BAB 2
TINJAUAN PUSTAKA

2.1. Tinjauan Pustaka

Penulis telah melakukan kajian terhadap beberapa penelitian dari beberapa sumber pustaka. Penelitian yang dilakukan oleh penulis berbeda dari penelitian-penelitian tersebut, namun terdapat beberapa kesamaan yang dapat penulis gunakan sebagai pendukung teori-teori dalam penelitian ini.

Leenen, Vorster, Roux (2010) melakukan penelitian tentang algoritma untuk menyelesaikan masalah *Dynamic Military Unit Path Finding Problem* (DMUPFP). *Military Unit Path Finding Problem* (MUPFP) adalah masalah pencarian jalur dari titik awal hingga titik tujuan, di mana sebuah unit militer harus berpindah secara aman selagi menghindari penghalang, dengan biaya perjalanan (*path cost*) seminimal mungkin pada suatu representasi digital dari suatu daerah sebenarnya. Dalam penelitian tersebut, peneliti berpendapat bahwa MUPFP sangat ideal dengan pendekatan berbasis *constraint* karena masalah ini membutuhkan fleksibilitas dalam pembuatan modelnya. Mereka memformulasikan MUPFP sebagai *constraint satisfaction problem* dan ekstensi berbasis *constraint* dari pencarian dengan algoritma A*. Pendekatan yang digunakan peneliti untuk menyelesaikan masalah yang diteliti adalah dengan mengoptimalkan biaya perjalanan (*path cost*) dan memastikan bahwa kriteria lain, misalnya keamanan jalur juga terpenuhi. Pendekatan berbasis *constraint* yang dipakai dalam penelitian tersebut, dapat diimplementasikan dengan pembedaan yang jelas antara tujuan untuk mendapatkan biaya yang optimal dengan tetap memenuhi aturan keamanan.

Penulis menelaah penelitian lain tentang *pathfinding* yang diteliti oleh Björnsson dan Halldórsson (2006). Penelitian tersebut berisi tentang pengembangan heuristik untuk *pathfinding* yang optimal dalam suatu *game map*. Dalam penelitian tersebut, peneliti berpendapat bahwa heuristik yang sering

digunakan dalam pencarian jalur dengan algoritma A* masih sangat sederhana untuk melakukan pencarian terpandu dalam suatu *game world* yang besar dan rumit. Mereka juga berpendapat bahwa hal tersebut yang menyebabkan agen cerdas seringkali menjelajahi seluruh *game world* untuk mencari jalur dengan algoritma A*. Pendekatan yang dilakukan dalam penelitian tersebut adalah dengan mengurangi *state-space exploration*, tetapi masih memperhitungkan kemungkinan adanya *dynamic obstacle*. Peneliti menggunakan *state-space abstraction* untuk mengembangkan heuristik yang digunakan dalam pencarian dengan algoritma A*. Hasil dari pengembangan heuristik dalam penelitian ini ada 2 yaitu *dead-end heuristic* dan *gateway heuristic*. *Dead-end heuristic* bekerja dengan mengeliminasi area-area pencarian yang buntu, sedangkan *gateway heuristic* bekerja dengan menghitung jarak antara titik masuk dan titik keluar suatu area. Untuk menggunakan kedua heuristik tersebut, proses pertama yang diperlukan adalah dengan membagi *map* menjadi area-area yang lebih kecil. Setelah map dibagi menjadi area-area yang lebih kecil, proses selanjutnya untuk *dead-end heuristic* adalah dengan menyusun suatu graf yang merepresentasikan setiap area (direpresentasikan dalam *node*) tersebut dan hubungan antar area (direpresentasikan dalam bentuk *edge*). Setelah graf terbentuk, akan dilakukan pencarian pada graf tersebut untuk menentukan rangkaian area yang valid (dapat sampai *goal*) dan membuang area yang buntu. Proses terakhir untuk *dead-end heuristic* adalah melakukan pencarian dengan algoritma A* biasa dari area-area valid yang telah dihimpun sebelumnya. Proses kedua untuk *gateway heuristic* adalah menghitung jarak antara suatu *node* dengan *node-node* batas dari suatu area (disebut *gateway*), kemudian dihitung pula jarak antara suatu *gateway* dengan *gateway* lain dan jarak antara *gateway* dengan *goal*. Proses terakhir dari *gateway heuristic* adalah mengembangkan nilai heuristik (*h*) dengan nilai-nilai yang telah diperoleh dari proses sebelumnya, nilai heuristik tersebut kemudian digunakan untuk pencarian algoritma A* biasa. Setelah melakukan evaluasi terhadap kedua heuristik tersebut, Björnsson dan Halldórsson menyimpulkan bahwa kedua herustik yang mereka sajikan lebih baik dibandingkan dengan standar heuristik algoritma A*, dilihat dari banyaknya *node expand* mapun waktu pencarian.

Penelitian yang dilakukan penulis juga menerapkan algoritma A* dalam mencari jalur terpendek yang akan dilalui oleh agen cerdas. Berbeda dari penelitian-penelitian yang telah penulis rangkum dalam tinjauan pustaka, pada penelitian ini, agen cerdas tidak harus mendapatkan jalur terpendek dalam menyelesaikan permainan, namun agen cerdas harus mengumpulkan poin dalam mencapai *goal*. Selain itu, dalam penelitian ini, user bertindak sebagai perancang *maze* yang harus diselesaikan oleh sistem dengan menggunakan algoritma A*, bukan sebagai pemain yang menggerakkan karakter tertentu.

2.2. Landasan Teori

Landasan teori memuat tentang teori dan konsep yang menjadi pedoman penulis dalam melakukan penelitian. Isi dari landasan teori ini juga didukung oleh berbagai sumber pustaka yang menjadi acuan penulis dalam penyusunannya. Landasan teori dalam penelitian ini antara lain :

2.2.1. Kecredasan Buatan

Kecerdasan buatan atau *Aritificial Intelligence* merupakan salah satu cabang ilmu dari ilmu komputer. Menurut Russell dan Norvig (2003) pengertian tentang kecerdasan buatan dari satu orang dengan yang lainnya dapat berbeda-beda, namun dari definisi-definisi tersebut, kecerdasan buatan dapat diklasifikasikan kedalam kategori berikut :

a. Sistem yang bertindak seperti manusia.
b. Sistem yang berfikir seperti manusia.
c. Sistem yang berfikir secara rasional.
d. Sistem yang bertindak secara rasional.

Salah satu pengertian dari kecerdasan buatan menurut Winston adalah studi tentang komputasi yang membuatnya dapat merasa, berfikir, dan bertindak. Pendapat lain tentang kecerdasan buatan disampaikan oleh Rich dan Knight yaitu bahwa kecerdasan buatan merupakan studi tentang bagaimana membuat komputer

melakukan hal dimana, pada saat ini, manusia dapat melakukannya dengan lebih baik (seperti yang dikutip dalam Russell dan Norvig, 2003, hlm. 2).

2.2.2. Manhattan Distance

Manhattan Distance menurut Black (2006) merupakan jarak antara 2 titik yang diukur sepanjang sumbu tegak lurus. *Manhattan Distance* merupakan penghitungan yang sederhana sehingga tidak memerlukan *computational cost* yang besar dalam prosesnya. Oleh karena itu, *Manhattan Distance* sering digunakan sebagai nilai heuristik dalam algoritma pencarian. Walaupun demikian, *Manhattan Distance* belum tentu optimal untuk semua kasus. *Manhattan Distance* sangat sesuai digunakan dalam pencarian jalur dalam suatu *grid map* dengan langkah pencarian lurus (tanpa diagonal).

Penghitungan nilai *Manhattan Distance Heuristic* antara 2 titik didapatkan dengan menjumlahkan nilai absolut dari selisih koordinat-koordinatnya. Rumus umum *Manhattan Distance* adalah :

$$h(n) = absolute(target.x - n.x) + absolute(target.y - n.y) \quad \textbf{[2.1]}$$

atau

$$h(n) = |dx - nx| + |dy - ny| \quad \textbf{[2.2]}$$

$h(n)$ = nilai heuristik *node n*
dx = posisi x dari *node goal* (*node* yang dituju)
dy = posisi y dari *node goal* (*node* yang dituju)
nx = posisi x suatu *node*
ny = posisi y suatu *node*

2.2.3. Algoritma A*

Algoritma A* menjelajahi *map* dengan membangkitkan *node-node* anak pada tiap *node* yang dilaluinya. Selain menyimpan lokasi dari tiap *node*, setiap *node* memiliki 3 atribut utama yaitu adalah *fitness* (f), *goal* (g), dan *heuristic* (h) (Rabin, 2002, hlm. 106). Berikut dijelaskan lebih jauh oleh Rabin (2002) mengenai ketiga atribut tersebut :

a. *g* adalah nilai *cost* antara titik awal (*start*) hingga *node* sekarang.
b. *h* adalah nilai perkiraan *cost* antara *node* sekarang hingga *node goal*. Dalam hal ini, h merupakan heuristik.
c. *f* adalah jumlah *g* dan *h*. Nilai *f* merepresentasikan tebakan *cost* optimal suatu *path* yang dilalui suatu *node*.

Dari ketiga atribut *node* tersebut, didapat persamaan sebagai berikut :

$$f(n) = g(n) + h(n) \quad \textbf{[2.3]}$$

Algoritma A* memiliki dua buah *list* yang digunakan dalam menjalankan pencariannya, yaitu *open list* dan *close list*. Menurut Rabin (2002) *open list* berisi *node-node* yang belum di-*explore*, sedangkan *close list* berisi semua *node* yang telah di-*explore*. Suatu *node* disebut telah di-*explore* jika dalam proses pencarian, *node* tersebut telah dikunjungi dan setiap *node* anak di sekitarnya telah diperiksa.

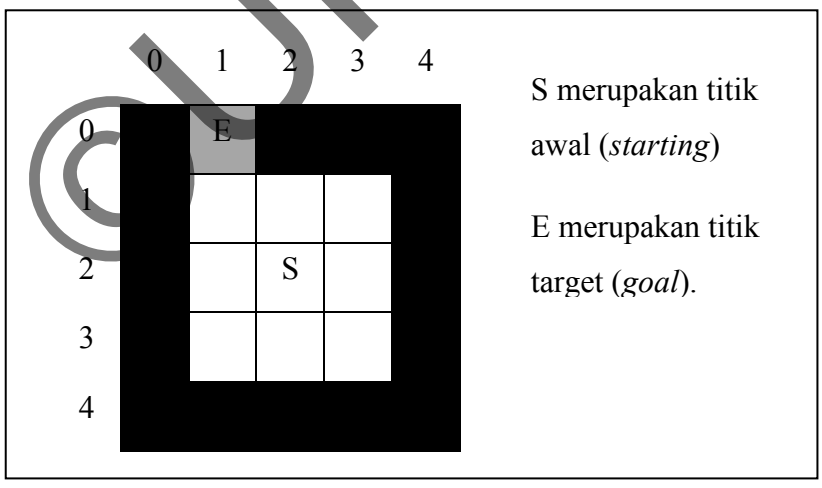
Pencarian dengan algoritma A* dapat digambarkan ke dalam *psudo-code* seperti berikut (Rabin, 2002) :

1. Let *P* = the starting point.
2. Assign *f*, *g* and *h* values to P.
3. Add *P* to the Open list. At this point, *P* is the only node on the Open list.
4. Let *B* = the best node from the Open list (best node has the lowest *f*-value).
   a. If *B* is the goal node, then quit—a path has been found.
   b. If the Open list is empty, then quit—a path cannot be found.
5. Let C = a valid node connected to B.
   a. Assign *f*, *g*, and *h* values to *C*.
   b. Check whether *C* is on the Open or Closed list.
      i. If so, check whether the new path is more efficient (lower *f*-value).
         1. If so, update the path.
      ii. Else, add *C* to the Open list.
   c. Repeat step 5 for all valid children of *B*.
6. Repeat from step 4.

