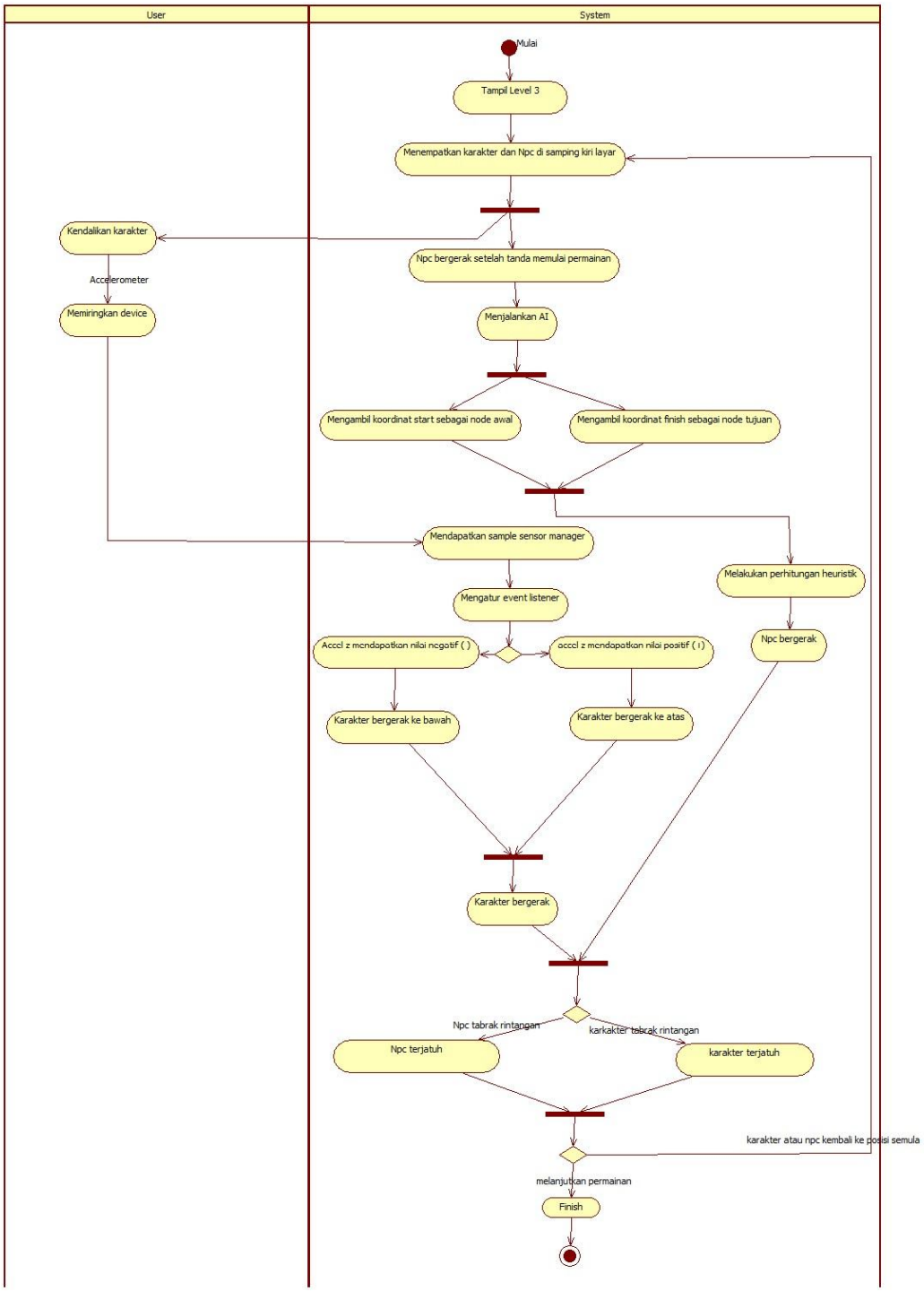
Aktivitas ini terjadi saat pemain memainkan permainan pada *level* 1. Diagram *activity* Permainan *level* 1 dapat dilihat pada gambar 3.20.



Gambar 3.20 *Activity Diagram level* 2

3. *Activity Diagram level* 3

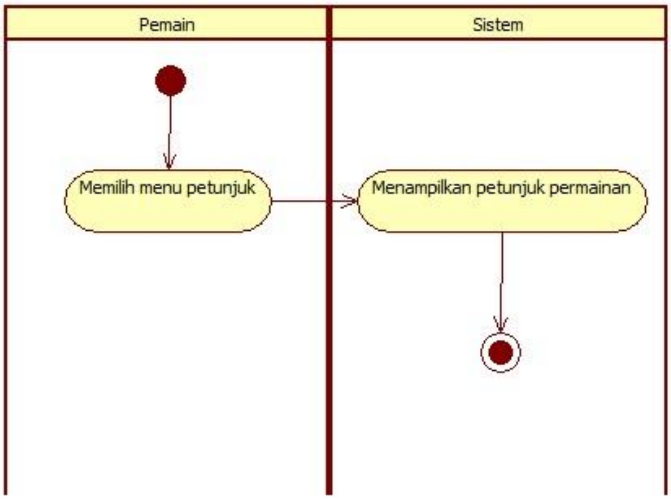
Aktivitas ini terjadi saat pemain memainkan permainan pada *level* 1. Diagram *activity* Permainan *level* 1 dapat dilihat pada gambar 3.21.



Gambar 3.21 *Activity Diagram level* 3

4. *Activity Diagram* petunjuk

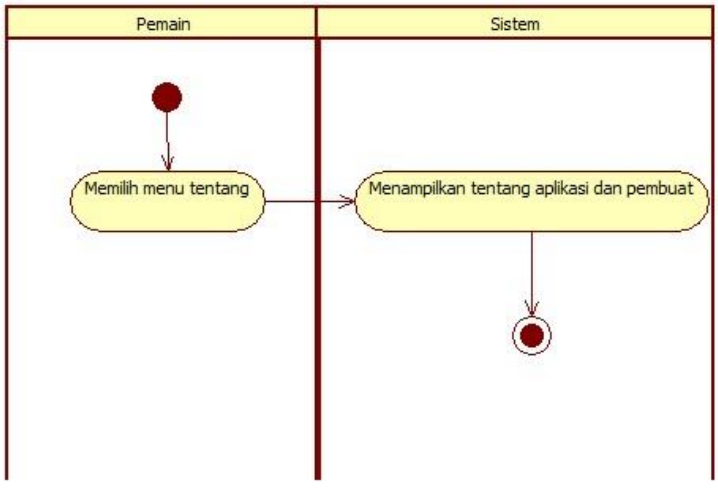
   Aktivitas ini terjadi saat pemain menekan tombol petunjuk pada tampilan menu utama. Diagram *activity* menampilkan petunjuk dapat dilihat pada gambar 3.22



