BAB III

ANALISIS DAN PERANCANGAN SISTEM

Pada bab ini akan dibahas mengenai analisis dan perancangan pada sistem yang dibangun yakni penerapan algoritma A* dalam pencarian jalan terpendek pada *game pathfinding*.

III.1 Deskripsi Masalah

Deskripsi masalah adalah suatu gambaran masalah yang diangkat dalam penulisan skripsi tentang penerapan algoritma A*. Algoritma A* *(A Star)* merupakan perbaikan dari metode BFS dengan memodifikasi fungsi *heuristic*nya. A* akan meminimumkan total biaya lintasan. Pada kondisi yang tepat, A* akan memberikan solusi yang terbaik dalam waktu yang optimal.

Algoritma A* *(A Star)* akan diterapkan dalam pencarian jalan terpendek pada *game pathfinding*. Pencarian jalan terpendek pada *game pathfinding* akan diasumsikan dengan titik awal (keadaan pertama) dan titik tujuan (keadaan akhir) yang diimplementasikan pada pencarian di ruang kosong maupun di ruang dengan penghalang seperti halnya permainan labirin. Ruang yang akan dibangun akan berbentuk *puzzle* (kotak) supaya mempermudah dalam penerapan cara kerja algoritma A*. Hasil pencarian algoritma tersebut akan menghasilkan jalan terpendek dari suatu titik awal ke titik tujuan.

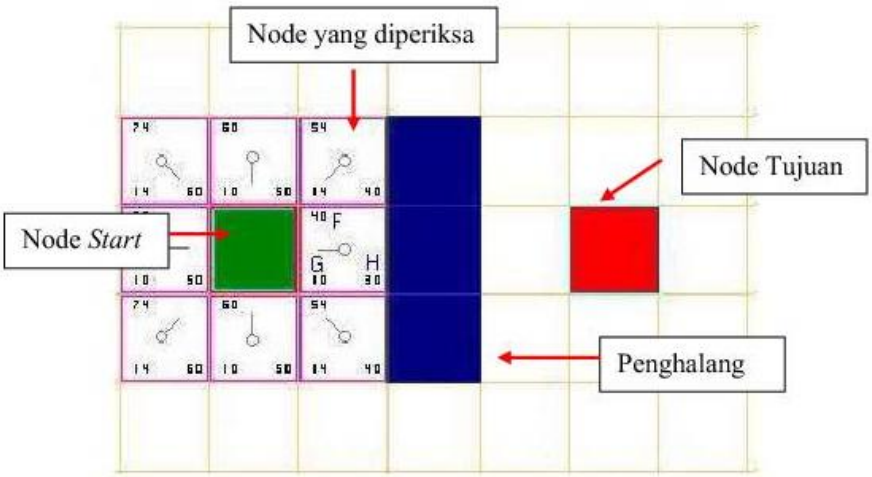
Untuk mengetahui performansi algoritma A* maka akan diadakan perbandingan dengan algoritma lain. Algoritma A* *(A Star)* adalah

pengembangan dari BFS yang digunakan untuk mencari jalan terpendek *(shortest path)* yang sering dipakai dalam *game programming*.

III.2 Analisis Kasus

Algoritma A* *(A Star)* akan diterapkan pada analisis kasus pencarian jalan terpendek menuju tujuan pada *game pathfinding* untuk mencari kecepatan, jarak langkah menuju tujuan dan simpul yang diperiksa.

Dalam kondisi jarak yang ditempuh sangat jauh dan memiliki banyak rintangan algoritma A* cocok untuk mencari solusi yang terbaik. Setiap pergerakan yang dilakukan pada kotak statusnya akan disimpan pada suatu *list*. *List* ini akan digunakan untuk melakukan pengecekan apakah kita sudah pernah membangun status tersebut atau belum agar kita tidak menggerakkan kotak yang sama berkali-kali ke status yang sama. Dengan menerapkan strategi ini, selain menemukan solusi, algoritma ini juga bisa menemukan langkah terpendek untuk mencapai solusi tersebut. Untuk lebih jelasnya, cara kerja algoritma A* dapat dilihat pada gambar III.1.

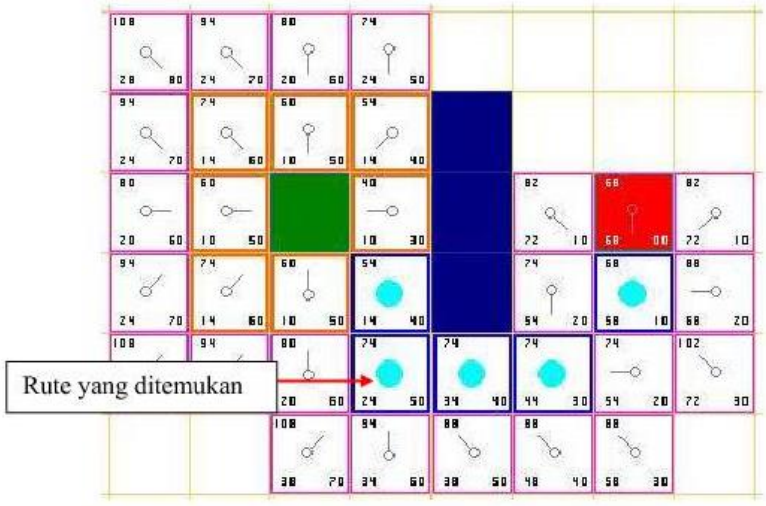


Gambar III.1. Proses algoritma A*

Pada gambar III.1, kotak warna hijau di sebelah kiri adalah *node* awal, dan kotak merah di sebelah kanan adalah tujuan. Tiga kotak biru di tengah adalah halangan yang tidak bisa dilewati. Angka-angka dalam kotak masing-masing adalah nilai f (kiri atas) dan h (kanan bawah). Tanda lingkaran dan garis di tengah kotak menunjukkan *parent* kotak tersebut. Untuk pergerakan horizontal dan vertikal, *cost*nya adalah sepuluh, sedangkan untuk pergerakan diagonal memakan *cost* 14.

Pada kotak di sebelah kanan *node* awal, g bernilai sepuluh dan h bernilai 30. Perhitungannya sederhana saja, g bernilai sepuluh sebab dari *node* awal ke kotak tersebut hanya perlu bergerak horizontal satu kali. H bernilai 30 didapat dari banyaknya langkah yang diperlukan untuk mencapai tujuan dari kotak tersebut.

Selanjutnya, algoritma ini akan memeriksa kotak mana yang bernilai f-nya terkecil dan mengembangkan jalan di sekitar kotak tersebut. Dengan cara seperti ini telah dijelaskan di atas sebelumnya, hasil akhir ini adalah seperti gambar III.2.



Gambar III.2. Hasil akhir pencarian A*

Pada gambar III.2, *rute* yang ditemukan digambarkan dengan bujur sangkar berbingkai biru dengan lingkaran biru muda di tengahnya. Maka jika diimplementasikan pada *game pathfinding* suatu karakter akan melewati halangan dan akhirnya akan menemukan *rute* terpendek ke tujuan.

III.3 Analisis Masalah

Game pathfinding akan berupa kotak-kotak dengan ordo X x Y, supaya pengguna bisa menentukan sendiri orde yang diinginkan.Pada ordo X x Y, 2 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 III.1.

Tabel III.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 = ...

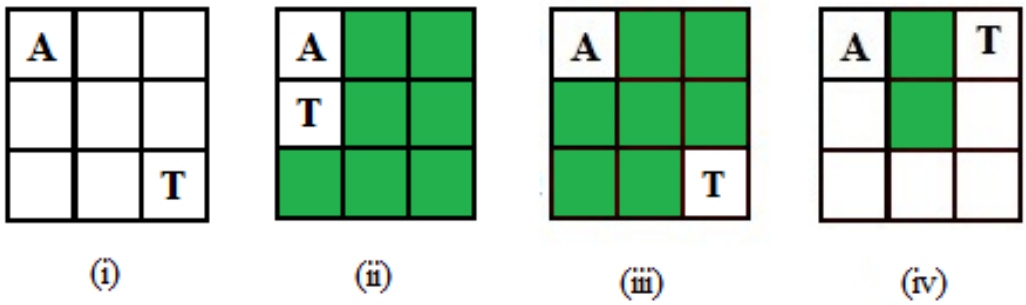
Keterangan :

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 titik awal dan titik tujuan saling berjauhan maka akan mengahasilkan 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 3x3 pada gambar III.3.



Gambar III.3. Ruang peta (*map*) ordo 3 x 3 dengan tiga kondisi

Keterangan :

A : Titik awal
T : Titik Tujuan
: Penghalang

Maksimal penghalang yang akan dipasangkan pada ordo lainnya adalah sebanyak ordo tersebut dikurangi dua untuk menempatkan titik awal dan titik tujuan. Pada gambar III.3 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 *orange* dan warna hijau diimplementasikan sebagai penghalang, yaitu kotak yang tidak dapat dilalui oleh titik awal.

Sekarang, kita akan mencari jalan terpendek dari posisi titik awal ke posisi titik tujuan. 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.

III.4 Analisis Pemecahan Masalah dengan Algoritma A* (*A Star*)

Terdapat beberapa hal yang perlu didefinisikan terlebih dahulu dalam kasus *game pathfinding* dengan penerapan algoritma A* *(A Star)*. Adapun istilah-istilah yang akan dibahas yaitu *path, open list, closed list,* nilai f, g dan n.

Algoritma A* menggunakan dua senarai yaitu *OPEN* dan *CLOSED*. *OPEN* adalah senarai (*list*) yang digunakan untuk menyimpan simpul-simpul yang pernah dibangkitkan dan nilai heuristiknya telah dihitung tetapi belum terpilih sebagai simpul terbaik (*best node*) dengan kata lain, *OPEN* berisi simpul-simpul masih memiliki peluang untuk terpilih sebagai simpul terbaik, sedangkan *CLOSED* adalah senarai untuk menyimpan simpul-simpul yang sudah pernah dibangkitkan dan sudah pernah terpilih sebagai simpul terbaik. Artinya, *CLOSED* berisi simpul-simpul yang tidak mungkin terpilih sebagai simpul terbaik (peluang untuk terpilih sudah tertutup).

1. *OPEN LIST* adalah *list* yang menyimpan kemungkinan *path* yang akan diperiksa. *OPEN LIST* dibuat terurut berdasarkan nilai f. *OPEN LIST* digunakan untuk menentukan secara selektif (berdasarkan nilai f) jalan yang dikira lebih dekat menuju pada *path* tujuan. *OPEN* berisi simpul-simpul yang masih memiliki peluang untuk terpilih sebagai simpul terbaik (*best node*).
2. *CLOSED* adalah senarai (*list*) untuk menyimpan simpul-simpul yang sudah pernah dibangkitkan dan sudah pernah terpilih sebagai simpul terbaik (*best node*) atau senarai yang menyimpan jalan yang sudah diperiksa dari *open list*. Artinya, *CLOSED* berisi simpul-simpul yang tidak mungkin terpilih sebagai simpul terbaik (peluang untuk terpilih sudah tertutup). Kedua list (*OPEN*

LIST dan *CLOSED LIST*) ini bertujuan juga untuk menghindari penelusuran berkali-kali jalan (*rute*) yang memang sudah diidentifikasi agar tidak masuk kembali ke dalam *OPEN LIST*.

3. Nilai F adalah *cost* perkiraan suatu *path* yang teridentifikasi. Nilai F merupakan hasil dari f(n).
4. Nilai G hasil dari fungsi g(n), adalah banyaknya langkah yang diperlukan untuk menuju ke *path* sekarang.
5. Setiap simpul *(node)* harus memiliki informasi nilai h(n), yaitu estimasi harga simpul tersebut dihitung dari simpul tujuan yang hasilnya menjadi nilai H. [1]

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

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

dengan :

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

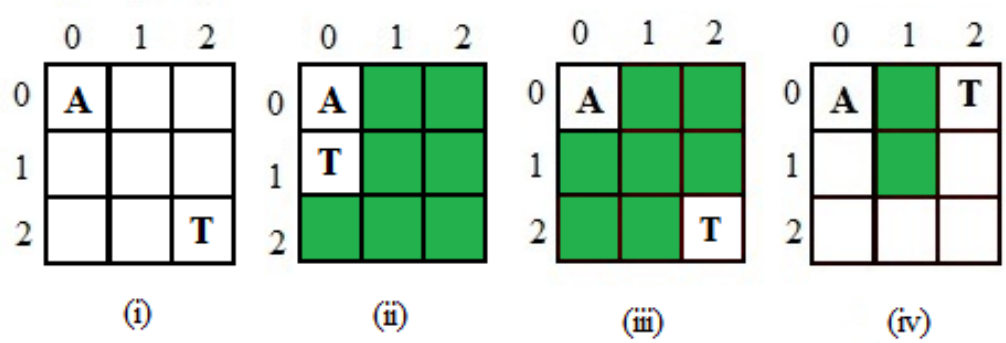
$g(n)$ = biaya *(cost)* yang dikeluarkan dari keadaan awal sampai keadaan n

$h(n)$ = estimasi biaya untuk sampai pada suatu tujuan mulai dari n [18]

Pergerakan diagonal pada *map* diperbolehkan, maka digunakan fungsi *heuristic Non-Manhattan Distance*. Maka fungsi *heuristic* yang digunakan adalah sebagai berikut :

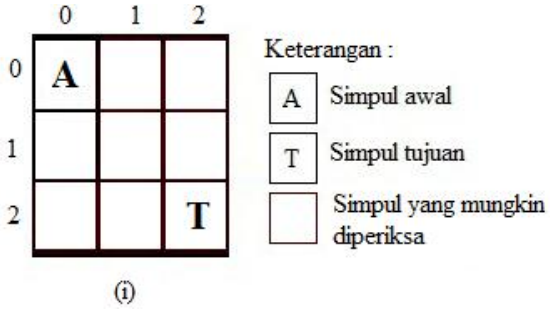
```
h_diagonal(n) = min(abs(n.x-goal.x) + abs(n.y-goal.y))
h_orthogonal(n) = (abs(n.x-goal.x) + abs(n.y-goal.y))
h(n) = h_diagonal(n) + (h_orthogonal (n) – (2 * h_diagonal(n)))
```

Sudah dijelaskan pada analisis masalah bahwa ordo dapat disesuaikan dengan rentang X x Y, salah satu contoh perhitungan pada *game pathfinding* ini akan dijelaskan dengan ordo minimal yaitu 3x3 dikarenakan perhitungan pada ordo berapa pun akan sama. Pada penentuan bobot setiap *node* akan diberikan nilai sesuai dengan jarak terdekat ke tujuan, misal *node* yang terjauh dari tujuan maka diberi bobot yang kecil sedangkan *node* yang terdekat dengan tujuan diberi bobot yang lebih besar. Contoh perhitungannya seperti pada gambar III.4 yang diberi nilai bobot yang terkecil yaitu 1 dan bobot selanjutnya ditambahkan 1.