Gambar 2.1 *Pseudo-code Algoritma A

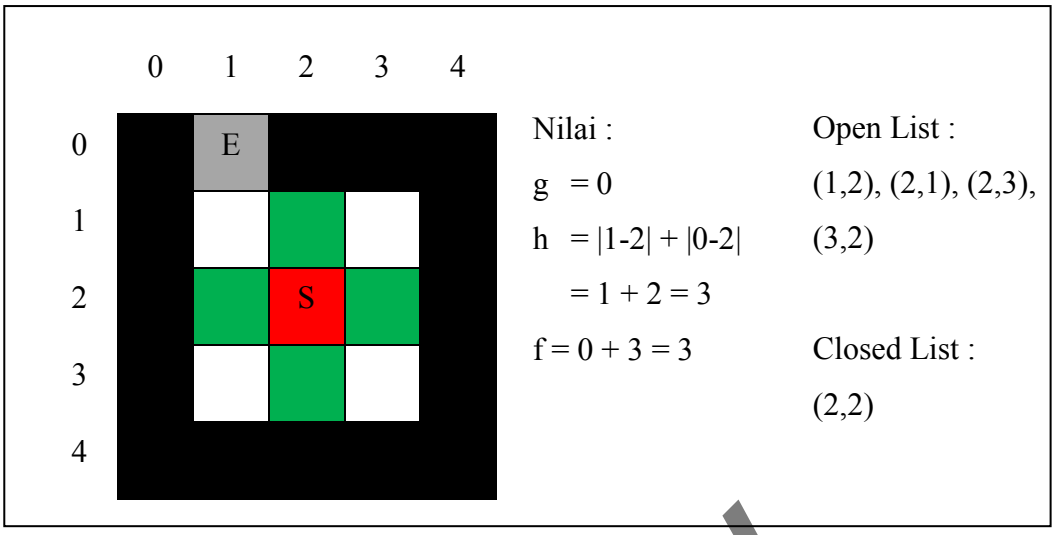
Dikutip dari : Rabin. Steve. (2002). AI Game Programming Wisdom. Rockland: Charles River Media Inc., hlm.107.

Berikut contoh langkah-langkah pencarian jalur pada *map* sederhana dengan menggunakan algoritma A* :



Gambar 2.2 Contoh *Map Maze* 5x5

Gambar 2.2 merupakan contoh *map grid* atau *tile*. *Node* S merupakan titik mulai (*start*). Tujuan atau *goal* dari *map* ini adalah titik E.



Gambar 2.3 Pencarian Jalur *Map Maze* 5x5 Langkah 1

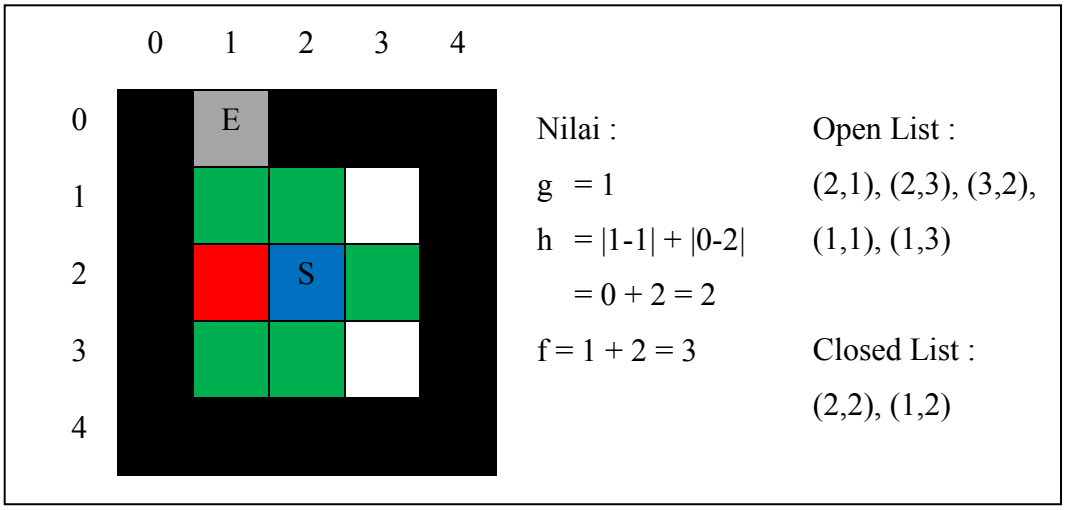
Gambar 2.3 menunjukkan langkah pertama dalam proses pencarian jalur *map* pada gambar 2.2. Nilai *g* merupakan nilai *cost* dari *start node* (S). Nilai h merupakan heuristik, nilainya didapat dari perhitungan *Manhattan Distance* pada rumus 2.2. Semua *children* untuk *node* pertama adalah:

Tabel 2.1

Node Children Untuk *Node* (2,2)

Child	*G*	*h*	*f*
(1,2)	1	|1-1|+|0-2| = 2	1 + 2 = 3
(2,1)	1	|1-2|+|0-1| = 2	1 + 2 = 3
(2,3)	1	|1-2|+|0-3| = 4	1 + 4 = 5
(3,2)	1	|1-3|+|0-2| = 4	1 + 4 = 5

Child (1,2) dan (2,1) memiliki nilai *f* yang sama, namun karena *node* selanjutnya dipilih hanya jika nilai *f* terkecil maka *child* yang dipilih sebagai *node* selanjutnya adalah (1,2).



Gambar 2.4 Pencarian Jalur *Map Maze* 5x5 Langkah 2

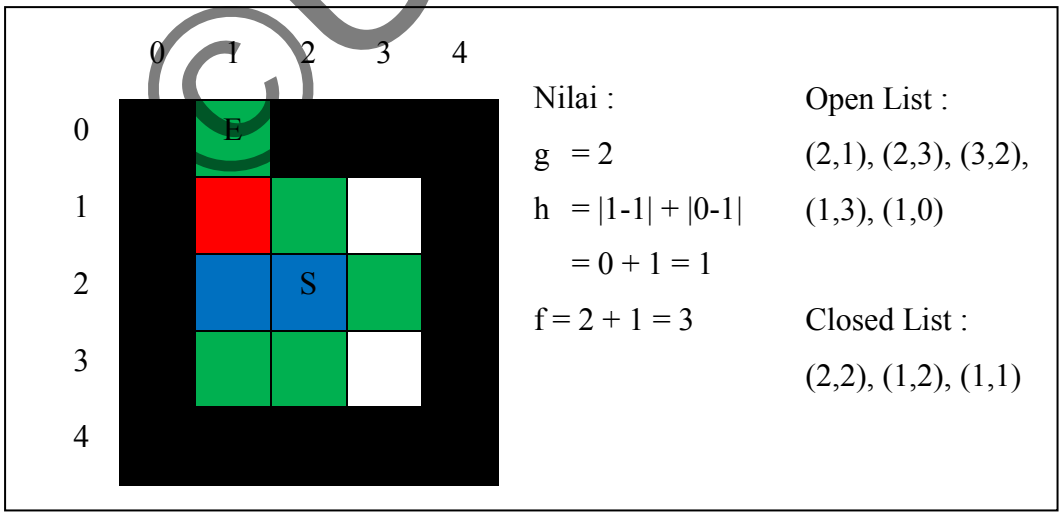
Semua *node* anak untuk *node* (1,2) adalah *node* :

Tabel 2.2

Node Children Untuk *Node* (1,2)

Child	*G*	*h*	*f*
(1,1)	2	|1-1|+|0-1| = 1	2 + 1 = 3
(1,3)	2	|1-1|+|0-3| = 3	2 + 3 = 5

Child yang dipilih sebagai *node* selanjutnya adalah (1,1).



Gambar 2.5 Pencarian Jalur *Map Maze* 5x5 Langkah 3

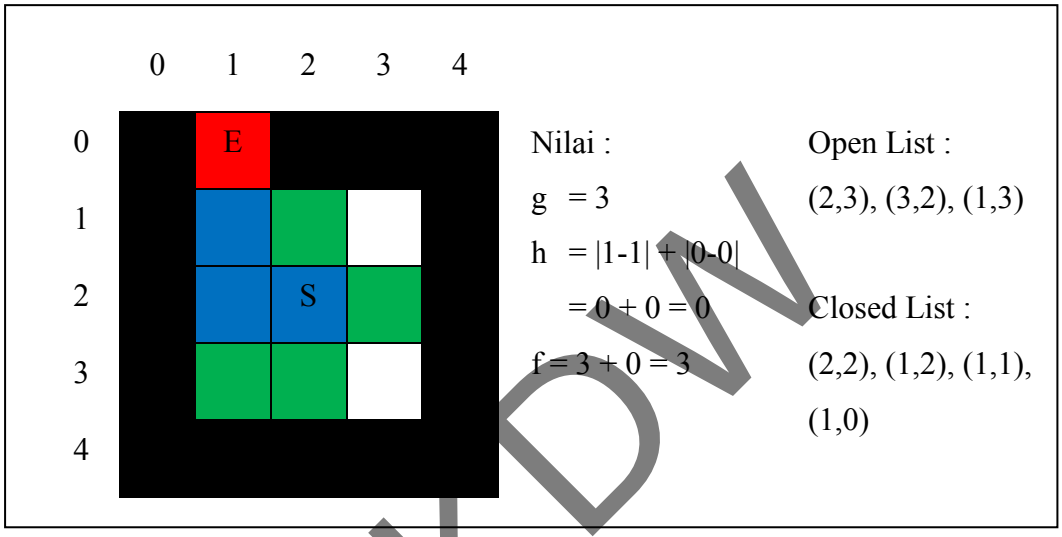
Semua *node* anak untuk *node* (1,1) adalah *node* :

Tabel 2.3

Node Children Untuk *Node* (1,1)

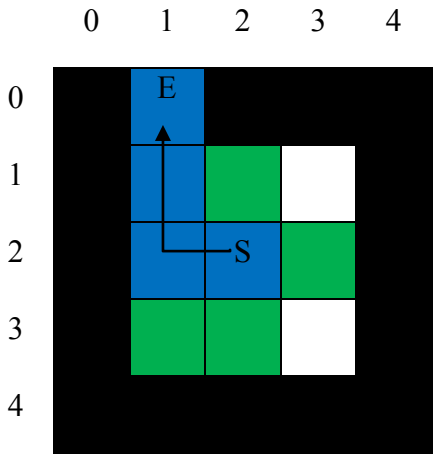
Child	*G*	*h*	*f*
(1,0)	3	|1-1|+|0-0| = 0	3 + 0 = 3
(2,1)	1	|1-2|+|0-1| = 2	1 + 2 = 3

Child yang dipilih sebagai *node* selanjutnya adalah (1,0).



Gambar 2.6 Pencarian Jalur *Map Maze* 5x5 Langkah 4

Langkah 4 (gambar 2.6) merupakan proses pemeriksaan *node* (1,0). *Node* yang diperiksa ternyata merupakan target (*goal*). Oleh karena itu, proses pencarian dihentikan dan jalur telah ditemukan. Jalur yang ditemukan untuk menyelesaikan *map* ini adalah : (2,2) → (1,2) → (1,1) → (1,0). Jalur yang didapat merupakan jalur yang optimal atau jalur terpendek untuk menyelesaikan masalah pencarian jalur *map maze* 5x5 tersebut.



Gambar 2.7 Penyelesaian Pencarian Jalur Untuk *Map Maze* 5x5

Gambar 2.7 menunjukkan *goal state* dari pencarian jalur *map maze* 5x5 pada gambar 2.2 dengan algoritma A*. Dalam gambar tersebut ditunjukkan bahwa jalur yang digunakan untuk menyelesaikan masalah pencarian *goal* pada *map maze* tersebut adalah *node* (2,2), *node* (1,2), *node* (1,1), *node* (1,0).

BAB 3
RANCANGAN SISTEM

3.1. Kebutuhan Sistem

Sistem yang dibuat memiliki spesifikasi tertentu agar dapat berjalan dengan baik. Spesifikasi tersebut termasuk kebutuhan minimal hardware dan software yang akan digunakan untuk menjalankan sistem. Kebutuhan hardware dan software untuk sistem yang dibangun adalah sebagai berikut :

3.1.1. Kebutuhan Hardware dan Software

Spesifikasi perangkat keras yang dibutuhkan untuk menjalankan sistem yang dibangun adalah : Prosesor Intel Pentium III 500 MHz atau AMD Athlon 600MHz, RAM 1024 MB. Untuk perangkat grafis disarankan menggunakan VGA yang mendukung DirectX 9 ke atas. Sistem yang dibangun membutuhkan media penyimpan dengan kapasitas minimal 10 MB. Dibutuhkan pula *mouse* sebagai perangkat *input* agar sistem dapat berjalan dengan baik.