Gambar 3.22 *Activity Diagram* petunjuk

5. *Activity Diagram* tentang

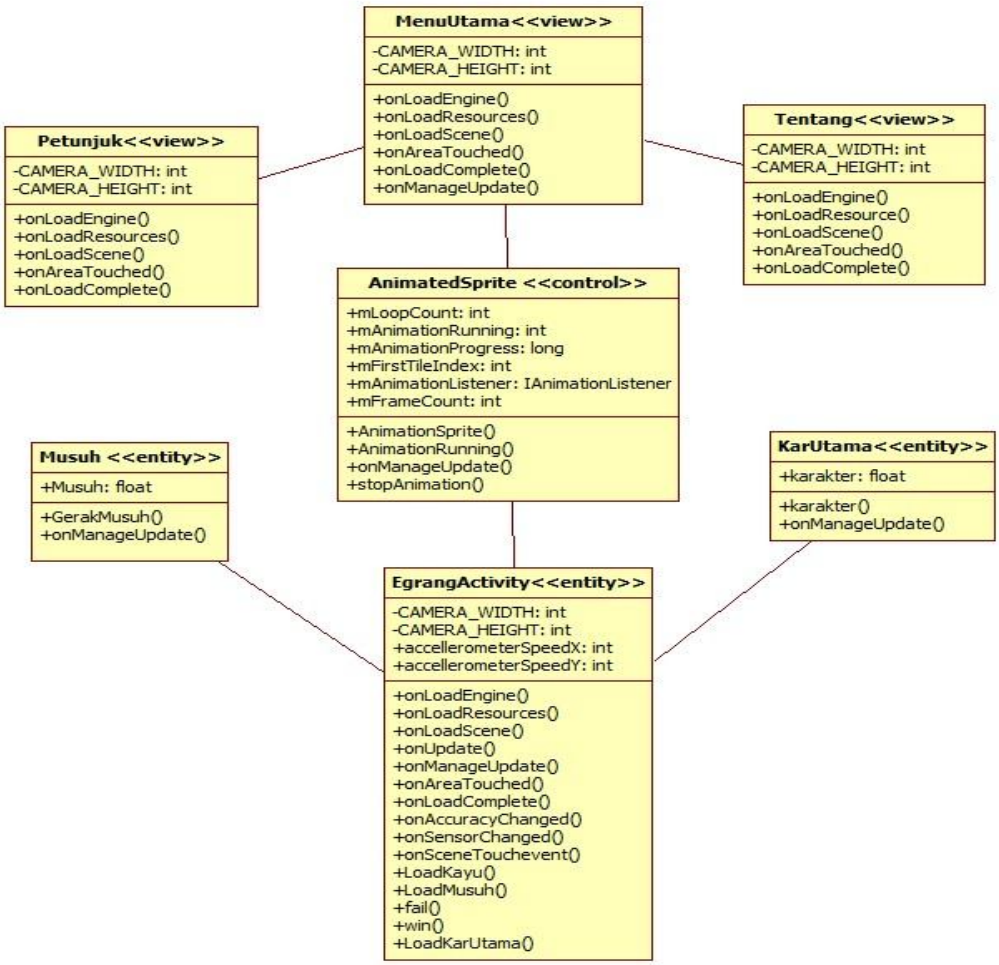
   Aktivitas ini terjadi saat pemain menekan tombol tentang pada tampilan menu utama. Diagram *activity* menampilkan tentang dapat dilihat pada gambar 3.23.



Gambar 3.23 *Activity Diagram* tentang

3.1.8.6 *Class Diagram*

Diagram kelas atau atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang dibuat untuk membangun sistem. Berikut ini adalah perancangan struktur sistem yang terdapat pada *game* tradisional egrang, yang digambarkan dengan *class diagram*.



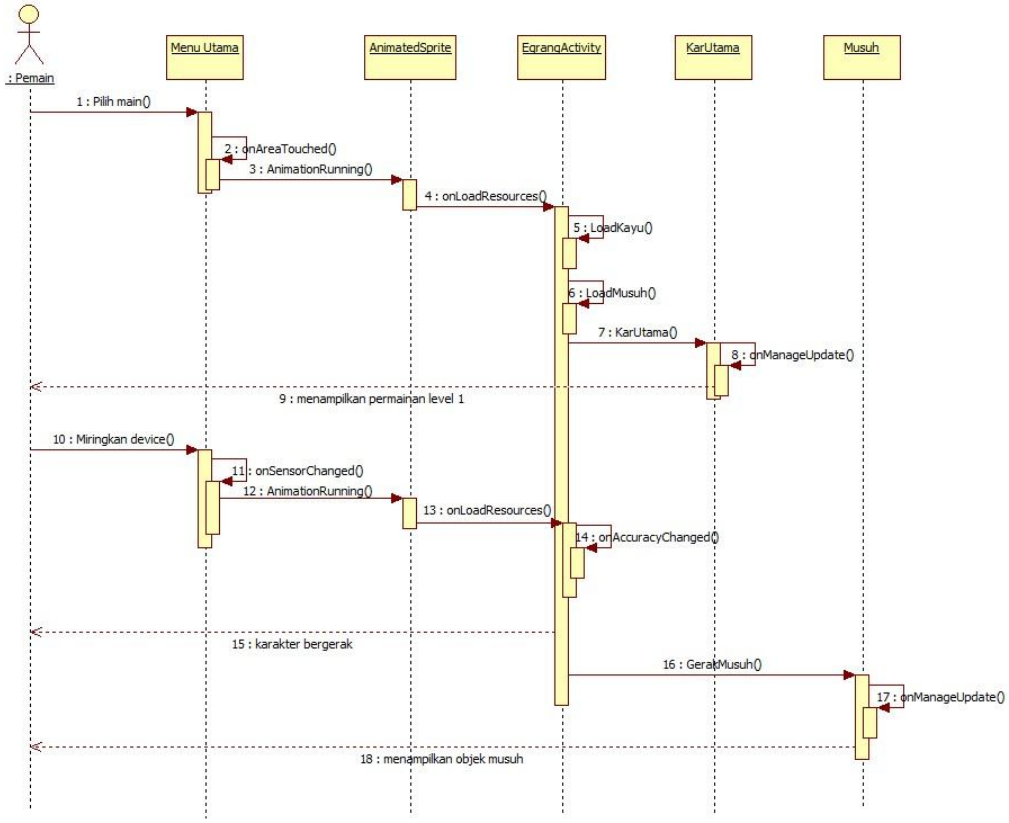
Gambar 3.24 *Class diagram game* tradisional egrang

3.1.8.7 *Sequence Diagram*

Sequence diagram merupakan suatu diagram yang menggambarkan interaksi antar objek dan mengindikasikan komunikasi diantara objek-objek tersebut.