Gambar III.4. Contoh kondisi ruang *map* yang akan dihitung dengan A*

Perhitungan yang dilakukan dengan algoritma A* dengan kondisi tanpa penghalang yang terlihat pada gambar III.5.



Gambar III.5. Contoh kondisi (i) tanpa penghalang dengan pencarian A*

Posisi simpul awal = Ax : 0, Ay : 0

Posisi simpul tujuan = goal x : 2, 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))

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)))

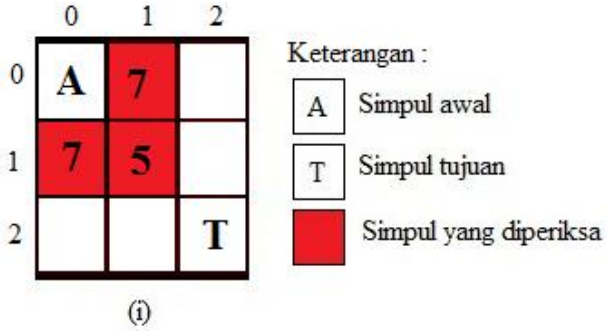
= -3 + 9

= 6

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

= 1 + 6

= 7



Gambar III.6. Langkah pertama pencarian *BestNode* pada kondisi (i)

Pada gambar III.6 terdapat tiga simpul yang mungkin menjadi *BestNode* yaitu (1,0) dengan f(n)=7, (1,1) dengan f(n)=5 dan (0,1) dengan f(n)=7. Dari ke tiga simpul yang mungkin maka dipilihlah simpul (1,1) 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)))

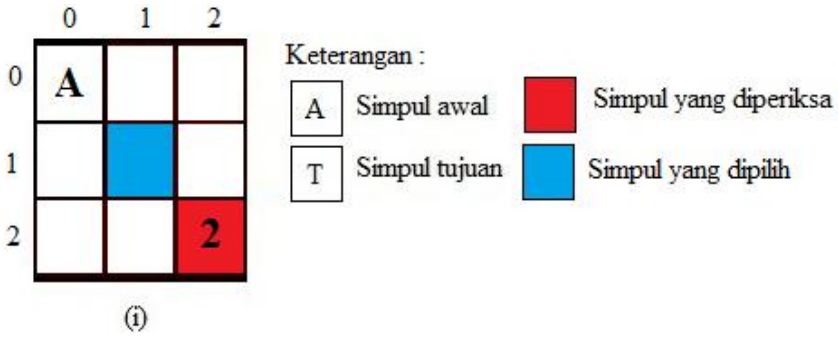
= 0 + 0

= 0

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

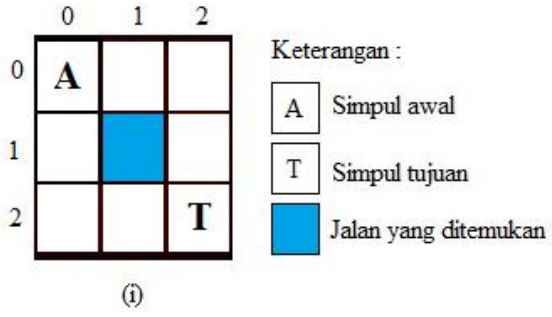
= 2 + 0

= 2



Gambar III.7. Langkah ke dua pencarian *BestNode* pada kondisi (i)

Pada gambar III.7 terdapat satu simpul yang mungkin menjadi *BestNode* yaitu (2,2) dengan f(n)=2, dan dikenali sebagai simpul tujuan yaitu (2,2) berarti solusi telah ditemukan.

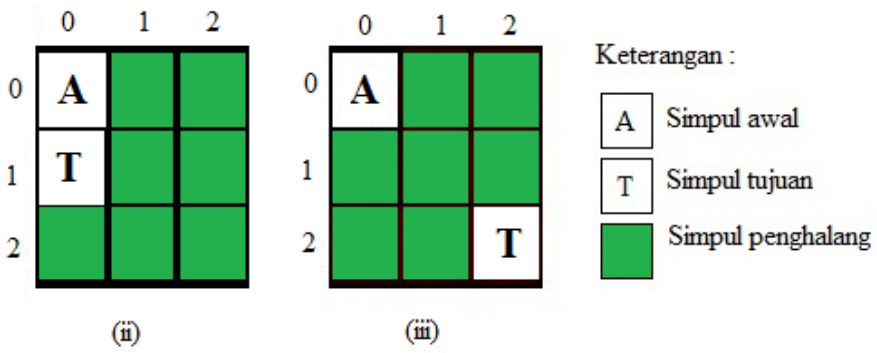


Gambar III.8. Hasil pencarian jalan dengan Algoritma A* pada kondisi (i)

Dari semua perhitungan yang telah dilakukan dipilihlah biaya/*cost* terkecil pada setiap langkahnya sehingga akan menghasilkan jalur terpendek yang terlihat pada gambar III.8.

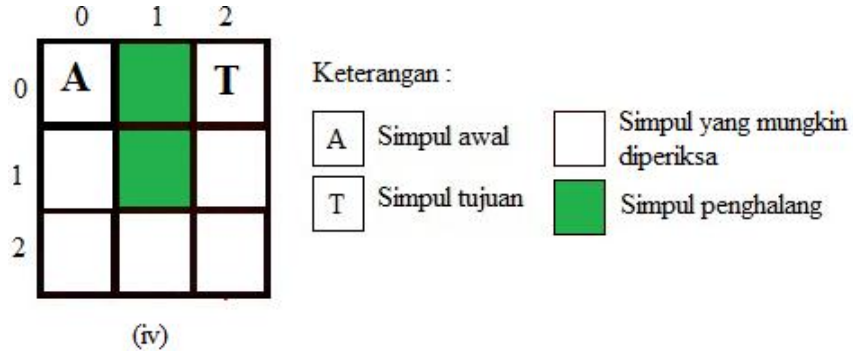
Pada contoh ordo 3x3 dengan kondisi penghalang maksimal dan simpul awal dengan simpul tujuan saling berdekatan pada kondisi (ii) dengan posisi simpul awal yaitu (0,0) dan posisi simpul tujuan yaitu (0,1), maka pencarian simpul yang diperiksa hanya satu simpul saja yaitu simpul (0,1) karena simpul lain yang berdekatan dengan posisi simpul awal merupakan penghalang yang tidak bisa dilewati, kemudian simpul (0,1) dikenali sebagai simpul tujuan yang berarti solusi telah ditemukan.

Jika kondisi simpul awal dengan simpul tujuan saling berjauhan pada kondisi (iii) dimanapun posisinya dengan penghalang maksimal, maka solusi tidak akan ditemukan sebab tidak ada jalan yang dapat dilewati untuk menuju tujuan. Kondisi (ii) dan (iii) dapat dilihat pada gambar III.9.



Gambar III.9. Contoh kondisi (ii) dan (iii) dengan maksimal penghalang

Pada contoh ordo 3 x 3 dengan kondisi (iv) yang terlihat pada gambar III.10, akan dihitung pencariannya dengan algoritma A*.



Gambar III.10. Contoh kondisi (iv) dengan pencarian A*

Posisi simpul awal = Ax : 0, Ay : 0

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

Langkah ke satu

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

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

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

= min 3

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

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

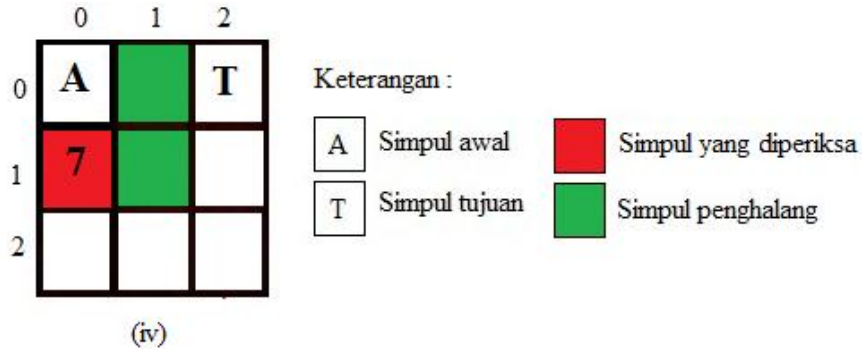
= -3 + 9

= 6

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

= 1 + 6

= 7



Gambar III.11. Langkah pertama pencarian *BestNode* pada kondisi (iv)

Pada gambar III.11 terdapat satu simpul yang mungkin menjadi *BestNode* yaitu (0,1) dengan f(n)=7, maka dipilihlah simpul (0,1) dengan biaya yaitu 7.

Langkah ke dua

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

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

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

= min 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

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

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

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

= min 3

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

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

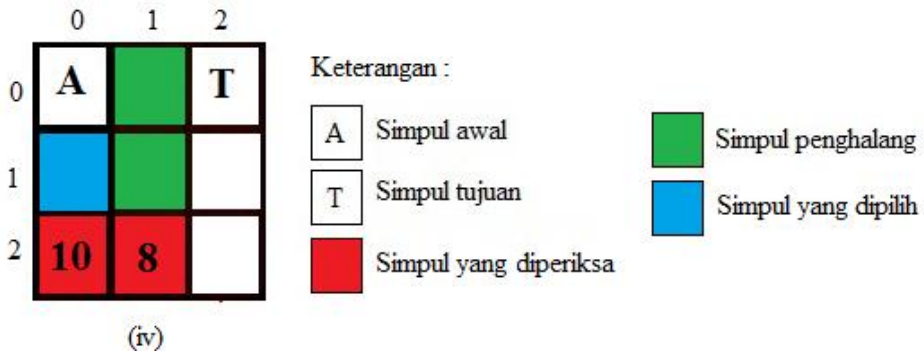
= -3 + 9

= 6

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

= 2 + 6

= 8



Gambar III.12. Langkah ke dua pencarian *BestNode* pada kondisi (iv)

Pada gambar III.12 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.

Langkah ke tiga

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

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

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

= min 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

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

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

= 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)))

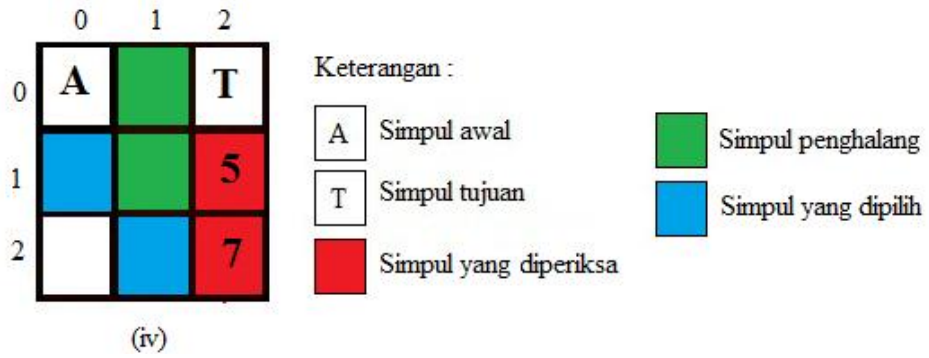
= -1 + 3

= 2

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

= 3+2

= 5



Gambar III.13. Langkah ke tiga pencarian *BestNode* pada kondisi (iv)

Pada gambar III.13 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.

Langkah ke empat

n (2,0) : g (2,0) = 4

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

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

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

= 0

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

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

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

= min 0

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

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

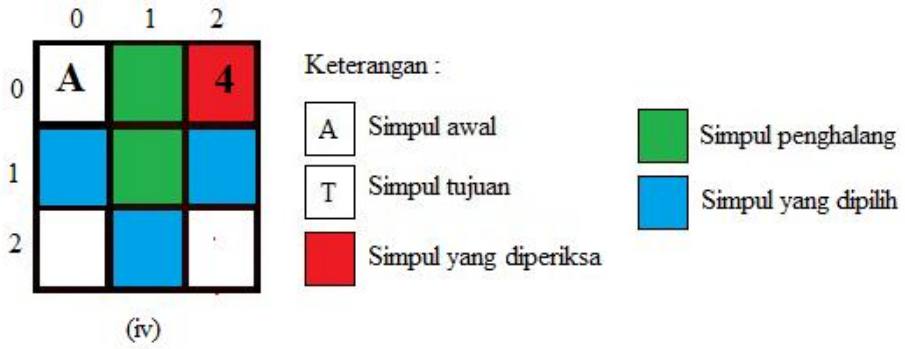
= 0 + 0

= 0

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

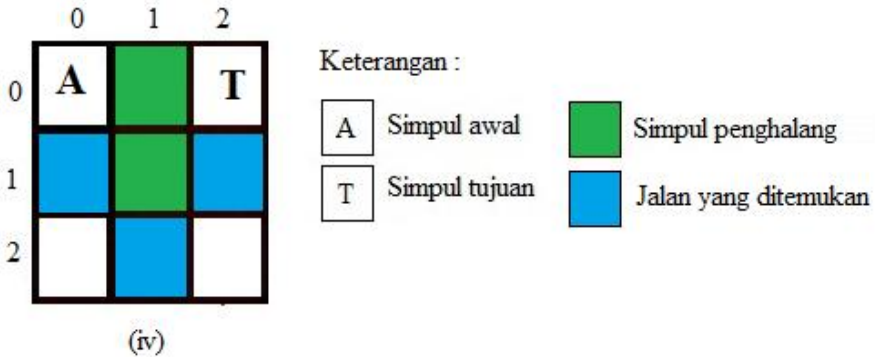
= 4 + 0

= 4



Gambar III.14. Langkah ke empat pencarian *BestNode* pada kondisi (iv)

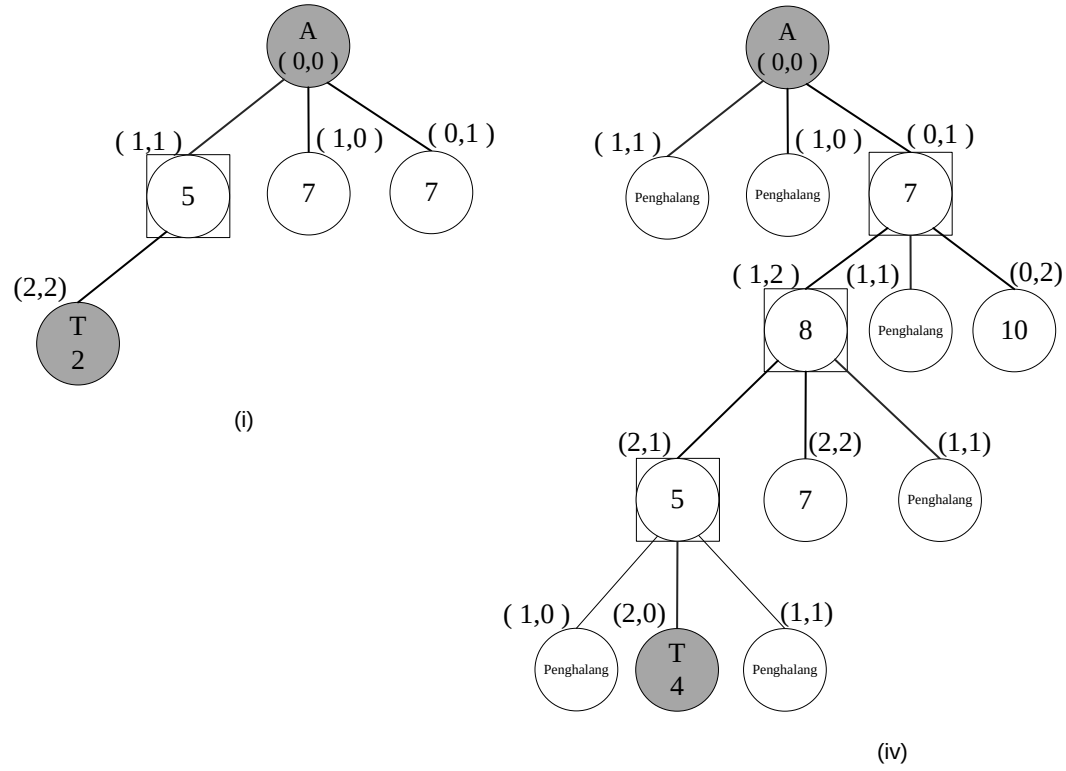
Pada gambar III.14 terdapat satu simpul (2,0) dengan f(n)=4 dan dikenali sebagai simpul tujuan yang berarti solusi telah ditemukan.



