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 hingga analisis berbagai kebutuhan baik fungsional maupun non-fungsional.

3.1.1 Analisis Aplikasi Permainan Sejenis

Berdasarkan hal ini diperlukan sebuah analisis yang dilakukan terhadap aplikasi permainan lain yang sejenis untuk menentukan dimana letak kekurangan aplikasi permainan tersebut, sehingga dapat kita kembangkan dengan membuat aplikasi permainan baru agar semakin menarik dibandingkan dengan aplikasi permainan yang dianalisis. Aplikasi permainan sejenis yang akan dianalisis adalah aplikasi permainan yang berjudul *Moon Invasion*, seperti pada gambar 3.1.



Gambar 3.1 *Screenshoot Moon Invasion*[11]

Moon Invasion menawarkan sebuah permainan game ber-*genre area defense.* Pada game tersebut, musuh terdiri dari 3 jenis dengan kekuatan yang berbeda-beda. Namun, mereka masih bergerak dengan pola yang sama. Musuh yang muncul dari bagian atas diharuskan untuk terus maju sampai *goal line* yang berada dibelakang *hero,* ketika musuh tersebut sampai pada *goal line* maka *point bar* yang berwarna kuning akan berkurang sesuai jenis musuh dan seiring

banyaknya musuh yang berhasil melewati *goal line*. Apabila *point bar* telah habis maka permainan diulang namun *live point* berkurang 1. Namun, jika *live point* telah habis maka permainan dinyatakan gagal.

Tugas hero adalah menghalau musuh yang datang bergelombang agar tidak sampai melewati goal line dengan cara menyerang musuh secara vertical saja dan hanya bisa bergerak di daerah tanah saja. Misi terselesaikan jika jumlah tertentu musuh telah terkalahkan dan *point bar* masih tersisa.

Adapun kelebihan dan kekurangan dari game ini adalah sebagai berikut[11]. Berikut kelebihan dari game ini:

a. Cukup sederhana untuk dipelajari dan dimainkan.
b. Tampilan yang bagus sehingga lebih menarik.

Adapun kekurangan dari game ini adalah sebagai berikut:

a. Perulangan cara bermain memberikan rasa bosan setelah beberapa saat.
b. Alur permainan tidak jelas dan masih sederhana.
c. Tidak adanya batasan waktu.
d. Permainan tidak terlalu menantang karena musuh terkesan monton.

3.1.2 Analisis Masalah

Berdasarkan analisis yang dilakukan terhadap game sejenis yang berjudul *Moon Invasion* dapat disimpulkan bahwa dalam game tersebut, dalam hal cara bermain masih terlalu monoton, hal ini dibuktikan dengan alur permainan yang sama untuk level-level selanjutnya dan masih memiliki tujuan akhir yang tidak jelas. Musuh bergerak monton dengan pola yang sama.

3.1.3 Analisis Game yang Akan Dibangun

3.1.3.1 Pengenalan

Clownfish : The Serious One merupakan aplikasi permainan ber-*genre Casual Games* yang akan dibuat. Kata *Clownfish* merupakan representasi untuk karakter utama permainan ini yaitu ikan badut. Sedangkan *The Serious One* merupakan sebuah kiasan ironi untuk ikan badut, karena badut sendiri sering diidentikan dengan pembawaan karakter yang polos dan lucu, namun ini sangat

berbeda sekali dengan karakter asli ikan badut. Keseriusannya dalam menjaga wilayah membuatnya menjadi sangat agresif terutama saat menjaga sarang bertelurnya. Hal ini seolah dua sisi koin yang tak sama.

Aplikasi ini selain sebagai sarana hiburan juga sebagai sarana pembelajaran, dalam hal ini mengangkat tema bermuatan lokal yaitu mengangkat salah satu fauna endemik yang berasal dari Indonesia. Tidak hanya itu, alur permainan game ini mendekati kehidupan asli di habitat aslinya. Adapun kelebihan-kelebihan yang terdapat dalam aplikasi ini adalah sebagai berikut:

a. Mengangkat karakter utama yang merupakan fauna endemik Indonesia
b. Penerapan konsep AI pada musuh di aplikasi permainan tersebut pada tingkatan tertentu
c. Memberikan pengetahuan tentang ekologi Ikan Badut

3.1.3.2 Story Line

Berawal karena eksploitasi yang berlebihan akan kebutuhan eksport ikan hias air laut dan ikan badut kuning menjadi komoditas utama hal ini membuat populasinya di alam bebas sangat sedikit. Si Giru nama karakter utama adalah anggota terakhir dari koloninya yang tersisa. Beruntung, karena dia adalah satu-satunya anggota yang selamat dari obat bius yang ditebar manusia dan yang tersisa dari koloninya hanyalah sarang atau anemon yang terdapat ribuan telur di dalamnya.

Sebagai anggota koloni yang terakhir, tugasnya adalah menjaga telur telurnya sampai mereka menetas. Namun, karena efek yang ditimbulkan dari ulah manusia tersebut membuat ekologi sistem menjadi timpang. Akibatnya banyak ikan lain yang kelaparan dan berusaha memangsa telur yang dilindunginya. Belum lagi pemangsa yang mengincar Si Giru. Hal ini memaksa Si Giru bekerja lebih sulit untuk menghalau ikan yang akan memangsa telur dan mensiasati predator yang akan memangsanya. Maka, di sini pemain harus membantu Si Giru menyelesaikan tugasnya yaitu menjaga telur-telur dalam sarang sampai menetas.

Berikut alur dan misi aplikasi permainan ini, yaitu:

1. Musuh yang memangsa telur akan bergerak dari titik awal (munculnya musuh) ke titik tujuan (tempat telur berada atau anemon).
2. Pemain mengendalikan pergerakan *hero* untuk menghalau musuh yang memangsa telur.
3. Selama hero bergerak *energy point* (stamina) akan berkurang. Jika *energy point menjadi* 0, maka *life point* (kesempatan bermain) berkurang.
4. Hero diharuskan memakan plankton untuk menambah *energy point.*
5. Jika *hero* berhasil memakan musuh yang memangsa telur, maka skor permainan akan bertambah dan musuh akan lenyap.
6. Jika musuh yang memangsa telur berhasil mencapai sarang, maka *health point* (jumlah telur) akan berkurang.
7. Pada level tertentu selain musuh yang memangsa telur akan ada predator yang memangsa *hero* dan ada NPC (*Non-Player Characters*) yang mengusir predator tersebut atau hanya predator yang memangsa *hero*.
8. Jika predator berhasil memangsa *hero* atau *hero* bersinggungan dengan NPC, maka *life point* akan berkurang.
9. Jika *life point* menjadi 0 atau Jika *health point* menjadi 0, maka permainan dinyatakan gagal.
10. Jika pemain berhasil bertahan selama 60 detik sebelum *life point* dan *health point* menjadi 0, maka permaianan dapat dilanjutkan ke level berikutnya atau permainan dinyatakan selesai.

3.1.3.3 Analisis Tingkat Kesulitan

Secara keseluruhan, pemain dapat memilih dua tingkat kesulitan yang berbeda, yaitu tingkat yang “mudah” tentunya disesuaikan agar pemain dapat menyelesaikan permainan, atau dapat memilih tingkat yang “sulit” sehingga permainan akan terasa sulit untuk diselesaikan.

Berdasarkan tabel 3.1 tingkat kesulitan tersebut lebih ditekankan pada perbedaan jumlah musuh yang dibangkitkan secara konstan dan kecepatan bergerak musuh pada setiap *level*. Pada tingkat kesulitan tersebut, jumlah musuh yang harus dihadang oleh pemain adalah sesuai dengan tabel 3.1 yang secara

konstan musuh tersebut akan dibangkitkan kembali apabila pemain telah mengalahkan musuh tersebut atau musuh telah mencapai sarang namun dengan posisi kemunculan yang berbeda dari sebelumnya. Musuh bergerak dengan kecepatan yang relative lambat tetapi tidak dalam kecepatan yang sama untuk masing-masing musuh. Sementara NPC khusus merupakan NPC yang diimplementasikan pencarian jalur terpendek ke tujuannya. NPC pertama adalah predator yang akan memangsa *hero* dan NPC kedua adalah yang akan membantu *hero* dengan mengejar predator.

Tabel 3.1. Analisis tingkat kesulitan

Level	**Tingkat kesulitan**	**Musuh yang dibangkitkan tiap gelombang**	**Banyaknya gelombang dalam 60 detik**	**Kecepatan musuh (pengali)**	**NPC khusus**
2	Mudah	15	10	30-75	-
	Sulit	20	11	50-100	-
3	Mudah	15	10	30-75	1
	Sulit	20	11	50-100	1
4	Mudah	15	10	30-75	2
	Sulit	20	11	50-100	2

3.1.3.4 Gameplay

Permainan ini membutuhkan konsentrasi agar dapat menyelesaikan permainan, yang secara langsung akan melatih kemampuan motorik, koordinasi, kecekatan dan kecermatan pemain. Selain itu, jalan cerita atau *story line* yang disuguhkan sebagian besar merupakan fakta unik yang terjadi di alam bebas sehingga setidaknya memberikan pengetahuan kepada pemainnya.

Saat aktivitasnya melindungi telur pemain harus memakan plankton yang menjadi sumber energinya yang mengapung mengikuti arus. Pemain harus melindungi telur yang berada di sarang dari ikan lain yang mengincar telur. Selain

itu, pemain juga harus tetap waspada terdahap predator yang akan memangsanya jika pemain terlalu jauh dari meninggalkan anemon atau sarangnya.

Permainan terselesaikan apabila pemain dapat menyelesaikan keempat level permainan. Permainan gagal apabila pemain tidak bisa mempertahankan jumlah telur dalam jangka watktu tertentu atau pemain kehabisan kesempatan bermain (*life*) yang dapat disebabkan karena kehabisan energi atau dimangsa predator pada setiap levelnya.

3.1.3.5 Scoring

Penilaian yang dilakukan untuk mengukur sejauh mana ketangkasan pemain dalam menyelesaikan permainan ini adalah dengan melihat *score* atau poin yang telah terkumpul selama memainkan permainan. Poin yang didapat tersebut merupakan poin gabungan dari poin *energy player* ditambah dengan poin dari telur yang tersisa. Dimana perhitungan untuk score diperoleh dari poin telur dikali 2 ditambah dengan poin energy yang tersisa, jika pon yang telah terkumpul telah mencapai ketentuan tertentu maka pemain berhak mendapatkan *life* poin tambahan.

3.1.4 Analisis Algoritma

Penggunaan algoritma pada penelitian ini akan diterapkan dalam pencarian jalan terpendek *(pathfinding)*. Pencarian jalan terpendek pada game ini akan diasumsikan dengan titik awal (keadaan pertama) dan titik tujuan (keadaan akhir) yang diimplementasikan pada pencarian target dalam hal ini pencarian karakter utama dan sarang oleh predator masing-masing. Hasil pencarian algoritma tersebut akan menghasilkan jalan terpendek dari suatu titik awal ke titik tujuannya

3.1.4.1 Analisis Pendekatan Algoritma SMA*

Salah satu masalah dari algoritma A* yaitu menggunakan *memory* yang besar untuk menyimpan simpul (*node*) sebelumnya atau *path* (jalan) sebelumnya [6]. Karena dalam prosesnya metode ini akan mengunjungi setiap node berdasarkan urutan yang dihasilkan dari estimasi heuristic. Karena representasi ruang yang akan dibangun bukan merupakan kotak (*puzzle*) dengan ordo yang

relative kecil namun dengan ordo yang relative besar sehingga simpul yang dibangkitkan dihawatirkan akan menghasbiskan *memory*. Maka, perlu mensiasati hal ini. Salah satunya dengan menggunakan algoritma *Simplified Memory-Bounded A* (SMA*)*.

Algoritma *SMA** merupakan penggabungan dari *greedy search* yang meminimalisir perkiraan pencarian harga dan *uniform cost search* yang meminimalisir harga sampai selesai [6]. SMA* ini dikembangkan karena algoritma A* menggunakan banyak *memory* sehingga menghabiskan *memory* untuk pencarian. Algoritma ini menjalankan *best first search* selama *memory* masih tersedia, apabila memory penuh maka *node* dengan nilai terburuk dibuang, namun nilai terbaik disimpan pada *node* atasnya. Jika ruang *memory* mencukupi untuk semua *node* pada *tree* dalam jalur pencarian, maka *repeated states* tidak akan diulang sehingga pencarian akan menjadi optimal.

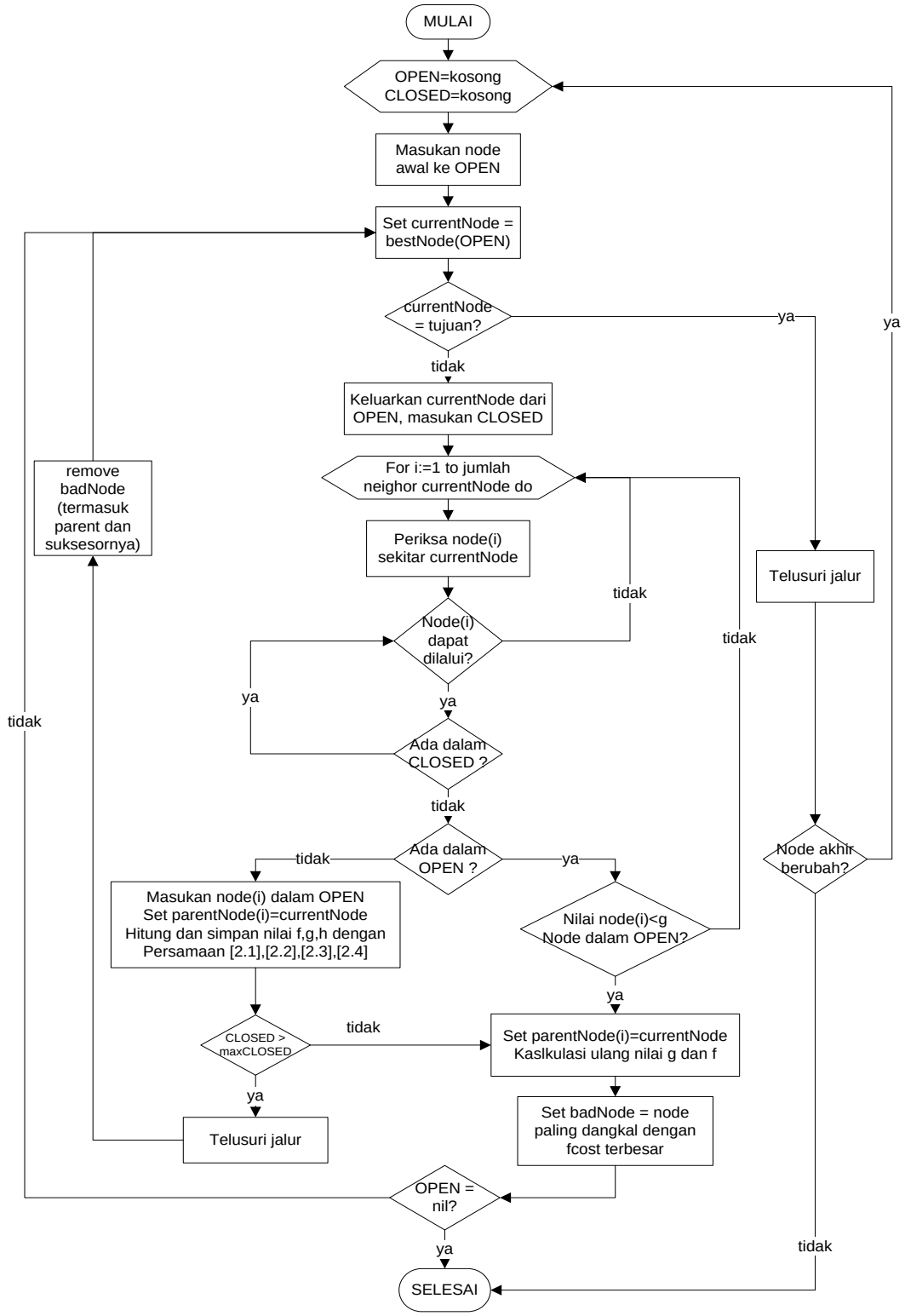
Algoritma SMA* tidak jauh berbeda karena merupakan pengembangan dari algoritma A*. Namun ada aturan-aturan SMA* yang harus diperhatikan:

1. Jika mempunyai lebih dari satu simpul, maka pilih salah satu yang mempunyai *f-cost* terkecil. Jangan hapus dulu simpul tersebut sebelum mengeceknya terlebih dahulu, karena bisa jadi simpul tersebut akan dipakai dulu oleh simpul yang lain.
2. Jika hasil *f-cost* dari suksesor baru yang telah dibangkitkan hasilnya lebih kecil, maka suksesor-suksesor sebelumnya dihapus. Tetapi jika lebih besar, maka lanjutkan dulu pencarian ke suksesor yang lebih kecil, sebelum menghapus suksesor yang lebih besar tersebut.
3. Jika menemukan *stat*e di level yang lebih dangkal, dan mmpunyai *f-cost* lebih besar, maka state tersebut juga harus dihapus.
4. Data di *CLOSED* dibatasi jumlahnya. Sehingga apabila telah melampaui jumlah tersebut, lakukan *backtrack* hingga list paling akhir kemudian jadikan simpul tersebut sebagai awal simpul dan lakukan kembali pencarian .

Berdasarkan pendekatan dari algoritma ini tentunya akan diterapkan kepada (*Non-Player Characters*) NPC yang ada di dalam game yang sesuai

dengan fungsinya sebagai *pathfinding*. Pada gambar 3.2 menjelaskan cara kerja algoritma SMA*. Algoritma ini bekerja dari titik awal ke titik tujuan dengan mencari jalan dengan dengan akumulasi bobot terendah (g) dijumlah dengan perkiraan jalan terendah bobotnya untuk mencapai tujuan (h). Rute yang memiliki node-node dengan nilai f(n) = g(n) + h(n) yang paling kecil akan dikembangkan lebih dulu, karena itu proses pemberian bobot pada node-node sangatlah penting dalam menentukan hasil pencarian.

Proses mencari node dengan nilai terkecil ini dilakukan berulang-ulang sampai menemukan node tujuan. Apabila node yang menjadi titik tujuan telah ditemukan, program akan melakukan backtrack ke parent dari tiap node untuk mendapatkan rangkaian node yang membentuk rute yang diharapkan memberikan hasil yang paling optimum. Namun karena dilakukan pembatasan maka hasil pencarian tidak menjamin memberikan hasil yang selalu optimal.

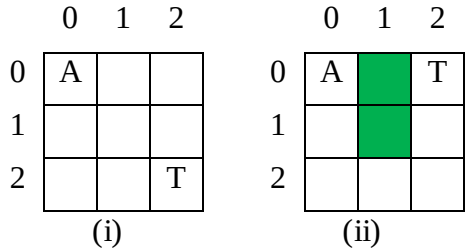


Gambar 3.2 *Flowchart* implementasi algoritma SMA*

3.1.4.2 Analisis Implementasi Pemecahan Masalah *Pathfinding*

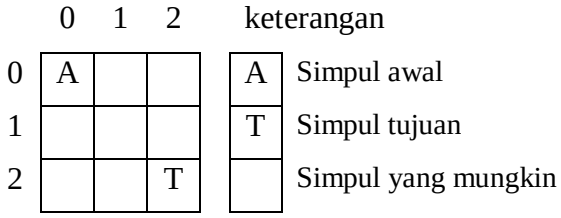
Analisis implementasi perhitungan dengan pendekatan SMA* pada *pathfinding* ini akan dijelaskan dengan ordo 3x3 dikarenakan perhitungan pada ordo berapa pun akan sama. Pada penentuan bobot setiap *node* akan diberikan nilai sesuai dengan jarak terdekat ke tujuan, misal *node* yang terjauh dari tujuan maka diberi bobot yang kecil sedangkan *node* yang terdekat dengan tujuan diberi bobot yang lebih besar. Penghematan penggunan *memory* dilakukan dengan cara membatasi jumlah data yang ada di *closed,* missal dengan pembatasan maksimum untuk *closed* sebanyak 3 data.