1. *Sequence* Diagram Permainan *Level* 1

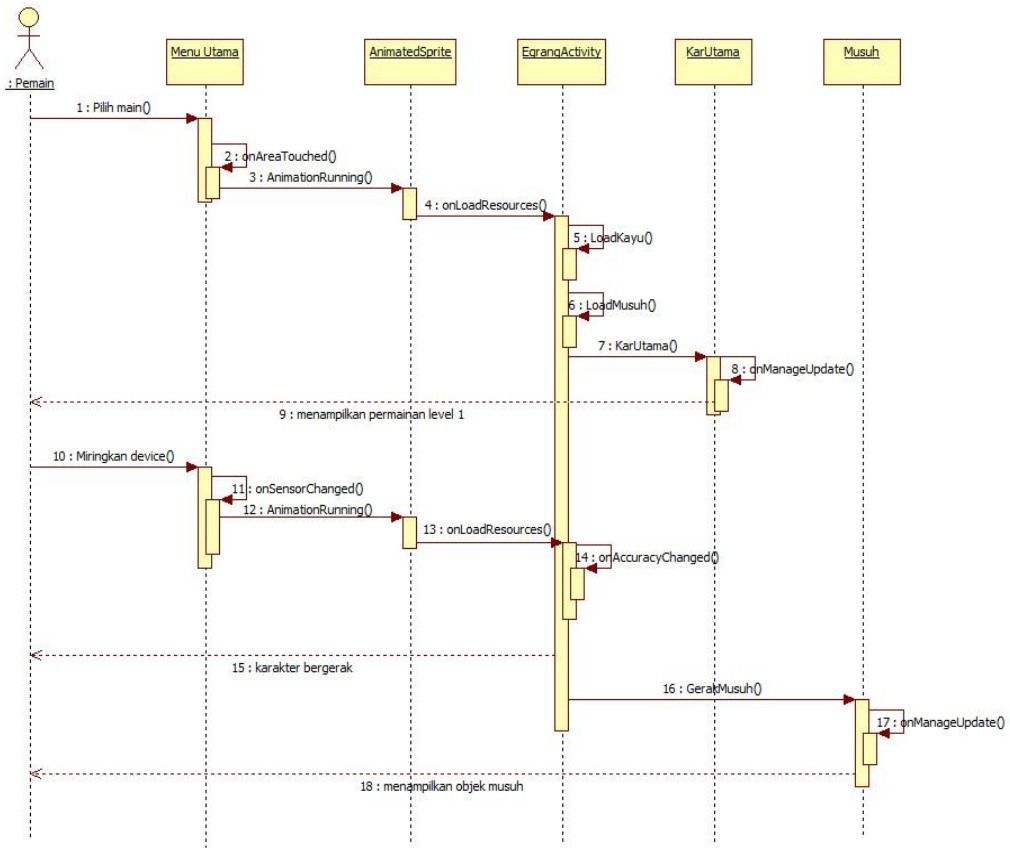
   Sequence diagram permainan *level* 1 dapat dilihat pada gambar 3.25



Gambar 3.25 *Sequence diagram* permainan *level* 1

2. *Sequence Diagram Level* 2

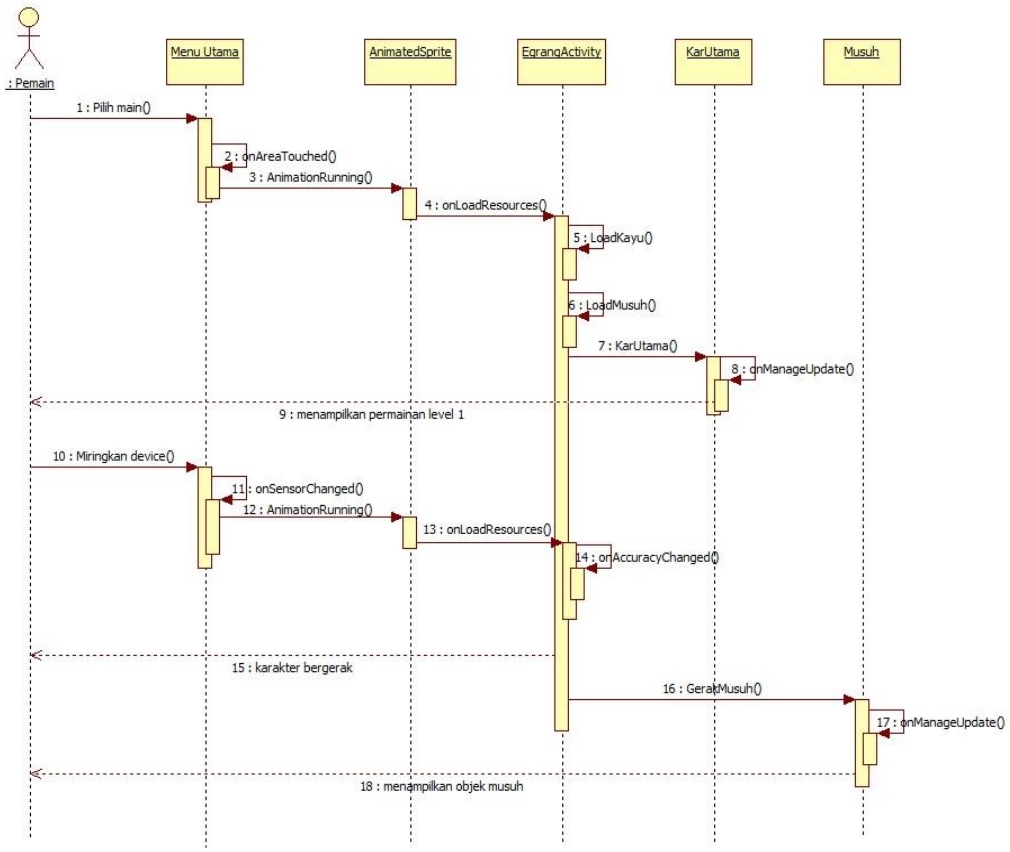
   Sequence diagram permainan *level* 1 dapat dilihat pada gambar 3.26



Gambar 3.26 *Sequence* diagram permainan *level* 2

3. *Sequence Diagram Level* 3

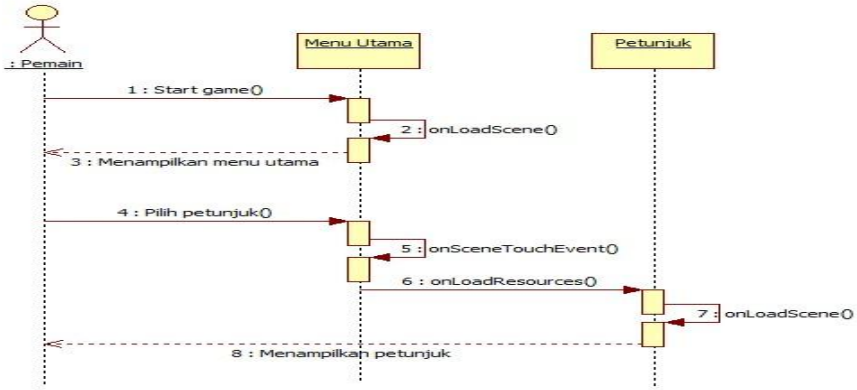
   Sequence diagram permainan *level* 1 dapat dilihat pada gambar 3.27