Gambar III.15. Hasil pencarian jalan dengan Algoritma A* pada kondisi (iv)

Dari perhitungan di atas dipilihlah biaya/*cost* terkecil pada setiap langkahnya sehingga akan menghasilkan jalur terpendek yang terlihat pada gambar III.15. Salah satu masalah dari algoritma A* yaitu menggunakan *memory* yang besar untuk menyimpan simpul (*node*) sebelumnya atau *path* (jalan) sebelumnya. Contoh kondisi ruang *map* yang akan dihitung dengan A* pada gambar III.4 bila digambarkan dengan pohon pelacakan, maka pohonnya dapat

dilihat pada gambar III.16, kecuali ruang *map* dengan kondisi (ii) karena simpul yang mungkin hanya satu.



Gambar III.16. Pohon pencarian jalan untuk masalah pada gambar III.4 dengan Algoritma A*

Pada gambar III.16 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. Untuk kondisi (i) pada gambar III.16 dapat dijelaskan langkahnya sebagai berikut :

a. Langkah pertama dalam kondisi (i), karena di *OPEN* hanya terdapat satu simpul yaitu A(0,0), maka A terpilih sebagai *BestNode* dan dipindahkan ke

CLOSED. 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)].

b. 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*. Oleh karena itu, *parent* harus diubah yang semula A(0,0) menjadi simpul (1,1). 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*, kemudian dicek nilai f nya. Setelah dihitung f(n) = 2 dengan simpul (2,2) terpilih sebagai *BestNode* dan sama dengan tujuan yang berarti solusi sudah ditemukan. 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 III.2. Jalur ini merupakan jalan yang dihasilkan oleh algoritma A* *search* pada kondisi (i).

Tabel III.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)]

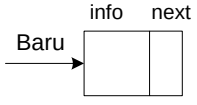
Ada beberapa istilah yang digunakan untuk menggambarkan *linked list* yaitu sebagai berikut :

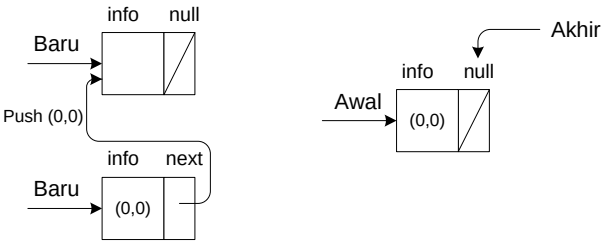
- Awal, variabel yang berisi alamat yang menunjuk lokasi simpul pertama *linked list*.
- Akhir, sebuah simpul yang tidak menunjuk pada simpul berikutnya.
- Info berisi informasi tentang elemen data yang bersangkutan.
- *Next* (*link field/next pointer field*), berisi alamat dari elemen (*node*) selanjutnya yang dituju.
- *Null,* tidak ada elemen (*node*) selanjutnya.
- Jika P adalah suatu variabel pointer, maka nilainya adalah alamat atau lokasi dari variabel lain yang dituju.

Linked list yang digunakan yaitu *single linked list*, pada penambahan data baru menggunakan penyisipan didepan dan penghapusan data memakai hapus data diawal. *Linked list* untuk penyelesaian pada contoh kondisi (i) gambar III.5 terdiri dari 2 (dua) langkah, yaitu :

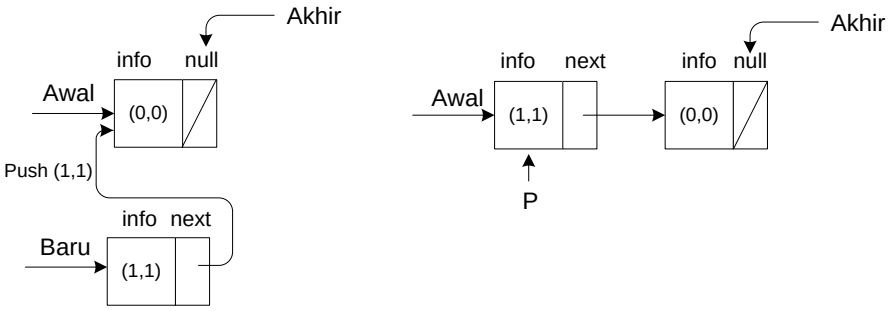
a. Langkah 1

- Penambahan data baru (0,0)

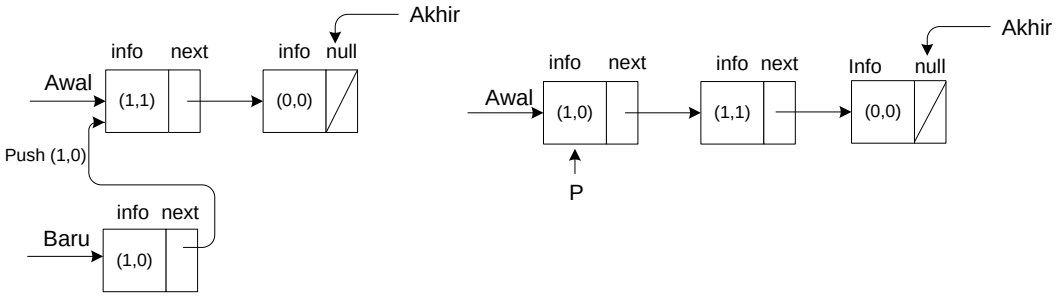




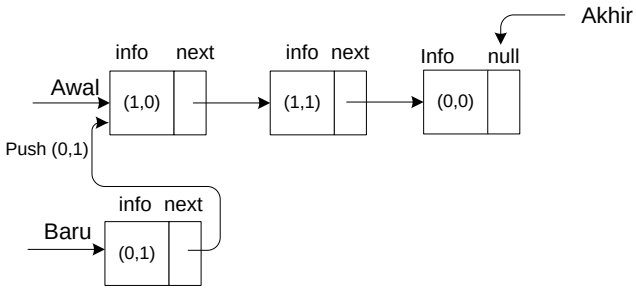
- Penambahan data baru (1,1)

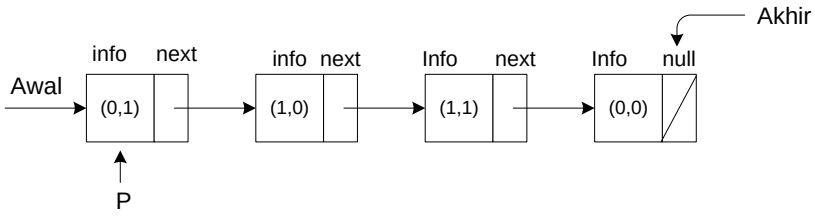


- Penambahan data baru (1,0)



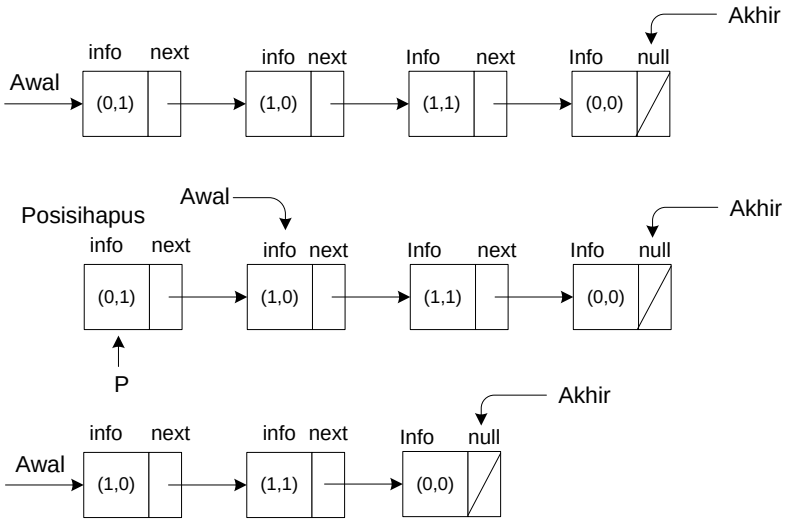
- Penambahan data baru (0,1)



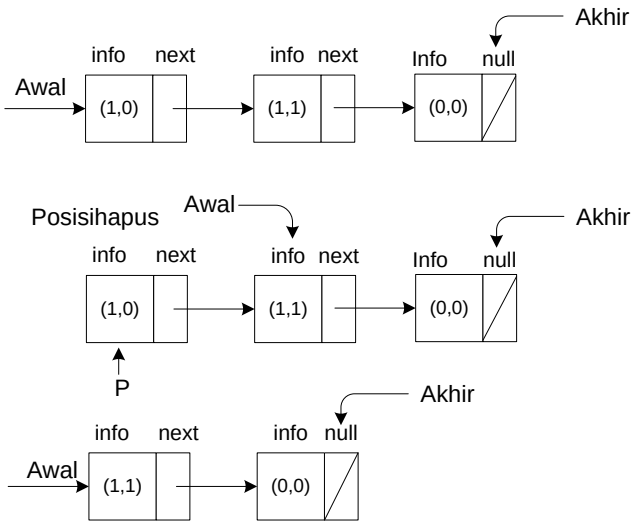


Node (1,0) dan (0,1) akan dihapus karena nilai f(n) nya lebih besar dari pada node (1,1), sehingga akan menghasilkan *list* sebagai berikut :

- Penghapusan data (0,1)

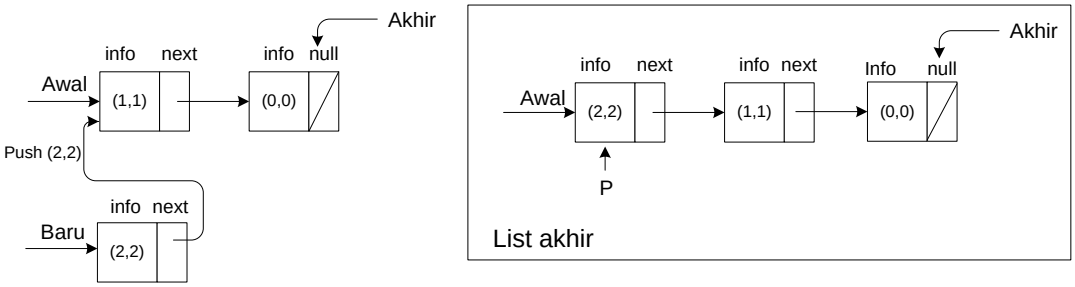


- Penghapusan data (1,0)



b. Langkah 2

- Penambahan data baru (2,2)



Langkah-langkah di bawah ini adalah penjelasan dari kondisi (iv) pada gambar III.16.

a. Langkah pertama dalam kondisi (iv), karena di *OPEN* hanya terdapat satu simpul yaitu A(0,0), maka A terpilih sebagai *BestNode* dan dipindahkan ke *CLOSED.* 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).

b. 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).

c. 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* kemudian dicek nilai f nya. Langkah ke tiga ini menghasilkan *OPEN* = [(0,2), (2,2), (2,1)] dan *CLOSED* = [A(0,0), (0,1), (1,2)]. Ternyata nilai f melaui 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).

d. 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,* kemudian dicek nilai f nya. Selanjutnya, simpul T(2,0) dengan f(n) = 4 terpilih sebagai *BestNode* dan sama dengan simpul tujuan, berarti solusi sudah ditemukan. Pada akhir langkah ke empat ini, *OPEN* = [(0,2), (2,2), T(2,0)] dan *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 jalan A(0,0) – (0,1) – (1,2) – (2,1) - T(2,0) dengan jumlah langkah 3 dan total simpul yang

diperiksa 7. Simpul-simpul yang diperiksa tiap langkah dapat dilihat pada tabel III.3. Jalur ini merupakan jalan yang dihasilkan oleh algoritma A* *search* pada kondisi (iv).