Berikut pada Gambar 3.3 merepresentasikan secara garis besar kemungkinan kondisi yang akan terjadi. Kotak berwarna hijau merepresentasikan objek yang tidak bisa dilewati atau penghalang, sementara kotak yang berwarna putih merepresentasikan jalur yang mungkin ditempuh. Misalkan bobot setiap simpul bernilai 1 dan bobot selanjutnya ditambah 1.



Gambar 3.3 Contoh kondisi ruang map pencarian jalan terpendek

Berikut pembahasan untuk mencari jarak terpendek dengan menggunakan formula [2.1], [2.2], [2,3] dan [2,4] yang telah dibahas pada pembahasan sebelumnya. Pada kondisi (i), kita akan mencari jalan terpendek dari posisi titik awal (A) ke posisi titik tujuan (T). 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. Kondisi ini serupa dengan kejadian dimana predator akan memangsa *hero* pada *level* 3 dan NPC yang membantu *hero* dalam menghalau predator yang memangsa *hero* pada *level* 4.



Gambar 3.4 Contoh kondisi ruang map tanpa penghalang

Perhitungan *pathfinding* dengan kondisi tanpa penghalang terlihat pada gambar 3.4 di atas dengan tiga kemungkinan node yang akan diambil yaitu: n(1,1), n(0,1) dan n(1,0) yang akan diuraikan sebagai berikut:

Posisi simpul awal = Ax : 0, Ay : 0

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

Langkah ke satu :

a. Simpul yang akan diambil n(1,1) sehingga kemungkinan simpul yang akan ditempuh adalah g(1,1) dengan perkiraan biaya diambil sebesar 1.

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

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

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

= -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. Simpul yang akan diambil n(1,0) sehingga kemungkinan simpul yang akan ditempuh adalah g(1,0) dengan perkiraan biaya diambil sebesar 1.

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

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

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

= -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. Simpul yang akan diambil n(0,1) sehingga kemungkinan simpul yang akan ditempuh adalah g(0,1) dengan perkiraan biaya diambil sebesar 1.

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

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

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

= -3

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

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

= -3 + 9

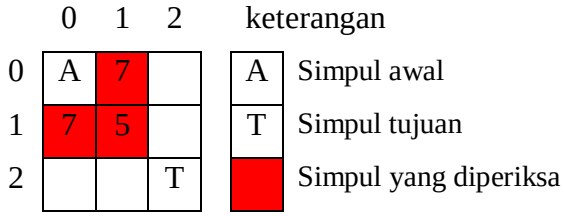
= 6

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

= 1 + 6

= 7

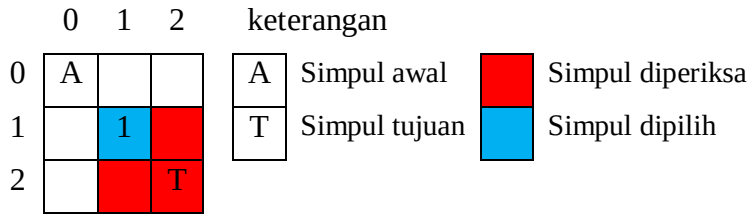
Pada gambar 3.5 di bawah terdapat tiga simpul yang mungkin menjadi *BestNode* yaitu (1,0) dengan f(n)=7, (1,1) dengan f(n)=5 dan (0,1) dengan f(n)=7. Dari ke tiga simpul yang maka dipilihlah simpul (1,1) dengan biaya terkecil yaitu 5 sebagai *BestNode*.



Gambar 3.5 Langkah pertama pencarian *BestNode* pada kondisi (i)

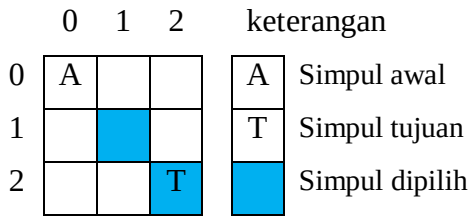
Perhitungan setelah langkah pertama menghasilkan simpul (1,1) sebagai *BestNode*, dari simpul ini ke simpul tujuan memiliki tiga kemungkinan node yang akan diambil yaitu: n(2,2), n(2,1) dan n(1,2) yang akan diuraikan sebagai berikut sebagai langkah selanjutnya.

Pada langkah ke dua, untuk simpul yang akan diambil n(2,2) sehingga kemungkinan simpul yang akan ditempuh adalah g(2,2) dengan perkiraan biaya diambil sebesar 2. Ternyata simpul ini dikenali sebagai simpul tujuan yaitu (2,2) berarti solusi telah ditemukan.



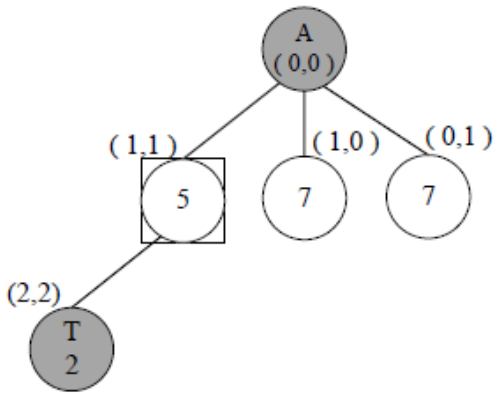
Gambar 3.6 Langkah kedua pencarian *BestNode* pada kondisi (i)

Dari semua perhitungan yang telah dilakukan dipilihlah biaya/*cost* terkecil pada setiap langkahnya sehingga akan menghasilkan jalur terpendek yang terlihat pada gambar 3.7.



Gambar 3.7 Hasil pencarian jalur pada kondisi (i)

Dari perhitungan di atas dipilihlah biaya atau *cost* terkecil pada setiap langkahnya sehingga akan menghasilkan jalur terpendek. Contoh kondisi ruang *map* di atas bila digambarkan dengan pohon pelacakan, maka pohonnya dapat dilihat pada gambar 3.8 dibawah:

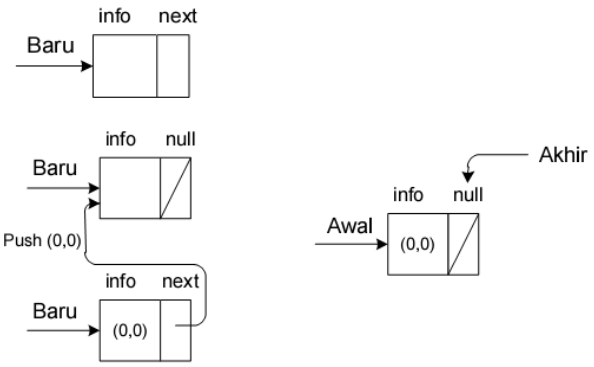


Gambar 3.8 Tree untuk masalah pada gambar 3.7

Pada gambar 3.8 terdapat simpul dengan tanda kotak yang menyatakan bahwa simpul tersebut berada di *closed* atau pernah terpilih sebagai *BestNode*, sedangkan simpul tanpa tanda kotak menyatakan simpul yang berada di *open*. Jika simpul sudah pada kondisi *closed* maka diabaikan, karena simpul ini sudah diuji.

Pada kondisi (i) pada gambar 3.3 atau penjelasan gambar 3.8 diatas dapat dijelaskan langkah-langkah secara struktur datanya dengan *linked list* sebagai berikut:

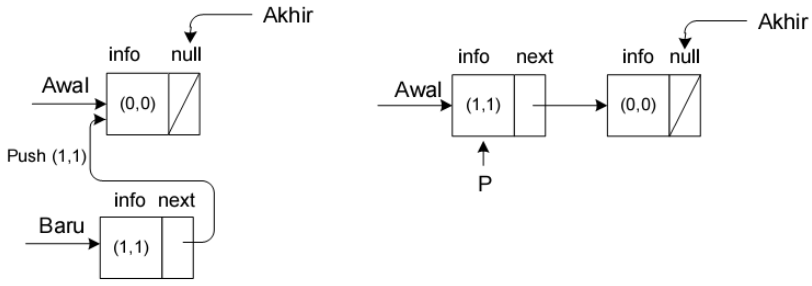
1. Langkah pertama, karena di *open* hanya terdapat satu simpul yaitu A(0,0), maka A terpilih sebagai *BestNode* dan dipindahkan ke *closed* seperti pada gambar 3.9.



Gambar 3.9 Penambahan simpul (0,0)

Kemudian dibangkitkan semua suksesor A(0,0), yaitu simpul (1,1), (1,0) dan (0,1). Karena ke tiga suksesor tidak ada di *open* maupun *closed*, maka ke tiganya dimasukkan ke *open.* Langkah pertama ini menghasilkan *open* = [(1,1), (1,0), (0,1)] dan *closed* = [A(0,0)].

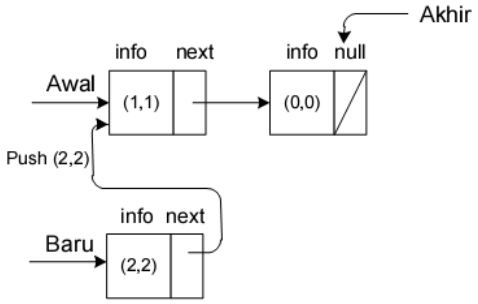
2. Langkah ke dua, semua suksesor (0,0) kemudian dicek nilai f nya. Ternyata nilai f melalui simpul (1,1) dari A(0,0) f(n) = 5 lebih kecil daripada biaya dari simpul (1,0) f(n) = 7 dan (0,1) f(n) = 7, maka simpul (1,1) terpilih sebagai *BestNode* dan ditambahkan ke *closed*.



Gambar 3.10 Penambahan simpul (1,1)

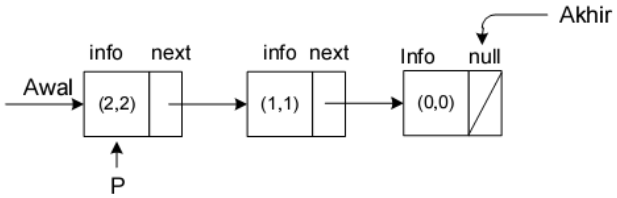
Oleh karena itu, *parent* harus diubah yang semula A(0,0) menjadi simpul (1,1) seperti pada gambar 3.10. Selanjutnya, semua suksesor (1,1) dibangkitkan yaitu simpul T(2,2), (2,1) dan (1,2). Karena simpul (2,2) yang pertama dicek dan ternyata sama dengan simpul tujuan maka simpul (2,1) dan (1,2) tidak perlu dicek. Simpul T(2,2) belum pernah ada di *open*

maupun *closed*, maka simpul T(2,2) dimasukkan ke *open*. Setelah dihitung f(n) = 2 dengan simpul (2,2) terpilih sebagai *BestNode* dan ditambahkan ke *closed* seperti pada gambar 3.11.



Gambar 3.11 Penambahan simpul (2,2)

Dikarenakan simpul ini sama dengan tujuan yang berarti solusi sudah ditemukan maka list akhir yang dihasilkan seperti pada gambar 3.12.



Gambar 3.12 list akhir yang dihasilkan

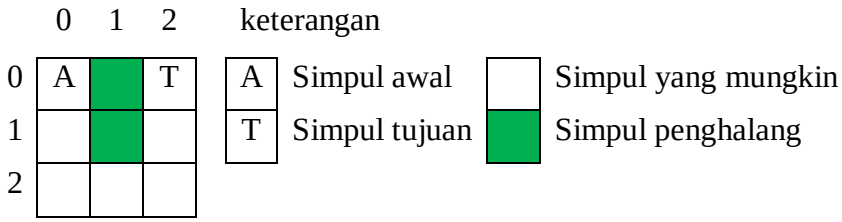
Jalan dan total biaya bisa ditelusuri balik dari T menuju A karena setiap simpul hanya memiliki satu *parent.* Penelusuran balik menghasilkan jalan A(0,0) – (1,1) – T(2,2) dengan jumlah langkah 1 dan total simpul yang diperiksa 5. Simpul-simpul yang diperiksa tiap langkah dapat dilihat pada tabel 3.2 jalur ini merupakan jalan atau solusi yang dihasilkan.

Tabel 3.2. Simpul yang diperiksa tiap langkah pada kondisi (i)

Langkah	***OPEN***	***CLOSED***
1	[(1,1), (1,0), (0,1)]	[A(0,0)]
2	[(1,0), (0,1), T(2,2)]	[A(0,0), (1,1)]

Contoh berikutnya pada gambar 3.3 kondisi (ii) atau pada gambar 3.13 di bawah. Pada kondisi ini, kita akan mencari jalan terpendek dari posisi titik awal (A) ke posisi titik tujuan (T) dengan penghalang. 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. Kondisi ini serupa dengan kejadian dimana predator akan memangsa *hero* pada *level* 4 namun predator harus menghindari NPC yang membantu *hero* dalam menghalau predator yang memangsa *hero.* Daerah NPC sendiri dianggap sebagai simpul yang tidak bisa dilewati atau sebagai penghalang.



Gambar 3.13 Contoh ruang dengan beberapa penghalang

Perhitungan *pathfinding* dengan kondisi di atas yang akan diuraikan sebagai berikut:

Posisi simpul awal = Ax : 0, Ay : 0

Posisi simpul tujuan = goal x : 2, goal y : 0

Langkah ke satu :

a. Simpul yang akan diambil n(0,1) sehingga kemungkinan simpul yang akan ditempuh adalah g(0,1) dengan perkiraan biaya diambil sebesar 1.

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 - 0))

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

= 3

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

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

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

= - 3

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

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

= -3 + 9

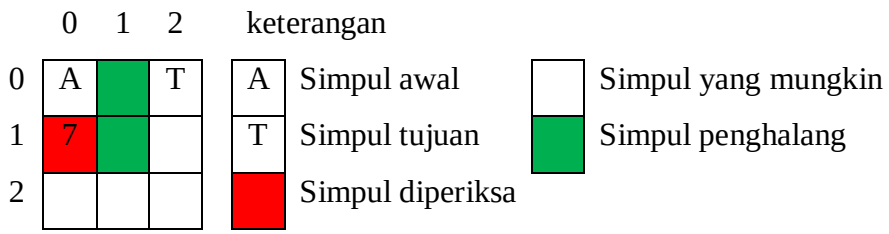
= 6

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

= 1 + 6

= 7

Pada gambar 3.14 terdapat satu simpul yang mungkin menjadi *BestNode* yaitu (0,1) dengan f(n)=7, maka dipilihlah simpul (0,1) dengan biaya yaitu 7.



Gambar 3.14 Langkah pertama pencarian *BestNode* kondisi (ii)

Perhitungan setelah langkah pertama menghasilkan simpul (0,1) sebagai *BestNode*, dari simpul ini ke simpul tujuan memiliki dua kemungkinan node yang akan diambil yaitu: n(0,2) dan n(1,2) yang akan diuraikan sebagai langkah selanjutnya sebagai berikut:

Langkah ke dua:

a. Simpul yang akan diambil n(0,2) sehingga kemungkinan simpul yang akan ditempuh adalah g(0,2) dengan perkiraan biaya diambil sebesar 2.

n (0,2) : g (0,2) = 2

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

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

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

= 4

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

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

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

= - 4

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

h(0,2) = (- 4)+ (4-(2*(-4)))

= -4 + 12

= 8

f (0,2) = g (0,2) + h (0,2)

= 2 + 8

= 10

b. Simpul yang akan diambil n(1,2) sehingga kemungkinan simpul yang akan ditempuh adalah g(1,2) dengan perkiraan biaya diambil sebesar 2.

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 - 0))

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

= 3

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

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

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

= - 3

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

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

= -3 + 9

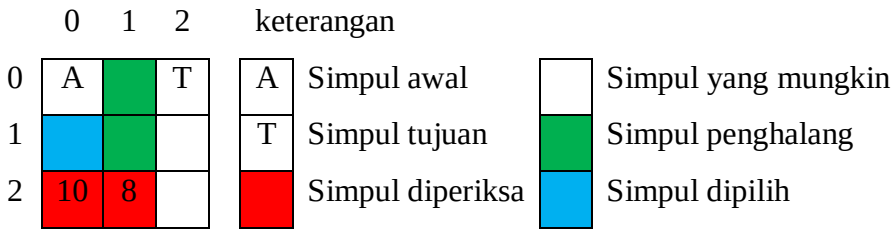
= 6

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

= 2 + 6

= 8

Pada gambar 3.15 terdapat dua simpul yang mungkin menjadi *BestNode* yaitu (0,2) dengan f(n)=10 dan (1,2) dengan f(n)=8. Dari ke dua simpul yang mungkin maka dipilihlah simpul (1,2) dengan biaya terkecil yaitu 8.



Gambar 3.15 Langkah kedua pencarian *BestNode* kondisi (ii)

Perhitungan setelah langkah kedua menghasilkan simpul (1,2) sebagai BestNode, dari simpul ini ke simpul tujuan memiliki dua kemungkinan node yang akan diambil yaitu: n(2,2) dan n(2,1) yang akan diuraikan sebagai langkah selanjutnya sebagai berikut:

Langkah ke tiga:

a. Simpul yang akan diambil n(2,2) sehingga kemungkinan simpul yang akan ditempuh adalah g(2,2) dengan perkiraan biaya diambil sebesar 3.

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

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

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

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

= 2

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

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

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

= - 2

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

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

= -2 + 6

= 4

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

= 3 + 4

= 7

b. Simpul yang akan diambil n(2,1) sehingga kemungkinan simpul yang akan ditempuh adalah g(2,1) dengan perkiraan biaya diambil sebesar 3.

n (2,1) : g (2,1) = 3

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

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

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

= 1

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

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

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

= - 1

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

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

= -1 + 3

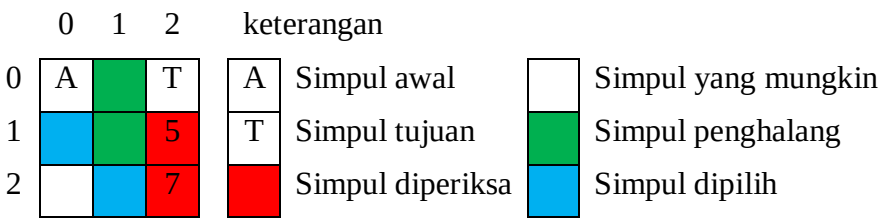
= 2

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

= 3+2

= 5

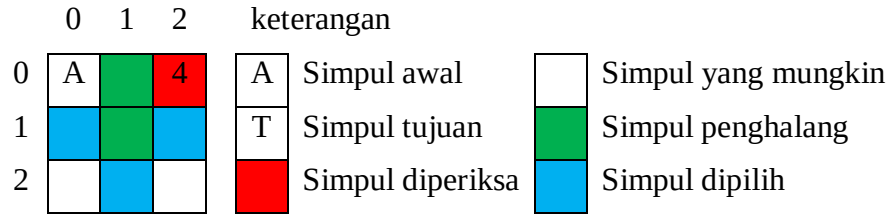
Pada gambar 3.16 terdapat dua simpul yang mungkin menjadi *BestNode* yaitu (2,2) dengan f(n)=7 dan (2,1) dengan f(n)=5. Dari ke dua simpul yang mungkin maka dipilihlah simpul (2,1) dengan biaya terkecil yaitu 5.