Gambar 3.27 *Sequence* diagram permainan *level* 3

4. *Sequence diagram* petunjuk

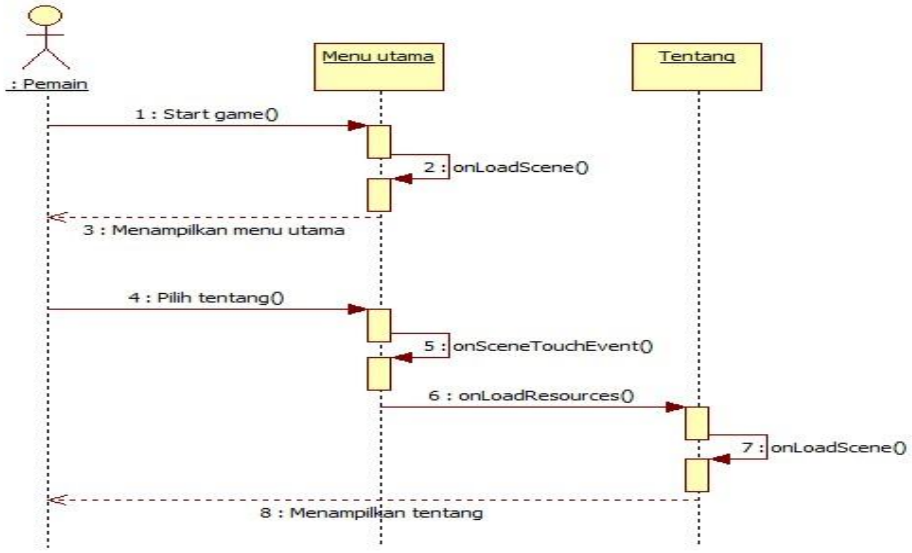
   Sequence diagram menampilkan cara bermain dapat dilihat pada gambar 3.28.



Gambar 3.28 *Sequence diagram* petunjuk

5. *Sequence diagram* tentang

Sequence diagram menampilkan tentang dapat dilihat pada gambar 3.29.



Gambar 3.26 *Sequence Diagram* Tentang

3.2 Perancangan Sistem

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

3.2.1 Perancangan Komponen Permainan

3.2.1.1 Karakter

Pengenalan karakter merupakan pembahasan mengenai karakter-karakter yang terlibat dalam *game* tradisional egrang ini.

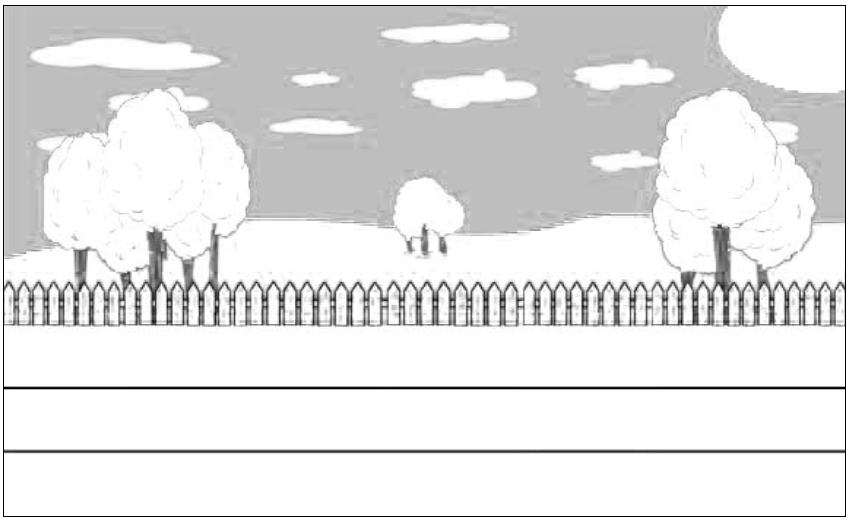
Pengenalan karakter dapat dilihat pada Tabel 3.13

Tabel 3.13 Pengenalan Karakter

No	Nama	Gambar	Keterangan
1	**Eman**		-.Karkter utama - Umur 10 tahun -. Selalu memakai baju merah.
2	**Asep**		-. Tinggal di Rt 01 - Terdapat di *level* 1, pemain harus mengalahkan terlebih dahulu agar dapat ke *level* berikutnya.
3	**Nana**		-. Tinggal di Rt 02 - Terdapat di *level* 2. - Dia akan menghalangi karakter utama agar tidak bisa melewatinya
4	**Dede**		- Tinggal di Rt 03 - Terdapat di *level* 3. - Akan berusaha menghalangi karakter utama agar tidak memenangkan balapan.

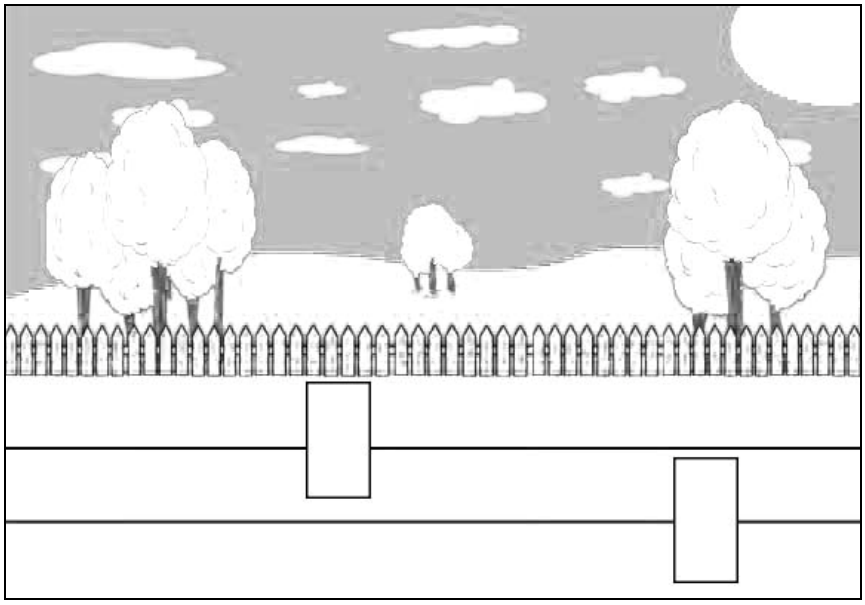
3.2.1.2 *Story Board*

Story board adalah sketsa gambar yang disusun berurutan sesuai dengan naskah. Berikut merupakan *story board* dari *game* tradisional egrang :