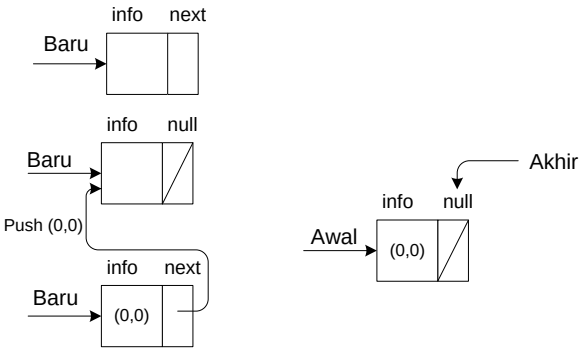
Tabel III.3. Simpul yang diperiksa tiap langkah pada kondisi (iv)

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)]

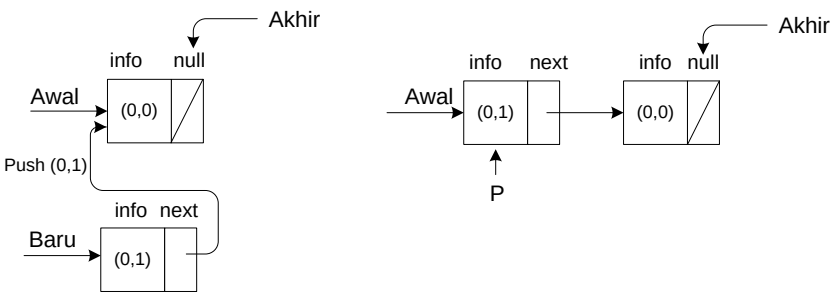
Linked list untuk penyelesaian pada contoh kondisi (iv) gambar III.10 terdiri dari 4 (empat) langkah, yaitu :

a. Langkah 1

- Penambahan data baru (0,0)

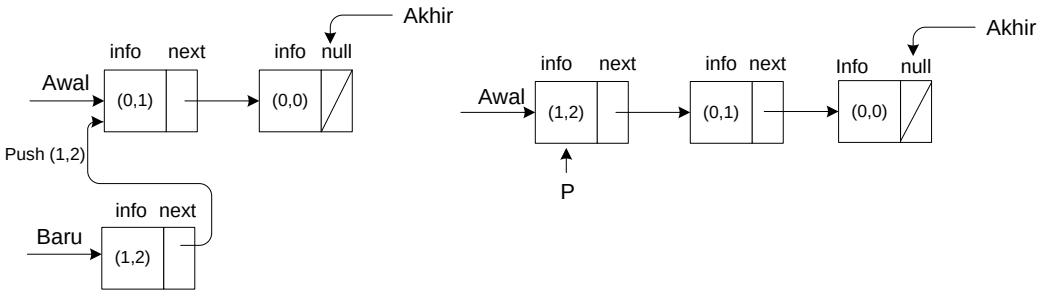


- Penambahan data baru (0,1)

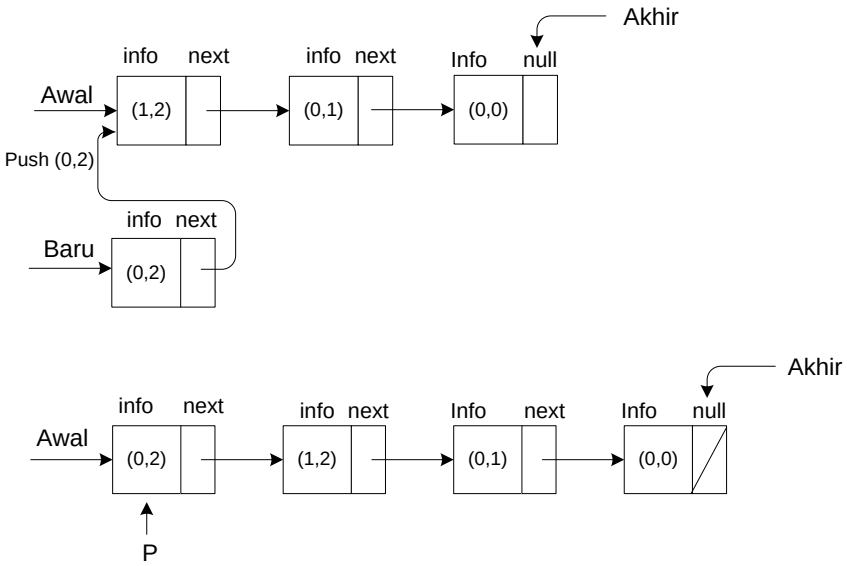


b. Langkah 2

- Penambahan data baru (1,2)

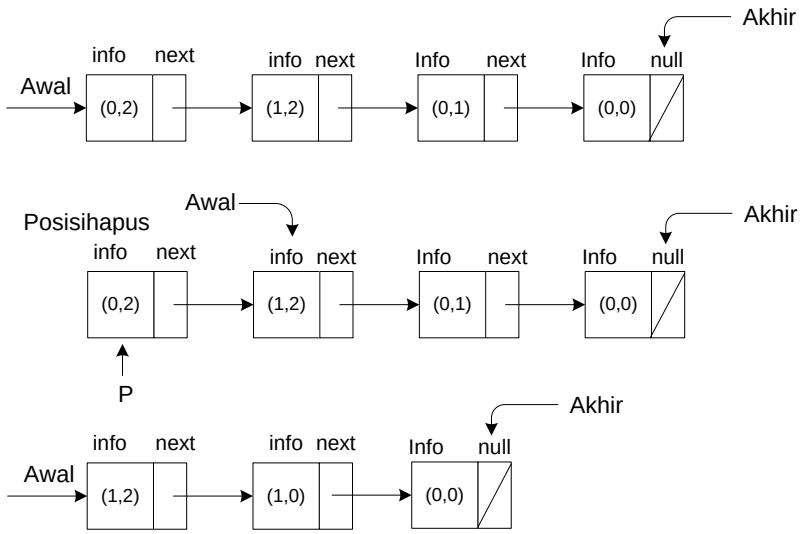


- Penambahan data baru (0,2)



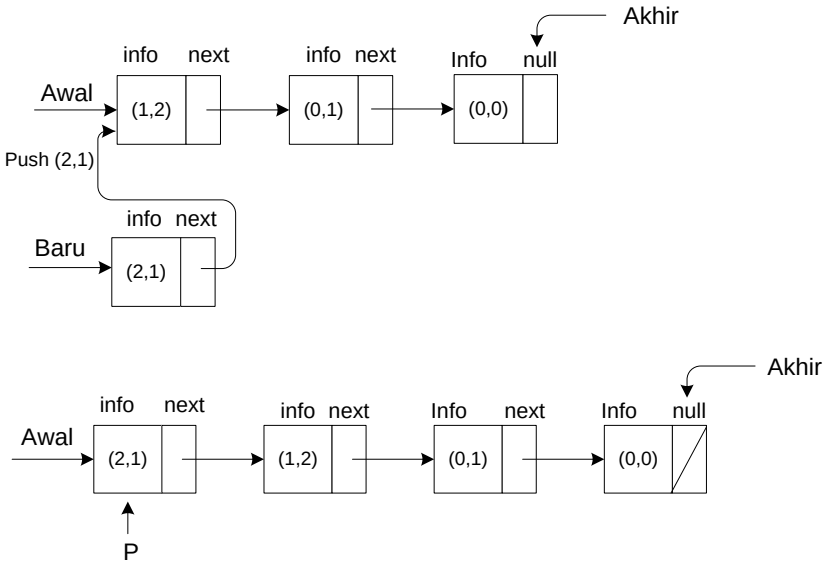
Node (0,2) akan dihapus karena nilai f(n) nya lebih besar dari pada *node* (1,2), sehingga akan menghasilkan *list* sebagai berikut :

- Penghapusan data (0,2)

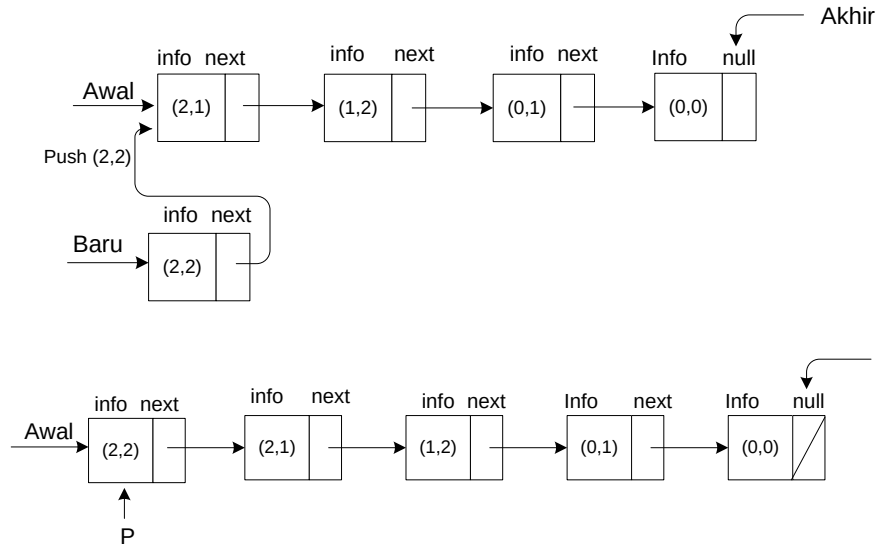


c. Langkah 3

- Penambahan data baru (2,1)

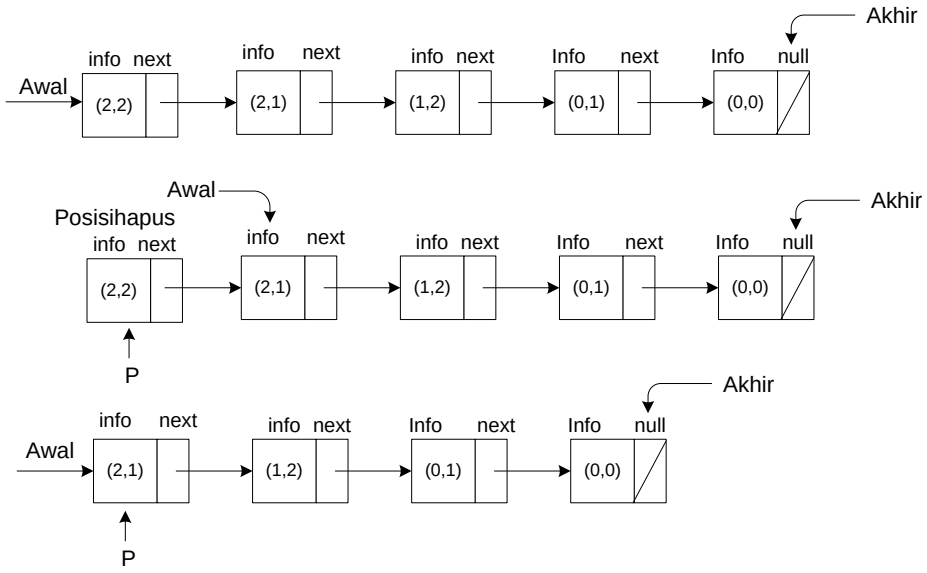


- Penambahan data baru (2,2)



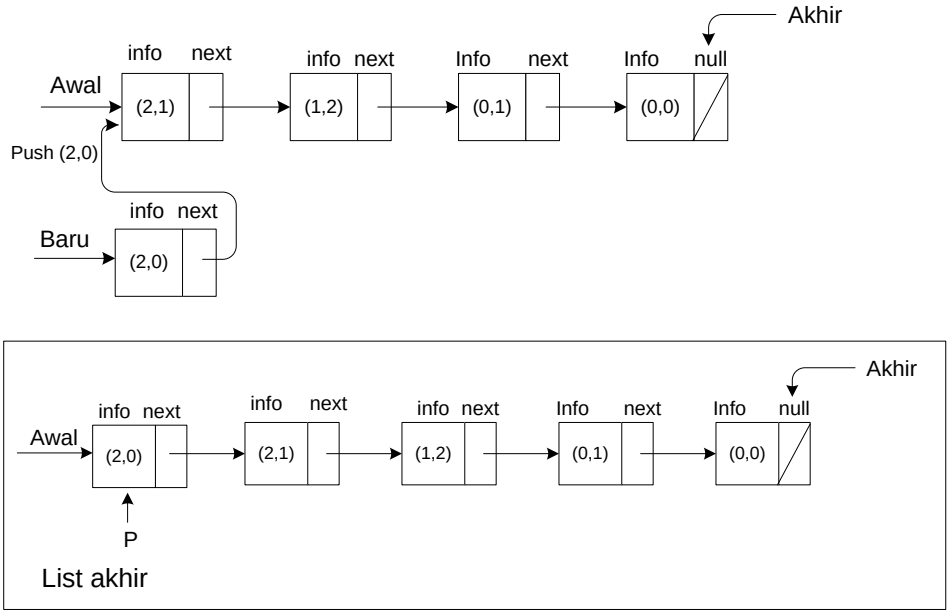
Node (2,2) akan dihapus karena nilai f(n) nya lebih besar dari pada *node* (2,1), sehingga akan menghasilkan list sebagai berikut :

- Penghapusan data (2,2)



d. Langkah 4

- Penambahan data baru (2,0)



III.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. Analisis kebutuhan ini juga menentukan spesifikasi masukan yang diperlukan sistem, keluaran yang akan dihasilkan sistem dan proses yang dibutuhkan untuk mengolah masukan sehingga menghasilkan suatu keluaran yang diinginkan.

Pada analisis kebutuhan sistem non fungsional ini dijelaskan analisis mengenai analisis masukan, analisis keluaran, perangkat keras (*hardware)*, perangkat lunak (*software)*, dan pengguna (*user*) sebagai bahan analisis

kekurangan dan kebutuhan yang harus dipenuhi dalam perancangan sistem yang akan diterapkan.

a. Analisis masukan

Analisis masukan dilakukan untuk mengetahui masukan data apa saja yang diperlukan oleh sistem. Masukan data yang diperlukan oleh sistem yaitu titik awal, titik tujuan, baris kotak, kolom kotak, penghalang maupun tanpa penghalang dan nilai heuristik.

b. Analisis keluaran

Analisis keluaran dilakukan untuk mengetahui keluaran dari sistem yang dibutuhkan oleh *user* yaitu berupa jalur terpendek dan pelacakan pencarian dari titik awal ke titik tujuan.

c. Analisis kebutuhan perangkat keras *(hardware)*

Agar aplikasi dapat berjalan dengan baik, maka dibutuhkan perangkat keras yang sesuai dengan kebutuhan aplikasi. Tabel III.4 adalah spesifikasi minimun perangkat keras yang dibutuhkan agar dapat menjalankan aplikasi *game pathfinding* secara optimal.

Tabel III.4. Spesifikasi minimum perangkat keras

Komputer	Spesifikasi perangkat keras
Prosesor	Prosesor 1.6 Ghz
Resolusi layar	1366 x 768 pixel
Memori	Memori 128 Mb
Harddisk	Harddisk 20 GB
Keyboard dan mouse	-

d. Analisis kebutuhan perangkat lunak (*software*)

Penerapan algoritma A* dalam pencarian jalan terpendek pada *game pathfinding* memiliki kebutuhan perangkat lunak (*software*) pada tabel III.5.

Tabel III.5. Spesifikasi perangkat lunak

Komputer	Spesifikasi perangkat lunak
Sistem Operasi	Window Xp, Window 7
Bahasa Pemograman	Borland Delphi 7.0

Dari 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 lunak yang telah direkomendasikan agar aplikasi dapat berjalan dengan baik. Jika pengguna ingin membuka *project*-nya maka terlebih dahulu harus menginstalkan borland delphi dan *component* yang mendukung.

e. Analisis pengguna

Pengguna yaitu *public* yang dapat mengerti dan memahami komputer sehingga dapat menggunakan aplikasi yang akan dibangun.