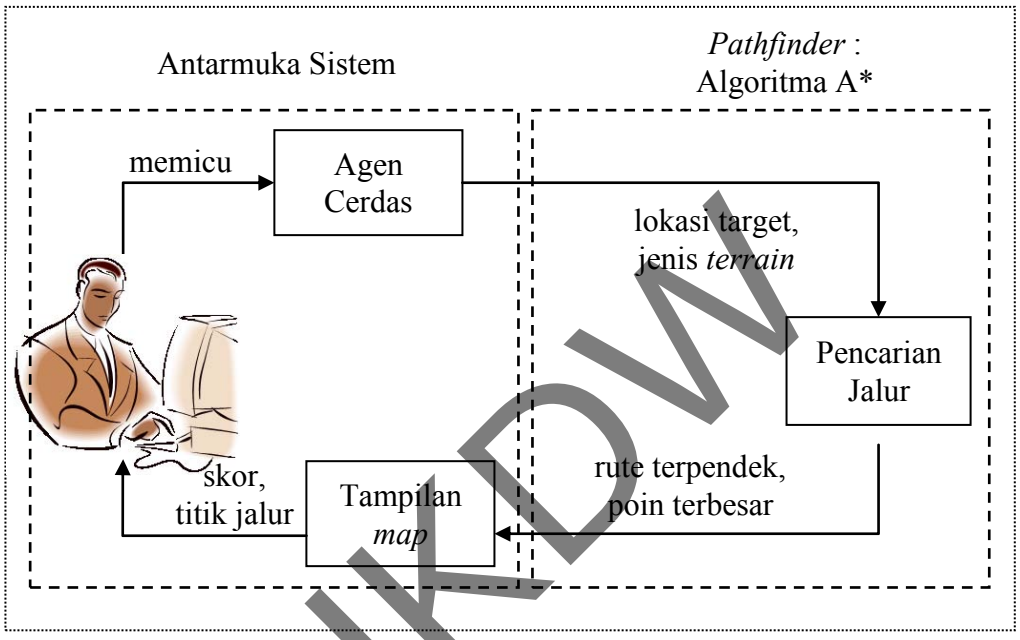
Kebutuhan minimum perangkat lunak untuk menjalankan sistem yang dibangun adalah : Sistem operasi Microsoft Windows 2000, Microsoft Windows XP, Microsoft Windows Vista (32/64 bit), Microsoft Windows 7 (32/64 bit), Mac OS X, dan Linux. Dibutuhkan Adobe Flash Player versi 9 ke atas. Selain itu dibutuhkan *web browser* seperti Mozilla Firefox versi 3.6, Google Chrome Browser, Opera Browser versi 10, atau Internet Explorer versi 7.

3.2. Perancangan Proses

Proses perancangan dalam penelitian ini bertujuan untuk mengubah kebutuhan-kebutuhan dan data yang telah dikumpulkan menjadi informasi yang mendukung dalam proses implementasi nantinya.

3.2.1. Rancangan Arsitektur Sistem

Sistem yang dibangun memiliki rancangan arsitektur sistem yang ditunjukkan oleh gambar 3.1. Di dalam diagram tersebut terdapat gambaran umum mengenai *layer-layer* yang ada di dalam sistem beserta aliran sistem yang terjadi di antara *layer-layer* tersebut :



Gambar 3.1 *Architecture Flow* Sistem

Gambar 3.1 merupakan gambaran besar proses yang terjadi di dalam sistem. Secara garis besar, proses tersebut berlangsung sebagai berikut : Pemain akan memicu bergeraknya agen cerdas, karena pemain yang memberi *inputan* untuk mengaktifkan agen cerdas. Kemudian agen cerdas akan membaca *inputan* dari pemain dan mengirimkan kepada modul pencarian jalur. Di modul ini terdapat proses pencarian jalur oleh algoritma A*. Setelah proses pencarian jalur selesai, hasilnya berupa rute terpendek dan poin, akan ditampilkan di dalam *interface* melalui *map*. Player mendapatkan hasil akhir berupa skor dan titik-titik jalur yang tertera di dalam *map*.

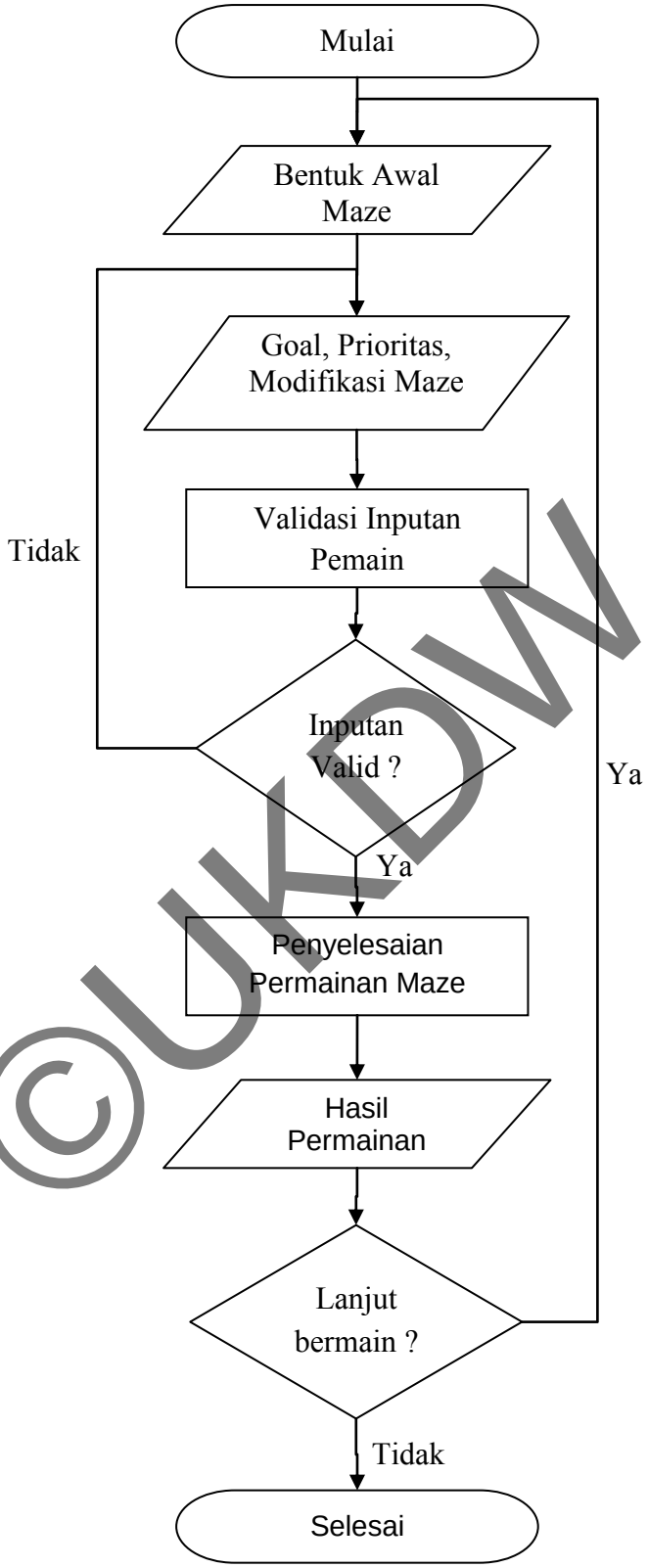
3.2.2. Diagram Alir Sistem

Sistem akan diawali dengan tampilan menu utama ketika mulai dijalankan. Dari tampilan menu utama tersebut, pemain dapat memilih untuk menuju halaman permainan, menuju halaman bantuan, atau mengakhiri permainan. Bila pemain memilih untuk bermain, maka sistem akan menampilkan halaman utama permainan dan menjalankan algoritma permainan. Permainan atau sistem yang dibangun memiliki agoritma sebagai berikut :

Langkah 0 : Mulai.
Langkah 1 : Sistem menampilkan bentuk *maze* awal.
Langkah 2 : Tentukan prioritas *goal* yang harus dicapai dan modifikasi *maze*.
Langkah 3 : Sistem melakukan validasi *input* dari pemain. Bila *input* yang diberikan tidak valid, ulang langkah 2
Langkah 4 : Sistem melakukan proses penyelesaian *maze*.
Langkah 5 : Sistem menampilkan langkah penyelesaian dan skor.
Langkah 6 : Pilih melanjutkan atau keluar dari sistem.
Langkah 7 : Jika memilih melanjutkan, ulang dari langkah 1.
Langkah 8 : Selesai.

Pada saat memasuki permainan utama, maka algoritma sistem mulai dijalankan. Sistem akan menampilkan *map maze* bentuk awal. Setelah itu, pemain dapat memodifikasi *map maze*, menentukan target, dan menentukan prioritas *goal*. *Input* yang diberikan pemain kemudian akan divalidasi sistem. Bila semua inputan valid, maka sistem dapat melakukan proses penyelesaian *map maze* dan menampilkan hasilnya kepada pemain.

Untuk memperjelas algoritma tersebut, berikut disajikan *flowchart* dari sistem yang akan dibangun :



Gambar 3.2 *Flowchart* Sistem

3.2.3. Perancangan *Game World*

Game world dalam permainan ini berbentuk *map maze* yang terbentuk dari *grid-grid*. Panjang maksimal *map maze* di dalam permainan ini adalah 25 *grid* horizontal dan 20 *grid* vertikal dengan setiap *grid* berbentuk persegi dengan ukuran 30 unit.

Grid-grid yang ada dalam satu *map maze* ini memiliki atribut masing-masing seperti warna, gambar, jenis, *id*, dan *label*. Jenis *grid* direpresentasikan dengan warna dan gambar yang unik. Jenis *grid* ini menentukan bahwa suatu *grid* dapat dilewati agen cerdas atau merupakan penghalang.

Warna tertentu dan gambar yang ada di dalam *grid* merupakan tanda khusus bahwa *grid* tersebut dapat digeser atau dirubah jenisnya. Terdapat 1 jenis *grid* khusus yaitu *grid* yang berisi koin atau hadiah. *Grid* tersebut dapat dilewati agen cerdas namun tidak bisa digeser.

3.2.4. Perancangan *Input* dan *Output*

Input yang dapat diberikan oleh pemain dalam permainan ini adalah *mouse click*. Input tersebut hanya berlaku untuk *button*, *slider*, *stepper*, dan *grid* di dalam *map maze*. Setiap *input* yang diterima akan diberi umpan balik (*feedback)* agar pemain mengerti bahwa *input* yang diberikan telah diterima sistem. Umpan balik tersebut berupa perubahan warna, pergerakan, perubahan angka, dan sebagainya.

Output yang diberikan kepada pemain berupa notifikasi dan hasil penyelesaian permainan. Notifikasi yang dimaksud adalah notifikasi jika terjadi kesalahan dalam permainan. Notifikasi tersebut akan ditampilkan dalam bentuk halaman khusus. Untuk hasil penyelesaian permainan, *output* yang diberikan adalah tulisan skor dan juga langkah pencarian pada *grid map maze*. Tanda-tanda tersebut berupa titik-titik atau gambar yang menunjukkan jalur pencarian yang dilakukan oleh agen cerdas.

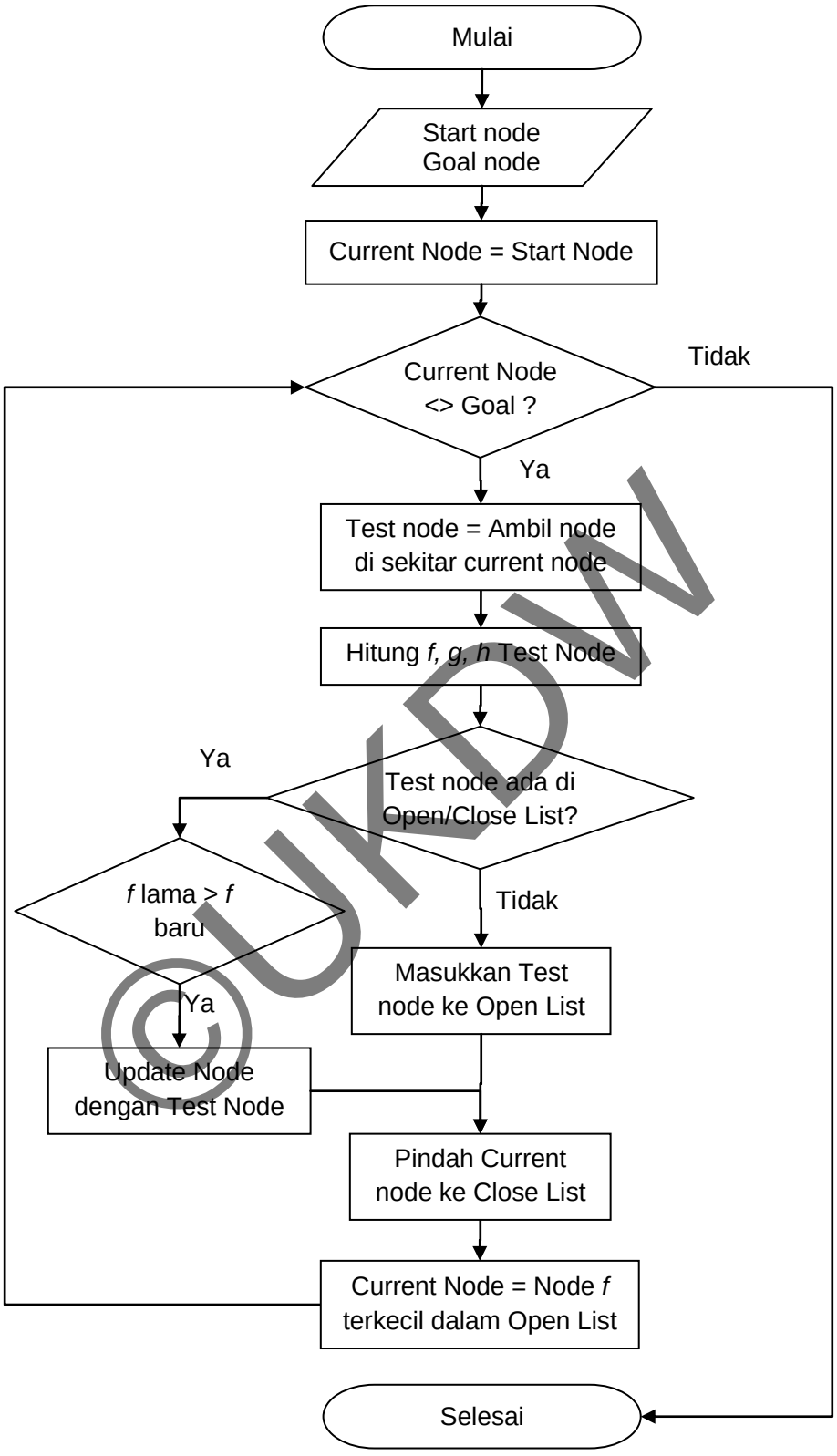
3.2.5. Perancangan Algoritma A* pada Permainan *Maze*

Pencarian jalur pada sistem yang dibangun menggunakan algoritma A*. Algoritma A* merupakan algoritma yang *complete* dan dapat *optimal* selama heuristik yang diberikan tepat (Russell dan Norvig, 2003).