Gambar 3.16 Langkah ke tiga pencarian *BestNode* kondisi (iv)

Perhitungan setelah langkah ketiga menghasilkan simpul (2,1) sebagai BestNode, dari simpul ini ke simpul tujuan memiliki satu kemungkinan node yang akan diambil yaitu: n(2,0) yang akan diuraikan sebagai langkah selanjutnya sebagai berikut:

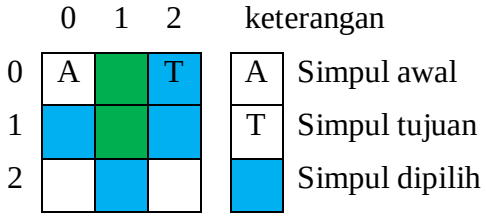
Pada langkah ke empat, untuk simpul yang akan diambil n(2,0) sehingga kemungkinan simpul yang akan ditempuh adalah g(2,2) dengan biaya diambil sebesar 4. Ternyata simpul ini dikenali sebagai simpul tujuan yaitu (2,2) berarti solusi telah ditemukan.



Gambar 3.17 Langkah ke empat pencarian *BestNode* kondisi (iv)

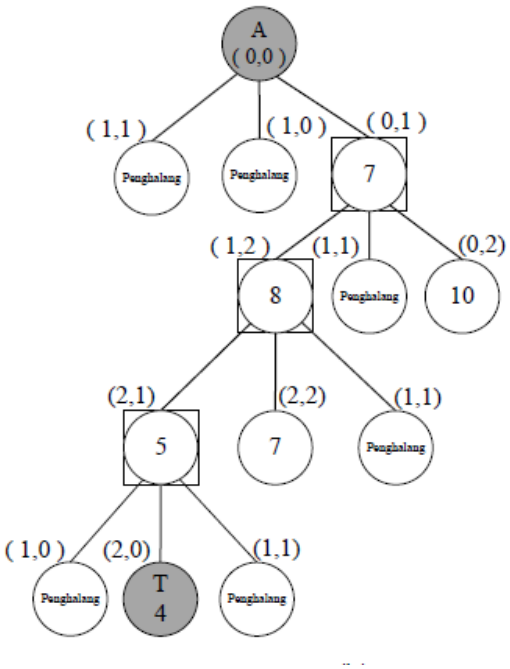
Pada gambar 3.17 terdapat satu simpul (2,0) dengan f(n)=4 dan dikenali sebagai simpul tujuan yang berarti solusi telah ditemukan.

Dari semua perhitungan yang telah dilakukan dipilihlah biaya/*cost* terkecil pada setiap langkahnya sehingga akan menghasilkan jalur terpendek yang terlihat pada gambar 3.18.



Gambar 3.18 Hasil pencarian jalur pada kondisi (iv)

Dari perhitungan di atas dipilihlah biaya atau *cost* terkecil pada setiap langkahnya sehingga akan menghasilkan jalur terpendek. Contoh kondisi ruang *map* di atas bila digambarkan dengan pohon pelacakan, maka pohonnya dapat dilihat pada gambar 3.19 dibawah:

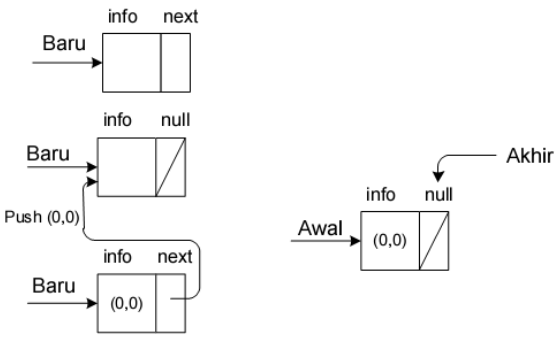


Gambar 3.19 Tree untuk masalah pada gambar 3.18

Pada gambar 3.19 terdapat simpul dengan tanda kotak yang menyatakan bahwa simpul tersebut berada di *closed* atau pernah terpilih sebagai *BestNode*, sedangkan simpul tanpa tanda kotak menyatakan simpul yang berada di *open*. Jika simpul sudah pada kondisi *closed* maka diabaikan, karena simpul ini sudah diuji.

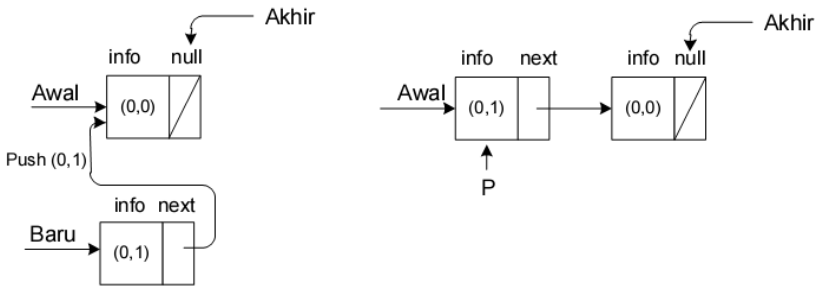
Pada kondisi (ii) pada gambar 3.3 atau penjelasan gambar 3.19 diatas, dapat dijelaskan langkah-langkah secara struktur datanya dengan *linked list* sebagai berikut:

1. Langkah pertama dalam kondisi diatas, karena di *open* hanya terdapat satu simpul yaitu A(0,0), maka A terpilih sebagai *BestNode* dan dipindahkan ke *closed* seperti pada gambar 3.20.



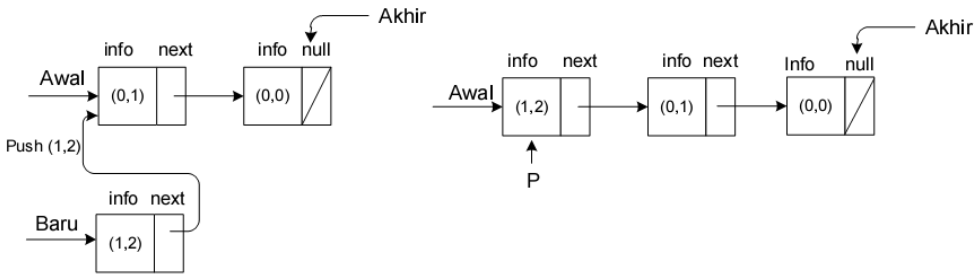
Gambar 3.20 Penambahan simpul (0,0)

Kemudian dibangkitkan semua suksesor A(0,0), yaitu simpul (1,1), (1,0) dan (0,1). Karena simpul (1,1) dan (1,0) dikenali sebagai penghalang maka tidak akan dimasukkan kedalam *list*, sedangkan untuk simpul (0,1) bukan penghalang dan tidak ada di *open* maupun *closed*, maka dimasukkan ke *open*. Langkah pertama ini menghasilkan *open* = [(0,1)] dan *closed* = [A(0,0)]. Selanjutnya, semua suksesor (0,0) kemudian dicek nilai f nya. Ternyata nilai f melalui simpul (0,1) dengan f(n) = 7 sebagai *BestNode* dan dipindahkan ke *closed*. Oleh karena itu, *parent* harus diubah yang semula A(0,0) menjadi simpul (0,1) seperti pada gambar 3.21.



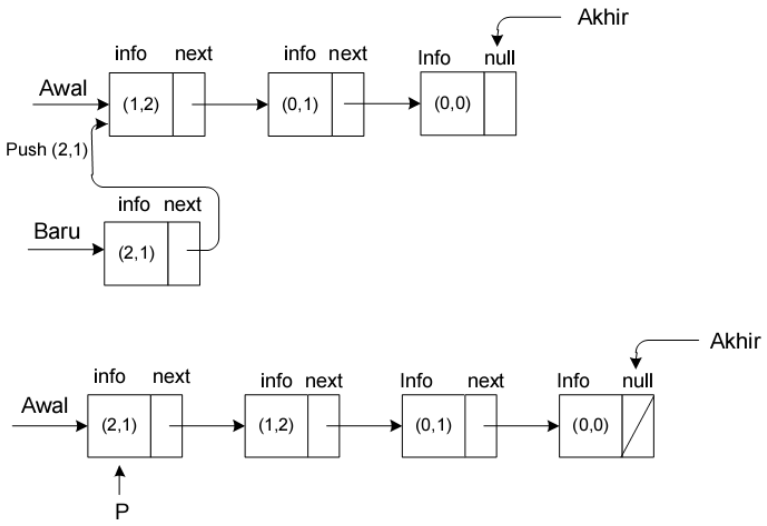
Gambar 3.21 Penambahan simpul (0,1)

2. Langkah ke dua semua suksesor (0,1) dibangkitkan yaitu simpul (1,2) dan (0,2). Karena belum pernah ada di *open* maupun *closed*, maka simpul (1,2) dan (0,2) dimasukkan ke *open*. Langkah ke dua ini menghasilkan *open* = [(1,2), (0,2)] dan *closed* = [A(0,0), (0,1)]. Selanjutnya, semua suksesor (0,1) kemudian dicek nilai f nya. Ternyata nilai f melalui simpul (1,2) menghasilkan f(n) = 8 dan simpul (0,2) menghasilkan f(n) = 10. Setelah dihitung terpilihlah simpul (1,2) dengan biaya terkecil sebagai *BestNode* dan dipindahkan ke *closed*. Oleh karena itu, *parent* harus diubah yang semula (0,1) menjadi simpul (1,2) seperti pada gambar 3.22.



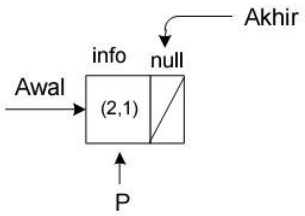
Gambar 3.22 Penambahan simpul (1,2)

3. Langkah ke tiga semua suksesor (1,2) dibangkitkan yaitu simpul (2,1) dan (2,2). Karena belum pernah ada di *open* maupun *closed*, maka simpul (2,1) dan (2,2) dimasukkan ke *open*. Langkah ke tiga ini menghasilkan *open* = [(0,2), (2,2), (2,1)] dan *closed* = [A(0,0), (0,1), (1,2)]. Selanjutnya, semua suksesor (1,2) kemudian dicek nilai f nya. Ternyata nilai f melalui simpul (2,1) menghasilkan f(n) = 5 dan simpul (2,2) menghasilkan f(n) = 7. Setelah dihitung terpilihlah simpul (2,1) dengan biaya terkecil sebagai *BestNode* dan dipindahkan ke *closed*. Oleh karena itu, *parent* harus diubah yang semula (1,2) menjadi simpul (2,1) seperti pada gambar 3.23.



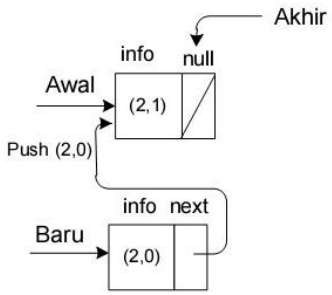
Gambar 3.23 Penambahan simpul (2,1)

4. Langkah ke empat semua suksesor (2,1) dibangkitkan yaitu simpul T(2,0). Karena belum pernah ada di *open* maupun *closed*, maka simpul T(2,0) dimasukkan ke *open*. Karena jumlah simpul saat ini di dalam *closed* yaitu 4 telah melebihi jumlah maksimum yang telah ditentukan yaitu 3. Maka, dilakukan *backtrack* atau penelusuran jalur yang ada dalam *closed* dari *parent* awal (titik awal/*pointer* akhir) sampai *parent* yang terakhir (atau sebanyak jumlah maksimum closed). Dengan catatan, simpul yang telah ditelusuri kemudian dihapus sehingga *closed* hanya menyisakan simpul (2,1) seperti pada gambar 3.24. Kemudian, simpul ini seolah dijadikan sebagai titik awal yang baru untuk proses *pathfinding* selanjutnya ke titik tujuan T(2,0).



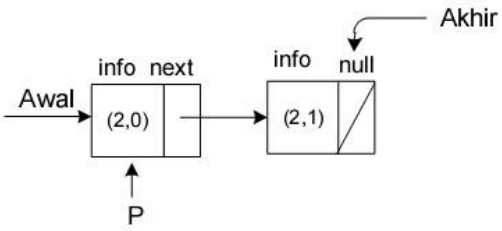
Gambar 3.24 *List* setelah penelusuran

Selanjutnya, simpul T(2,0) dengan f(n) = 4 terpilih sebagai *BestNode* dan dan ditambahkan ke *closed* seperti pada gambar 3.25



Gambar 3.25 Penambahan simpul (2,0)

Dikarenakan simpul ini sama dengan tujuan yang berarti solusi sudah ditemukan maka *list* akhir yang dihasilkan yaitu sebagai jalur yang kedua seperti pada gambar 3.26.



Gambar 3.26 *List* akhir yang dihasilkan

Pada akhir langkah ke empat ini, *open* = [(0,2), (2,2), T(2,0)] dan *closed* = [A(0,0), (0,1), (1,2)] dan [(2,1)] atau secara keseluruhan *closed* = [A(0,0), (0,1), (1,2), (2,1)]. Jalan dan total biaya bisa ditelusuri balik dari T menuju A karena setiap simpul hanya memiliki satu *parent*. Penelusuran balik menghasilkan 2 tahap. Tahap pertama A(0,0) – (0,1) – (1,2) – (2,1) dan tahap kedua (2,1) - T(2,0) dengan jumlah langkah 4 dan total simpul yang diperiksa 7. Simpul-simpul yang diperiksa tiap langkah dapat dilihat pada tabel 3.3.

Tabel 3.3. Simpul yang diperiksa tiap langkah pada kondisi (ii)

Langkah	***OPEN***	***CLOSED***
1	[(0,1)]	[A(0,0)]
2	[(1,2), (0,2)]	[A(0,0), (0,1)]
3	[(0,2), (2,1), (2,2)]	[A(0,0), (0,1), (1,2)]
4	[(0,2), (2,2), T(2,0)]	[A(0,0), (0,1), (1,2), (2,1)]

3.1.5 Analisis Kebutuhan Non Fungsional

Analisis non-fungsional merupakan analisis yang dibutuhkan untuk menentukan spesifikasi kebutuhan sistem. Spesifikasi ini juga meliputi elemen atau komponen-komponen apa saja yang dibutuhkan untuk sistem yang akan dibangun sampai dengan sistem tersebut diimplementasikan.

Pada analisis kebutuhan sistem non fungsional ini dijelaskan analisis mengenai kebutuhan perangkat keras (hardware), kebutuhan perangkat lunak (software), dan kebutuhan perangkat pikir (pengguna / *user)*.

3.1.5.1 Analisis Kebutuhan Perangkat Keras

Agar aplikasi dapat berjalan dengan baik, maka dibutuhkan perangkat keras yang sesuai dengan kebutuhan aplikasi. Tabel 3.5 adalah spesifikasi minimun perangkat keras yang dibutuhkan agar dapat menjalankan aplikasi secara optimal.

Tabel 3.4. Spesifikasi minimum perangkat keras

Komputer	**Spesifikasi perangkat keras**
Prosesor	Prosesor 1 Ghz ~
Resolusi layar	1024 x 768 pixel
Memori	Memori 512 Mb
VGA	64 MB
Keyboard dan mouse	Standar

3.1.5.2 Analisis Kebutuhan Perangkat Lunak

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 ini adalah sebagai berikut.

Tabel 3.5. Spesifikasi minimum perangkat lunak

Komputer	**Spesifikasi perangkat lunak**
Sistem Operasi	Window Xp, Window 7
Tools	IDE Pocessing 2.0b3

Berdasarkan hasil pengamatan perangkat lunak, terdapat beberapa hal penting yang harus diperhatikan agar sistem yang akan dirancang dapat berjalan dengan baik dan sesuai dengan yang diharapkan. Pengguna diharapkan dapat memenuhi kebutuhan perangkat keras yang telah direkomendasikan agar aplikasi dapat berjalan dengan baik. Adapun spesifikasi perangkat lunak yang digunakan untuk menjalankan aplikasi permainan ini adalah perangkat komputer yang telah terinstal JVM (*Java Virtual Machine*).

3.1.5.3 Analisis Kebutuhan Perangkat Pikir

Sistem pada aplikasi ini hanya terdiri dari satu bagian yaitu *front end* dimana *front end* digunakan dan berfungsi sebagai antar muka utama aplikasi bagi pengguna aplikasi.

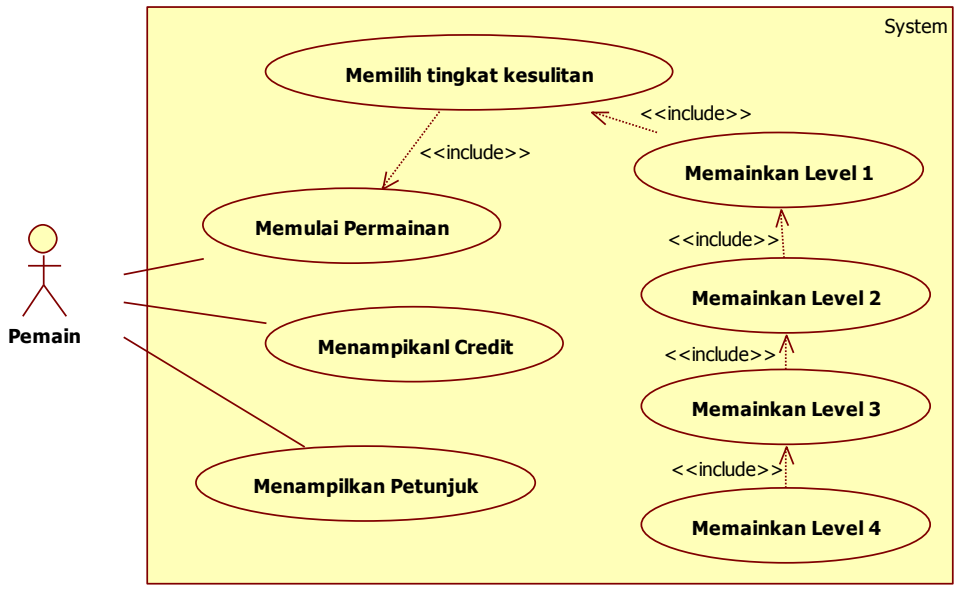
Tabel 3.6. Spesifikasi minimum perangkat pikir

	Pengguna
Peran	Pemain
Hak akses	Menggunakan aplikasi
Tingkat pendidikan	-
Tingkat keterampilan	Dapat mengoperasikan komputer dan membaca
Umur (tahun)	6 - 10

3.1.6 Analisis Kebutuhan Fungsional

3.1.6.1 Diagram *Use Case*

Secara umum, interaksi antara pengguna yaitu pemain dengan aplikasi permainan ini digambarkan pada diagram *Use Case*, seperti pada gambar 3.27.



Gambar 3.27 Diagram *use case* aplikasi

Berikut definisi *use case* yang menjelaskan setiap *use case* yang tersedia dalam tabel 3.7.

Tabel 3.7 Definisi *use case*

No	*Use Case*	Deskripsi
1	Menampilkan Petunjuk	Proses untuk melihat petuntuk cara bermain
2	Menampilkan *Credit*	Proses untuk melihat *credit*
3	Memulai Permainan	Proses untuk memulai permainan
4	Memilih Tingkat Kesulitan	Proses untuk memilih tingkat kesulitan
5	Memainkan Level 1	Representasi story board pertama
6	Memainkan Level 2	Representasi story board kedua
7	Memainkan Level 3	Representasi story board ketiga
8	Memainkan Level 4	Representasi story board keempat

3.1.6.2 Skenario *Use Case*

Skenario pada diagram *use case* berisi tentang alur kejadian '*flow of event' use case* utama. Di dalamnya menjelaskan urutan interaksi antara aktor dengan *use case* tersebut dari awal sampai akhir. Adapun scenario *use case* yang terdapat pada aplikasi permainan ini adalah sebagai berikut :

1. Skenario *Use Case* Menampilkan Petunjuk

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain dengan *use case* menampilkan petunjuk yang dijelaskan pada tabel 3.8.