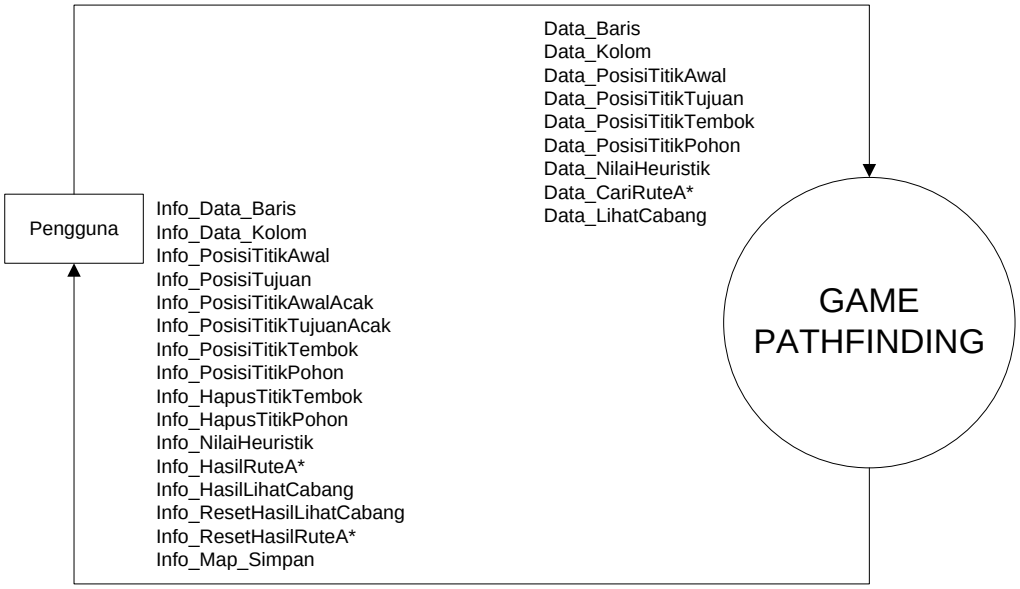
III.6 Analisis Kebutuhan Fungsional

Pemodelan yang digunakan pada aplikasi yang akan dibuat adalah aliran data terstuktur sebagai alat bantu yaitu diagram konteks dan DFD (*Data Flow Diagram*).

III.6.1 Diagram Konteks

Diagram konteks atau disebut juga dengan model aplikasi fundamental yang merepresentasikan seluruh elemen aplikasi sebagai sebuah *bubble* tunggal

dengan data masukan-keluaran yang ditunjukkan oleh anak panah yang masuk dan keluar secara berurutan. Diagram konteks dari aplikasi yang akan dibangun adalah seperti gambar III.17. [5]



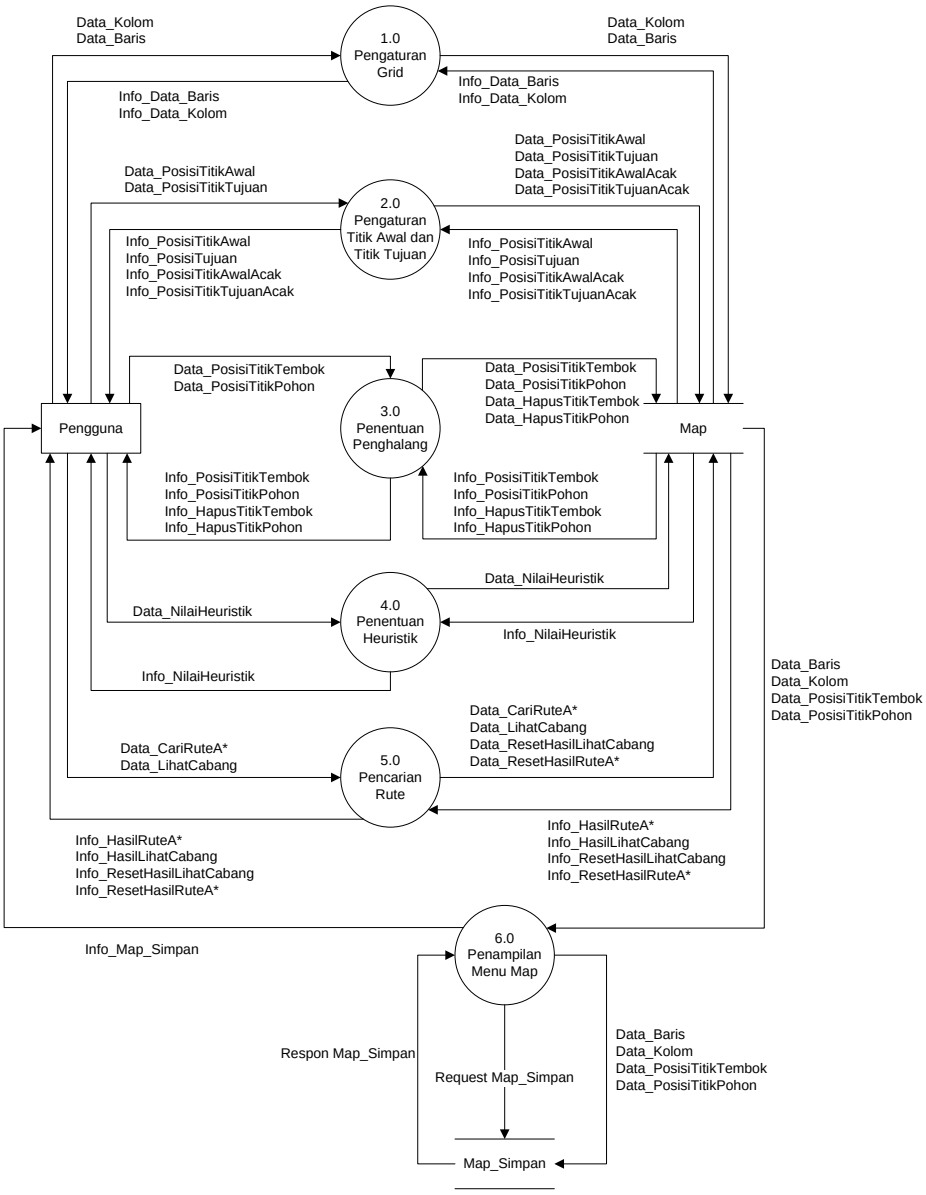
Gambar III.17. Diagram Konteks

III.6.2 *Data Flow Diagram* (DFD)

Data flow diagram merupakan model dari sistem untuk menggambarkan pembagian sistem ke modul yang lebih kecil. DFD sering digunakan untuk menggambarkan suatu sistem yang telah ada atau sistem baru yang akan dikembangkan secara logika tanpa mempertimbangkan lingkungan fisik dimana data tersebut mengalir atau lingkungan fisik dimana data tersebut akan disimpan. Salah satu keuntungan DFD adalah memudahkan pemakai atau *user* yang kurang menguasai bidang komputer untuk mengerti sistem yang akan dikerjakan. [5]

III.6.2.1 DFD Level 0

DFD Level 0 dibuat jika pada diagram konteks masih terdapat proses yang harus dijelaskan lebih rinci. Pada DFD Level 0 terdapat proses - proses yaitu pengaturan *grid*, pengaturan titik awal dan titik tujuan, penentuan penghalang, penentuan heuristik, proses pencarian *rute* dan penampilan menu *map* yang dapat dilihat pada gambar III.18.



Gambar III.18. DFD Level 0 *game pathfinding*

1. Pengaturan Grid

   Proses pengaturan untuk menentukan berapa baris dan kolom yang telah yang diberi nilai.

2. Pengaturan Awal dan Tujuan

   Proses penempatan titik awal dan titik tujuan yang diinginkan.

3. Penentuan Penghalang

   Proses pemasangan penghalang yaitu tembok dan pohon pada *map* .

4. Penentuan Heuristik

   Proses penentuan nilai heuristik yang akan digunakan saat pencarian berlangsung.

5. Pencarian *Rute*

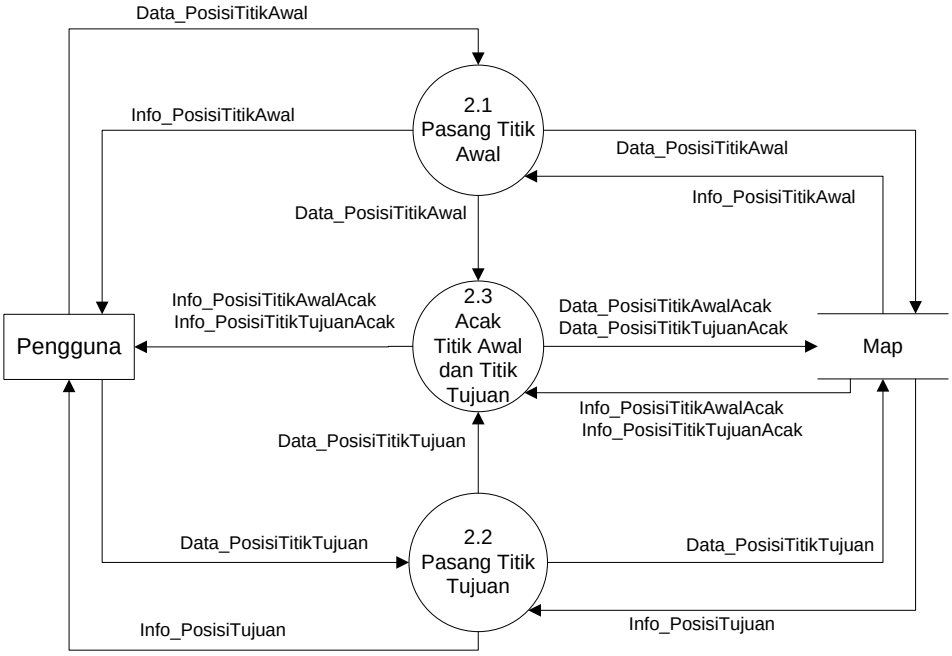
   Proses berlangsungnya pencarian jalur pada *map.*

6. Penampilan Menu *Map*

   Proses ini berisi tentang membuka *map* simpan, menyimpan *map* baru, dan membuat *map* baru.

III.6.2.2 DFD Level 1 Proses 2.0 Pengaturan Titik Awal dan Titik Tujuan

Proses yang terdapat pada DFD level 1 proses 2.0 adalah proses pengaturan titik awal dan titik tujuan yang terdiri atas proses 2.1 pasang titik awal, proses 2.2 pasang titik tujuan dan proses 2.3 acak titik awal dan titik tujuan. DFD level 1 untuk proses 2.0 dapat dilihat pada gambar III.19.



Gambar III.19. DFD Level 1 Proses 2.0 pengaturan titik awal dan titik tujuan

2.1 Pasang Titik Awal

Proses untuk pemasangan titik awal pada *map*.

2.2 Pasang Titik Tujuan

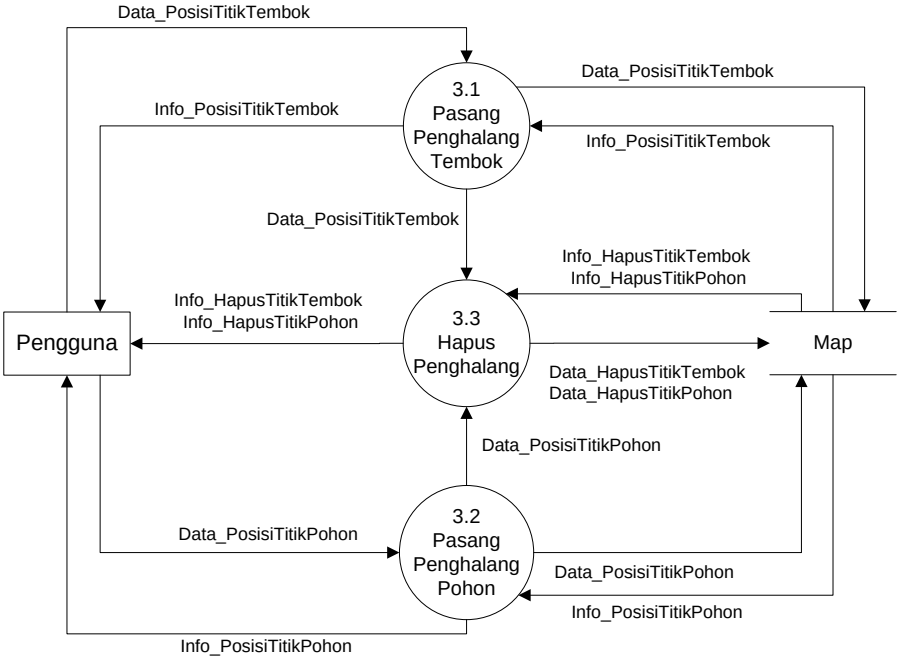
Proses untuk pemasangan titik tujuan pada *map*.

2.3 Acak Titik Awal dan Titik Tujuan

Proses pengacakan posisi titik awal dan titik tujuan.

III.6.2.3 DFD Level 1 Proses 3.0 Penentuan Penghalang

Proses yang terdapat pada DFD level 1 proses 3.0 adalah proses penentuan penghalang yang terdiri atas proses 3.1 pasang penghalang tembok, proses 3.2 pasang penghalang pohon dan proses 3.3 hapus penghalang. DFD level 1 untuk proses 3.0 dapat dilihat pada gambar III.20.



Gambar III.20. DFD Level 1 Proses 3.0 penentuan penghalang

3.1 Pasang Penghalang Tembok

Proses awal meletakkan posisi tembok pada *map* (*grid*) yang telah disediakan.