Pencarian dengan algoritma A* dalam sistem yang dibangun berjalan sesuai langkah-langkah sebagai berikut :

1. Masukkan *node start* ke dalam *Open List*.
2. Ambil *node start* sebagai *node current,* sebut sebagai *node* C;
3. Periksa apakah *node current* adalah *node goal*, jika tidak maka jalankan langkah 4.
    a. Jika *node current* adalah *node goal*, maka jalur telah ditemukan, pencarian berakhir.
    b. Jika tidak ada *node current* (*Open List* kosong), jalur tidak mungkin ditemukan, hentikan pencarian.
4. Periksa semua *node* valid di sekitar *node* C dengan ketentuan row – 1, row + 1, col – 1, col + 1 dan bukan merupakan penghalang, sebut sebagai *node* N.
    a. Tentukan nilai *f*, *g*, dan *h* dari *node* N.
    b. Periksa apakah *node* N ada di dalam *Open List* atau *Close List*.
        i. Jika *node* N tidak ada di dalam *Open List* atau *Close List*, masukkan *node* N ke dalam *Open List*.
        ii. Jika *node* N ada di dalam *Open List* atau *Close List*, periksa apakah nilai baru *f* dari *node* N lebih kecil.
            1. Jika nilai baru *f* dari *node* N lebih kecil maka perbaharui *node* lama dengan *node* N.
5. Pindahkan *node current* dari *Open List* ke dalam *Close List*.
6. Ambil *node* terbaik (*f* terkecil) dari *Open List* sebagai *node current*.
7. Kembali ke langkah 3.

Gambar 3.3 berikut menunjukkan *flowchart* dari algoritma A* yang diterapkan dalam sistem.



Gambar 3.3 Flowchart Algoritma A*

Fungsi pencarian (*f*) dengan menggunakan algoritma A* membutuhkan nilai *goal* (*g*) dan nilai *heuristic* (*h*) dalam melakukan pencarian. Untuk algoritma A* di dalam sistem yang dibangun, nilai *g* dari suatu *node N'* adalah jarak yang telah ditempuh (*cost*) dari *node start* hingga *node N'* tersebut. Nilai ini dapat diperoleh dari nilai *cost node* sebelumnya (*node N*) ditambah dengan nilai *cost* suatu *node* untuk melakukan 1 langkah atau :

$$g(N') = g(N) + \ tile\ cost$$

[3.1]

g(*N'*) = nilai cost dari *node N'*
g(*N*) = nilai cost dari node *N* (satu *node* sebelum menuju *node N'*)
tile cost = nilai *cost* untuk melakukan satu langkah antar *node*

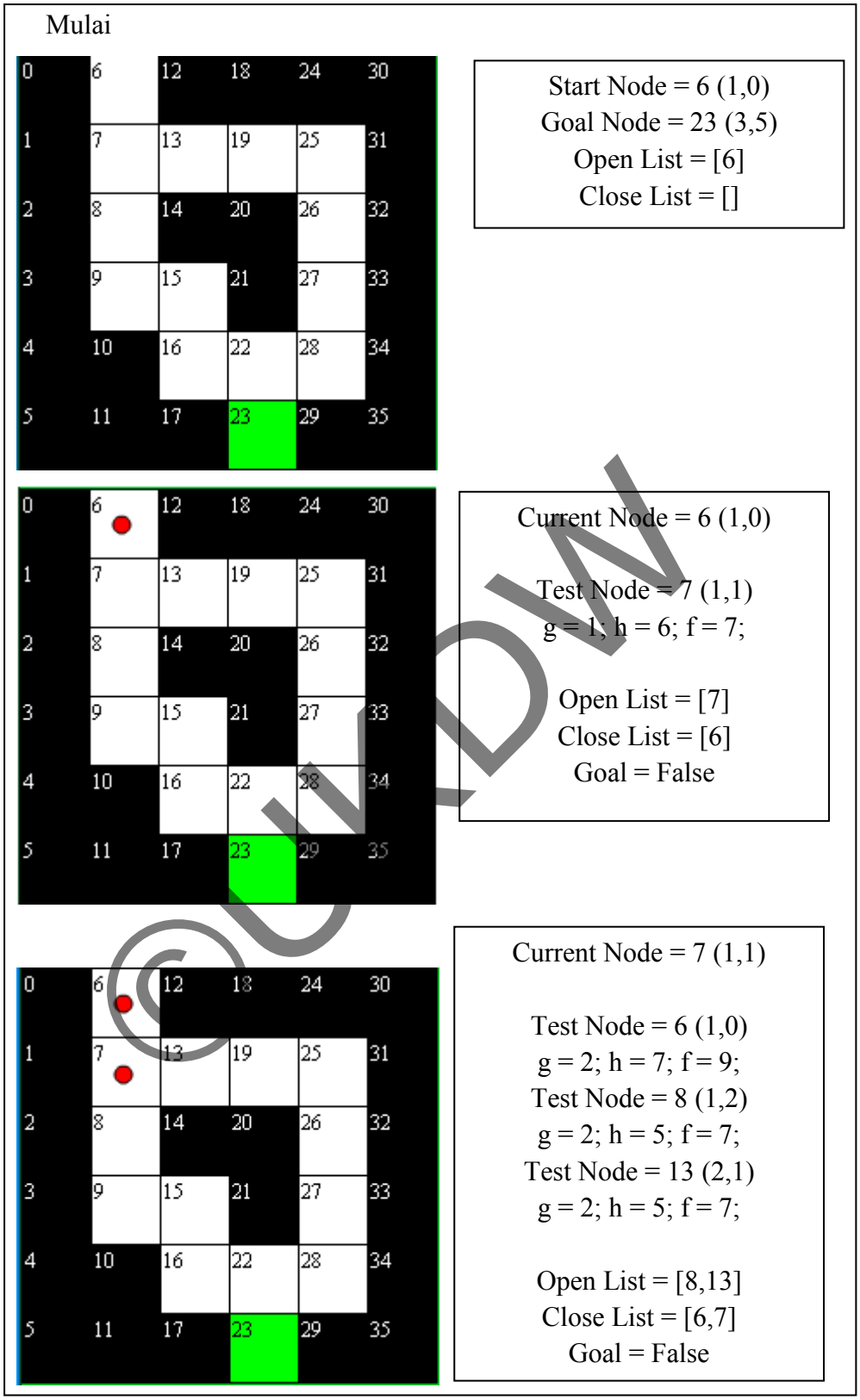
Nilai heuristik (*h*) untuk algoritma A* di dalam sistem yang dibangun didapatkan dengan menghitung jarak antara suatu *node* dengan *node goal* dengan menggunakan perhitungan *Manhattan Distance*. Secara notasi, nilai *h* dari suatu *node N* menuju *node goal G* adalah :

$$h(N) = |Gx - Nx| + \ |Gy - Ny|$$

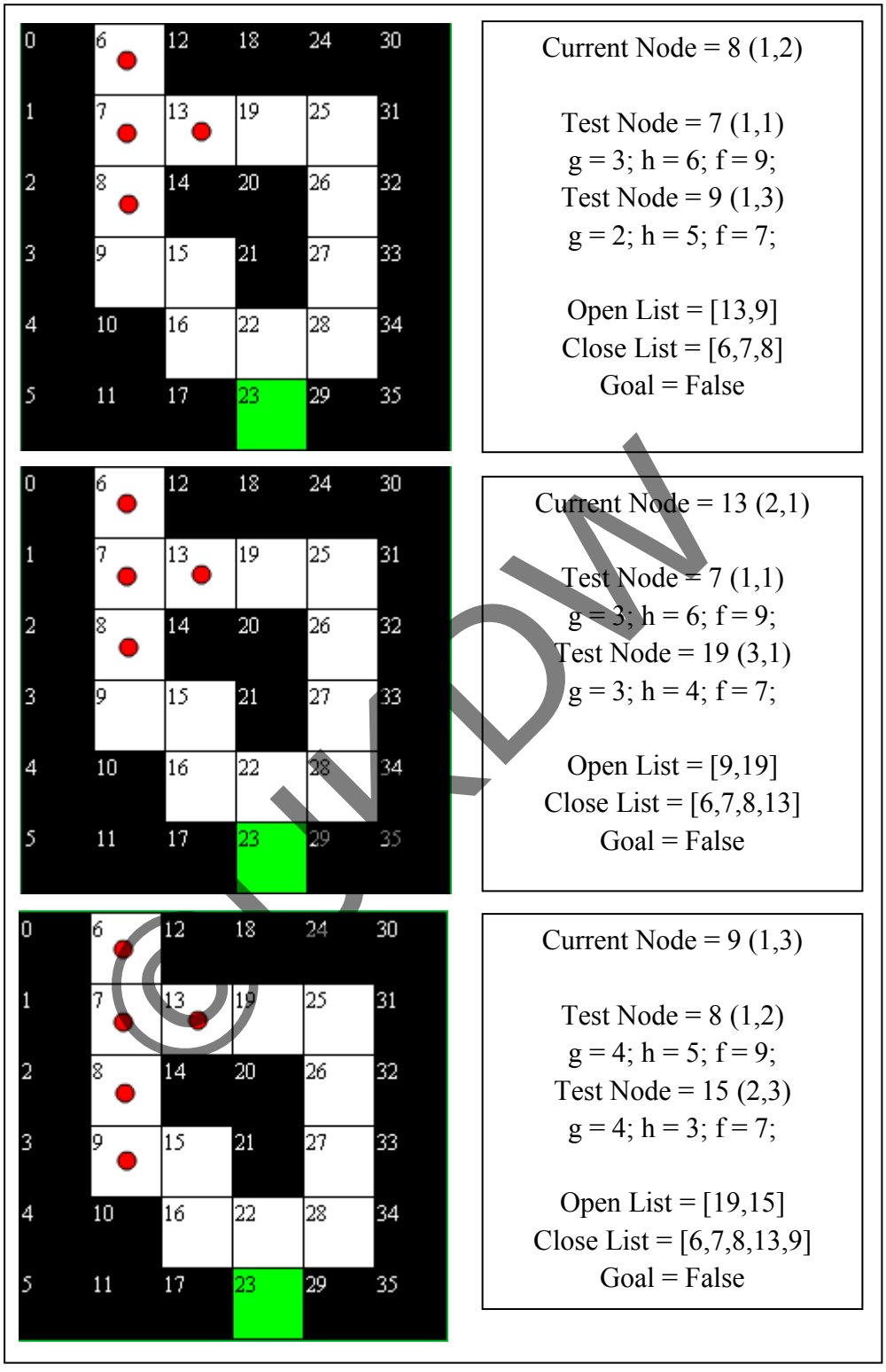
[3.2]

h(*N*) = nilai heuristik dari *node N*
Gx = posisi x dari *node goal* (*node* yang dituju)
Gy = posisi y dari *node goal* (*node* yang dituju)
Nx = posisi x suatu *node*
Ny = posisi y suatu *node*