Tabel 3.8 Skenario *use case* menampilkan petunjuk

Identifikasi	
Nama use case	Menampilkan petunjuk
Aktor	Pemain
Tujuan	Melihat informasi mengenai petunjuk permainan
Kondisi Awal	Pemain berada di menu utama aplikasi
Kondisi Akhir	Pemain melihat informasi petunjuk permainan
Skenario	
Aksi Aktor	**Reaksi Sistem**
1. Memilih menu petunjuk	
	2. Menampilkan informasi petunjuk permainan
	3. Jika setelah sistem menampilkan informasi petunjuk permainan pemain memilih menu keluar petunjuk, maka sistem akan menampilkan menu utama. 4. Jika pemain tidak memilih menu keluar petunjuk, maka sistem akan tetap menampilkan informasi petunjuk permainan.

2. Skenario *Use Case* Menampilkan *Credit*

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain dengan *use case* menampilkan *credit* yang dijelaskan pada tabel 3.9.

Tabel 3.9 Skenario *use case* menampilkan c*redit*

Identifikasi	
Nama use case	Menampilkan *credit*
Aktor	Pemain
Tujuan	Melihat informasi mengenai *credit*
Kondisi Awal	Pemain berada di menu utama aplikasi
Kondisi Akhir	Pemain melihat informasi *credit*
Skenario	
Aksi Aktor	**Reaksi Sistem**
1. Memilih menu *credit*	
	2. Menampilkan informasi *credit* permainan
	3. Jika setelah sistem menampilkan informasi *credit* permainan pemain memilih menu keluar petunjuk, maka sistem akan menampilkan menu utama. 4. Jika pemain tidak memilih menu keluar *credit*, maka sistem akan tetap menampilkan informasi *credit* permainan.

3. Skenario *Use Case* Memulai Permainan

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain dengan *use case* memulai permainan yang dijelaskan pada tabel 3.10

Tabel 3.10 Skenario *use case* memulai permainan

Identifikasi	
Nama use case	Memulai permainan
Aktor	Pemain
Tujuan	Memulai permainan aplikasi
Kondisi Awal	Pemain berada di menu utama aplikasi
Kondisi Akhir	Masuk ke dalam menu pilihan tingkat kesulitan
Skenario	
Aksi Aktor	**Reaksi Sistem**
1. Memilih menu mulai permainan	
	2. Menampilkan pemilihan tingkat kesulitan

4. Skenario *Use Case* Memilih Tingkat Kesulitan

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain dengan *use case* memilih tingkat kesulitan yang dijelaskan pada tabel 3.11.

Tabel 3.11 Skenario *use case* memilih tingkat kesulitan

Identifikasi	
Nama use case	Memilih tingkat kesulitan
Aktor	Pemain
Tujuan	Memilih tingkat kesulitan yang akan dilalui pemain
Kondisi Awal	Pemain berada di menu pemilihan tingkat kesulitan
Kondisi Akhir	Pemain memilih salah satu tingkat kesulitan
Skenario	
Aksi Aktor	**Reaksi Sistem**
1. Memilih menu pada tingkat kesulitan	
	2. Memeriksa menu yang telah terpilih

	3. Jika Pemain memilih tingkat kesulitan sulit, maka sistem akan menginisialisasi permainan dengan tingkat kesulitan yang sulit. 4. Jika Pemain memilih tingkat kesulitan mudah, maka sistem akan menginisialisasi permainan dengan tingkat kesulitan yang mudah. 5. Jika pemain memilih menu kembali, maka sistem akan menampilkan menu utama.
	6. Sistem menginisialisasi informasi permainan sesuai dengan tingkat kesulitan permainan yang dipilih
	7. Menampilkan scene pembuka
	8. Menampilkan arena permainan level 1 sesuai dengan tingkat kesulitan yang dipilih

5. Skenario *Use* Case Memainkan Level 1

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain dengan *use case* memainkan level 1 yang dijelaskan pada tabel 3.12.

Tabel 3.12 Skenario *use case* memainkan level 1

Identifikasi	
Nama use case	Memainkan Level 1
Aktor	Pemain
Tujuan	Menyelesaikan permainan level 1
Kondisi Awal	Arena permainan level 1 ditampilkan
Kondisi Akhir	Permainan dinyatakan berhasil atau gagal
Skenario	

Aksi Aktor	Reaksi Sistem
1. Mengendalikan hero	
	2. Menggerakan hero sesuai input pemain.
	3. Cek posisi hero, jika hero mencapai titik akhir, maka permainan dinyatakan berhasil. 4. Jika hero belum mencapai titik akhir, cek collision hero dengan rintangan.
	5. Jika terjadi tabrakan, maka cek kesempatan bermain. 6. Jika tidak terjadi tabrakan, pemain masih bisa mengendalikan hero.
	7. Jika terjadi tabrakan dan kesempatan bermain habis (*lifepoint* <0), maka permainan dinyatakan gagal. 8. Jika tidak terpenuhi, maka hero mati (*lifepoint* -= 1) kemudian reset posisi hero.
	9. Jika permainan dinyatakan berhasil, maka sistem akan menampilkan informasi berhasil. 10. Jika Pemain dinyatakan gagal, maka sistem akan menampilkan informasi gagal.
	11. Setelah sistem menampilkan informasi berhasil, maka sistem akan menginisialisasi permainan

	level 2. 12. Setelah sistem menampilkan informasi gagal, maka permainan selesai (gagal) dan kembali ke menu utama.
	13. Setelah sistem menginisialisasi permainan level 2, maka sistem akan menampilkan arena permainan level 2.

6. Skenario *Use Case* Memainkan Level 2

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain dengan *use case* memainkan level 2 yang dijelaskan pada tabel 3.13.

Tabel 3.13 Skenario *use case* memainkan level 2

Identifikasi	
Nama use case	Memainkan Level 2
Aktor	Pemain
Tujuan	Menyelesaikan permainan level 2
Kondisi Awal	Arena permainan level 2 ditampilkan
Kondisi Akhir	Pemain dinyatakan berhasil atau gagal
Skenario	
Aksi Aktor	**Reaksi Sistem**
1. Mengendalikan hero	
	2. Menggerakan hero sesuai input pemain.
	3. Cek collision, jika hero menabrak food, maka update energy hero kemudian cek jumlah lifepoint. 4. Jika hero tidak menabrak food, cek collision hero menabrak musuh.

	5. Jika hero menabrak musuh, maka update lifepoint hero kemudian cek jumlah lifepoint. 6. Jika hero tidak menabrak musuh, cek collision musuh menabrak sarang.
	7. Jika musuh menabrak sarang, maka update hp hero kemudian cek jumlah lifepoint. 8. Jika musuh tidak menabrak sarang, cek jumlah lifepoint
	9. Jika lifepoint habis permainan dinyatakan gagal. Jika lifepoint belum habis cek jumlah hp.
	10. Jika jumlah hp habis permainan dinyatakan gagal. Jika jumlah hp belum habis cek waktu bermain.
	11. Jika waktu bermain habis permainan dinyatakan berhasil. Jika waktu belum habis pemain masih mengendalikan hero.
	12. Jika Pemain dinyatakan berhasil, maka sistem akan menampilkan informasi berhasil. 13. Jika Pemain dinyatakan gagal, maka sistem akan menampilkan informasi gagal.
	14. Setelah sistem menampilkan informasi berhasil, maka sistem

	akan menginisialisasi permainan level 3. 15. Setelah sistem menampilkan informasi gagal, maka permainan selesai (*game over*) dan kembali ke menu utama.
	16. Setelah sistem menginisialisasi permainan level 3, maka sistem akan menampilkan arena permainan level 3.

7. Skenario *Use Case* Memainkan Level 3

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain dengan *use case* memainkan level 3 yang dijelaskan pada tabel 3.14.

Tabel 3.14 Skenario *use case* memainkan level 3

Identifikasi	
Nama use case	Memainkan Level 3
Aktor	Pemain
Tujuan	Menyelesaikan permainan level 3
Kondisi Awal	Arena permainan level 3 ditampilkan
Kondisi Akhir	Pemain dinyatakan berhasil atau gagal
Skenario	
Aksi Aktor	**Reaksi Sistem**
1. Mengendalikan hero	
	2. Menggerakan hero sesuai input pemain.
	3. Cek collision, jika hero menabrak food, maka update energy hero kemudian cek jumlah lifepoint.

	4. Jika hero tidak menabrak food, cek collision hero menabrak musuh.
	5. Jika hero menabrak musuh, maka update lifepoint hero kemudian cek jumlah lifepoint. 6. Jika hero tidak menabrak musuh, cek collision musuh menabrak sarang.
	7. Jika musuh menabrak sarang, maka update hp hero kemudian cek jumlah lifepoint. 8. Jika musuh tidak menabrak sarang, cek jumlah lifepoint
	9. Jika lifepoint habis permainan dinyatakan gagal. Jika lifepoint belum habis cek jumlah hp.
	10. Jika jumlah hp habis permainan dinyatakan gagal. Jika jumlah hp belum habis cek waktu bermain.
	11. Jika waktu bermain habis permainan dinyatakan berhasil. Jika waktu belum habis pemain masih mengendalikan hero.
	12. Jika Pemain dinyatakan berhasil, maka sistem akan menampilkan informasi berhasil. 13. Jika Pemain dinyatakan gagal, maka sistem akan menampilkan informasi gagal.

	14. Setelah sistem menampilkan informasi berhasil, maka sistem akan menginisialisasi permainan level 4. 15. Setelah sistem menampilkan informasi gagal, maka permainan selesai (*game over*) dan kembali ke menu utama.
	16. Setelah sistem menginisialisasi permainan level 4, maka sistem akan menampilkan arena permainan level 4.

8. Skenario *Use Case* Memainkan Level 4

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu pemain dengan *use case* memainkan level 4 yang dijelaskan pada tabel 3.15.

Tabel 3.15 Skenario *use case* memainkan level 4

Identifikasi	
Nama use case	Memainkan Level 4
Aktor	Pemain
Tujuan	Menyelesaikan permainan level 4
Kondisi Awal	Arena permainan level 4 ditampilkan
Kondisi Akhir	Pemain dinyatakan berhasil atau gagal
Skenario	
Aksi Aktor	**Reaksi Sistem**
1. Mengendalikan hero	
	2. Menggerakan hero sesuai input pemain.
	3. Cek collision, jika hero menabrak food, maka update energy hero

	kemudian cek jumlah lifepoint. 4. Jika hero tidak menabrak food, cek collision hero menabrak musuh.
	5. Jika hero menabrak musuh, maka update lifepoint hero kemudian cek jumlah lifepoint. 6. Jika hero tidak menabrak musuh, cek collision musuh menabrak sarang.
	7. Jika musuh menabrak sarang, maka update hp hero kemudian cek jumlah lifepoint. 8. Jika musuh tidak menabrak sarang, cek jumlah lifepoint
	9. Jika lifepoint habis permainan dinyatakan gagal. Jika lifepoint belum habis cek jumlah hp.
	10. Jika jumlah hp habis permainan dinyatakan gagal. Jika jumlah hp belum habis cek waktu bermain.
	11. Jika waktu bermain habis permainan dinyatakan berhasil. Jika waktu belum habis pemain masih mengendalikan hero.
	12. Jika Pemain dinyatakan berhasil, maka sistem akan menampilkan informasi berhasil. 13. Jika Pemain dinyatakan gagal, maka sistem akan menampilkan informasi gagal.

	14. Setelah sistem menampilkan informasi berhasil, maka sistem akan menampilkan scene penutup. 15. Setelah sistem menampilkan informasi gagal, maka permainan selesai (*game over*) dan kembali ke menu utama.
	16. Setelah sistem menampilkan scene penutup, maka sistem akan kembali ke menu utama.

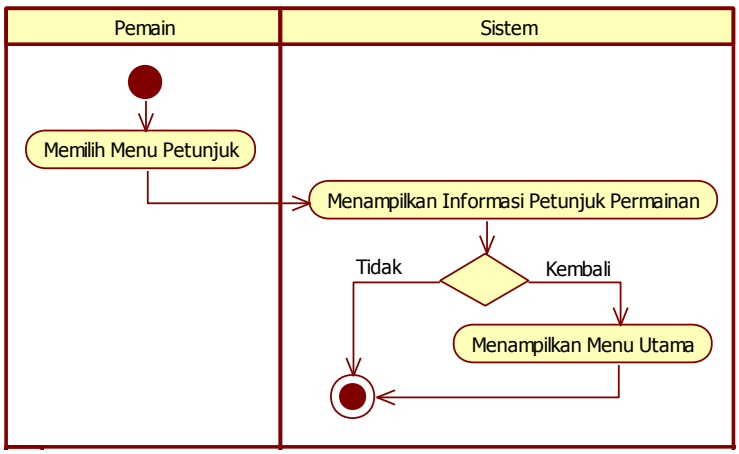
3.1.6.3 Diagram Aktivitas

Diagram ini berisi penjelasan secara lengkap mengenai urutan aktivitas yang terjadi antara pemain dengan aplikasi permainan ini.

1. *Activity Diagram* Menampilkan Petunjuk

Proses ini terjadi saat pemain memilih menu petunjuk pada menu utama aplikasi permainan ini. Sistem akan menampilkan petunjuk permainan dari aplikasi permainan ini.

Diagram *activity* menampilkan petunjuk terdapat pada gambar 3.28.

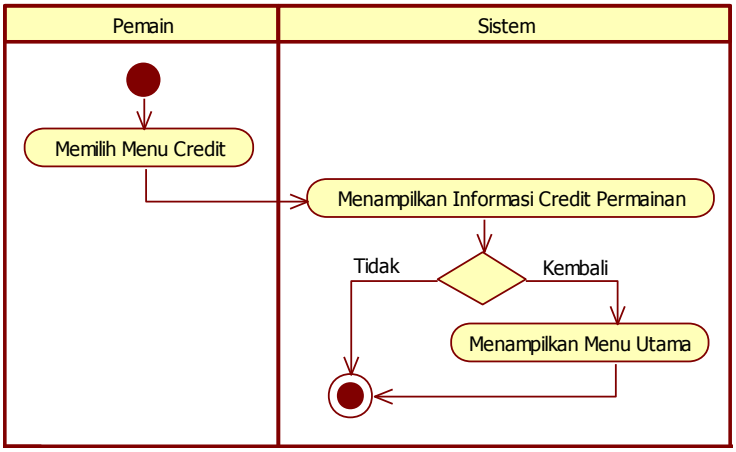


Gambar 3.28 *Activity diagram* menampilkan petunjuk

2. *Activity Diagram* Menampilkan *Credit*

Proses ini terjadi saat pemain memilih menu tampil *credit* pada menu utama aplikasi permainan ini. Sistem akan menampilkan *credit* permainan dari aplikasi permainan ini.

Diagram *activity* menampilkan *credit* terdapat pada gambar 3.29.

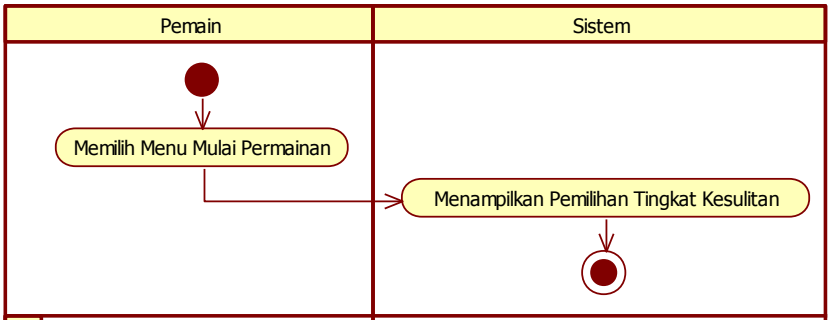


Gambar 3.29 *Activity diagram* tampil *credit*

3. *Activity Diagram* Memulai Permainan

Proses ini terjadi saat pemain memilih menu mulai permainan pada menu utama aplikasi permainan, kemudian sistem akan menampilkan menu pemilihan tingkat kesulitan atau pemain dapat kembali lagi ke menu utama.

Activity diagram mulai permainan terdapat pada gambar 3.30.



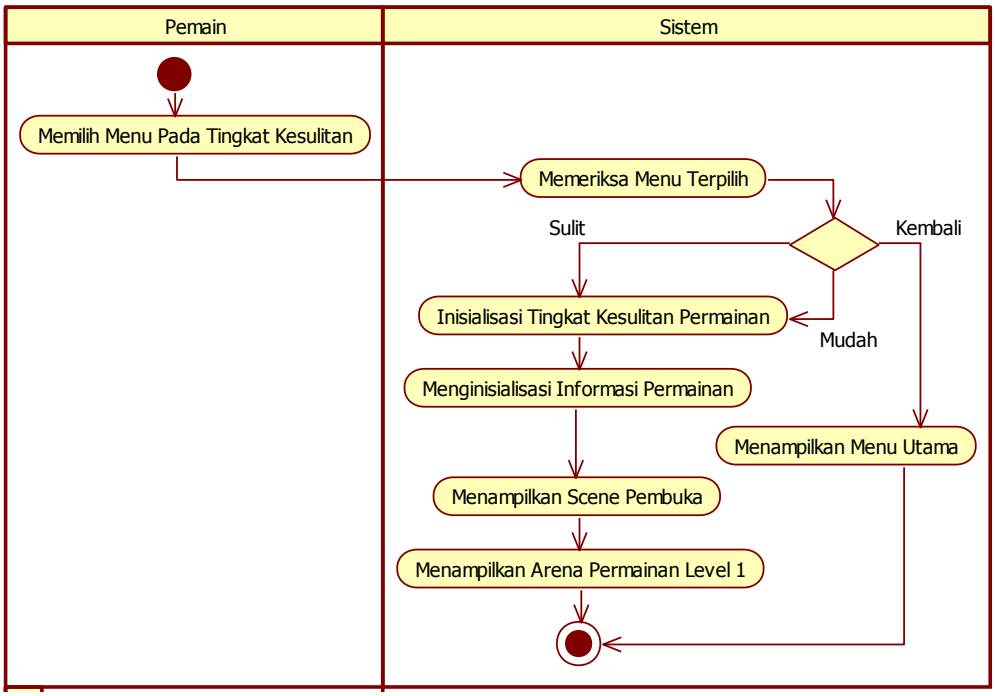
Gambar 3.30 *Activity diagram* memulai permainan

4. *Activity Diagram* Memilih Tingkat Kesulitan

Proses ini terjadi saat pemain telah memilih menu mulai permainan pada menu utama aplikasi permainan. Sistem akan menampilkan pilihan tingkat

kesulitan dengan tingkat kesulitan mudah dan sulit. Setelah inisialisasi informasi sistem maka pemain langsung dapat memainkan permainan pada level 1. Apabila tidak memilih salah satu tingkat kesulitan.

Activity diagram memilih tingkat kesulitan terdapat pada gambar 3.31.