3.2 Pasang Penghalang Pohon

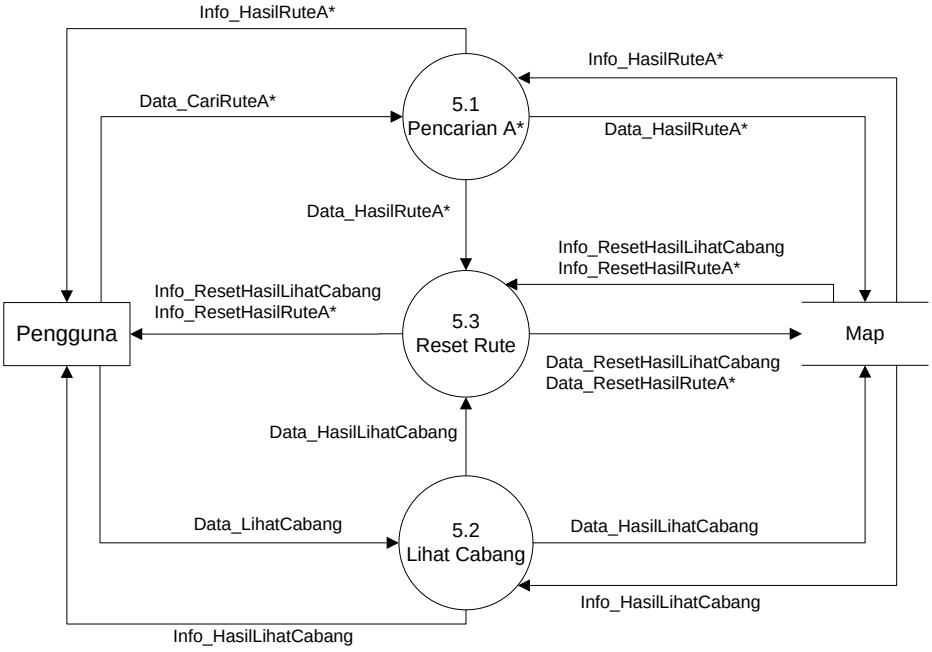
Proses awal meletakkan posisi pohon pada *map (grid)* yang telah disediakan.

3.3 Hapus Penghalang

Proses menghapus baik itu penghalang tembok ataupun pohon yang telah dibuat.

III.6.2.4 DFD Level 1 Proses 5.0 Pencarian *Rute*

Proses yang terdapat pada DFD level 1 proses 5.0 adalah proses pencarian *rute* yang terdiri atas proses 5.1 pencarian A*, proses 5.2 lihat cabang dan proses 5.3 reset *rute*. DFD level 1 untuk proses 5.0 dapat dilihat pada gambar III.21.



Gambar III.21. DFD level 1 Proses 5.0 pencarian *rute*

5.1 Pencarian A*

Proses pencarian *rute* dengan algoritma A* *search*.

5.2 Lihat Cabang

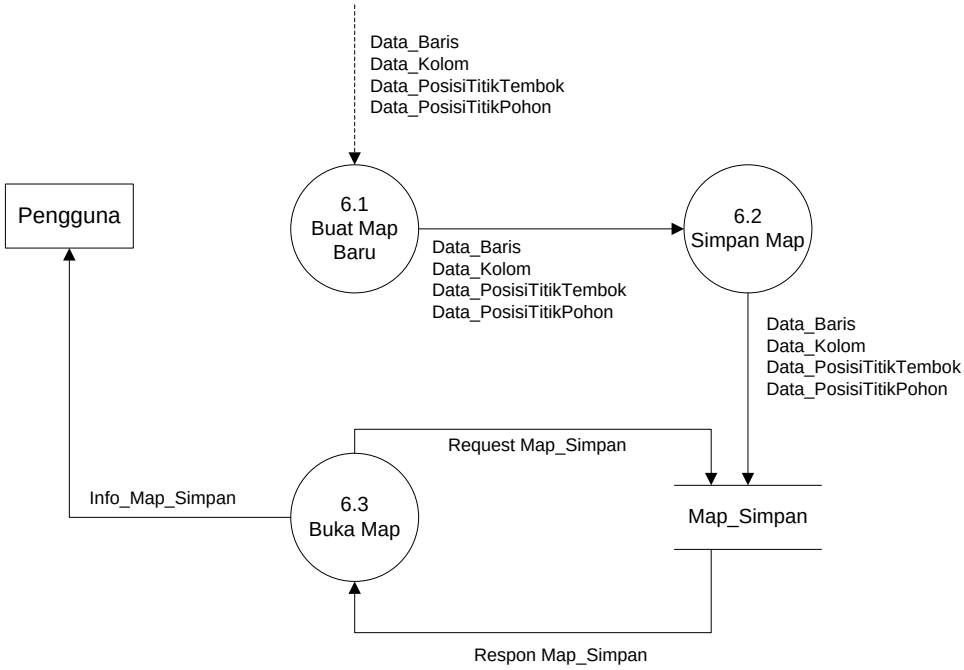
Proses memperlihatkan pelacakan cabang untuk A*.

5.3 Reset *rute*

Proses menghapus hasil *rute* dari pencarian A* dan pelacakan cabangnya.

III.6.2.5 DFD Level 1 Proses 6.0 Penampilan Menu *Map*

Proses yang terdapat pada DFD level 1 proses 6.0 adalah proses penampilan menu *map* yang terdiri atas proses 6.1 buka *map* baru, proses 6.2 simpan *map* dan proses 6.3 buka *map*. DFD level 1 untuk proses 6.0 dapat dilihat pada gambar III.22.



Gambar III.22. DFD Level 1 Proses 6.0 penampilan menu *map*

6.1 Buat *Map* Baru

Proses pembuatan *map* baru yang akan disimpan.

6.2 Simpan *Map*

Proses penyimpanan *map* setelah dilakukan pengaturan.

6.3 Buka *Map*

Proses membuka *map* yang telah disimpan sebelumnya.

III.6.3 Spesifikasi Proses

Untuk menjelaskan lebih lanjut tentang proses-proses yang ada di dalam diagram alir data atau DFD maka dibuatlah spesifikasi proses. Spesifikasi proses merupakan deskripsi dari setiap elemen proses yang terdapat dalam aplikasi, yang meliputi nama proses, masukan, keluaran dan keterangan dari proses. Adapun spesifikasi proses untuk diagram alir data aplikasi *game pathfinding* dengan

penerapan algoritma A* untuk mencari jalan terpendek dapat dilihat pada tabel III.6.

Tabel III.6. Spesifikasi proses

No.	Proses	Keterangan
1.	No. Proses	1.0
	Nama Proses	Pengaturan *Grid*
	Source (sumber)	Pengguna
	Deskripsi	Proses pengaturan untuk menentukan nilai baris dan kolom yang telah ditentukan.
	Masukan	Data baris dan data kolom.
	Keluaran	Info baris dan info kolom.
	Logika proses	1. Pengguna memasukkan nilai baris dan nilai kolom. 2. Jika nilai baris kurang dari 3 atau lebih dari 31 maka akan tampil info pesan data nilai baris tidak boleh kurang dari 3 atau info pesan data nilai baris tidak boleh lebih dari 31. 3. Jika nilai kolom kurang dari 3 atau lebih dari 39 maka akan tampil info pesan data nilai kolom tidak boleh kurang dari 3 atau info pesan data nilai kolom tidak boleh lebih dari 39. 4. Jika nilai baris atau nilai kolom tidak diisi maka akan tampil pesan data nilai baris atau nilai kolom tidak boleh kosong. 5. Jika nilai baris lebih dari 31 dan nilai kolom lebih dari 39 maka akan tampil konfirmasi pesan nilai baris tidak boleh lebih dari 31 dan nilai kolom tidak boleh lebih dari 39.
2.	No. Proses	2.0
	Nama Proses	Pengaturan titik awal dan titik tujuan
	Source (sumber)	Pengguna
	Deskripsi	Proses penempatan titik awal dan titik tujuan yang diinginkan.
	Masukan	Data posisi titik awal dan titik tujuan.
	Keluaran	Info posisi titik awal dan titik tujuan.
	Logika proses	Pengguna menempatkan posisi titik awal dan posisi titik tujuan.

Tabel III.6. Spesifikasi proses (lanjutan)

No.	Proses	Keterangan
3.	No. Proses	3.0
	Nama Proses	Penentuan penghalang
	Source (sumber)	Pengguna
	Deskripsi	Proses pemasangan penghalang yaitu tembok dan pohon pada *map.*
	Masukan	Data posisi titik tembok dan titik pohon
	Keluaran	Info posisi titik tembok dan titik pohon
	Logika proses	1. Pengguna menempatkan penghalang titik tembok atau titik pohon maupun keduanya. 2. Jika penghalang ditempatkan penuh pada *map* maka akan ada info pada status bar *rute* tidak ditemukan pada saat pencarian berlangsung.
4.	No. Proses	4.0
	Nama Proses	Penentuan heuristik
	Source (sumber)	Pengguna
	Deskripsi	Proses penentuan nilai heuristik yang akan digunakan saat pencarian berlangsung.
	Masukan	Data nilai heuristik
	Keluaran	Info nilai heuristik
	Logika proses	1. Pengguna menentukan nilai heuristik. 2. Jika tidak ditentukan oleh pengguna maka nilai heuristik otomatis dimulai dengan nilai 1.
5.	No. Proses	5.0
	Nama Proses	Pencarian *rute*
	Source (sumber)	Pengguna
	Deskripsi	Proses berlangsungnya pencarian jalur pada *map*.
	Masukan	Data pencarian
	Keluaran	Info hasil *rute*
	Logika proses	1. Jika pengguna memilih pencarian dengan A* tanpa mencentang “lihat pelacakan” maka akan berlangsung proses dengan perhitungan algoritma tersebut tanpa meperlihatkan pelacakan cabangnya. 2. Jika pengguna memilih pencarian dengan A* dengan mencentang “lihat pelacakan” maka akan berlangsung proses dengan perhitungan algoritma tersebut dengan meperlihatkan pelacakan cabangnya.

Tabel III.6. Spesifikasi proses (lanjutan)

No.	Proses	Keterangan
6.	No. Proses	2.1
	Nama Proses	Pasang titik awal
	Source (sumber)	Pengguna
	Deskripsi	Proses menempatkan posisi titik awal pada *map*.
	Masukan	Data posisi titik awal
	Keluaran	Info posisi titik awal
	Logika proses	Jika pengguna menempatkan titik awal maka akan langsung terlihat posisinya pada *map*.
7.	No. Proses	2.2
	Nama Proses	Pasang titik tujuan
	Source (sumber)	Pengguna
	Deskripsi	Proses menempatkan titik tujuan pada *map*.
	Masukan	Data posisi titik tujuan
	Keluaran	Info posisi titik tujuan
	Logika proses	Jika pengguna menempatkan titik tujuan maka akan langsung terlihat posisinya pada *map*.
8.	No. Proses	2.3
	Nama Proses	Acak titik awal dan titik tujuan
	Source (sumber)	Pengguna
	Deskripsi	Proses pengacakan posisi titik awal dan titik tujuan
	Masukan	Data posisi titik awal dan titik tujuan
	Keluaran	Info posisi titik awal dan titik tujuan yang telah diacak
	Logika proses	Jika pengguna memilih proses acak maka posisi titik awal dan titik tujuan akan berubah.
9.	No. Proses	3.1
	Nama Proses	Pasang penghalang tembok
	Source (sumber)	Pengguna
	Deskripsi	Proses awal meletakkan posisi tembok pada *map* (*grid*) yang telah disediakan.
	Masukan	Data posisi titik tembok
	Keluaran	Info posisi titik tembok
	Logika proses	Jika pengguna menempatkan titik tembok maka akan langsung terlihat posisinya pada *map*.

Tabel III.6. Spesifikasi proses (lanjutan)

No.	Proses	Keterangan
10.	No. Proses	3.2
	Nama Proses	Pasang penghalang pohon
	Source (sumber)	Pengguna
	Deskripsi	Proses awal meletakkan posisi pohon pada *map (grid)* yang telah disediakan.
	Masukan	Data posisi titik pohon
	Keluaran	Info posisi titik pohon
	Logika proses	Jika pengguna menempatkan titik pohon maka akan langsung terlihat posisinya pada *map*.
11.	No. Proses	3.3
	Nama Proses	Hapus penghalang
	Source (sumber)	Pengguna
	Deskripsi	Proses berlangsungnya pencarian jalur pada *map*.
	Masukan	Data pencarian
	Keluaran	Info hasil rute
	Logika proses	Jika pengguna memilih hapus penghalang maka akan terhapus tembok maupun pohon yang telah dipilih untuk dihapus.
12.	No. Proses	5.1
	Nama Proses	Pencarian A*
	Source (sumber)	Pengguna
	Deskripsi	Proses pencarian *rute* dengan algoritma A* *search*.
	Masukan	Data cari *rute* A*
	Keluaran	Info hasil *rute* A*
	Logika proses	Jika pengguna memilih pencarian A* maka akan tampil hasil *rute* A* dari titik awal sampai titik tujuan.
13.	No. Proses	5.2
	Nama Proses	Lihat Cabang
	Source (sumber)	Pengguna
	Deskripsi	Proses memperlihatkan pelacakan cabang untuk A*.
	Masukan	Data lihat cabang
	Keluaran	Info hasil lihat cabang
	Logika proses	Jika pengguna memilih lihat cabang maka akan tampil hasil pelacakan cabang dari titik awal sampai titik tujuan.

Tabel III.6. Spesifikasi proses (lanjutan)

No.	Proses	Keterangan
14.	No. Proses	5.3
	Nama Proses	Reset *rute*
	Source (sumber)	Pengguna
	Deskripsi	Proses menghapus hasil *rute* dari pencarian A* dan pelacakan cabang.
	Masukan	Data cari *rute* A* atau data lihat cabang
	Keluaran	Data reset hasil *rute* A*, data reset hasil lihat cabang, info reset hasil *rute* A* dan info reset hasil lihat cabang.
	Logika proses	Jika pengguna ingin menghapus *rute* A* atau pelacakan cabang maka tinggal melakukan proses reset *rute*.
15.	No. Proses	6.1
	Nama Proses	Buat *map* baru
	Source (sumber)	Pengguna
	Deskripsi	Proses pembuatan *map* baru yang akan disimpan.
	Masukan	Data baris, data kolom, data posisi titik tembok dan data posisi titik pohon dari data sementara pada *map*.
	Keluaran	Data baris, data kolom, data posisi titik tembok dan data posisi titik pohon.
	Logika proses	Jika pengguna ingin membuat *map* baru yang akan disimpan maka semua data masukan akan masuk ke proses 6.2.
16.	No. Proses	6.2
	Nama Proses	Simpan *map*
	Source (sumber)	Pengguna
	Deskripsi	Proses penyimpanan *map* setelah dilakukan pengaturan.
	Masukan	Data baris, data kolom, data posisi titik tembok dan data posisi titik pohon dari proses 6.1.
	Keluaran	Data baris, data kolom, data posisi titik tembok dan data posisi titik pohon.
	Logika proses	Jika pengguna ingin menyimpan *map* baru maka semua data masukan akan masuk penyimpanan *map* simpan.

Tabel III.6. Spesifikasi proses (lanjutan)

No.	Proses	Keterangan
17.	No. Proses	6.3
	Nama Proses	Buka *map*
	Source (sumber)	Pengguna
	Deskripsi	Proses membuka *map* yang telah disimpan sebelumnya.
	Masukan	Request *map* simpan dari penyimpanan *map* simpan.
	Keluaran	Respon *map* simpan dari penyimpanan *map* simpan.
	Logika proses	Jika pengguna ingin membuka *map* dari penyimpanan *map* simpan maka file *map* yang dipilih akan tampil pada *map*.