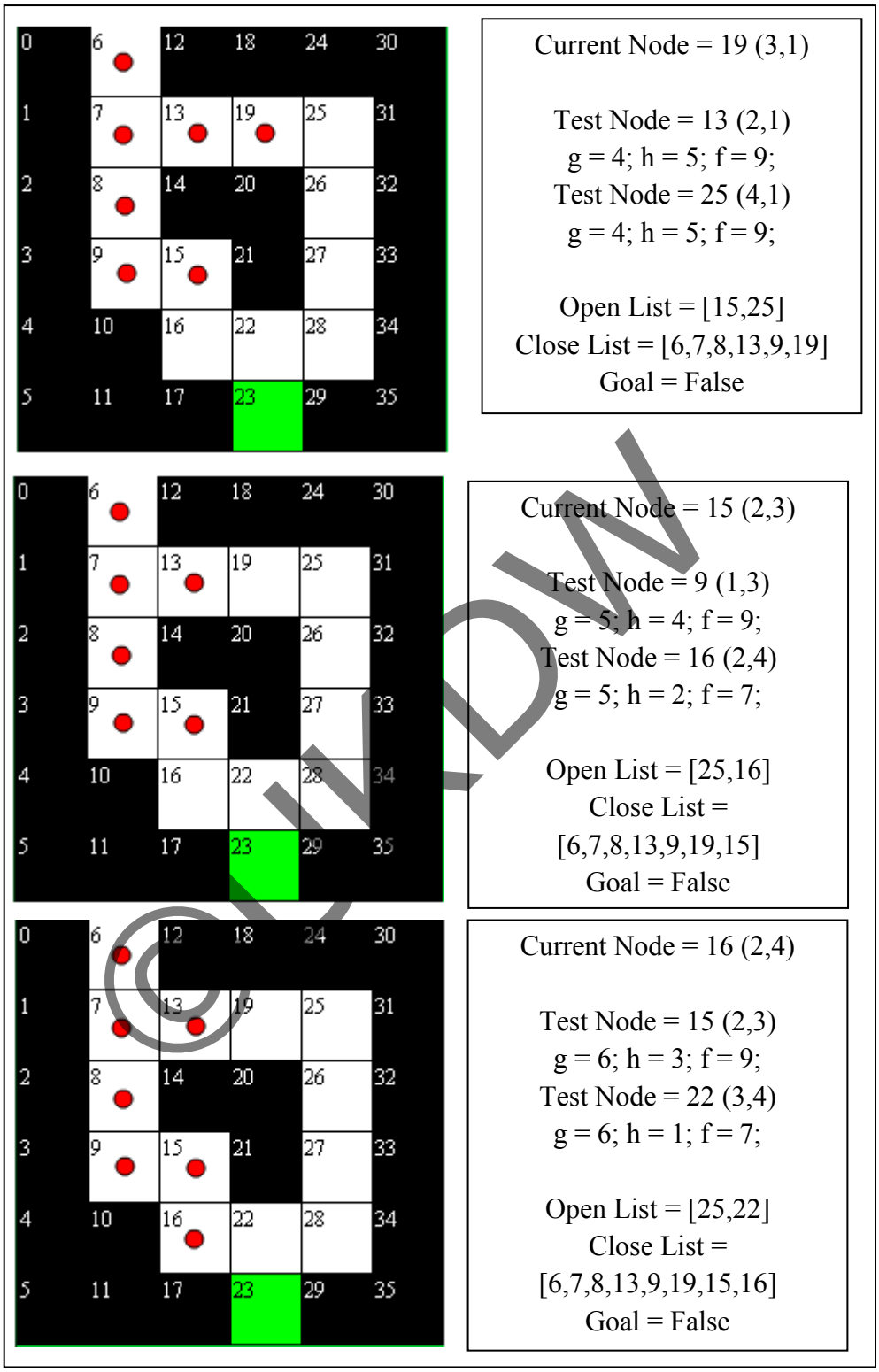
Untuk memperjelas proses algoritma yang diterapkan dalam sistem, penulis menyajikan gambar simulasi langkah-langkah pencarian jalur dari sistem yang dibuat dengan menerapkan algoritma A* :



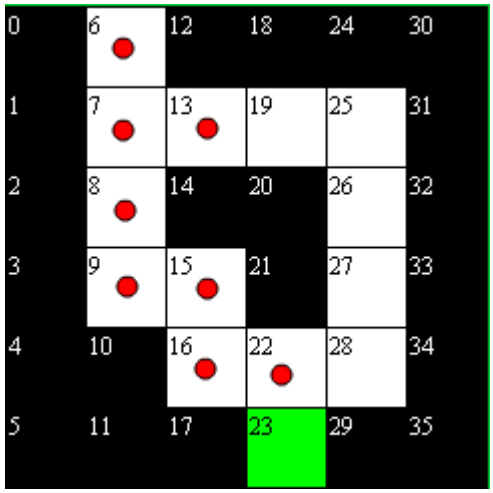
Gambar 3.4 Simulasi Langkah Pencarian dengan Algoritma A*



Gambar 3.4 Simulasi Langkah Pencarian dengan Algoritma A* (lanjutan)

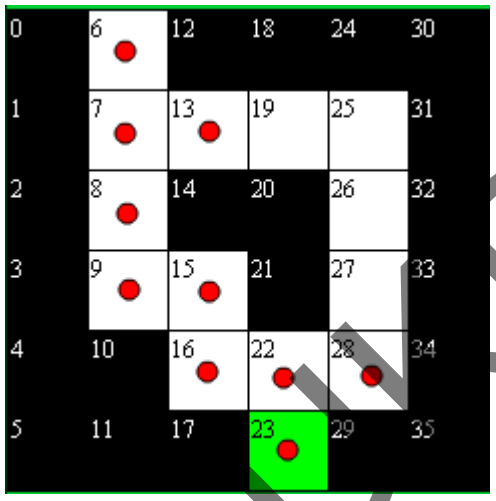


Gambar 3.4 Simulasi Langkah Pencarian dengan Algoritma A* (lanjutan)



Current Node = 22 (4,3)
Test Node = 16
g = 7; h = 2; f = 9;
Test Node = 23
g = 7; h = 0; f = 7;
Test Node = 28
g = 7; h = 2; f = 9;

Open List = [23,28]
Close List =
[6,7,8,13,9,19,15,16,22]
Goal = False

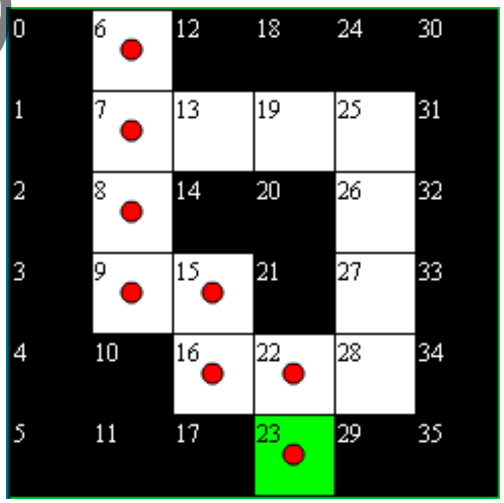


Current Node = 23 (3,5)

Open List = [25, 28]
Close List =
[6,7,8,13,9,19,15,16,23]

Goal = True
Goal Ditemukan !

Gambar 3.4 Simulasi Langkah Pencarian dengan Algoritma A* (lanjutan)



Gambar 3.5 Pencarian Algoritma A* Telah Selesai

Setelah pencarian jalur yang dilakukan oleh algoritma A* selesai, akan ditampilkan output berupa hasil pencarian. Jika ternyata algoritma A* tidak dapat menyelesaikan pencarian, maka akan keluar notifikasi *error*.

3.2.6. Perancangan Pengembangan Algoritma A* pada Permainan *Maze*

Implementasi algoritma yang digunakan di dalam sistem membutuhkan pengembangan algoritma A*. Hal ini dilakukan agar memungkinkan agen cerdas untuk menemukan jalur yang optimal secepat mungkin dan mendapatkan poin-poin sebanyak mungkin. Agar dapat memenuhi tujuan tersebut, penulis perlu merancang pembobotan dan heuristik yang tepat untuk algoritma A* yang diterapkan di dalam sistem. Deskripsi dari *goal* di dalam penelitian ini adalah agen cerdas dapat mencapai titik tujuan *(node goal)* dengan mendapatkan poin sebanyak mungkin. Dalam penelitian ini, *goal* tersebut dapat dibagi menjadi 2 yaitu *primary goal* dan *secondary goal*. Untuk *map maze* di dalam sistem ini, *primary goal* adalah *node goal* sedangkan *secondary goal* adalah *node-node* koin. Seperti halnya *node goal*, *node-node* koin menempati lokasi tertentu yang dapat dilewati oleh agen cerdas di dalam *map maze*, informasi ini dapat digunakan untuk mengembangkan heuristik algoritma A*.

Nilai heuristik yang digunakan untuk pengembangan algoritma A* ini diperoleh dengan menghitung jarak antara suatu *node N* dengan *node koin* terdekat (*node K)*. Perhitungan jarak ini dapat dilakukan dengan *Manhattan Distance*. Perhitungan heuristik untuk algoritma A* yang dikembangkan dapat dinotasikan sebagai berikut :

$$h(N) = |Kx - Nx| + |Ky - Ny| \quad \textbf{[3.3]}$$

$h(N)$ = nilai heuristik dari *node N*

Kx = posisi x dari *node* koin terdekat dengan *node N*

Ky = posisi y dari *node* koin terdekat dengan *node N*

Nx = posisi x suatu *node*

Ny = posisi y suatu *node*

Goal yang ada di dalam penelitian ini memungkinkan sistem untuk mendapatkan lebih dari 1 *goal*. Bila terdapat lebih dari 1 *goal*, maka nilai heuristik yang dipakai adalah nilai heuristik yang terkecil dari suatu *node* terhadap semua *goal* yang ada. Sehingga bila suatu *node N* memiliki nilai herusitik terhadap *node koin* K_1 sebesar h_1, terhadap *node koin* K_2 sebesar h_2, dan terhadap *node koin* K_3 sebesar h_3, maka nilai heuristik untuk *node N* adalah :

$$h(N) = min(h_1, h_2, h_3)$$

[3.4]

$h(N)$ = nilai heuristik *node N*
h_1 = nilai heursitik dari *node N* menuju *node* koin K_1
h_2 = nilai heursitik dari *node N* menuju *node* koin K_2
h_3 = nilai heursitik dari *node N* menuju *node* koin K_3

Selain nilai heuristik, penulis juga merancang pembobotan yang digunakan dalam proses pencarian jalur. Pembobotan ini berfungsi untuk mengarahkan agen cerdas agar dapat memilih jalur dengan koin yang lebih banyak. Pembobotan dapat disertakan dalam menghitung *cost* atau nilai *g*. Pembobotan ini dinotasikan sebagai nilai *alpha*. Nilai *alpha* merupakan skala untuk menentukan besarnya *cost*, nilainya dapat berupa 0, 1, atau antara 0 dan 1. Nilai *alpha* ini dirancang agar dapat memberikan *cost* yang kecil untuk *node* dengan koin. Perhitungan nilai *g* untuk pengembangan algoritma A* mengembangkan perhitungan yang dilakukan oleh Patel (2012). Perhitungannya adalah sebagai berikut :

$$g(N') = tile\ cost + alpha * (g(N) - tile\ cost)$$

[3.5]

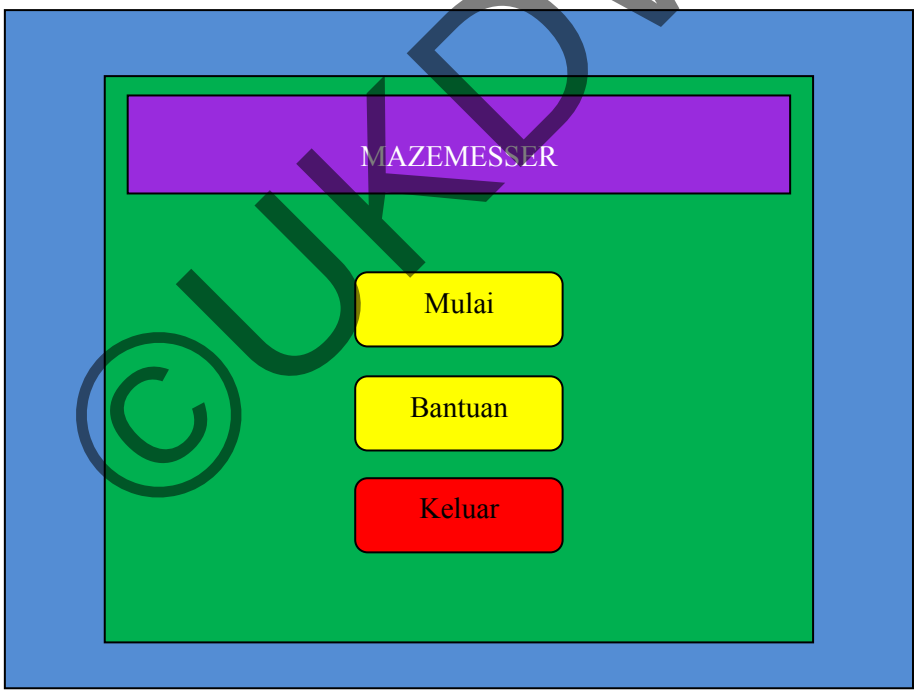
$g(N')$ = nilai cost dari *node N'*
$g(N)$ = nilai cost dari node *N* (satu *node* sebelum menuju *node N'*)
tile cost = nilai *cost* untuk melakukan satu langkah antar *node*
alpha = nilai pembobotan *cost*

3.3. Rancangan Antarmuka

Rancangan antarmuka sistem terdiri dari tampilan antarmuka menu utama (gambar 3.15), tampilan antarmuka bantuan (gambar 3.16), tampilan antarmuka halaman peringatan (gambar 3.17), dan tampilan antarmuka permainan *maze* (gambar 3.18). Berikut adalah rancangan antarmuka yang ada di dalam sistem yang dibangun.