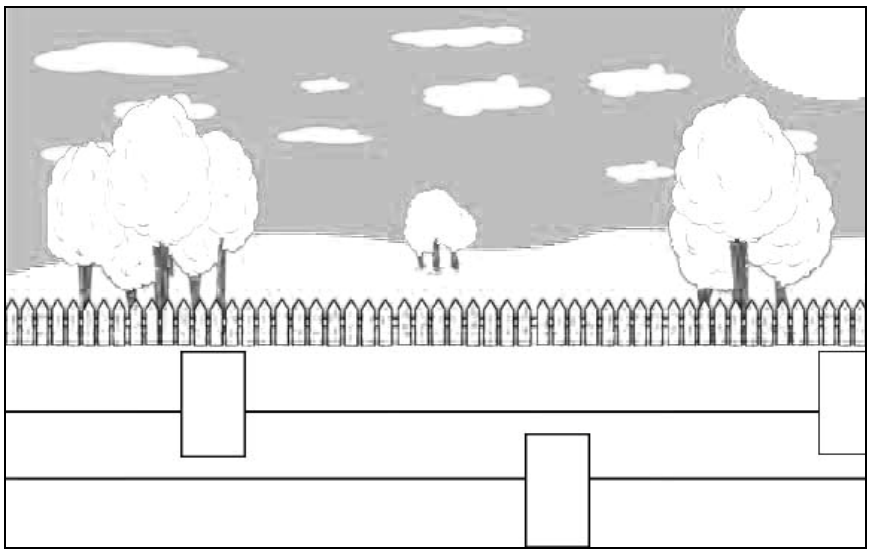
Gambar 3.30 *Story Board Level* 1 Rt 01

Pada Rt 01, pemain akan berlomba atau balapan dengan satu orang lawan dengan kondisi lintasan lurus. Pemain harus memenangkan balapan untuk dapat melanjutkan ke *level* berikutnya. Jika pemain menang maka akan muncul informasi tentang egrang.



Gambar 3.31 *Story Board Level* 2 Rt 02

Pada Rt 02, pemain masih harus berlomba atau balapan dengan satu orang lawan tentu saja harus memenangkan balapan di *level* sebelunya tapi dengan kondisi lintasan terdapat rintangan. Jika pemain menang maka akan muncul informasi tentang egrang.



Gambar 3.32 *Story Board Level* 3 Rt 03

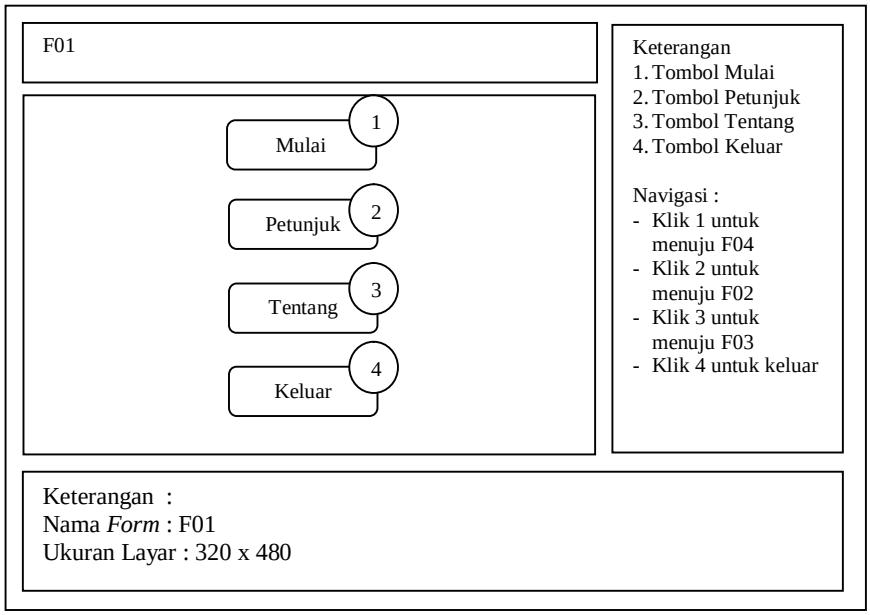
Pada Rt 03, pemain masih harus berlomba atau balapan dengan satu orang lawan tentu saja harus memenangkan balapan di *level* sebelunya tapi dengan kondisi lintasan terdapat rintangan sedikit lebih sulit dari *level* sebelumnya. Jika pemain menang maka akan muncul informasi tentang egrang.

3.2.2 Perancangan Antar Muka

Perancangan antarmuka merupakan sebuah penggambaran, perencanaan, dan pembuatan sketsa atau pengaturan dari beberapa elemen yang terpisah ke dalam satu kesatuan yang utuh dan berfungsi. Adapun perancangan antarmuka *game* tradisional egrang adalah sebagai berikut :

3.2.2.1 Perancangan Antar Muka Menu Utama

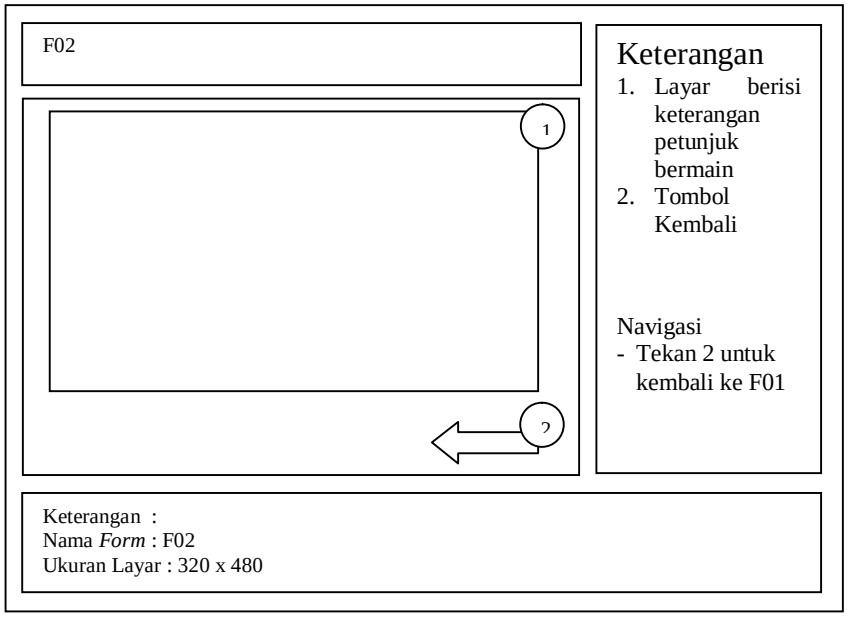
Form menu utama adalah *form* yang digunakan sebagai tampilan awal ketika pemain membuka aplikasi. Berikut adalah tampilan f*orm* menu utama :