Berdasarkan pengolahan DFD (*data flow diagram)* yang ada maka akan digunakan dua *file* untuk penyimpanan yang terdiri dari *file map* dan *file map* simpan. File map merupakan penyimpanan sementara untuk mempermudah tampilan data titik awal, titik tujuan, penghalang, dan hasil *rute* pada *map* sedangkan *file map* simpan yaitu penyimpanan untuk *map* yang telah dibuat yang berupa dari data nilai baris, nilai kolom dan penghalang yang akan disimpan dengan gambar bertype .*map*.

III.6.4 Kamus Data

Kamus data atau *data dictionary* adalah katalog fakta tentang data dan kebutuhan-kebutuhan informasi dari suatu sistem informasi. Kamus data untuk diagram arus data pada *game pathfinding* dalam pencarian jalan terpendek dapat dilihat pada tabel III.7. [5]

Tabel III.7. Kamus data

Nama aliran data	Pengolahan data *map* simpan
Digunakan pada	Pengguna – Proses 6.0
Deskripsi	Berisi penyimpanan data *map*
Struktur data • Nilai baris • Nilai kolom • Penghalang (titik tembok atau titik pohon)	Berisi nilai baris, nilai kolom, penghalang : *Type numeric* (integer) [0-9], *width* 2 : *Type numeric* (integer) [0-9], *width* 2 : *Image* ber*type .map*

III.7 Perancangan Sistem

Perancangan sistem merupakan metodologi 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 kepada penggunaan tentang sistem yang akan dibangun.

III.7.1 Perancangan Komponen *Game Pathfinding*

Perancangan komponen pada *game pathfinding* terdiri dari pengguna, titik awal dan titik tujuan, penghalang dan *map* yang akan dijelaskan sebagai berikut :

1. Pengguna

Game ini akan dimainkan oleh satu orang. Pengguna dapat menempatkan titik awal, titik tujuan dan penghalang pada *map* yang telah disediakan dengan posisi yang diinginkan.

2. Titik awal dan titik tujuan

Titik awal merupakan titik pertama jalan yang akan dicari sedangkan titik tujuan merupakan titik akhir yang akan ditemukan. Titik awal akan digambarkan dengan gambar pinguin dan titik tujuan dengan gambar peti harta karun supaya tampilan lebih menarik.

3. Penghalang

Penghalang yaitu suatu titik yang tidak dapat dilewati. Penghalang yang akan digunakan yaitu tembok dan pohon agar tampilan yang dihasilkan lebih menarik.

4. Map

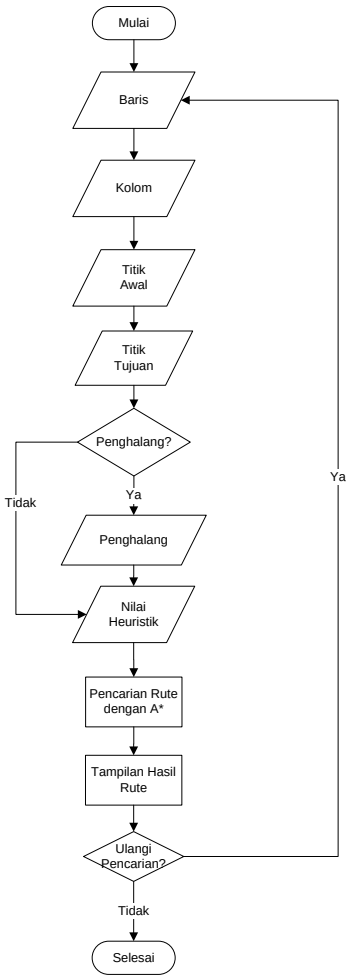
Map yang digunakan pada *game pathfinding* ini akan berupa kotak-kotak dengan ordo X x Y. Terdiri dari kotak (*grid*) kecil dengan ordo minimal 3 x 3 dan maksimal 31 x 39. Dalam perancangan tampilan *map* awal diwakili dengan kotak berwarna putih. *Map* ini merupakan tempat untuk memasangkan titik awal, titik tujuan dan penghalang serta untuk menampilkan jalan yang ditemukan. Hasil akhir *game pathfinding* ditandai oleh tampilnya jalur dan performansi yang dihasilkan oleh A* *search*.

III.7.2 Perancangan Prosedural

Bagan alir sistem *(flowchart system)* merupakan bagan-bagan yang mempunyai arus yang menggambarkan langkah-langkah penyelesaian suatu

masalah dan merupakan cara penyajian dari suatu algoritma, sehingga dapat menjelaskan urutan prosedur-prosedur yang ada dalam sistem. [5]

Pada gambar III.23. dijelaskan bahwa cara kerja sistem yang akan dibangun yaitu dimulai dari memasukkan data berupa titik awal, titik tujuan dan penghalang maupun tanpa penghalang, kemudian masuk kedalam pencarian *rute* dengan algoritma A* *search*. Setelah pencarian *rute* yang dipilih A* maka akan muncul jalan yang ditemukan oleh algoritma A*, jika sekaligus dicentang "lihat cabang pencarian" maka akan muncul pelacakan cabangnya. Setelah selesai *user* dapat menghapus *rute* dan dapat memasukkan masukan baru.



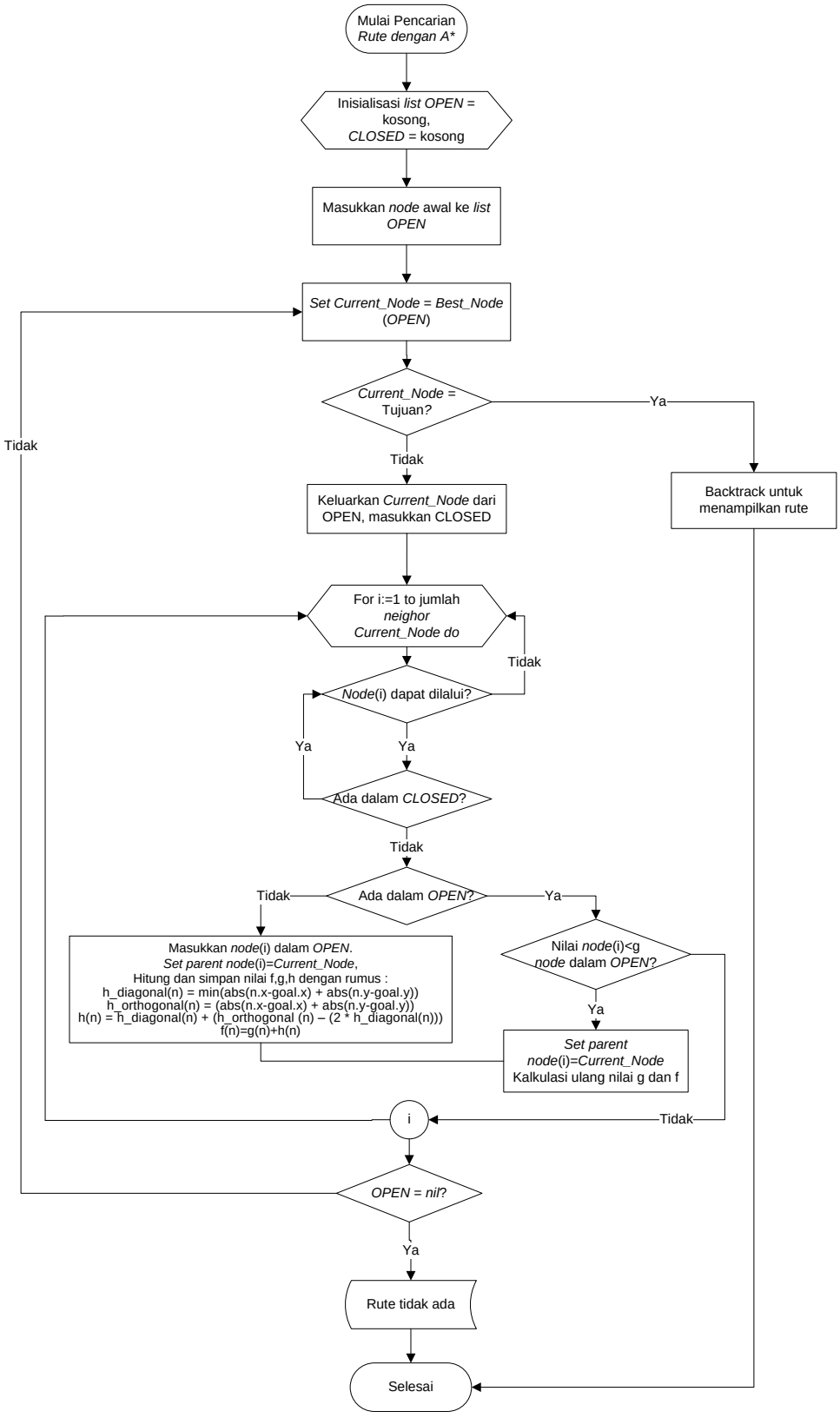
Gambar III.23. *Flowchart* pencarian *rute* pada *game pathfinding*

III.7.2.1 Bagan Alir Algoritma A* *(A Star)*

Setelah *user* memilih tujuan, maka pencarian pun dipilih yaitu dengan menggunakan algoritma A*. Diagram alir algoritma A* dapat dilihat pada gambar III.24.

Pada gambar III.24 menjelaskan cara kerja algoritma A* bekerja dalam mencari *rute* paling optimum pada peta. Algoritma ini bekerja dari titik awal ke titik tujuan dengan mencari jalan dengan dengan akumulasi bobot terendah (dilambangkan dengan notasi g) dijumlah dengan perkiraan jalan terendah bobotnya untuk mencapai tujuan (dilambangkan dengan notasi 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.

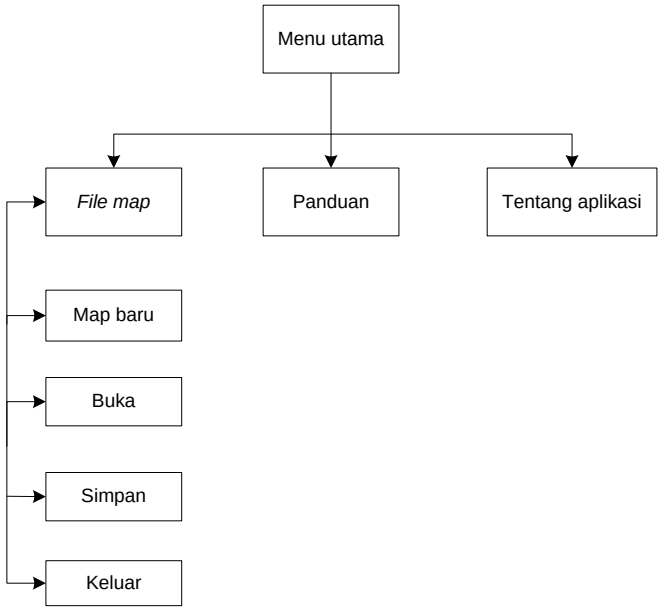
Proses mencari *node* dengan nilai terkecil ini dilakukan berulang-ulang sampai menemukan *node* tujuan. Apabila *node* yang menjadi titik tujuan telah ditemukan, program akan melakukan *backtrack* ke *parent* dari tiap *node* untuk mendapatkan rangkaian *node* yang membentuk *rute* yang akan paling optimum yang diinginkan. Penjelasan lebih lengkap tentang algoritma A* beserta *pseudocode*-nya telah dituliskan pada bab 2.



Gambar III.24. *Flowchart* algoritma A* *Search*

III.7.3 Perancangan Struktur Menu

Perancangan struktur menu berisikan menu dan submenu yang berfungsi memudahkan *user* didalam menggunakan sistem. Pada gambar III.25 dijelaskan perancangan struktur menu aplikasi *game pathfinding*.



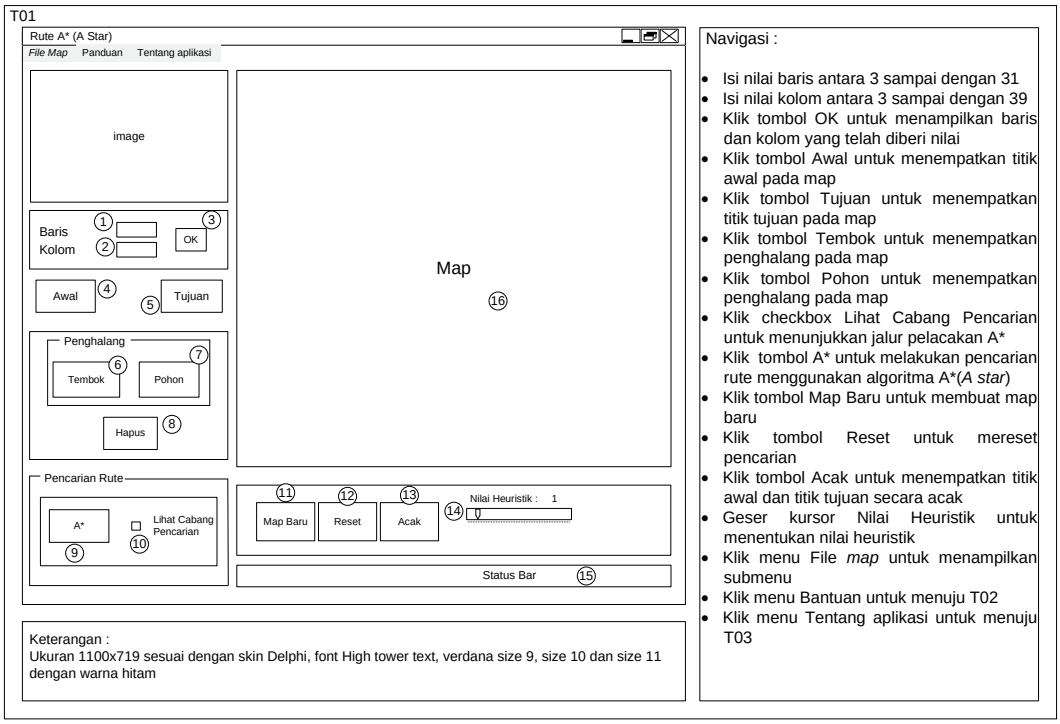
Gambar III.25. Struktur menu

III.7.4 Perancangan Antarmuka

Perancangan antarmuka bertujuan untuk memberikan gambaran tentang aplikasi yang akan dibangun, sehingga akan mempermudah dalam mengimplementasikan aplikasi serta akan memudahkan pembuatan aplikasi. Perancangan antarmuka aplikasi *game pathfinding* terbagi menjadi tiga rancangan antarmuka yaitu rancangan antarmuka tampilan utama, rancangan antarmuka panduan dan rancangan antarmuka tentang aplikasi.

a. Rancangan tampilan utama

Rancangan antarmuka tampilan utama digunakan untuk menampilkan aplikasi utama. Pada rancangan antarmuka terdapat tombol-tombol yang mempunyai fungsi tertentu yang berguna untuk menjalankan aplikasi. Rancangan antarmuka utama dapat dilihat pada gambar III.26.