3.3.1. Rancangan Antarmuka Menu Utama

Gambar 3.6 berikut merupakan rancangan antarmuka halaman menu utama. Halaman tersebut berisi pilihan kepada pemain untuk bermain atau membuka halaman bantuan.

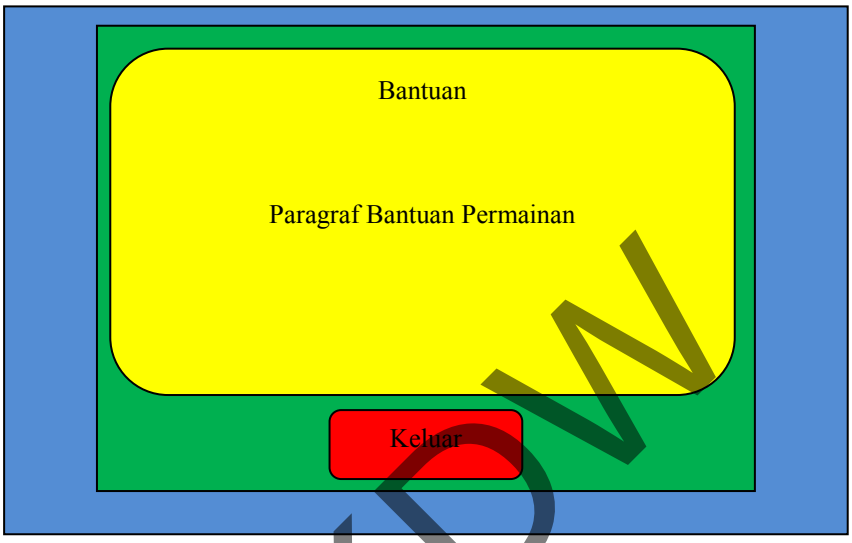


Gambar 3.6 Rancangan Antarmuka Menu Utama

Antarmuka menu utama dirancang dengan bentuk yang sederhana. Di dalam halaman tersebut, akan disediakan tombol-tombol pilihan yaitu tombol “mulai”, tombol “bantuan”, dan tombol “keluar”.

3.3.2. Rancangan Antarmuka Halaman Bantuan

Halaman bantuan berisi petunjuk permainan ini. Pemain dapat mengakses halaman ini hanya lewat menu utama. Di dalam halaman ini berisi tentang bagaimana cara bermain dan cara berinteraksi dengan permainan.



Gambar 3.7 Rancangan Antarmuka Halaman Bantuan

Gambar 3.7 merupakan rancangan antarmuka halaman bantuan. Halaman bantuan dirancang dengan bentuk yang sederhana, namun penuh informasi tentang permainan ini di antaranya tentang aturan permainan dan cara bermain.

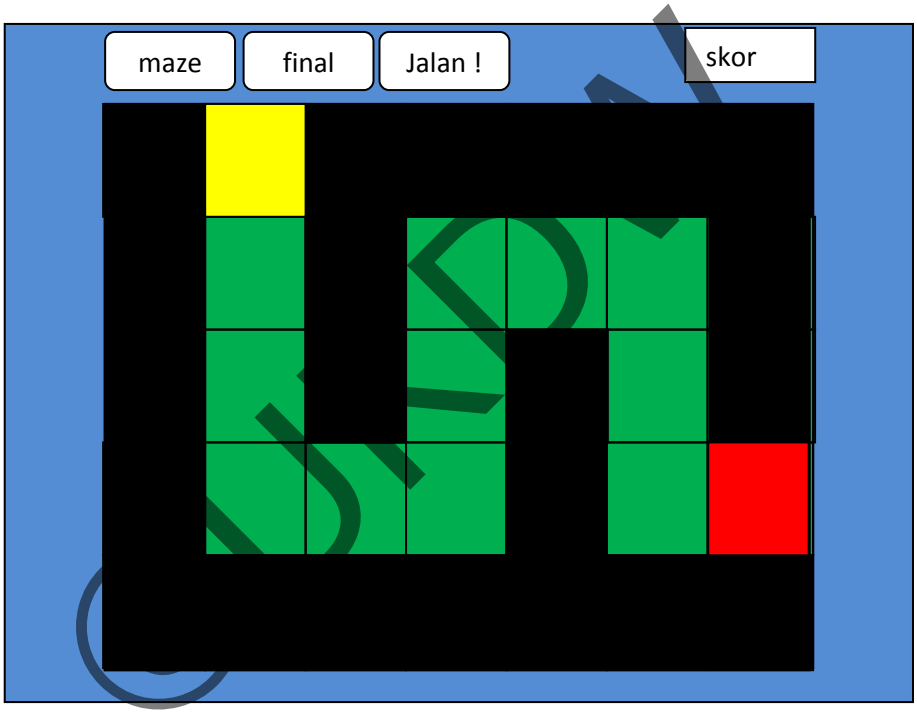
3.3.3. Rancangan Antarmuka Halaman Peringatan

Gambar 3.8 Rancangan Antarmuka Halaman Peringatan

Gambar 3.8 merupakan rancangan halaman peringatan. Halaman tersebut akan muncul jika terjadi kesalahan baik yang dilakukan oleh pemain maupun terjadi kesalahan dalam sistem.

3.3.4. Rancangan Antarmuka Permainan Maze

Rancangan antarmuka permainan *maze* ditunjukkan pada gambar 3.9. Terdapat beberapa elemen utama seperti tombol-tombol untuk bermain, tampilan skor, dan tampilam *map maze* yang sedang dimainkan.



Gambar 3.9 Rancangan Antarmuka Permainan Maze

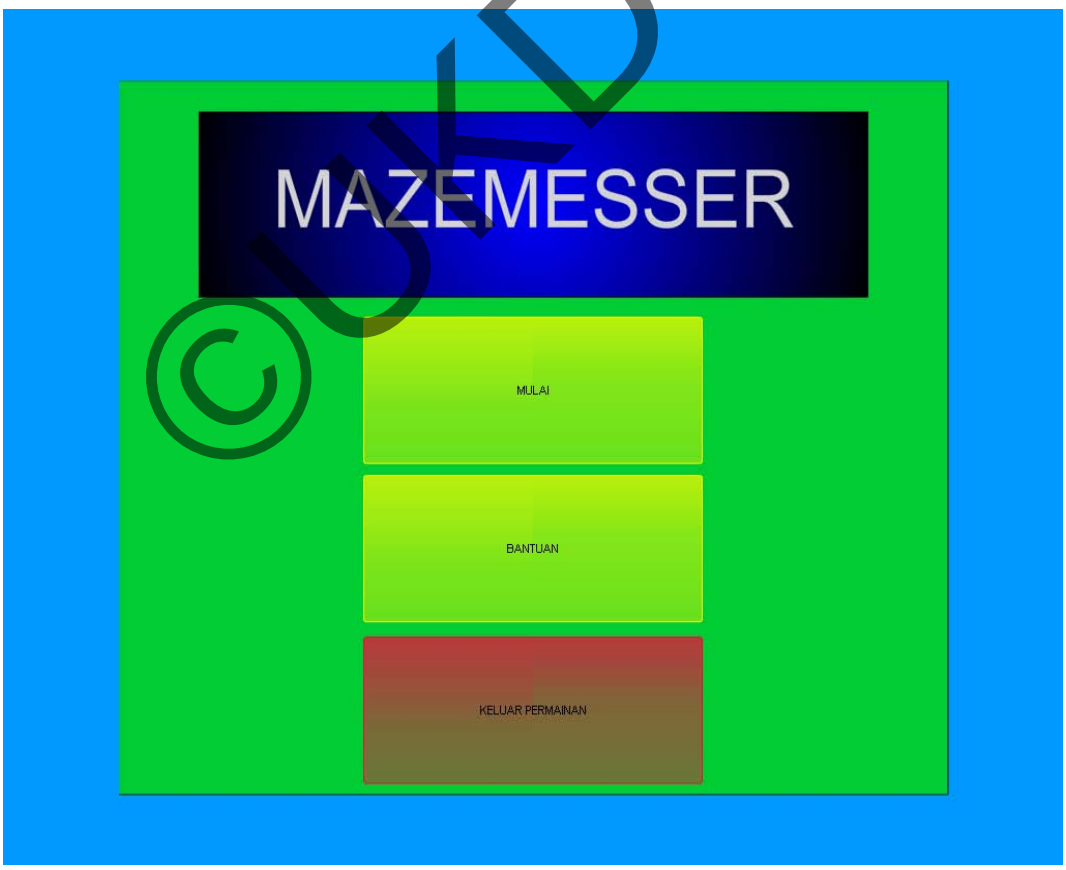
Map maze akan ditampilkan di tengah halaman permainan agar mudah diakses pemain. Map maze dirancang memiliki warna yang beragam agar terlihat lebih menari dan juga memudahkan pemain dalam memodifikasi *map maze*.

BAB 4
IMPLEMENTASI DAN ANALISIS KINERJA SISTEM

4.1. Antarmuka Sistem

Antarmuka sistem yang telah dibangun terdiri dari tampilan antarmuka menu utama (gambar 4.1), tampilan antarmuka bantuan (gambar 4.2), tampilan antarmuka halaman peringatan (gambar 4.3), dan tampilan antarmuka permainan *maze* (gambar 4.4). Berikut adalah hasil dari implementasi antarmuka sistem yang dibangun.

4.1.1. Antarmuka Menu Utama Sistem



Gambar 4.1 Tampilan Antarmuka Halaman Menu Utama

Gambar 4.1 adakah tampilan yang muncul ketika permainan dijalankan. Halaman ini merupakan halaman “Menu Utama” permainan. Terdapat 3 tombol yang memiliki fungsi berbeda-beda. Tombol “Mulai” digunakan untuk memulai permainan, tombol “Bantuan” digunakan untuk memanggil halaman yang berisi petunjuk permainan, tombol “Keluar” digunakan untuk keluar dari sistem. Halaman ini hanya berupa tampilan kepada pengguna untuk mempermudah pemain dalam memulai permainan, tidak terdapat implementasi permainan yang dibuat oleh penulis.

4.1.2. Antarmuka Halaman Bantuan

Halaman bantuan dapat diakses melalui halaman menu utama. Gambar 4.2 berikut menunjukkan tampilan dari halaman bantuan yang terdapat dalam permainan ini.