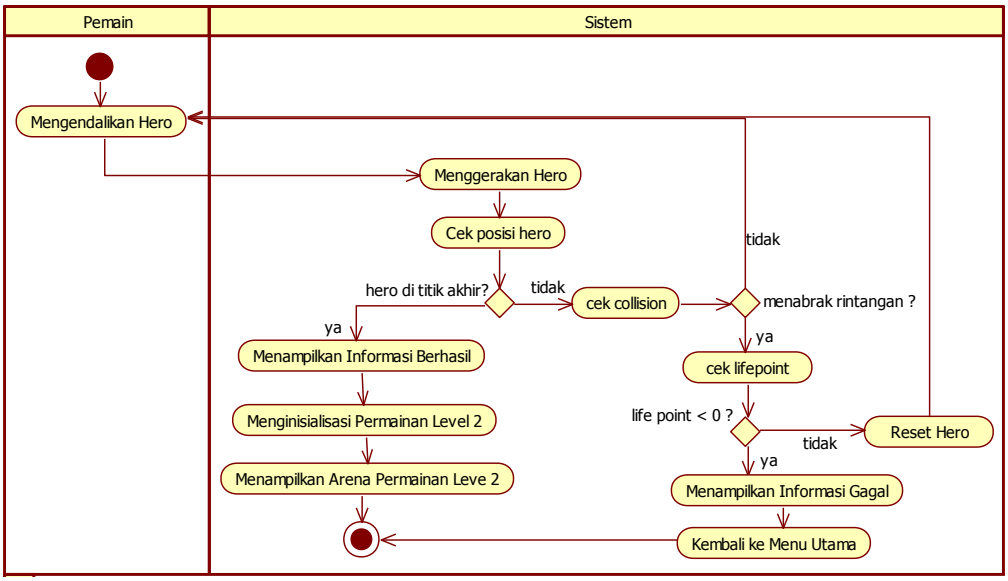


Gambar 3.31 *Activity diagram* memilih tingkat kesulitan

5. *Activity Diagram* Memainkan Level 1

Proses ini terjadi saat pemain telah memilih salah satu tingkat kesulitan permainan yang akan dilalui pada aplikasi permainan. Sistem akan menampilkan arena permainan dimana pemain dapat langsung memainkan *level* 1 dan menyelesaikan misi permainan pada *level* 1. Jika misi permainan terselesaikan maka permainan berlanjut ke *level* 2, apabila misi tidak terselesaikan maka pemain kembali ke menu utama untuk memulai permainan kembali.

Activity diagram memainkan level 1 terdapat pada gambar 3.32.

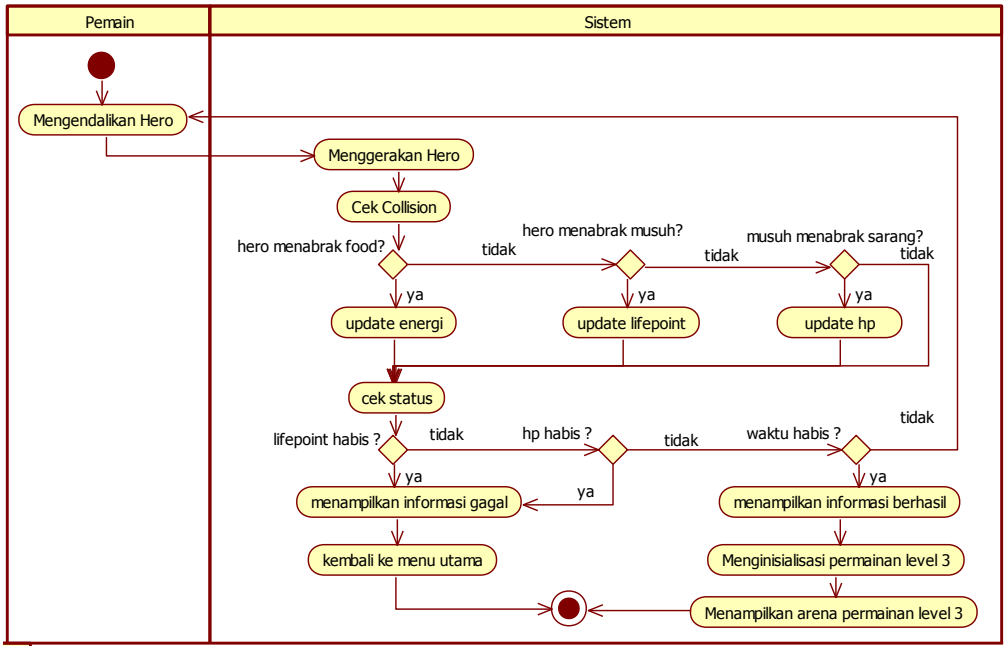


Gambar 3.32 *Activity diagram* memainkan level 1

6. *Activity Diagram* Memainkan Level 2

Proses ini terjadi saat pemain telah menyelesaikan aplikasi permainan pada *level* 1. Sistem akan menampilkan arena permainan *level* 2 dan pemain harus menyelesaikan misi permainan pada *level* 2. Jika misi permainan terselesaikan maka permainan berlanjut ke *level* 3, apabila misi permainan tidak terselesaikan maka pemain dinyatakan gagal dan akan kembali ke menu utama untuk memulai permainan dari awal.

Diagram *activity* memainkan *level* 2 terdapat pada gambar 3.33.

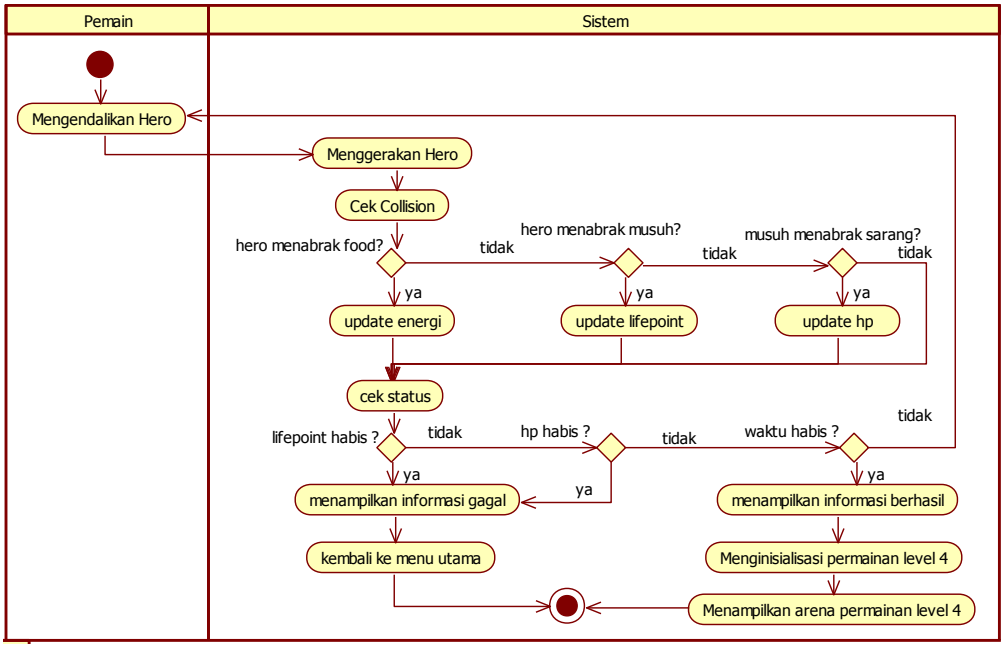


Gambar 3.33 *Activity diagram* memainkan level 2

7. *Activity Diagram* Memainkan Level 3

Proses ini terjadi saat pemain telah menyelesaikan aplikasi permainan pada *level* 2. Sistem akan menampilkan arena permainan *level* 3 dan pemain harus menyelesaikan misi permainan pada *level* 3. Jika misi permainan terselesaikan maka permainan berlanjut ke *level* 4, apabila misi permainan tidak terselesaikan maka pemain dinyatakan gagal dan akan kembali ke menu utama untuk memulai permainan dari awal.

Diagram *activity* memainkan *level 3* terdapat pada gambar 3.34.

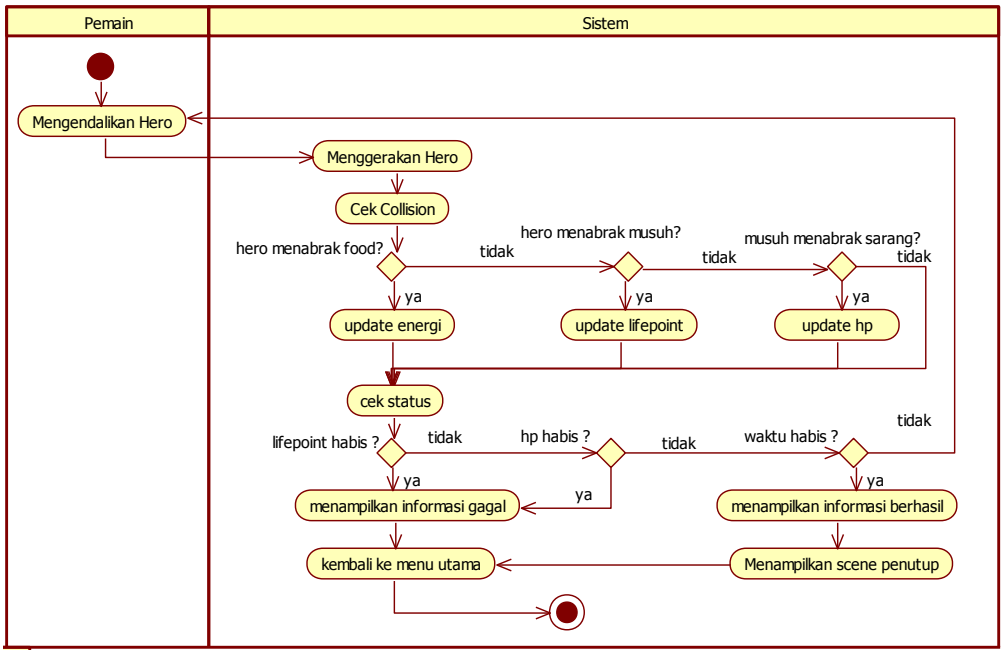


Gambar 3.34 *Activity diagram* memainkan level 3

8. *Activity Diagram* Level 4

Proses ini terjadi saat pemain telah menyelesaikan aplikasi permainan pada *level 3*. Sistem akan menampilkan arena permainan *level 4* dan pemain harus menyelesaikan misi permainan pada *level 4*. Jika misi permainan terselesaikan maka permainan ini telah selesai, kemudian setelah informasi terselesaikannya permainan selesai maka sistem akan kembali ke menu utama. Apabila misi permainan tidak terselesaikan maka pemain dinyatakan gagal dan akan kembali ke menu utama untuk memulai permainan dari awal.

Diagram *activity* memainkan *level 4* terdapat pada gambar 3.35.



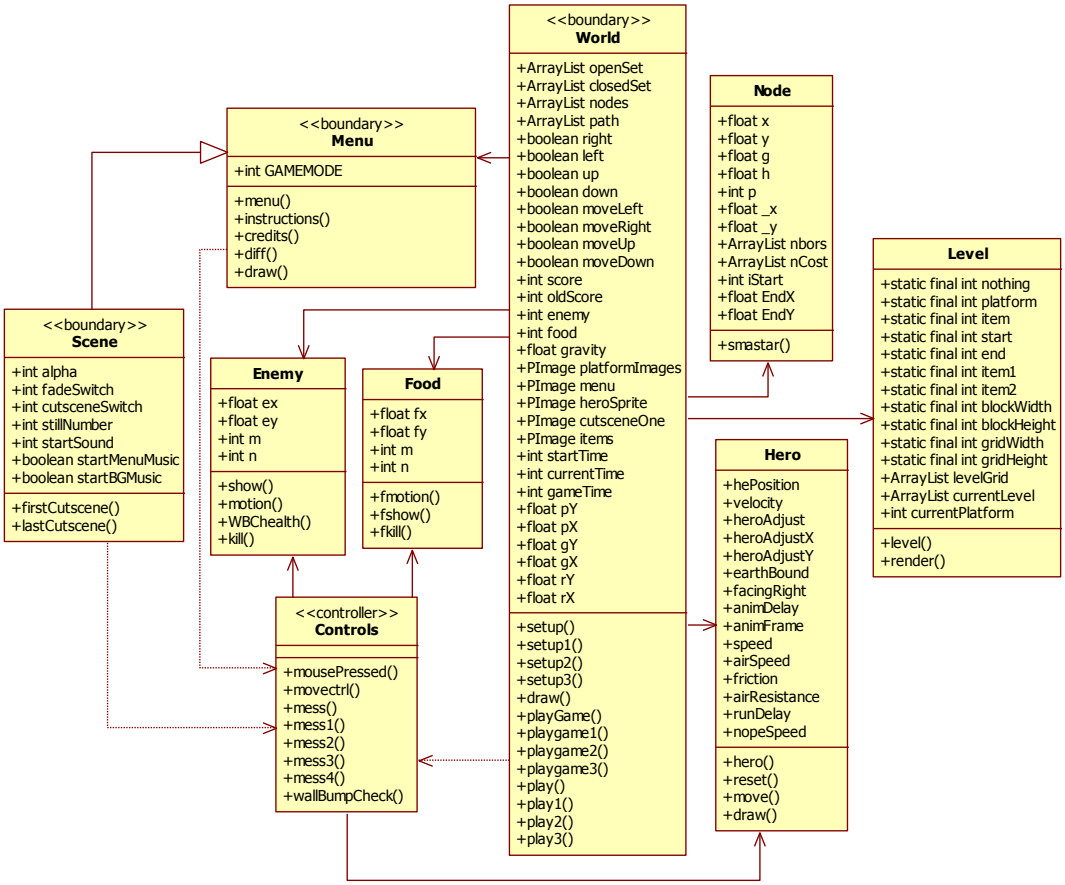
Gambar 3.35 *Activity diagram* memainkan level 4

3.1.6.4 Diagram Kelas

Diagram *class* pada aplikasi permainan ini terdapat pada gambar 3.36. Berikut adalah komposis kelas berdasarkan jenis dan fungsi umum dari kelas yang dibutuhkan.

Tabel 3.16 Komposisi kelas

Jenis *Class*	**Nama *Class***	**Keterangan**
Boundary	World	Antarmuka arena permainan
	Menu	Antarmuka pemilihan menu
	Scene	Antarmuka ilustrasi alur permainan
Control	Controls	Memodelkan perilaku khusus untuk mengatur objek lain
Entity	Level	Menyimpan dan mengatur informasi permainan
	Node	Menyimpan dan mengatur informasi permainan
	Hero	Menyimpan dan mengatur informasi hero
	Enemy	Menyimpan dan mengatur informasi enemy
	Food	Menyimpan dan mengatur informasi food



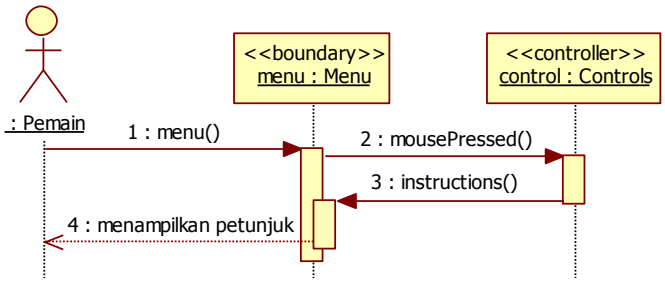
Gambar 3.36 Kelas diagram aplikasi

3.1.6.5 Diagram *Sequence*

Berikut merupakan diagram *sequence* yang terdapat pada aplikasi permainan yang berdasarkan *use case* yang ada:

A. Diagram *Sequence* dari *Use Case* : Menampilkan Petunjuk

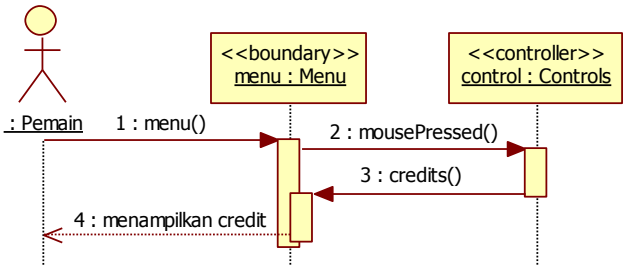
Diagram *sequence* menampilkan petunjuk terdapat pada gambar 3.37



Gambar 3.37 Diagram *sequence* menampilkan petunjuk

B. Diagram *Sequence* dari *Use Case* : Menampilkan Kredit

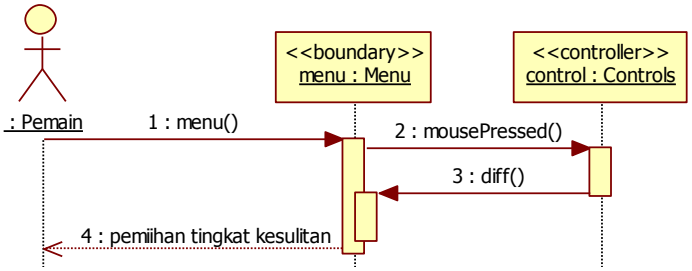
Diagram *sequence* menampilkan kredit terdapat pada gambar 3.38



Gambar 3.38 Diagram *sequence* tampil kredit

C. Diagram *Sequence* dari *Use Case* : Memulai Permainan

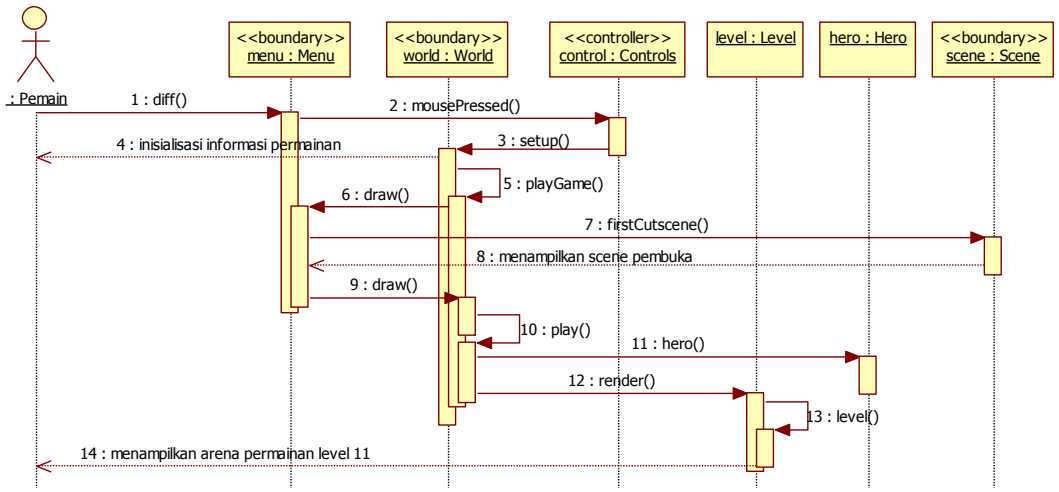
Diagram *sequence* memulai permainan terdapat pada gambar 3.39