Gambar III.26. Rancangan antarmuka menu utama

Keterangan antarmuka tampilan utama dijelaskan pada tabel III.8.

Tabel III.8. Keterangan antarmuka tampilan utama

No.	Nama	*Background* dan *font*	Keterangan
1.	E_Baris	Skin delphi + verdana size 10 hitam	Tempat pengisian nilai baris
2.	E_Kolom	Skin delphi + verdana size 10 hitam	Tempat pengisian nilai kolom

Tabel III.8. Keterangan antarmuka tampilan utama (lanjutan)

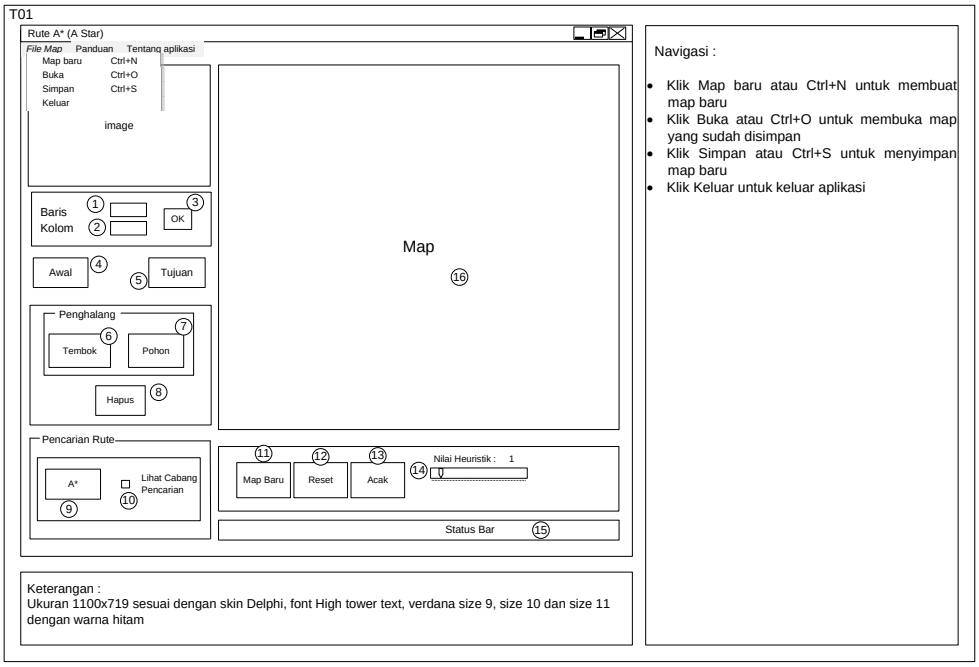
No.	Nama	*Background* dan *font*	Keterangan
3.	Tbl_Grid	Skin delphi + high tower text size 9 hitam	Proses untuk menampilkan nilai baris dan nilai kolom pada *map*
4.	SbStart	Skin delphi + gambar (.bmp) + high tower text size 9 hitam	Proses untuk memasangkan titik awal pada *map*
5.	SbEnd	Skin delphi + gambar (.bmp) + high tower text size 9 hitam	Proses untuk memasangkan titik tujuan pada *map*
6.	SbWalls	Skin delphi + gambar (.bmp) + high tower text size 10 hitam	Proses untuk memasangkan titik tembok pada *map*
7.	SbPohon	Skin delphi + gambar (.bmp) + high tower text size 10 hitam	Proses untuk memasangkan titik pohon pada *map*
8.	SbClear	Skin delphi + gambar (.bmp) + high tower text size 10 hitam	Proses untuk menghapus penghalang pada *map*
9.	btnAStarFindPath	Skin delphi + high tower text size 10 hitam	Proses untuk menampilkan jalur dengan A* pada *map*
10.	CB_ShowSearch	Skin delphi + high tower text size 11 hitam	Proses untuk menunjukkan pelacakan pencarian dengan A*
11.	btnClear	Skin delphi + high tower text size 10 hitam	Proses membuat *map* baru

Tabel III.8. Keterangan antarmuka tampilan utama (lanjutan)

No.	Nama	*Background* dan *font*	Keterangan
12.	btnRandomize	Skin delphi + high tower text size 10 hitam	Proses mengacak posisi titik awal dan titik tujuan
13.	btnReset	Skin delphi + high tower text size 10 hitam	Proses untuk menghapus jalur yang telah ada pada *map*
14.	tbHeuristic	Skin delphi	Proses menggeserkan kursor untuk memilih nilai heuristik yang akan digunakan
15.	StatusBar1	Skin delphi + hitam	Proses untuk menampilkan status proses yang telah dilakukan pengguna
16.	sgMap	Skin delphi	Tempat untuk pemasangan posisi titik awal, titik tujuan, penghalang dan menampilkan jalur pencarian

b. Rancangan tampilan menu *file map*

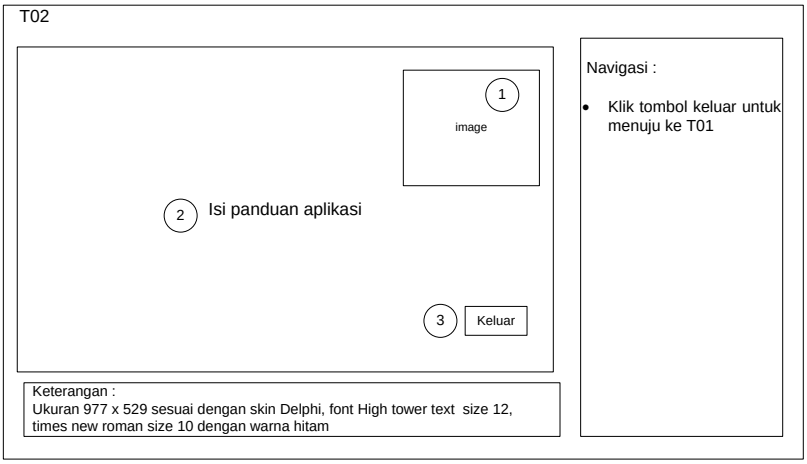
Pada rancangan antarmuka tampilan *file map* terdapat fungsi-fungsi yaitu membuat map baru, buka map, simpan map dan keluar aplikasi. Rancangan antarmuka menu *file map* dapat dilihat pada gambar III.27.



Gambar III.27. Rancangan antarmuka menu *file map*

c. Rancangan tampilan panduan

Rancangan antarmuka tampilan panduan digunakan untuk menampilkan data panduan yang bermanfaat membantu pengguna untuk mengetahui fungsi tombol yang terdapat pada aplikasi. Rancangan antarmuka panduan dapat dilihat pada gambar III.28.



Gambar III.28. Rancangan antarmuka panduan

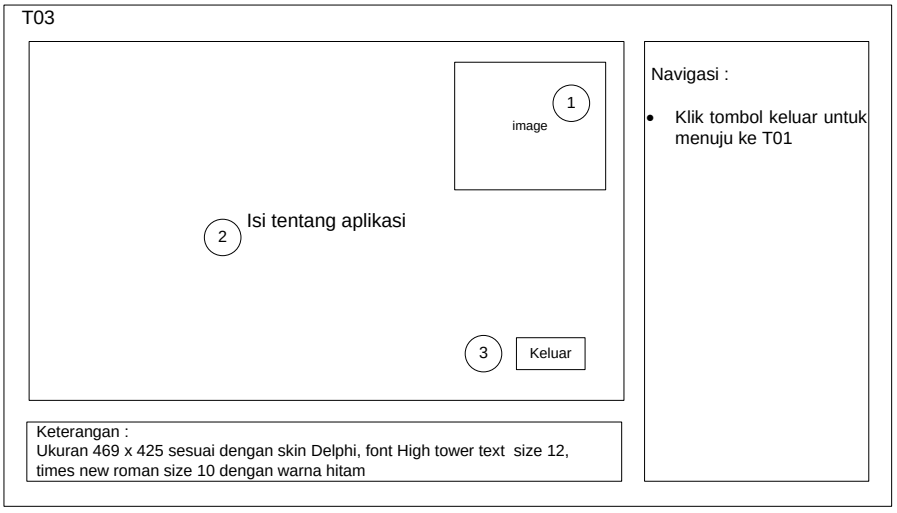
Keterangan antarmuka panduan dijelaskan pada tabel III.9.

Tabel III.9. Keterangan antarmuka panduan

No.	Nama	*Background* dan *font*	Keterangan
1.	Image	Gambar (.jpg)	Tampilan untuk gambar
2.	Panel_IsiBantuan	Gambar (.jpg) + high tower text size 12 hitam	Tampilan panduan penggunaan aplikasi
3.	Tbl_KeluarBantuan	Skin delphi + gambar (.bmp) + times new roman size 10 hitam	Proses keluar dari T02 menuju T01

d. Rancangan tampilan tentang aplikasi

Rancangan antarmuka tampilan tentang aplikasi digunakan untuk menampilkan data tentang aplikasi. Rancangan antarmuka tentang aplikasi dapat dilihat pada gambar III.29.



Gambar III.29. Rancangan antarmuka tentang aplikasi

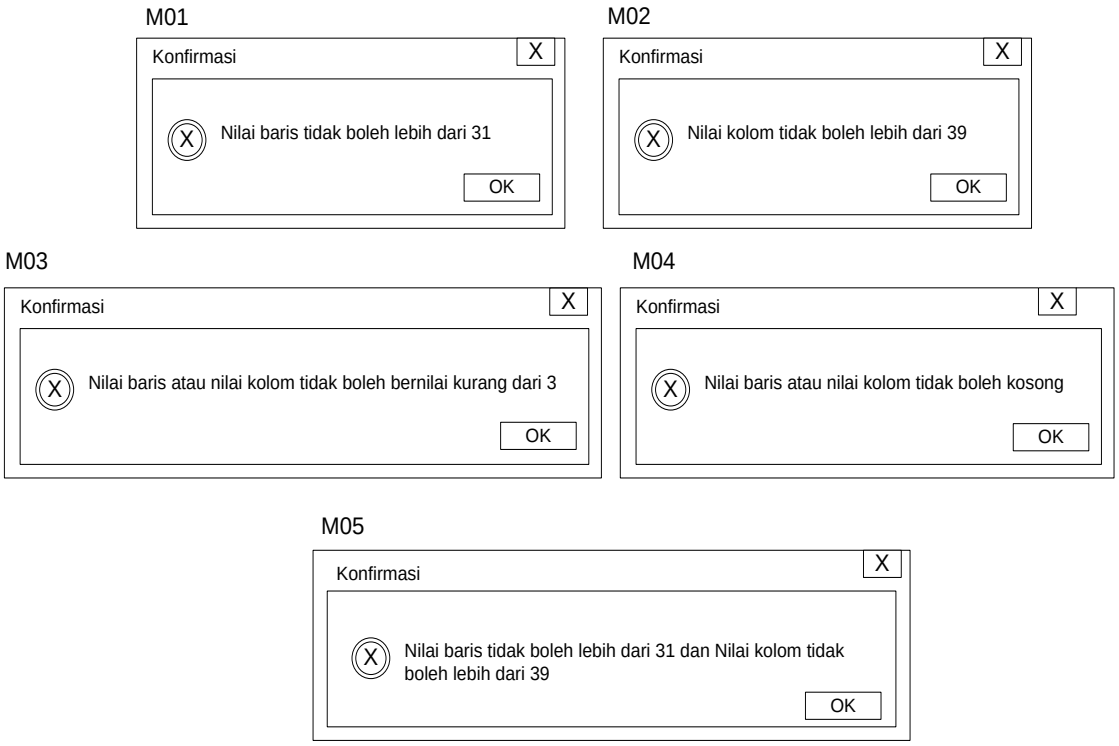
Keterangan antarmuka tentang aplikasi dijelaskan pada tabel III.10.

Tabel III.10. Keterangan antarmuka tentang aplikasi

No.	Nama	*Background* dan *font*	Keterangan
1.	Image	Gambar (.jpg)	Tampilan untuk gambar
2.	Panel_IsiAbout	Gambar (.jpg) + high tower text size 12 hitam	Tampilan isi tentang aplikasi
3.	Tbl_KeluarAbout	Skin delphi + gambar (.bmp) + times new roman size 10 hitam	Proses keluar dari T03 menuju T01

e. Rancangan tampilan pesan

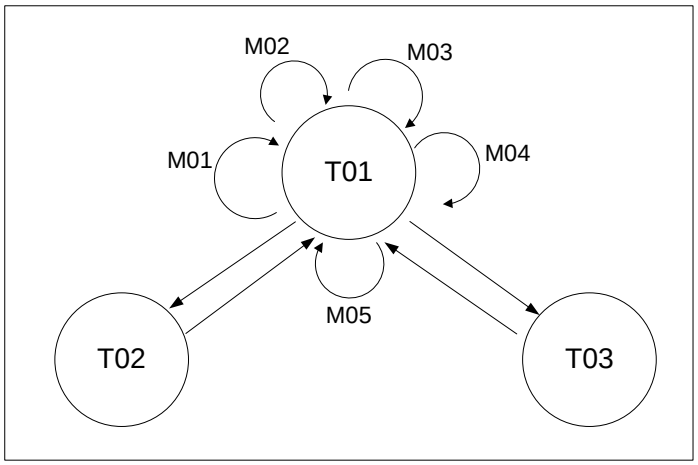
Pada gambar III.30 adalah perancangan pesan yang muncul sebagai informasi kepada pengguna dalam mengoperasikan sistem.



Gambar III.30. Rancangan antarmuka pesan

f. Jaringan semantik

Perancangan jaringan semantik menggambarkan alur informasi dari setiap antarmuka aplikasi yang dijelaskan pada gambar III.31.



Gambar III.31. Jaringan semantik

Keterangan :

- T01 : Tampilan utama
- T02 : Tampilan panduan
- T03 : Tampilan tentang aplikasi
- M01 : *Message* konfirmasi nilai baris tidak boleh lebih dari 31
- M02 : *Message* konfirmasi nilai kolom tidak boleh lebih dari 39
- M03 : *Message* konfirmasi nilai baris atau nilai kolom kurang dari 3
- M04 : *Message* konfirmasi nilai baris atau nilai kolom tidak boleh kosong
- M05 : *Message* konfirmasi nilai baris tidak boleh lebih dari 31 dan nilai kolom tidak boleh lebih dari 39