Gambar 4.2 Tampilan Antarmuka Halaman Bantuan

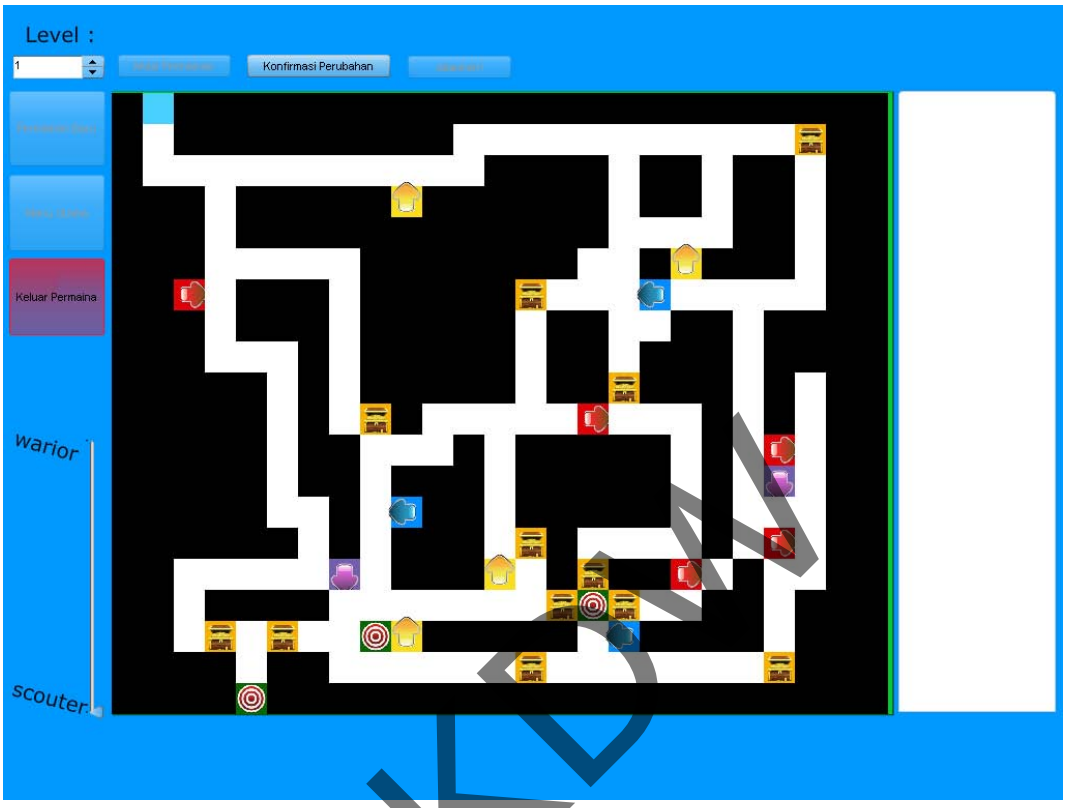
Halaman bantuan berisi panduan untuk pengguna dalam memainkan permainan ini. Dalam halaman ini, terdapat *objective* pemain dalam bermain. Selain itu terdapat pula aturan dalam permainan dan keterangan fungsi-fungsi tombol yang ada didalam tampilan permainan. Pemain dapat kembali ke menu utama dengan meng-klik tombol "Selamat Bermain".

4.1.3. Antarmuka Halaman Peringatan

Gambar 4.3 Tampilan Antarmuka Halaman Peringatan

Gambar 4.3 merupakan tampilan pesan *error* atau halaman peringatan bila terjadi kesalahan di dalam sistem. Halaman ini akan muncul jika *input* yang diberikan pemain tidak valid atau terjadi kesalahan di dalam sistem dalam menjalankan permainan.

4.1.4. Antarmuka Halaman Permainan Maze



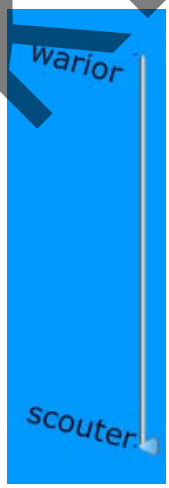
Gambar 4.4 Tampilan Antarmuka Halaman Permainan Maze

Gambar 4.4 merupakan tampilan awal jika pengguna memulai permainan. Halaman ini dapat diakses melalui tombol "Mulai" yang telah dijelaskan pada subbab 4.1.1.

Dalam halaman permainan ini, ada terdapat 3 buah tombol yang dapat digunakan pemain untuk menjalankan permainan, satu buah tombol untuk kembali ke "Menu Utama", satu buah tombol "Permainan Baru", satu buah tombol untuk keluar permainan, satu buah *setting* level, satu buah *slider* untuk pengaturan prioritas *goal*, dan juga terdapat tampilan *maze* yang sedang dimainkan. Selain itu, terdapat pula skor yang akan ditampilkan setelah sistem menyelesaikan permainan *maze*. Tombol-tombol yang ada dalam halaman permainan ini berfungsi sebagai berikut :

1. Tombol “Mulai” untuk menampilkan *maze* awal.
2. Tombol “Konfirmasi Perubahan” untuk menyelesaikan perubahan pada mze yang dibuat pengguna dan memvalidasinya.
3. Tombol “Jalankan” untuk memicu agen cerdas untuk melakukan pencarian jalur pada *maze* yang sedang dimainkan.
4. Tombol “Permainan Baru” untuk mengulang permainan dari awal. Tombol ini akan aktif jika permainan telah diselesaikan.
5. Tombol “Menu Utama” untuk kembali ke “Menu Utama”.
6. Tombol “Keluar Permainan” untuk keluar dari permainan.

Dalam halaman permainan ini, terdapat sebuah *slider* yang berisi prioritas *goal* yaitu memprioritaskan kecepatan jalur dan mempertimbangkan besaran poin. Indikator menuju pada *scouter* berarti prioritas goal untuk agen cerdas adalah pada kecepatan jalur (jalur terpendek) sedangkan indikator ada pada *warior* berarti gabungan prioritas kecepatan jalur dan besarnya poin. Gambar 4.5 merupakan tampilan *slider* yang terdapat dalam permainan ini.



Gambar 4.5 Pengaturan Prioritas Target

Tampilan *maze* yang sedang dimainkan juga terdapat dalam halaman ini. Pemain dapat berinteraksi dengan *maze* dengan meng-klik pada kotak-kotak yang sudah ditandai. Interaksi kotak-kotak yang ditandai tersebut akan memberi umpan balik sebagai berikut :

1. Kotak  akan bergeser ke kanan ketika diklik.
2. Kotak  akan bergeser ke kiri ketika diklik.
3. Kotak  akan bergeser ke atas ketika diklik.
4. Kotak  akan bergeser ke bawah ketika diklik.
5. Kotak  akan menjadi *goal* ketika diklik.
6. Kotak  kembali menjadi kotak nomor 5 ketika diklik.

Selain itu, terdapat pula kotak khusus berbentuk  yang merupakan koin atau poin yang dapat dikumpulkan agen cerdas. Kotak tersebut tidak dapat diberi input.

4.2. Implementasi Sistem

Implementasi sistem dilakukan dengan proses pengkodean dengan menggunakan *script* Action Script 3.0. Perancangan yang dilakukan pada bab sebelumnya membantu penulis untuk mengimplementasikan pemenuhan kebutuhan-kebutuhan ke dalam sistem yang dibangun.

4.2.1. Implementasi Pembuatan *Game World*

Ketika pengguna memulai permainan, permainan dimulai dengan tampilan kondisi *game world* berupa *map maze* asli. Data yang digunakan untuk membangun *game world* atau *map maze* berasal dari *file* XML. *File* tersebut akan dibaca oleh sistem dan diubah ke dalam bentuk *array* 2 dimensi. *Array* tersebut kemudian kembali di ubah oleh sistem ke dalam bentuk *grid-grid* dan akan ditampilkan menjadi *map maze*.

Untuk membaca dan memproses *file* XML, penulis membuat *class* khusus yaitu *XmlLoader*. Gambar 4.6 merupakan fungsi utama dalam *class XmlLoader* yang berfungsi untuk membaca dan *load file* XML.

```
dataXML = new XML();
fileNameXML = url;
urlReqXML = new URLRequest(fileNameXML);
urlLoadXML = new URLLoader(urlReqXML);

urlLoadXML.addEventListener(Event.COMPLETE, loadCompleteXML);
```

Gambar 4.6 Potongan Kode untuk Membaca *Load File* XML

Class XmlLoader akan membaca *file* eksternal yang berupa *file* XML, kemudian dan menampunnya ke dalam data XML. Sistem kemudian merubah data XML tersebut ke dalam *array* 2 dimensi agar mudah digunakan dalam membentuk *map*. *Array* 2 dimensi terseubut kemudian digunakan sebagai data untuk membentuk *grid-grid map maze*.

Grid yang ditampilkan merupakan *class* "Grid" yang telah dibuat oleh penulis. Dalam *class Grid* tersebut, terdapat fungsi untuk merepresentasikan *array map maze* menjadi tampilan visual di dalam permainan. Proses instansiasi dari *class grid* ini yang membentuk *map maze* sebagai *game world* dalam permainan ini. Gambar 4.7 berikut merupakan potongan program yang berfungsi untuk membaca dan men-*generate map* yang digunakan dalam permainan ini.

```
    for(var row:int=0; row<gameMap.length; row++)
    {
        mapArray.push(new Array());
        for(var col:int=0; col<gameMap[0].length; col++)
        {
            // e.g : node 20 -> col = 1, row = 0. 20/20 = 1, 20%20 = 0 -> col 1, row 0 ; array 0,1
            grid = new Grid(col,row,105+(col*MAP_TILE_SIZE),85+(row*MAP_TILE_SIZE),MAP_TILE_SIZE,gameMap
row][col],(col * gameMapY) + row);
            mapArray[row].push(grid);
```

Gambar 4.7 Potongan Kode Untuk Menciptakan *Grid*

Dalam potongan program yang ada pada gambar 4.7 tersebut memperlihatkan bahwa jumlah *grid* disesuaikan dengan banyaknya elemen data dalam *array gameMap*. Kemudian jenis *grid* dibaca dari isi dari data *gameMap*.

Penulis memilih menggunakan *array* 2 dimensi sebagai basis data *game world* (*map maze*) karena penggunaannya dan pembacaannya yang mudah. Penggunaannya mudah karena sangat sesuai dengan basis desain permainan yang dibangun oleh penulis yaitu *tile-based game*. Data berbentuk array 2 dimensi tersebut memiliki tambahan informasi yang diperlukan dalam pembuatan *game world*, seperti koordinat (kolom dan baris). Selain itu, dengan menggunakan *array* 2 dimensi, pembacaan data baik oleh komputer maupun manusia cukup mudah karena data tersebut dapat merepresentasikan bentuk visual dari *game world* yang akan digunakan. Contoh dari data map yang digunakan dalam pembuatan sistem dapat di lihat pada gambar 4.8 berikut ini :

```
<tilemap>
    <row>1,8,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1</row>
    <row>1,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,9,1,1</row>
    <row>1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,1,1,0,1,1,0,1,1</row>
    <row>1,1,1,0,1,1,1,1,1,5,1,1,1,1,1,1,0,1,1,0,1,1,0,1,1</row>
    <row>1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,1,0,1,1</row>
    <row>1,1,1,0,0,0,0,0,1,1,1,1,1,1,1,0,0,1,5,1,1,1,0,1,1</row>
    <row>1,1,3,0,1,1,1,0,1,1,1,1,1,9,0,0,0,4,0,0,0,0,0,1,1</row>
    <row>1,1,1,0,1,1,1,0,1,1,1,1,1,0,1,1,0,0,1,1,0,1,1,1,1</row>
    <row>1,1,1,0,0,0,1,0,1,1,1,1,1,0,1,1,0,1,1,1,0,1,1,1,1</row>
    <row>1,1,1,1,1,0,1,0,1,1,1,1,1,0,1,1,9,1,1,1,0,1,0,1,1</row>
    <row>1,1,1,1,1,0,1,0,9,1,0,0,0,0,0,3,0,0,0,1,0,1,0,1,1</row>
    <row>1,1,1,1,1,0,1,1,0,0,0,1,0,1,1,1,1,1,1,0,1,0,3,0,1,1</row>
    <row>1,1,1,1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,0,1,0,6,0,1,1</row>
    <row>1,1,1,1,1,0,0,1,0,4,1,1,0,1,1,1,1,1,1,0,1,0,0,0,1,1</row>
    <row>1,1,1,1,1,1,0,1,0,1,1,1,0,9,1,0,0,0,0,1,0,3,0,1,1</row>
    <row>1,1,0,0,0,0,0,6,0,1,1,1,5,0,1,9,1,1,3,0,1,0,0,1,1</row>
    <row>1,1,0,1,1,1,1,1,0,0,0,0,0,0,0,9,7,9,0,0,1,1,0,1,1,1</row>
    <row>1,1,0,9,0,9,0,0,7,5,1,1,1,1,1,1,0,4,1,1,1,1,0,1,1,1</row>
    <row>1,1,1,1,0,1,1,0,0,0,0,0,0,9,0,0,0,0,0,0,0,9,1,1,1</row>
    <row>1,1,1,1,7,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1</row>
</tilemap>
```


Gambar 4.8 Contoh Data *Map Maze*

Dari contoh data *array* 2 dimensi yang ada dalam *file* XML tersebut dapat dilihat bagaimana bentuk *map maze* akan dibaca dan ditampilkan oleh sistem.

4.2.2. Implementasi *User Costumized Maze*

Permainan ini memungkinkan pemain untuk merubah *maze* awal yang telah ditampilkan sebelumnya. Sistem akan mencari jalur dengan menggunakan Algoritma A* dari *maze* yang telah dimodifikasi tersebut. Untuk implementasi fitur tersebut, sistem dapat menerima *input* berupa *mouse click* yang dilakukan pada *grid-grid* khusus dalam papan permainan. Dalam subbab 4.1.4 telah dijelaskan secara lebih rinci tentang umpan balik yang terjadi ketika pengguna men-klik *grid-grid* tersebut.