Gambar 3.39 Diagram *sequence* memulai permainan

D. Diagram *Sequence* dari *Use Case* : Memilih Tingkat Kesulitan

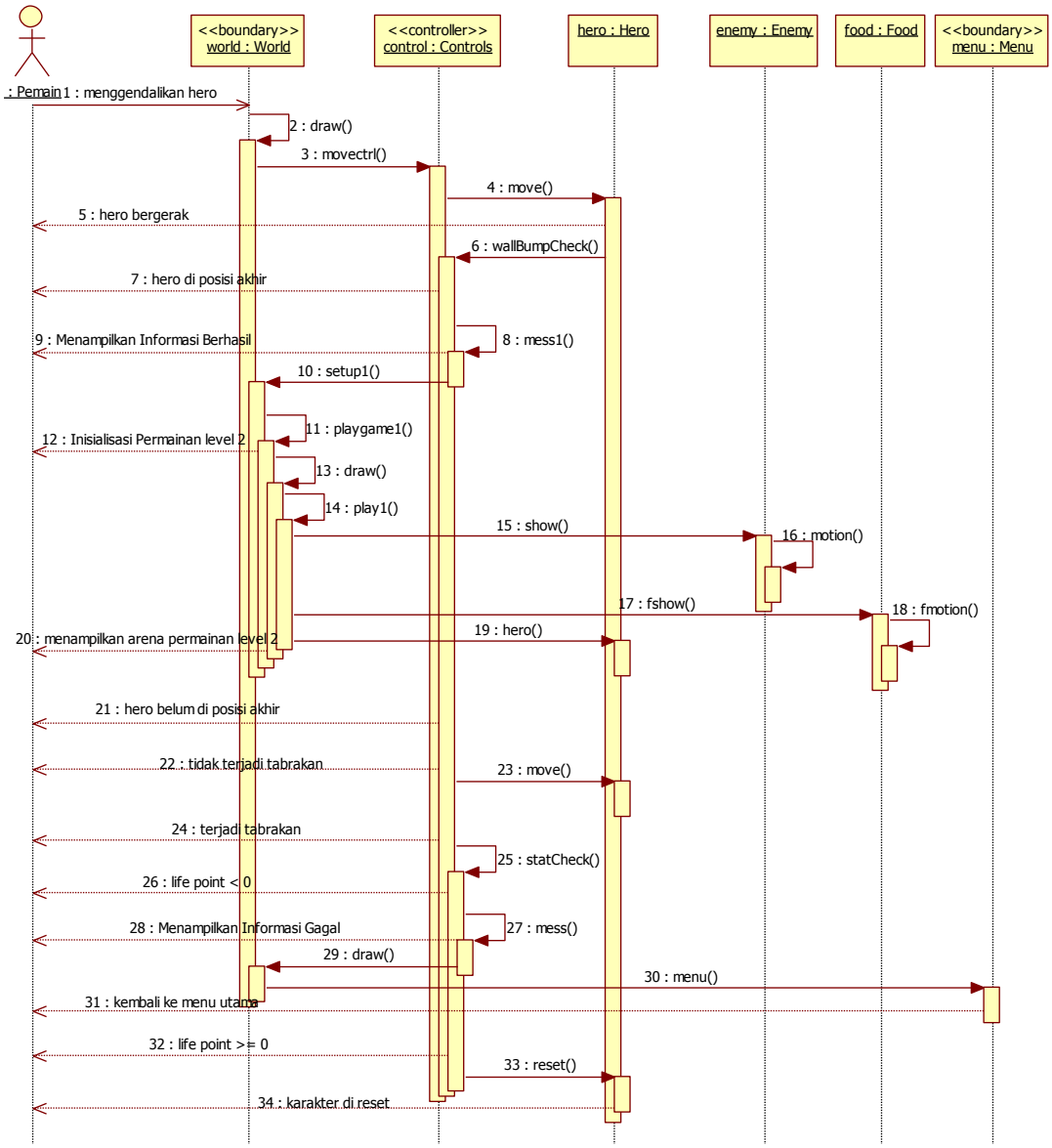
Diagram *sequence* memilih tingkat kesulitan terdapat pada gambar 3.40



Gambar 3.40 Diagram *sequence* memilih tingkat kesulitan

E. Diagram *Sequence* dari *Use Case* : Memainkan Level 1

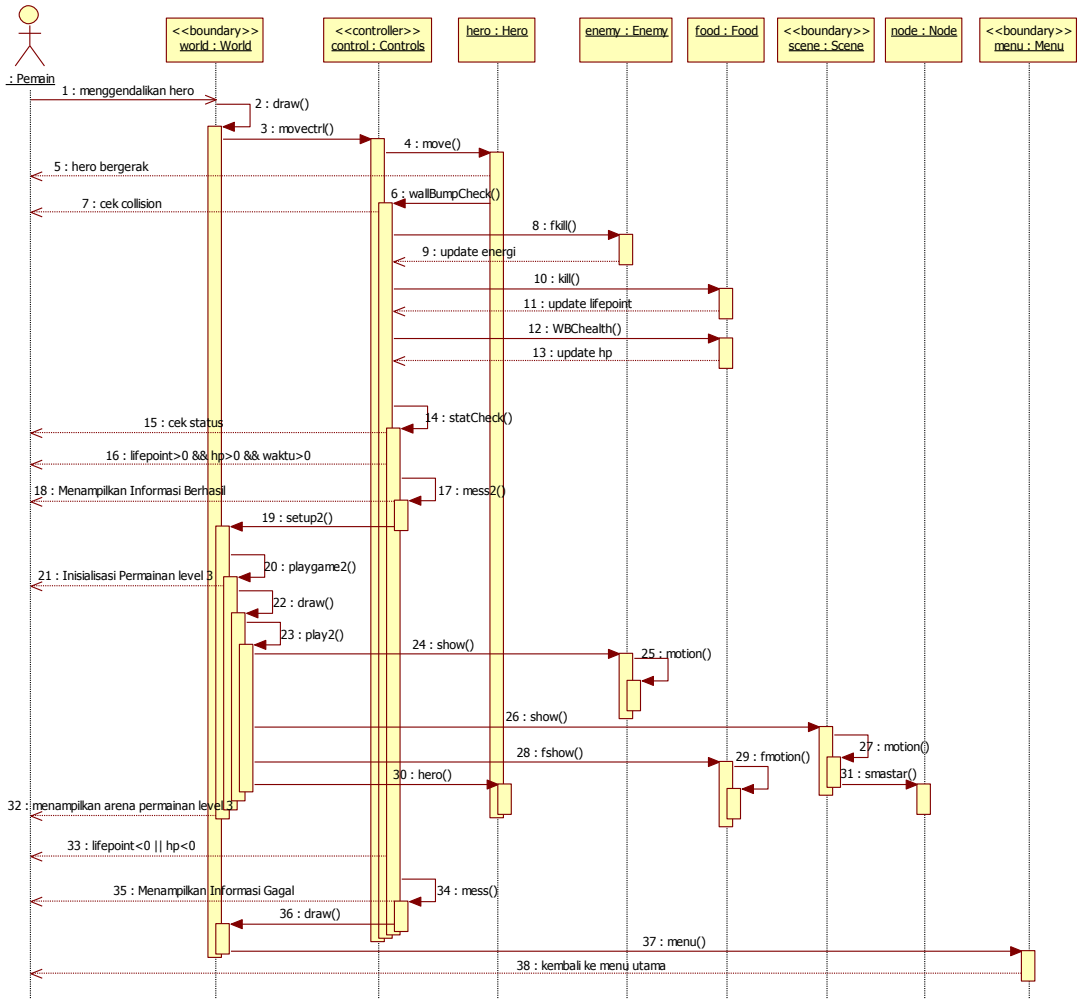
Diagram *sequence* memainkan level 1 terdapat pada gambar 3.41



Gambar 3.41 Diagram *sequence* memainkan level 1

F. Diagram *Sequence* dari *Use Case* : Memainkan Level 2

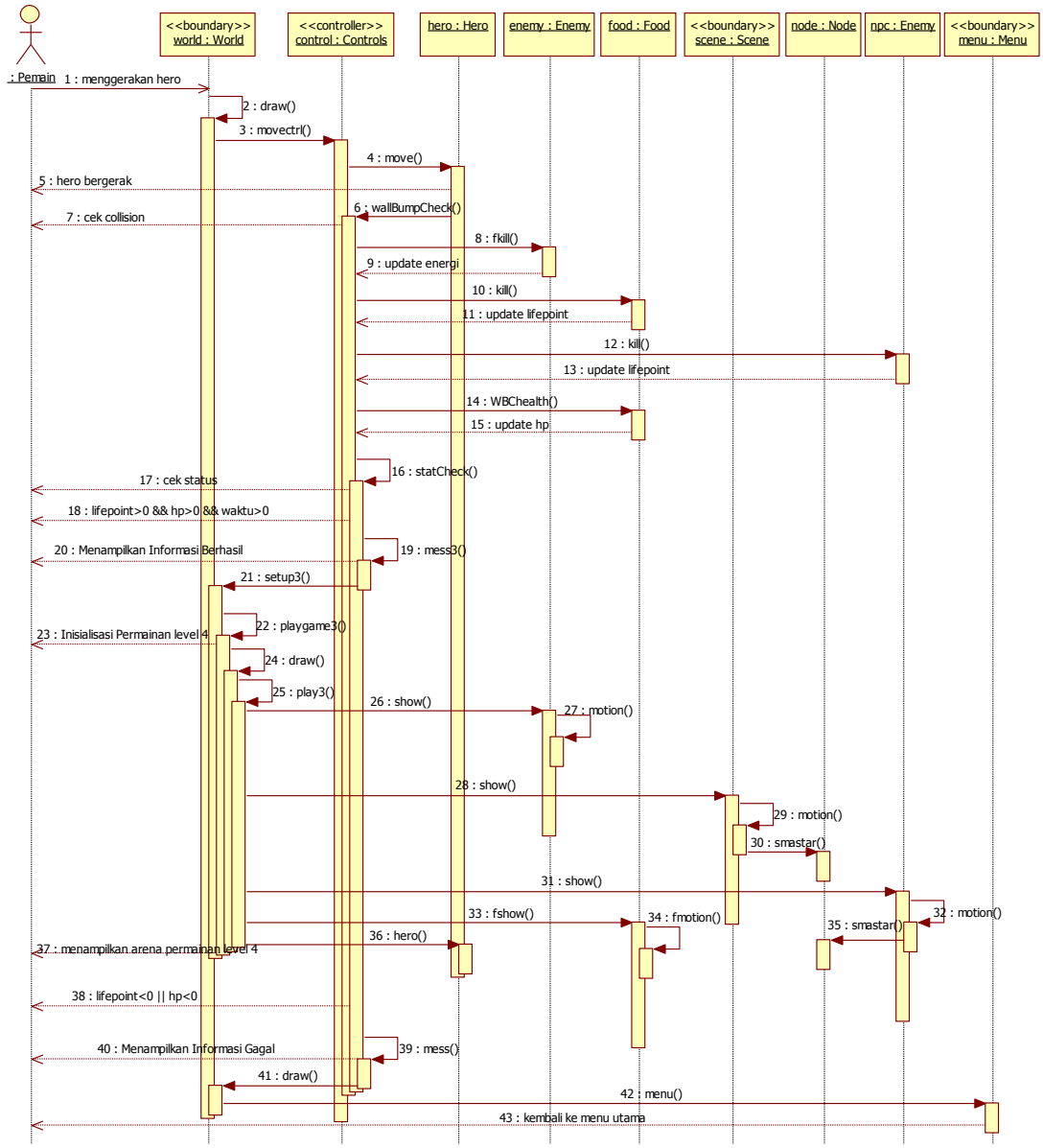
Diagram *sequence* memainkan level 2 terdapat pada gambar 3.42



Gambar 3.42 Diagram *sequence* memainkan level 2

G. Diagram *Sequence* dari *Use Case* : Memainkan Level 3

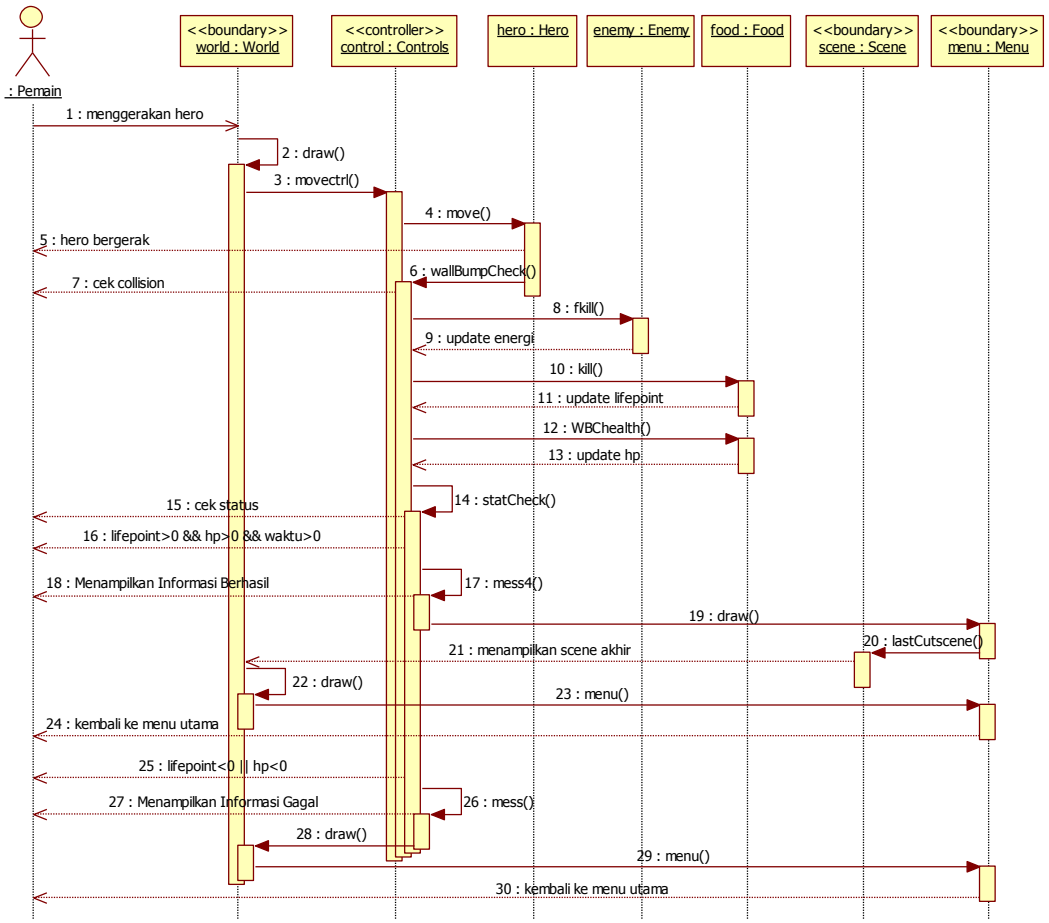
Diagram *sequence* memainkan level 3 terdapat pada gambar 3.43



Gambar 3.43 Diagram *sequence* memainkan level 3

H. Diagram *Sequence* dari *Use Case* : Memainkan Level 4

Diagram *sequence* memainkan level 4 terdapat pada gambar 3.44



Gambar 3.44 Diagram *sequence* memiankan level 4

3.2 Perancangan Sistem

Perancangan sistem merupakan pengembangan suatu perangkat lunak yang dilakukan setelah melalui tahapan analisis. Perancangan bertujuan untuk memberikan gambaran secara terperinci dan merupakan tahap lanjutan dari analisis sistem. Dimana dalam hal ini digambarkan rancangan sistem yang akan dibangun sebelum dilakukan pengodean ke dalam suatu bahasa pemrograman. Desain umum yang akan diaplikasikan bertujuan untuk memberikan gambaran secara umum tentang sistem yang akan dibangun.

3.2.1 Perancangan Struktur Atribut Kelas

Kelas yang terdapat pada sistem yang akan dibangun terdiri dari 3 kelas *boundary*, 1 kelas *control* dan 5 kelas *entity*. Struktur dari atribut kelas *entity* yang memuat informasi yang harus disimpan oleh sistem tersebut adalah sebagai berikut:

Tabel 3.17 Struktur atribut kelas Hero

Atribut	Tipe Data	Akses
hePosition	PVector	*public*
velocity	PVector	*public*
heroAdjust	int	*public*
heroAdjustX	int	*public*
heroAdjustY	int	*public*
earthBound	Boolean	*public*
facingRight	Boolean	*public*
animDelay	int	*public*
animFrame	int	*public*
speed	static final float	*public*
airSpeed	static final float	*public*
friction	static final float	*public*
energy	int	*public*
lifepoint	int	*public*
hp	int	*public*

Tabel 3.18 Struktur atribut kelas Level

Atribut	Tipe Data	Akses
nothing	static final int	*public*
platform	static final int	*public*
item	static final int	*public*
start	static final int	*public*
end	static final int	*public*

item1	static final int	*public*
item2	static final int	*public*
blockWidth	static final int	*public*
blockHeight	static final int	*public*
gridWidth	static final int	*public*
gridHeight	static final int	*public*
levelGrid	ArrayList	*public*
currentLevel	ArrayList	*public*
currentPlatform	int	*public*

Tabel 3.19 Struktur atribut kelas Node

Atribut	Tipe Data	Akses
x	float	*public*
y	float	*public*
g	float	*public*
h	float	*public*
p	int	*public*
_x	float	*public*
_y	float	*public*
nbors	ArrayList	*public*
nCost	ArrayList	*public*
iStart	int	*public*
EndX	float	*public*
EndY	float	*public*

Tabel 3.16 Struktur atribut kelas Food

Atribut	Tipe Data	Akses
fx	float	*public*
fy	float	*public*
m	int	*public*
n	int	*public*

Tabel 3.20 Struktur atribut kelas Enemy

Atribut	Tipe Data	Akses
ex	float	*public*
ey	float	*public*
m	int	*public*
n	int	*public*

3.2.2 Perancangan Komponen Permainan

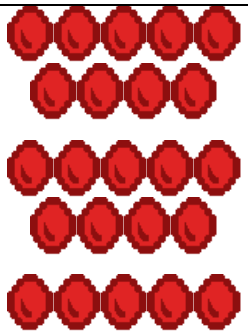
3.2.2.1 Entitas Game

Entitas yang terdapat di dalam *game* terbagi menjadi dua, yaitu karakter *hero* dan NPC yang berupa musuh, selain itu terdapat juga item yang digunakan untuk membantu *hero* dalam pertahanan.

Entitas karakter dan item yang terdapat di dalam game ini ditunjukkan pada tabel 3.16.

Tabel 3.21. Entitas katakter dan item

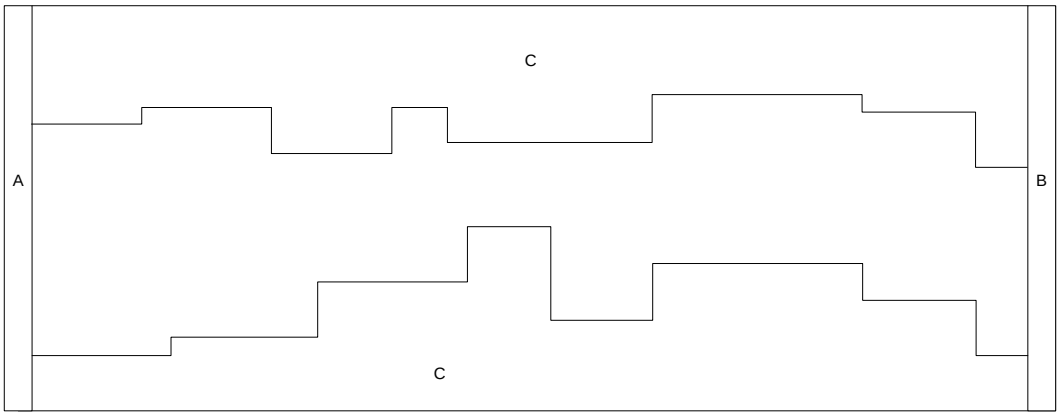
NO	NAMA	GAMBAR	PENJELASAN
1	Giru		Pada karakter *hero* ini, merupakan karakter utama yang dapat dimainkan.
2	Mrelfish		Pada NPC (*Non-Player Characters*) ini, merupakan NPC yang membantu pertahanan *hero,* dengan mencoba mengusir NPC Madpuff
3	Babybarac		NPC ini merupakan karakter musuh yang akan menyerang sarang. Ketika musuh ini mencapai sarang jumlah telut akan berkurang sebanyak 5.
4	Madpuff		NPC ini merupakan karakter musuh yang akan memangsa *hero,* apabila hero termangsa olehnya maka *life*

			poin akan berkurang 1.
7	Babyshrimp		Jika item ini diambil maka poin energi akan bertambah sebanyak 5.
8	Anemon		Anemon merupakan tempat penyimpanan telur atau sarang yang harus dilindungi oleh hero dan merupakan tujuan akhir dari yang memangsa telur

3.2.2.2 Story Board

Storyboard merupakan sketsa gambar yang disusun berurutan sesuai naskah yang ada, dan dengan storyboard kita dapat menyampaikan ide cerita kita kepada orang lain dengan mudah.