Gambar 3. 33 Perancangan Antar Muka Menu Utama

3.2.2.2 Perancangan Antar Muka Petunjuk

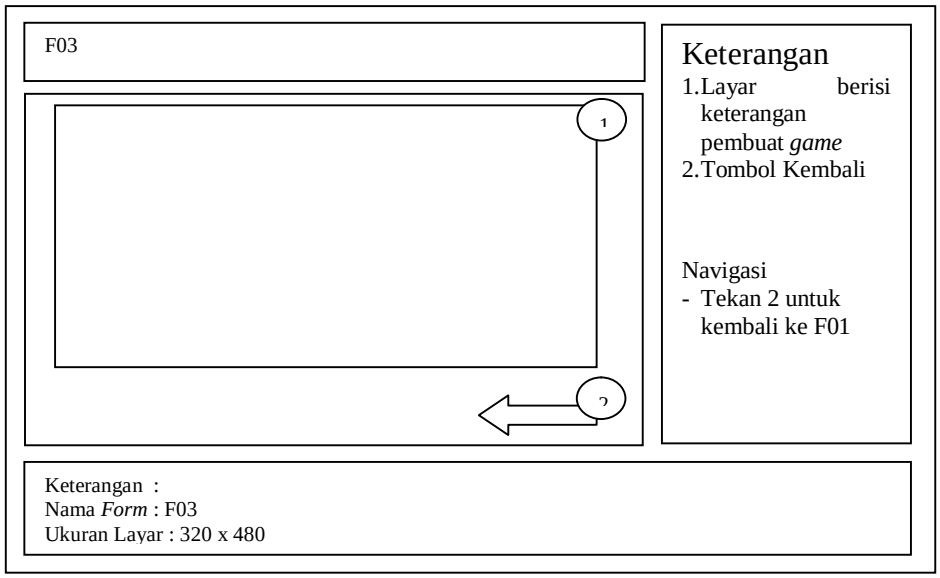
Form petunjuk adalah *form* yang digunakan sebagai tampilan ketika pemain ingin mengetahui petunjuk bermain *game* ini. Berikut adalah tampilan *form* petunjuk :



Gambar 3.34 Perancangan Antar Muka Petunjuk

3.2.2.3 Perancangan Antar Muka Tentang

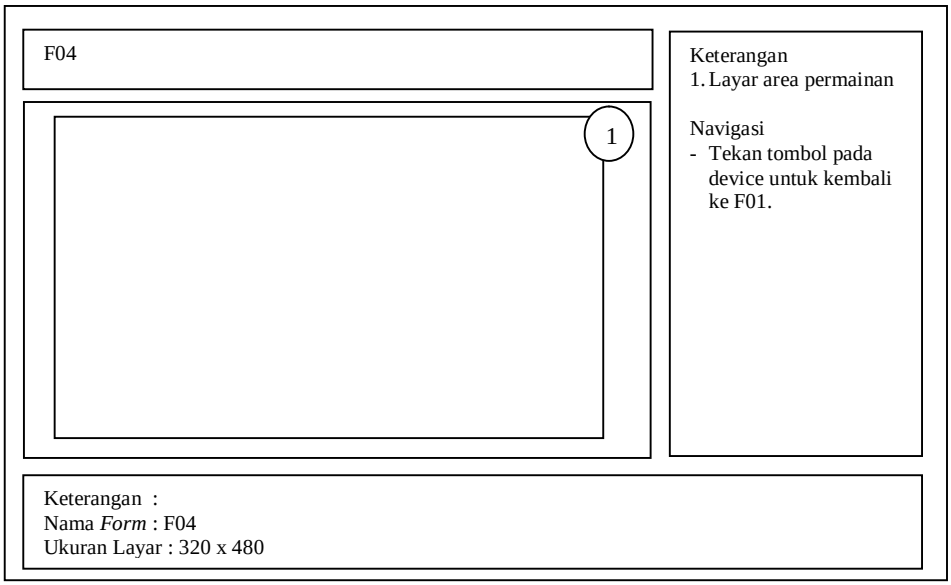
Form tentang adalah *form* yang digunakan sebagai tampilan ketika pemain ingin mengetahui pembuat dari *game* ini. Berikut adalah tampilan *form* tentang :



Gambar 3.35 Perancangan Antar Muka Tentang

3.2.2.4 Perancangan Antar Muka *Level*

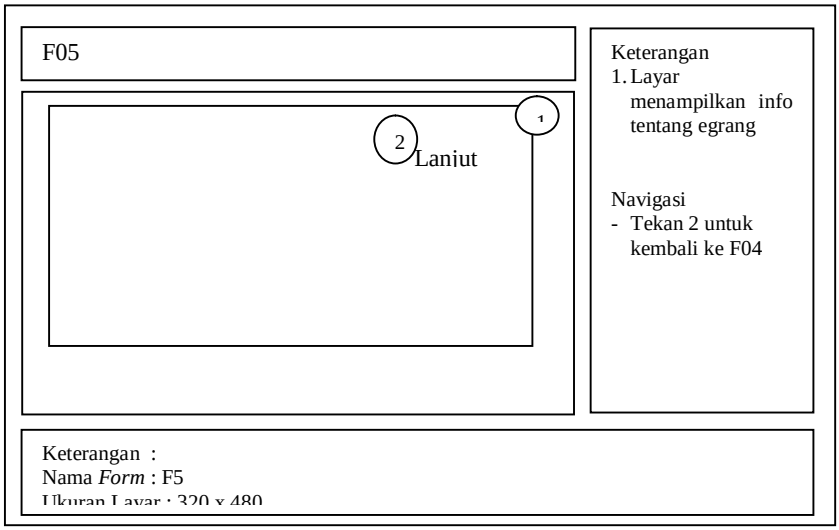
Form level adalah *form* yang digunakan sebagai tampilan area bermain dalam game ini. Berikut adalah tampilan *form level* :



Gambar 3.36 Perancangan Antar Muka *Level*

3.2.2.5 Perancangan Antar Muka Informasi Egrang

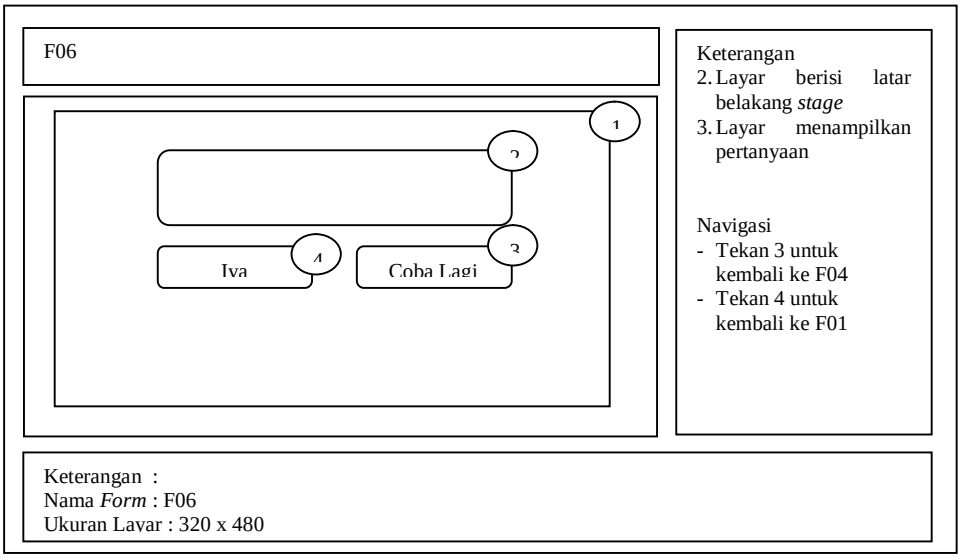
Form informasi egrang adalah *form* yang digunakan sebagai tampilan apabila pemain berhasil memenangkan permainan di salah satu *stage* dalam *game* ini. Berikut adalah tampilan *form* informasi egrang :