Storyboard level 1 dari aplikasi *game* ini, terdapat pada gambar 3.45



Gambar 3.45 *Storyboard level* 1

Pada *level* 1 ini, *hero* diharuskan mencapai daerah B dari daerah A tanpa bersinggungan dengan daerah C. Pada level ini lebih menekankan perkenalan user interface dan user control dalam game ini. Jika berhasil sampai di daerah B berarti level pertama terselesaikan.

Storyboard level 2, *level 3* dan *level* 4 dari aplikasi *game* ini, terdapat pada gambar 3.46



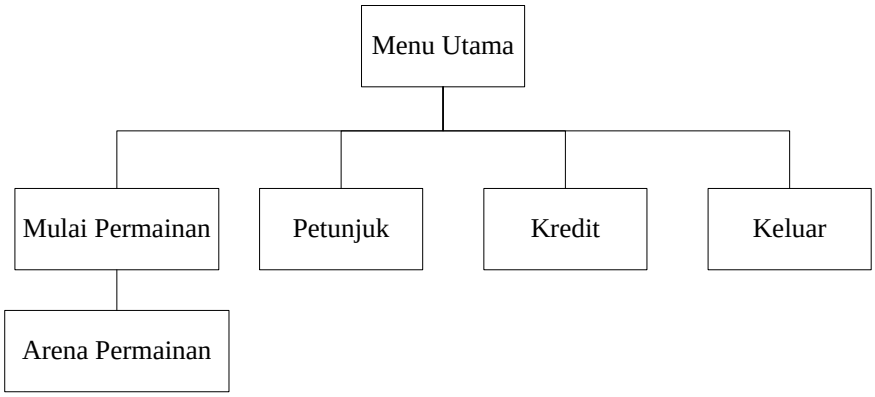
Gambar 3.46 *Storyboard level* 2, *level* 3 dan *level* 4

Pada *level* 2 ini, musuh akan keluar secara acak dari daerah A menuju daerah B(tempat sarang) dalam rentang waktu tertentu, musuh yang muncul pada *level* ini yaitu musuh yang hanya mengincar telur di daerah sarang. Hampir sama dengan level kedua, namun pada level ketiga daerah muncul dan keluarnya musuh berubah, yaitu musuh muncul dari daerah B menuju ke daerah A serta munculnya musuh yang akan mencoba memangsa *hero*.

Pada *level* 4 ini, musuh akan keluar secara acak dari daerah A menuju daerah B(tempat sarang) dalam rentang waktu tertentu. Musuh yang muncul pada *level* ini ada dua jenis musuh, yang pertama yaitu jenis musuh yang hanya mengincar telur di daerah sarang dan jenis yang kedua musuh yang akan mencoba memangsa *hero*. Jenis musuh pertama akan mencoba menuju sarang dan hero harus menghalaunya. Jenis musuh kedua akan mencoba memangsa *hero*, namun *hero* akan dibantu NPC yang akan mengusir musuh yang mengincar *hero*. NPC ini dianggap sebagai penghalang yang bergerak bagi jenis musuh yang kedua dalam upayanya memangsa hero. Jenis musuh yang ke dua telah diterapkan algoritma SMA* untuk mencari rute terpendek ke posisi hero, sedangkan NPC yang membantu her akan dicarikan rute terpendek untuk memangsa musuh jenis ke dua.

3.2.3 Perancangan Struktur Menu

Selain untuk mempermudah dalam 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 nantinya akan diakses terlebih dahulu setelah program selesai dibuat. Perancangan arsitektur menu pada aplikasi permainan ini dapat dilihat pada gambar 3.47.

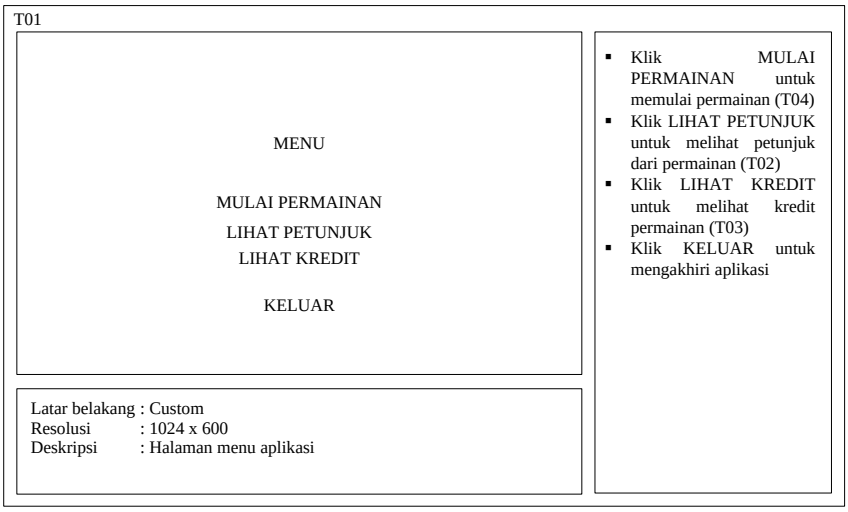


Gambar 3.47 Arsitektur menu aplikasi

3.2.4 Perancangan Antar Muka

1. Perancangan Antarmuka Menu Utama

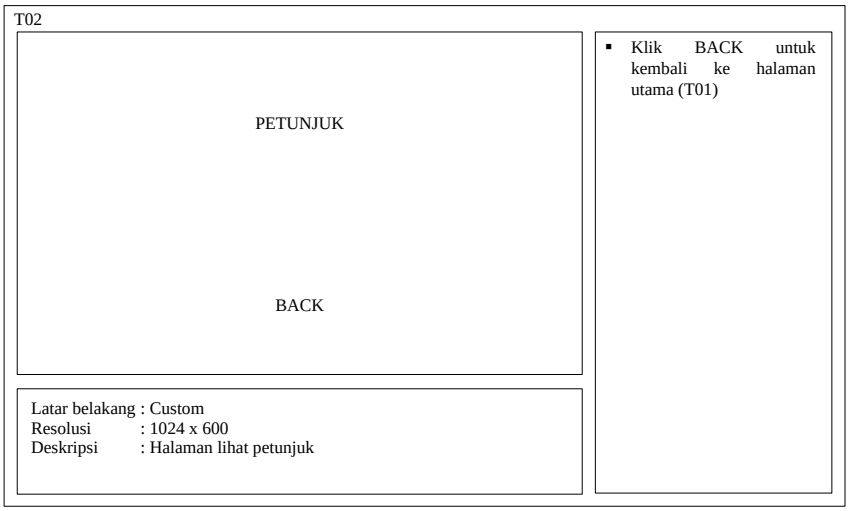
Perancangan antarmuka menu pada aplikasi permainan ini dapat dilihat pada gambar 3.48.



Gambar 3.48 Perancangan antarmuka menu

2. Perancangan Antarmuka Lihat Petunjuk

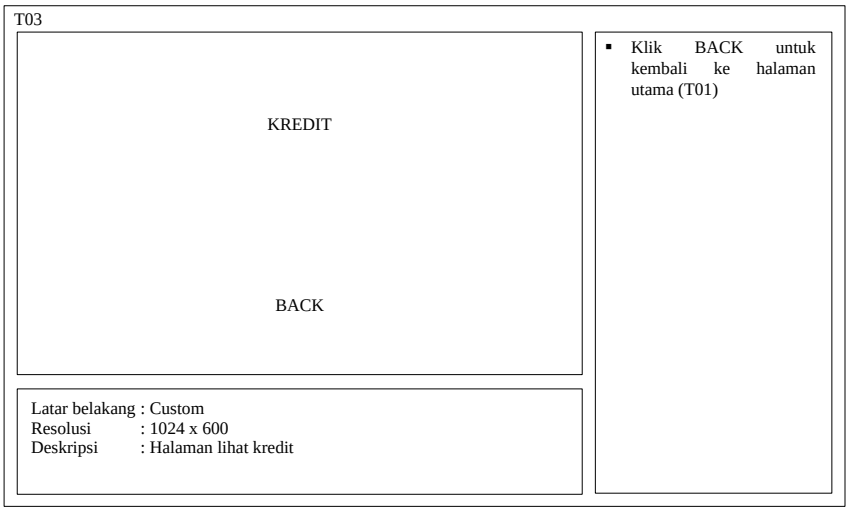
Perancangan antarmuka lihat petunjuk pada aplikasi permainan ini dapat dilihat pada gambar 3.49.



Gambar 3.49 Perancangan antarmuka lihat petunjuk

3. Perancangan Antarmuka Lihat Kredit

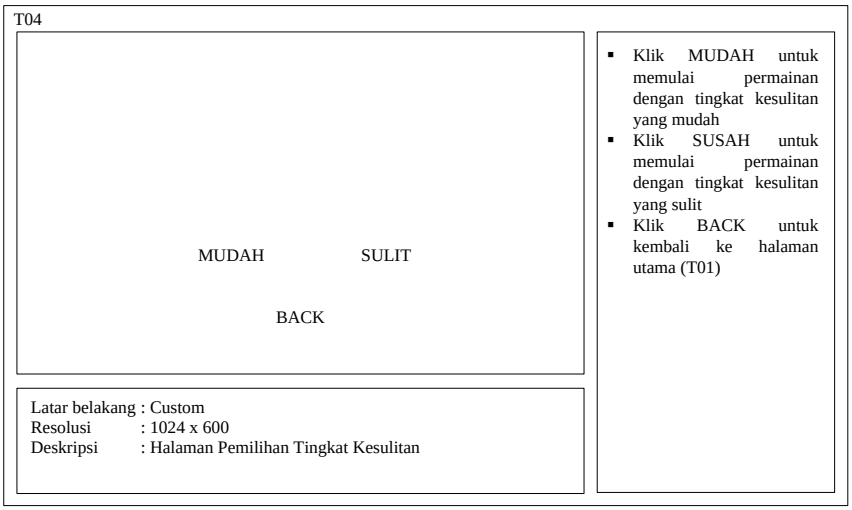
Perancangan antarmuka lihat kredit pada aplikasi permainan ini dapat dilihat pada gambar 3.50.



Gambar 3.50 Perancangan antarmuka lihat kredit

4. Perancangan Antarmuka Mulai Permainan

Perancangan antarmuka mulai permainan pada aplikasi permainan ini akan menampilkan pemilihan tingkat kesulitan yang dapat dilihat pada gambar 3.51.

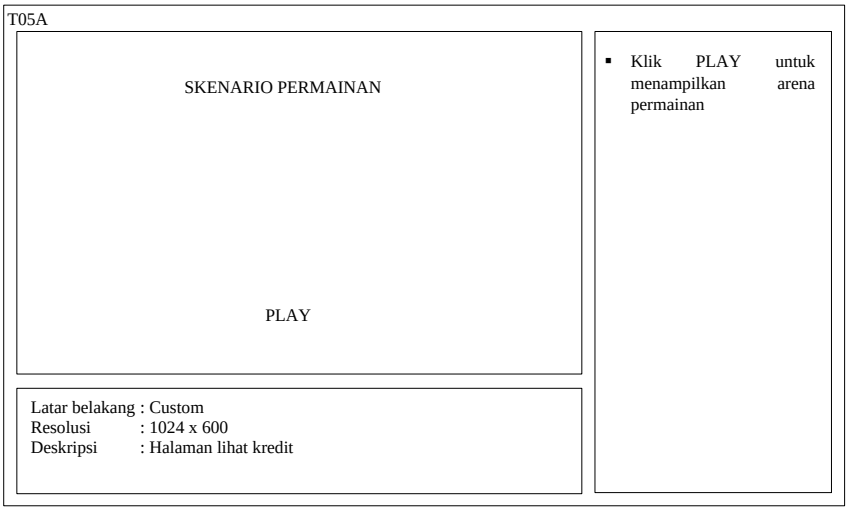


Gambar 3.51 Perancangan antarmuka mulai permainan

5. Perancangan Antarmuka Arena Permainan

a. Antarmuka Skenario Permainan

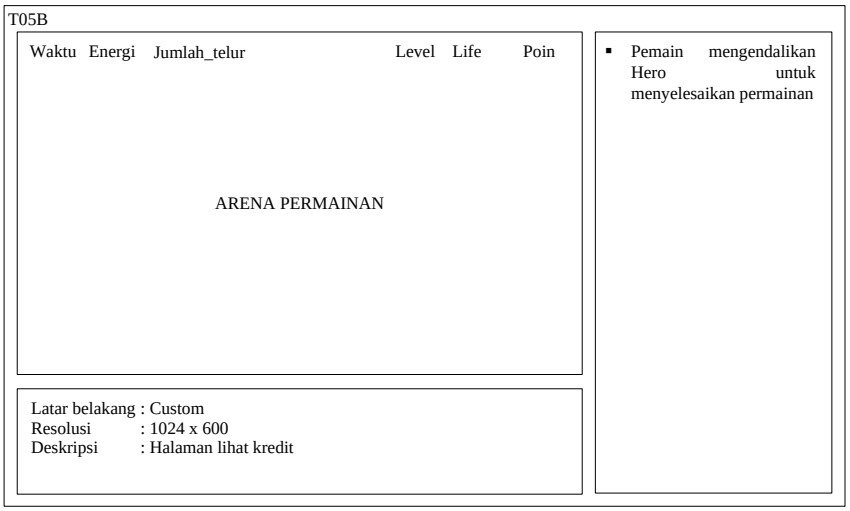
Perancangan antarmuka skenario permainan pada aplikasi permainan ini dapat dilihat pada gambar 3.52.



Gambar 3.52 Perancangan antarmuka skenario permainan

b. Antarmuka Arena Permainan

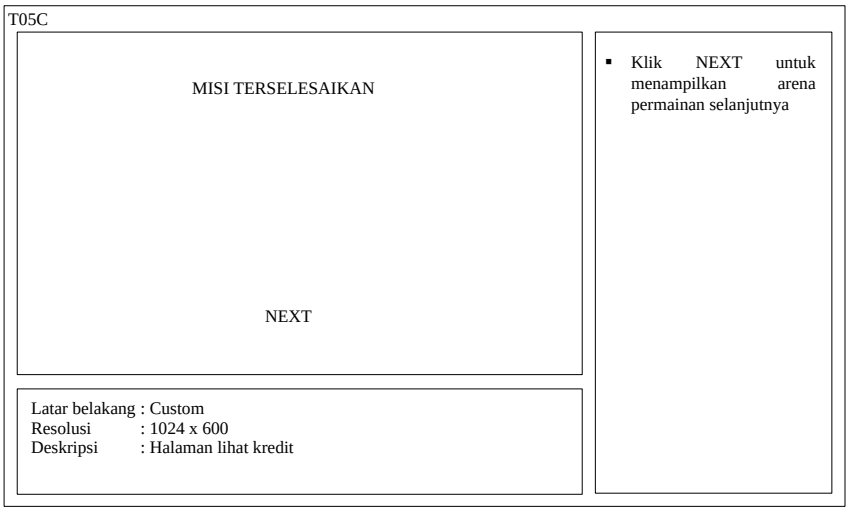
Perancangan antarmuka arena permainan pada aplikasi permainan ini dapat dilihat pada gambar 3.53.



Gambar 3.53 Perancangan antarmuka arena permainan

c. Antarmuka Misi Terselesaikan

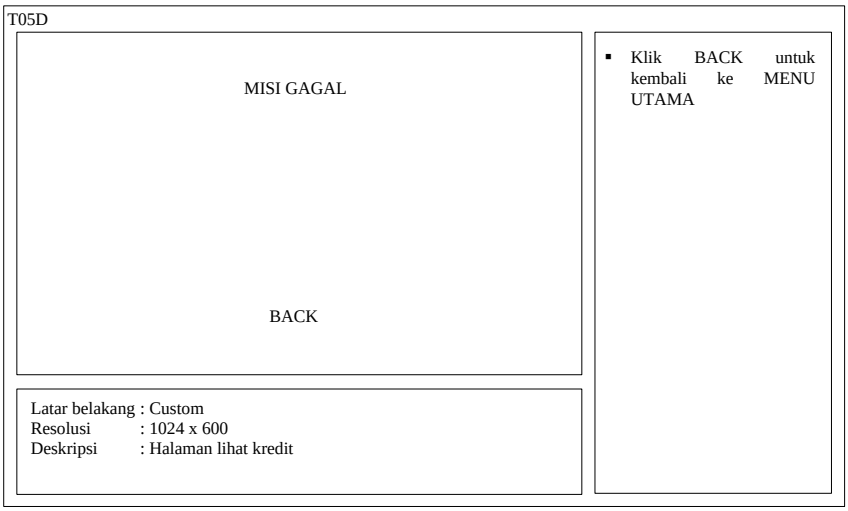
Perancangan antarmuka misi terselesaikan pada aplikasi permainan ini dapat dilihat pada gambar 3.54.



Gambar 3.54 Perancangan antarmuka misi terselesaikan

d. Antarmuka Permainan Gagal

Perancangan antarmuka permainan gagal pada aplikasi permainan ini dapat dilihat pada gambar 3.55.

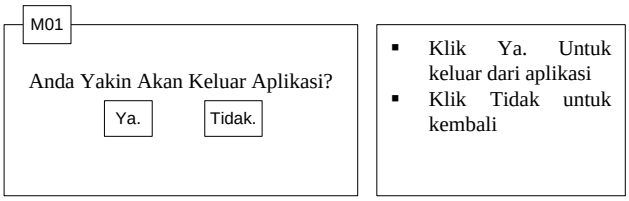


Gambar 3.55 Perancangan antarmuka permainan gagal

3.2.5 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.

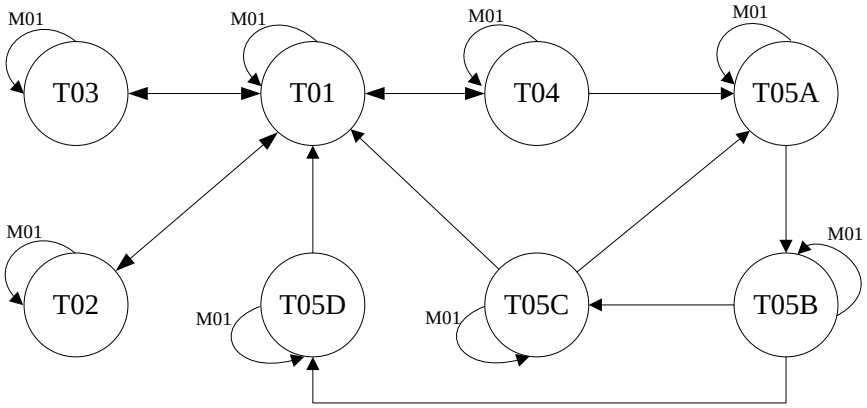
Perancangan pesan peringatan keluar aplikasi pada aplikasi permainan ini dapat dilihat pada gambar 3.56.



Gambar 3.56 Perancangan pesan peringatan keluar aplikasi

3.2.6 Jaringan Semantik

Jaringan semantik pada aplikasi permainan ini dapat dilihat pada gambar 3.57.