Gambar 3.37 Perancangan Antar Muka Informasi Egrang

3.2.2.6 Perancangan Antar Muka Kalah

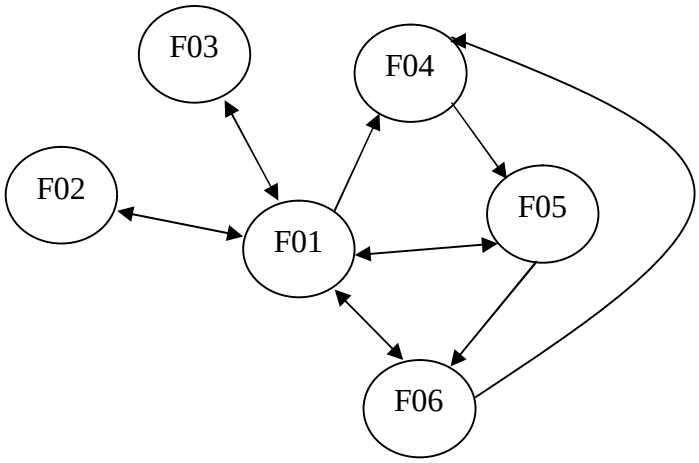
Form kalah adalah *form* yang digunakan sebagai tampilan kalah bermain egrang dan tidak dapat melanjutkan ke stage berikutnya dalam game ini. Berikut adalah tampilan *form* kalah :



Gambar 3.38 Perancangan Antar Muka Kalah

3.2.3 Jaringan Semantik

Perancangan jaringan semantik menggambarkan alur informasi dari setiap antarmuka aplikasi yang dijelaskan pada gambar berikut:



Gambar 3.39 Jaringan Semantik

3.2.4 Perancangan *Method*

Perancangan *method* menjelaskan penjabaran alur proses-proses yang ada pada sistem dan dibangun menggunakan diagram *flowchart*.*Method* yang ada pada aplikasi *game* ini terdiri dari dua jenis *method*, yaitu *method* yang sudah ada di dalam *tools* pembangun aplikasi (hanya perlu dipanggil sesuai kebutuhan) dan *method* yang dirancang berdasarkan kebutuhan aplikasi.

3.2.4.1 Penggunaan *Method* yang Tersedia di Dalam *Tools* Pembangun

Method yang digunakan dari *tools* pembangun aplikasi dalam pembangunan *game Tradisional Kelereng*ini adalah sebagai berikut.

Tabel 3.14 *Method* yang Sudah Ada Dalam *Tools* Pembangun

No	Nama Method	Keterangan
1	onLoadEngine()	Menseting *screen orientation*, *camera* dan resolusi
2	onLoadResources()	Menentukan *resource game*

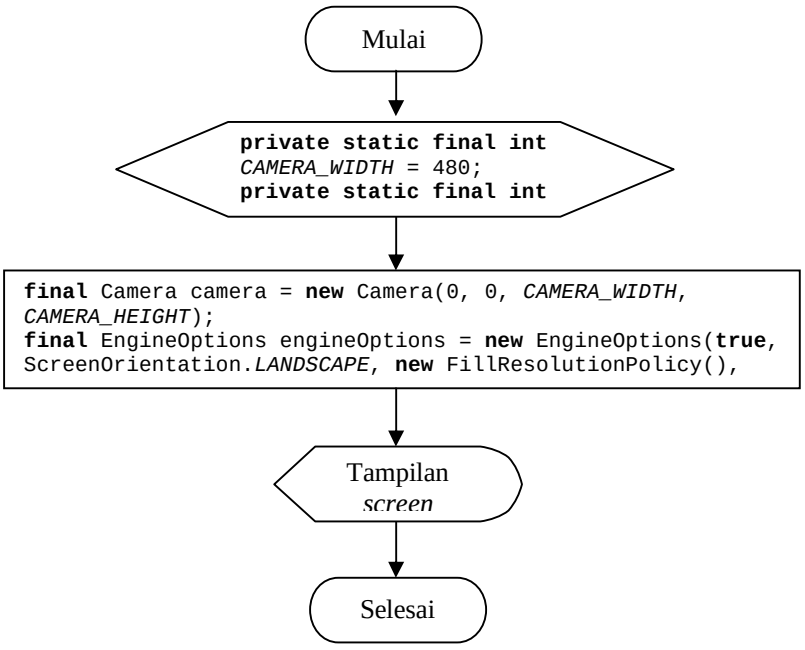
3	onLoadScene()	Menampilkan *sprite* dari objek
4	onLoadComplete()	Mengecek *resource*
5	onManagedUpdate()	Mengupdate objek
6	Remove()	Menghilangkan objek *sprite*
7	setAssetBasePath()	Menentukan *path*didalam folder
8	onCreateDialog()	Menampilkan dialog tertentu
9	restart()	Mengulang aplikasi

3.2.4.2 Perancangan Method yang Akan Digunakan di Dalam Game

Method yang dirancang berdasarkan kebutuhan yang ada pada *game* tradisional egrang ini adalah sebagai berikut.

1. *Method* onLoadEngine

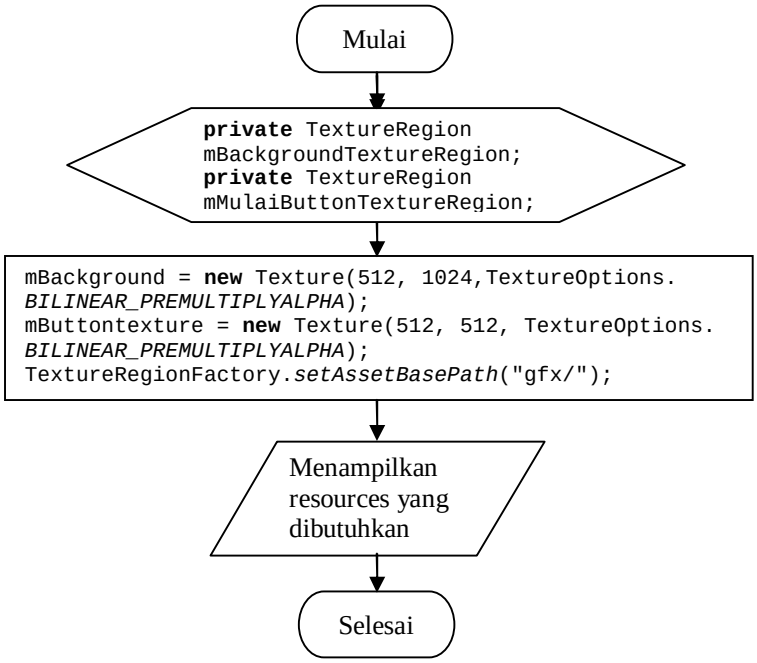
   Method ini digunakan untuk menjalankan fungsi *game* (*gameplay* dan *rules*).



Gambar 3.40 *Method* onLoadEngine

2. *Method* onLoadResources

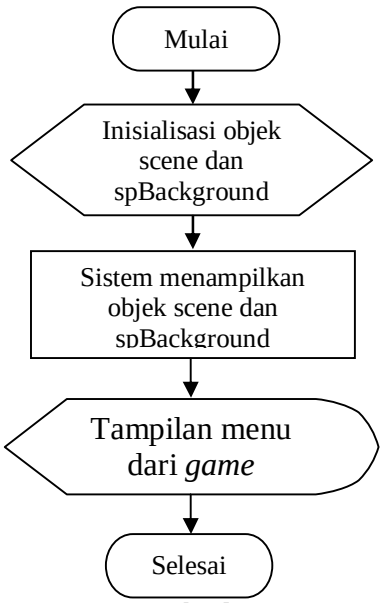
Method ini akan mengambil semua kebutuhan yang akan digunakan dalam *game* dari *database* sistem.



Gambar 3.41 *Method* onLoadResources

3. *Method* onLoadScene

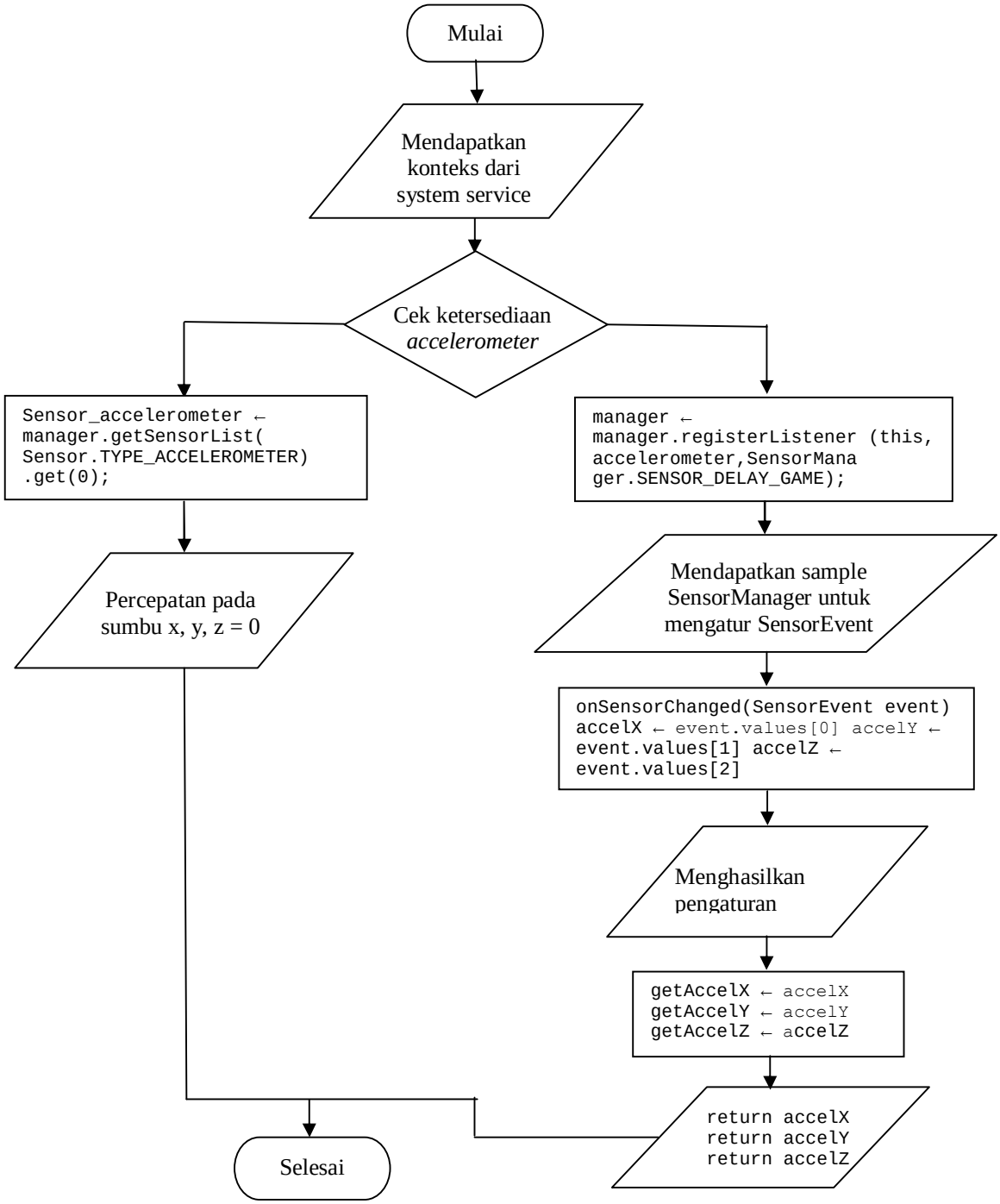
Method ini digunakan untuk menampilkan gambar, grafik, dan animasi yang terdapat pada *game*.



Gambar 3.42 *Method* onLoadScene

4. *Method Accelerometer Handler*

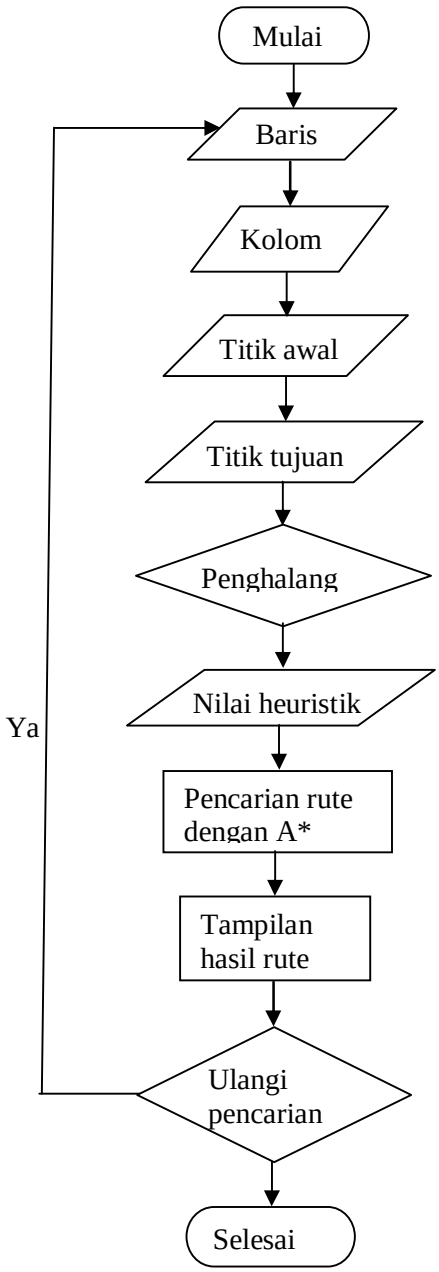
AccelerometerHandler merupakan *class* yang berfungsi untuk menangani sensor *accelerometer*.Untuk lebih jelas, berikut ini merupakan *flowchart* dari *AccelerometerHandler* :



Gambar 3.43 *Method AccelerometerHandler*

5. *Method A Star* (A*)

Method ini ditanamkan pada NPC untuk mencari rute terpendek yang terdapat pada *game*.



Gambar 3.44 *Method A Star* (A*)