Gambar 3.57 Jaringan semantik aplikasi

3.2.7 Perancangan *Method*

Perancangan komponen *method* merupakan perancangan yang dibuat setelah perancangan arsitektur menu, antarmuka, pesan, dan jaringan semantik. Perancangan ini berfungsi untuk mendeskripsikan *method-method* yang berada di dalam aplikasi permainan.

3.2.7.1 Penggunaa *Method* yang Tersedia Pada *Tools* Pembangun

Berikut adalah daftar method yang tersedia dari IDE kemudian digunakan dalam pembangunan aplikasi permainan ini:

Tabel 3.22 Daftar method yang tersedia dari IDE

No.	Nama Method	Keterangan
1.	setup	Fungsi inisialisasi aplikasi utama
2.	level	Mengatur keluaran lingkungan permainan
3.	draw	Menampilkan *frame* objek yang terbentuk
4.	reset	Mereset aplikasi
5.	movectrl	Mengatur pergerakan *hero*
6.	move	Menampilkan pergerakan hero
7.	firstCutscene	Menampung sprite scene awal
8.	lastCutscene	Menampung sprite scene akhir
9.	show	Menampilkan animasi musuh

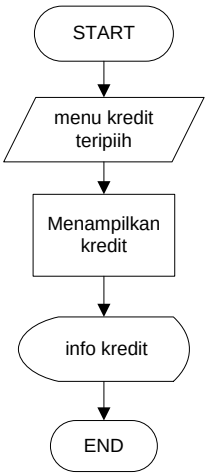
10.	motion	Menampilkan pergerakan musuh
11.	kill	Memperbaharui info pemain
12.	wallBumpCheck	Memeriksa tabrakan antar objek entitas yang terbentuk
13.	fshow	Menampilkan animasi makanan
14.	fmotion	Menampilkan pergerakan makanan
15.	fkill	Memperbaharui info pemain
16.	render	Menampung grid arena permainan yang dihasilkan
17.	mousePressed	Mengatur proses input dari mouse
18.	hero	Menampilkan animasi hero

3.2.1.2 Perancangan Method yang Akan Digunakan

Adapun *method-method* yang terdapat dalam aplikasi permainan yang akan dibangun untuk kebutuhan aplikasi ini adalah sebagai berikut :

A. *Method* Credits

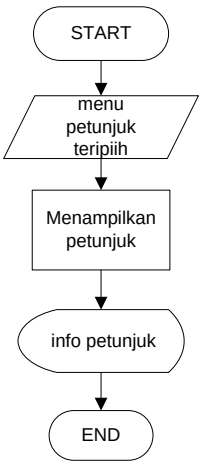
Method Credits akan dipanggil ketika memilih menu Kredit, dapat dilihat pada gambar 3.59.



Gambar 3.59 *Flowchart method* credits

B. *Method* Instructions

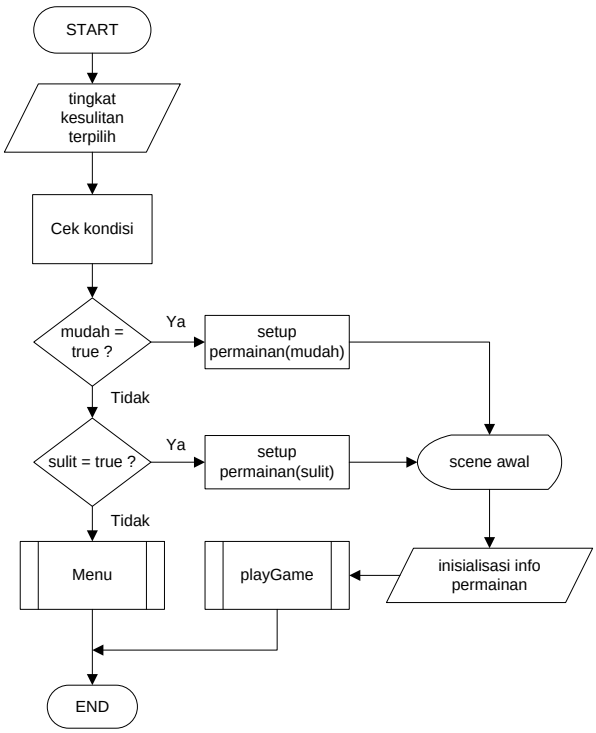
Method instructions akan dipanggil ketika memilih menu petunjuk, dapat dilihat pada gambar 3.60.



Gambar 3.60 *Flowchart method* instructions

C. *Method* Diff

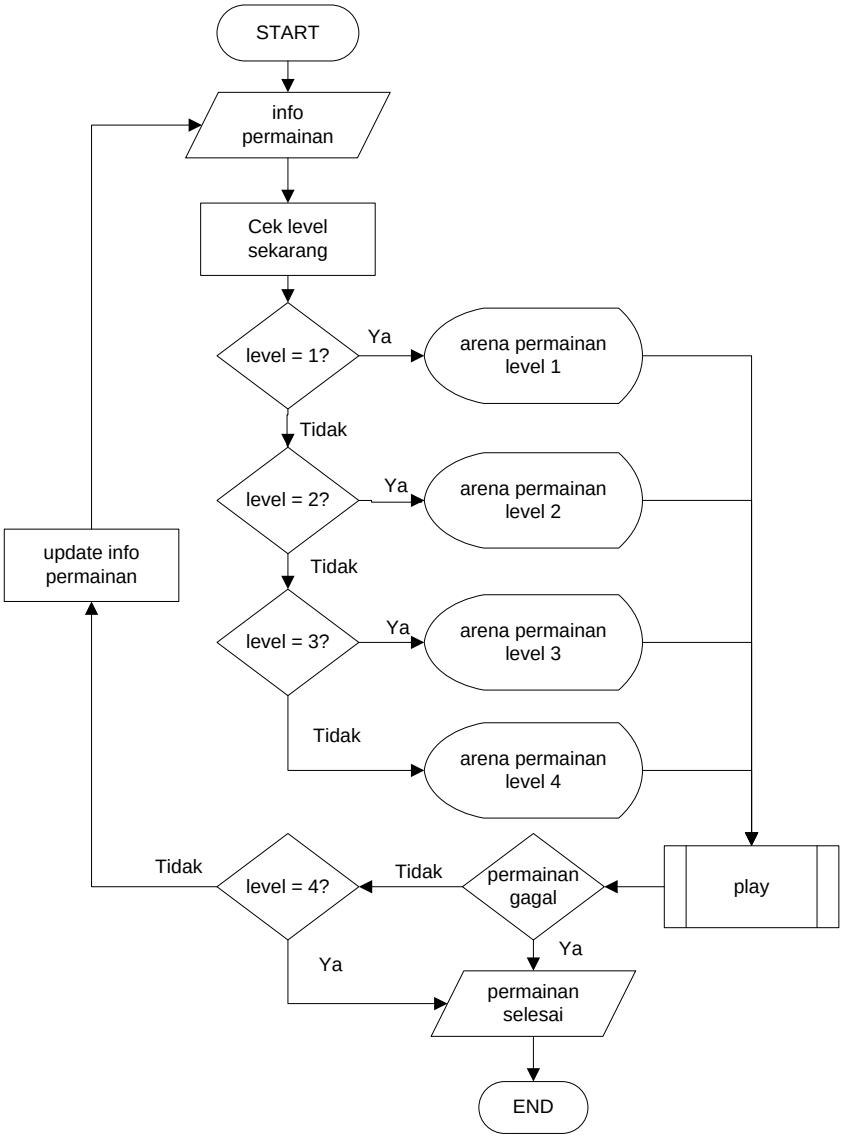
Method diff akan dipanggil menu ketika memilih mulai permainan kemudian memilih tingkat kesulitan permainan yang akan dimainkan, dapat dilihat pada gambar 3.61.



Gambar 3.61 *Flowchart method* diff

D. *Method* PlayGame

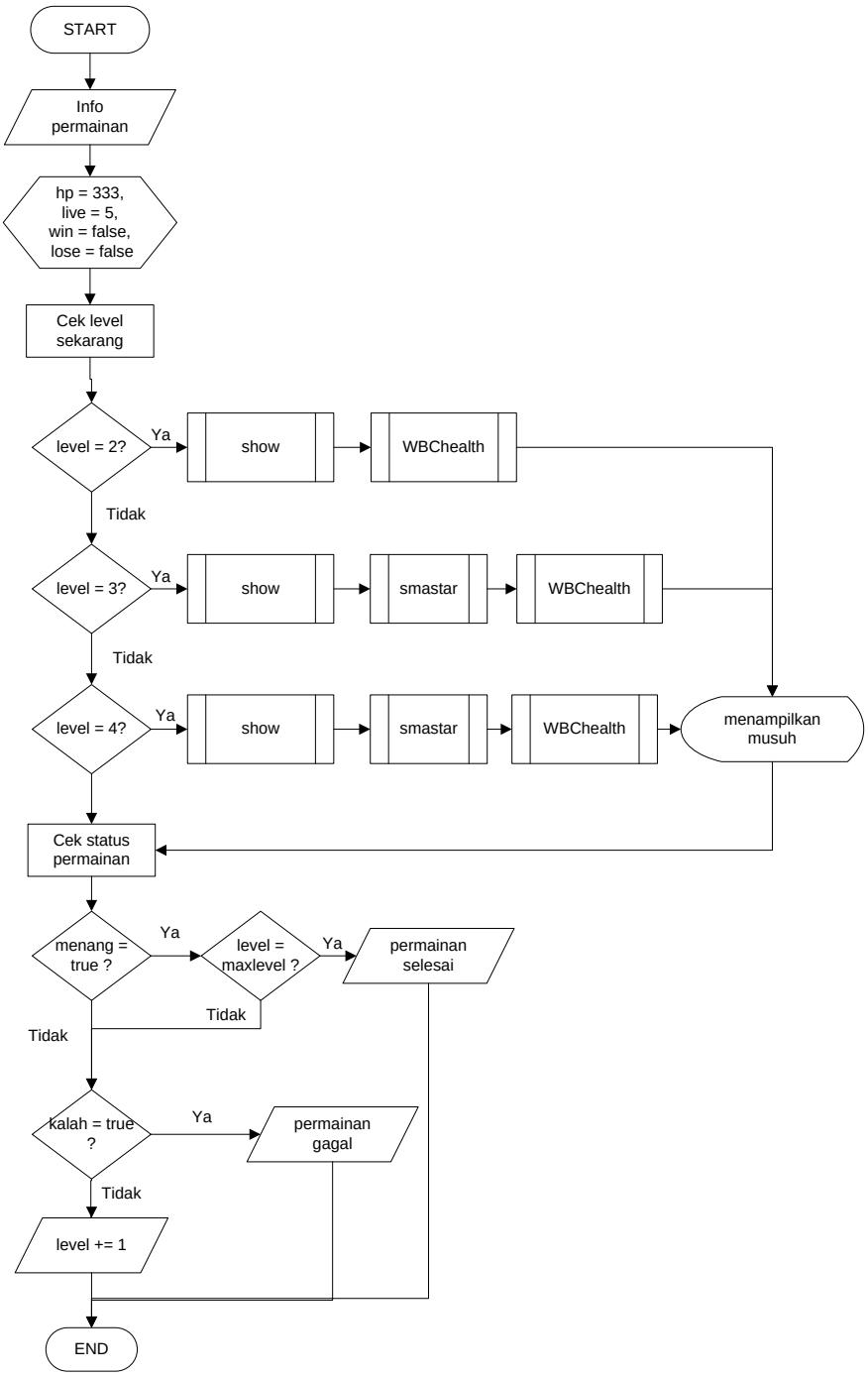
Method playGame akan dipanggil dalam *Method* diff dapat dilihat pada gambar 3.62.



Gambar 3.62 *Flowchart method* playGame

E. *Method* Play

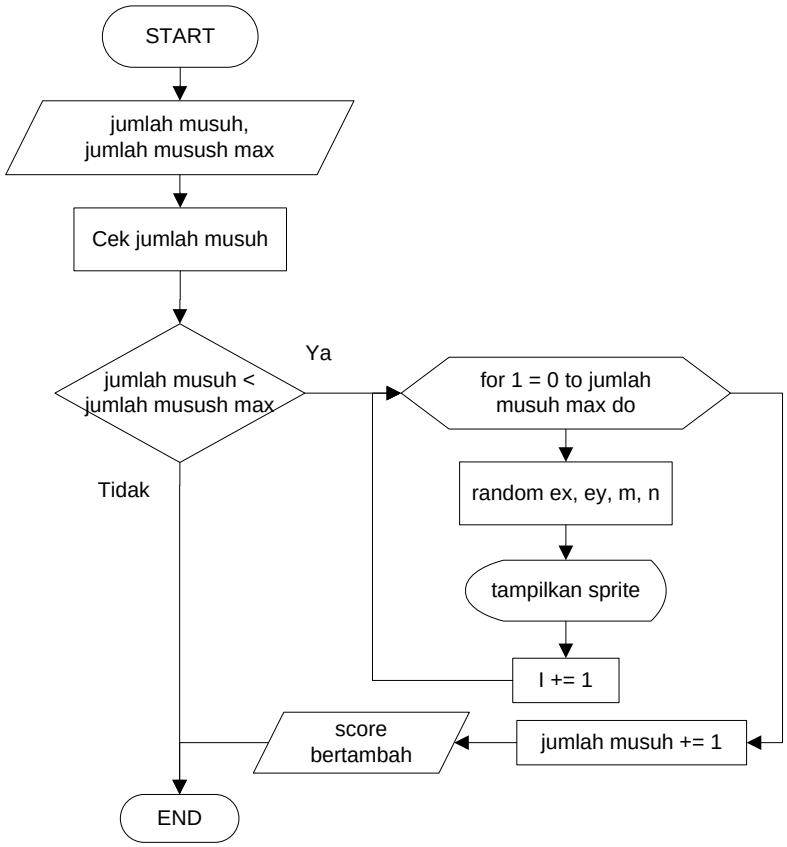
Method play akan dipanggil untuk memulai permainan, dapat dilihat pada gambar 3.63.



Gambar 3.63 *Flowchart method* play

F. *Method* Show

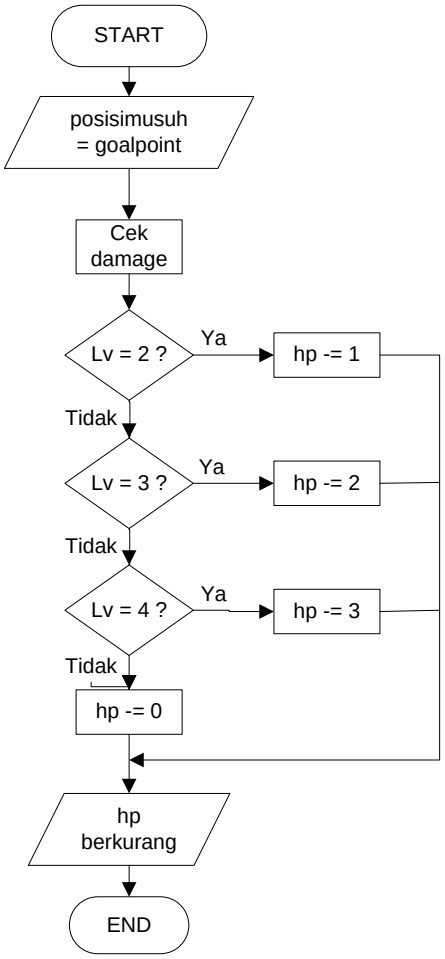
Method show akan dipanggil jika hero berhasil menghalau musuh, dapat dilihat pada gambar 3.64.



Gambar 3.64 *Flowchart method* mobSpawner

G. *Method* WBChealth

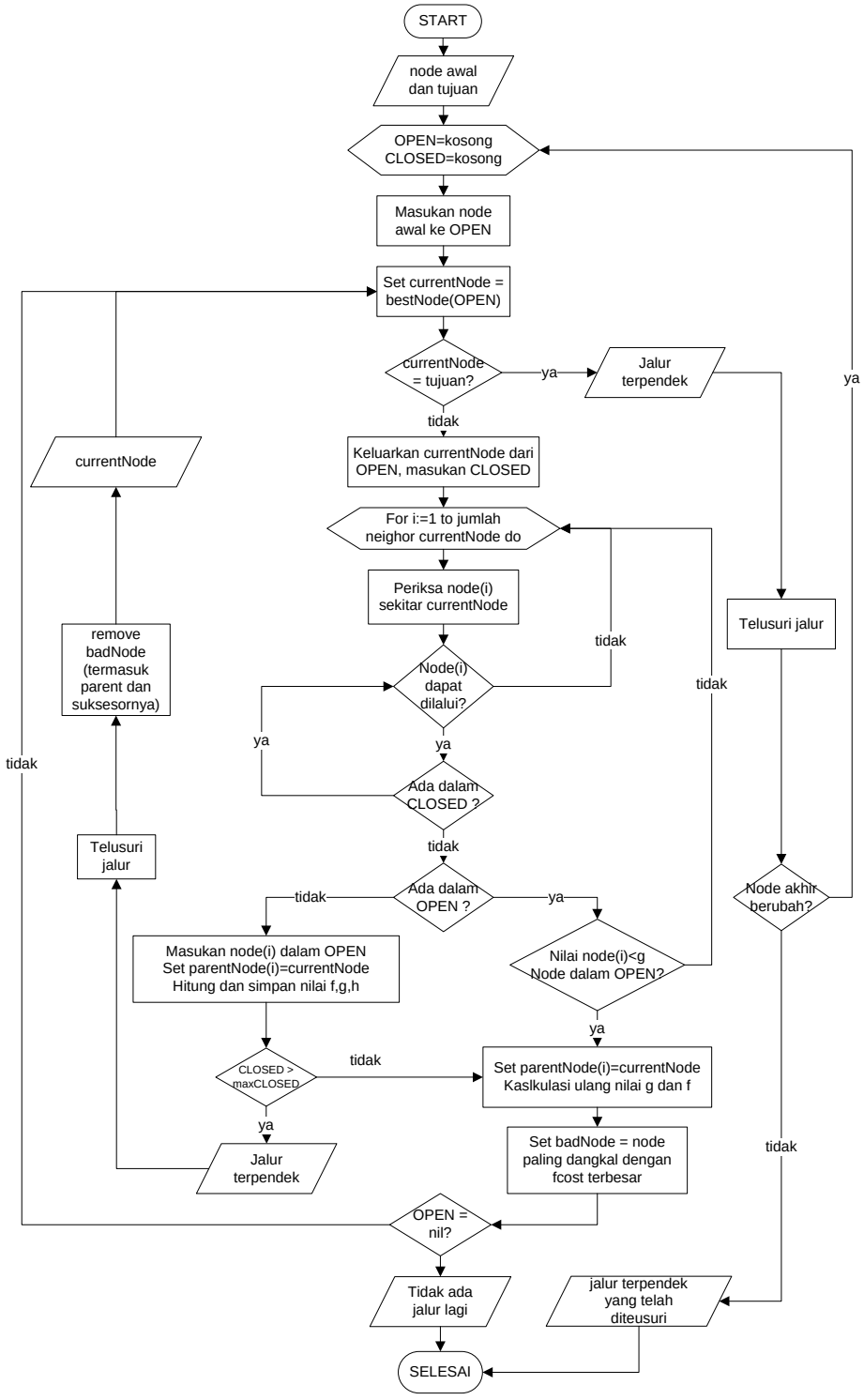
Method WBChealt akan dipanggil jika musuh berhasil mencapai daerah yang harus dilindungi (*goal point)* oleh *hero*, dapat dilihat pada gambar 3.65.



Gambar 3.65 *Flowchart method* WBChealth

H. *Method* Smastar

Method smastar akan dipanggil pada *method* play sebagai *pathfinding* dapat dilihat pada gambar 3.66.



Gambar 3.66 *Flowchart method* smastar