Potongan program pada gambar 4.9 menggambarkan contoh *input handling* yang digunakan untuk mengakomodasi *input* yang diberikan pemain. Dalam implementasinya, *grid* yang diklik oleh pengguna akan dideteksi jenisnya. Bila pengguna mengklik *grid-grid* khusus, maka sistem akan memberikan umpan balik berupa perubahan jenis *grid* dan atau pertukaran suatu *grid* dengan *grid* disekitarnya.

```
    if(evt.currentTarget.tipe == MAP_TIPE_DOWN)
    {
        switchTmp = gameMap[(evt.currentTarget.row)+1][evt.currentTarget.col];
        gameMap[(evt.currentTarget.row)+1][evt.currentTarget.col] = MAP_TIPE_UP;
        gameMap[evt.currentTarget.row][evt.currentTarget.col] = switchTmp;
        mapArray[(evt.currentTarget.row)+1][evt.currentTarget.col].tipe = gameMap[(evt.currentTarget.row
+1][evt.currentTarget.col];
        mapArray[evt.currentTarget.row][evt.currentTarget.col].tipe = gameMap[evt.currentTarget.row][evt
.currentTarget.col];
        //redraw seperlunya
        mapArray[(evt.currentTarget.row)+1][evt.currentTarget.col].fungsiRedraw();
    }
```

Gambar 4.9 Potongan Kode untuk Mengakomodasi *Input* Pemain

Setelah menerima *input* pemain, sistem akan melakukan validasi terhadap *input* tersebut. *Input* yang diperiksa berupa *goal* yang ditentkan oleh pemain. Bila *goal* tidak memenuhi persyaratan, misalnya karena *goal* tidak ditemukan atau terdapat *goal* lebih dari 1, maka akan ditampilkan pesan *error*. Validasi berfungsi sebagai tindakan preventif terhadap kemungkinan terjadinya *error* pada sistem. Bila *error handling* tersebut tidak diberikan, *input* tersebut akan digunakan untuk

menjalankan algoritma A* yang diimplementasikan di dalam sistem. Kesalahan dalam memberikan input dapat menyebabkan proses tersebut tidak berjalan dengan baik dan dapat beresiko menyebabkan *freeze*.

4.2.3. Implementasi Pencarian Jalur dengan Algoritma A*

Penulis mengimplementasikan proses pencarian dengan menggunakan Algoritma A* ke dalam *class* tersendiri yaitu *class* AStar. Proses pencarian jalur tersebut dapat dibagi ke dalam 3 proses utama yaitu pembuatan *node-node* jalur, proses pengecekan dan pemilihan *node* selanjutnya, dan proses pembentukan jalur dari *node-node* terpilih. Pencarian jalur dengan Algoritma A* dalam penelitian ini dimulai dengan proses pembuatan *node-node*. *Node-node* tersebut berupa *object* yang memiliki atribut sebagai berikut :

Tabel 4.1
Atribut pada setiap *Object Node*

Atribut	Fungsi / deskripsi
row	menampung letak baris *node* dalam *map.*
col	menampung letak kolom *node* dalam *map.*
f	menampung nilai f *node* tersebut.
g	menampung nilai g *node* tersebut.
h	menampung nilai h *node* tersebut.
parent	menampung refrensi *object node* sebelumnya.
id	merupakan id yang unik dari tiap *node.*
tipe	merupakan jenis *node* yang dibaca dari *map maze* (*array*).
poin	menampung nilai poin dari *node* tersebut.

Proses pembuatan *node* ini diawali dengan pembacaan data *array map maze*. *Array* tersebut kemudian direpresentasikan dalam bentuk *object node* dan disusun dalam bentuk *map* ke dalam *array* 2 dimensi. Gambar 4.10 berikut merupakan *listing* program dalam pembuatan *object node* dan *array*.

```
for(row = 0;row < loadedMap.length; row++) //row
{
    //Array 2D
    map[row] = new Array();

    //for(col = 0;col < loadedMap[0].length; col+
    for(col = 0;col < loadedMap[0].length; col++)
    {
        //var node:Array = [];
        //Nic: 25-3-2013. Tiap node punya propert
        var node:Object = new Object();
```

Gambar 4.10 Potongan Kode untuk Pembuatan *Node*

Object node yang baru saja terbentuk kemudian diberi nilai atribut awal. Atribut ini dapat berganti dalam proses selanjutnya. Setelah pemberian nilai atribut awal, *node* akan dimasukkan ke dalam *array map*. Gambar 4.11 berikut merupakan *listing* program dalam pemberian nilai dari tiap *object node*.

```
//Nic: 25-3-2013. Property A* yang dibutuh
node.f = 0;
node.g = 0;
node.h = 0;
node.parent = null;
// node.next = null;

// Nic: 25-3-2013. Row, Col, ID, Tipe/warn
// e.g : node 20 -> col = 1, row = 0. 20/2
node.row = row;
node.col = col;
node.id = (col * /*Y*/ MAP_MAX_ROW) + row;
node.tipe = loadedMap[row][col]; //0 passe
```


Gambar 4.11 Potongan Kode untuk Pemberian Nilai Tiap *Node*

Setelah *object node* terbentuk, pencarian jalur dengan algoritma A* dapat dimulai. Dalam melakukan pencarian dengan algoritma A*, pencarian diawali dari titik *start* dan berakhir di titik *goal*, implementasi di dalam kode ditunjukan pada gambar 4.12 berikut.

```
var curNode:Object = nodeMap[uint(startNodeID % MAP_MAX_ROW)][uint(startNodeID / MAP_MAX_ROW)];
var goalNode:Object = nodeMap[uint(goalNodeID % MAP_MAX_ROW)][uint(goalNodeID / MAP_MAX_ROW)];
```

Gambar 4.12 Untuk Memberikan Titik Awal dan Akhir Pencarian

curNode adalah *node* yang saat ini sedang dibangkitkan, *startNode* dimasukkan ke dalam *node* tersebut. *goalNode* adalah titik *goal* sebagai tujuan akhir dari pencarian. Sistem akan melakukan *looping* untuk mencari jalur selama titik *goalNode* belum ditemukan atau hingga *open list* dalam algoritma A* kosong. Bila *open list* kosong berarti tidak ada lagi jalur yang ditemukan.

Setiap *node* yang menjadi *curNode* akan dibangkitkan untuk setiap tetangganya (*testNode*). Setiap test*Node* yang dibangkitkan tersebut kemudian diperiksa apakah dapat dilalui (bukan penghalang). Bila ternyata *node* yang dibangkitkan dapat dilalui, maka *node* tersebut akan dihitung nilai *f*, *g*, dan *h* nya. Nilai pertama yang dihitung adalah nilai *g*. Nilai g dalam algoritma A* merupakan *cost* yang didapat dari titik *start* hingga titik sekarang (*testNode*). Gambar 4.13 berikut merupakan implementasi perhitungan nilai g di dalam sistem.

```
var alpha:Number = 0.75;
if(heuristic == HEURISTIC_MULTI)
{
    if(curNode.poin > 0)
    {
        alpha = 0.75;
    }

    var fG:Number;
    fG = alpha * (curNode.g - MAP_TILE_SIZE);
    g = MAP_TILE_SIZE + Math.ceil(fG);
}
else
{
    g = curNode.g + cost; // costnya pake tile
    : dari node sblumnya
}
```

Gambar 4.13 Potongan Kode Untuk Menghitung Nilai *g*

Selain nilai g, pencarian jalur dengan algoritma A* juga mengandalkan heuristik (h). Nilai heuristik merupakan nilai *cost* yang dibutuhkan dari *testNode* untuk sampai ke *goal*. Nilai heuristik ini sangat menentukan kecepatan dan keberhasilan algoritma A* dalam melakukan pencarian, karena itu penentuan nilai heuristik sangat penting. Salah satu cara dalam menghitung nilai heuristik adalah dengan metode *Manhatan Distance* seperti yang telah dijelaskan pada sub-bab 2.2.2. Dalam penelitian ini, peneliti menggunakan Manhatan Distance dalam menghitung nilai heuristik. Nilai heuristik ini didapat dari menghitung jarak vertikal dan horizontal titik sekarang (*testNode*) dengan titik *goal*. Implementasi perhitungan heuristik di dalam sistem ditunjukan dalam gambar 4.14.

```
var h2Temp:uint;
for(var coinIdx=0;coinIdx<coinArray.length; coinIdx++)
{
    nodeCoin = nodeMap[coinArray[coinIdx].row][coinArray[coinIdx].col];

    h2 = (Math.abs(nodeCoin.col - tmpNextNode.col)) * MAP_TILE_SIZE + (Math.abs(
    nodeCoin.row - tmpNextNode.row)) * MAP_TILE_SIZE;

    if(coinIdx == 0)
    {
        h2Temp = h2;

    }
    else
    {
        if(h2 < h2Temp) {
            h2Temp = h2;
        }
    }

}
if(heuristic == HEURISTIC_MULTI)
{
    h = h2Temp;
}
else
{
    h = (Math.abs(goalNode.col - tmpNextNode.col)) * MAP_TILE_SIZE + (Math.abs(
    goalNode.row - tmpNextNode.row)) * MAP_TILE_SIZE;
}
```

Gambar 4.14 Potongan Kode Untuk Menghitung Nilai *h*

Setelah mendapatkan nilai *g* dan *h*, kemudian akan di hitung nilai *f* sesuai rumus [2.3] pada bab 2.

Salah satu tujuan penelitian ini adalah untuk mengimplementasikan dan pengembangan Algoritma A*. Perbedaan antara algoritma A* biasa dengan algoritma A* yang dikembangkan ini adalah pada pencarian jalur dengan *goal* sekunder. Pada kasus maze, algoritma A* biasa memiliki 1 *goal* yang harus dicapai yaitu untuk mencari jalur terpendek. Algoritma A* yang dikembangkan tidak hanya mencari jalan keluar (*goal*) namun juga mencari poin terbanyak dengan melewati *node-node* khusus dengan poin (dalam kasus ini adalah koin). Peneliti mengembangkan algoritma A* dengan memberikan pembobotan pada nilai *cost* (gambar 4.13) dan perhitungan nilai heuristik (gambar 4.14) untuk mendukung diperolehnya *goal* sekunder.

Implementasi perhitungan nilai g pada penelitian ini ada 2 yaitu untuk algoritma A* biasa dan algoritma A* yang dikembangkan. Pada pengembangan algoritma A*, penulis memberikan pembobotan untuk nilai *cost* dengan menggunakan *alpha* (gambar 4.13). Nilai h algoritma A* biasa didapat dengan menghitung *Manhattan Distance* antara titik sekarang (*testNode*) dengan titik *goal*, sedangkan dalam algoritma A* yang dikembangkan, nilai h didapat dari menghitung *Manhattan Distance* antara titik sekarang (*testNode*) dengan titik *goal* sekunder (dalam hal ini posisi koin) dan mencari nilai h paling kecil jika terdapat beberapa *goal* sekunder (gambar 4.14).

Setelah mendapatkan nilai f, sistem akan memeriksa apakah *node testNode* sudah ada di *dalam open list* atau *close list*. Jika *node testNode* tidak ada di *open* atau *close list*, maka *testNode* akan dimasukkan ke dalam *open list*. Namun jika *testNode sudah ada di open list* atau *close list* maka *testNode* hanya akan di-*update* nilai refrensi dan *parent*-nya jika nilai *f testNode* yang baru lebih kecil. Potongan program pada gambar 4.15 merupakan implementasi proses *update open list* atau *update* refrensi *node*.


```
if(!(bIsOnCloseList || bIsOnOpenList))
{
    tmpNextNode.f = f; //masukkan f,g,h
    tmpNextNode.g = g;
    tmpNextNode.h = h;
    tmpNextNode.parent = curNode; //di
    openList.push(tmpNextNode); // masu
}
else // kalau di dalam list, tmpNextNod
{
    trace(tmpNextNode.id+" ada di open/
    if(tmpNextNode.f > f)    //kalau ada
    {                        //sudah pun
        //klw teryata node
        tmpNextNode.f = f;
        tmpNextNode.g = g;
        tmpNextNode.h = h;
        tmpNextNode.parent = curNode;
    }
} //if(!(bIsOnCloseList || bIsOnOpenLis
```

Gambar 4.15 Potongan Kode Untuk *Update Node* dan *Open List*

Setelah menghitung dan membangkitkan semua *node* tetangga (*testNode*) milik *node* sekarang (*cestNode*), *node* sekarang dimasukkan ke dalam *close list*. Proses pencarian selanjutnya adalah pengambilan *node* sekarang (*curNode*). *Node* yang diambil adalah *node* dengan nilai *f* paling kecil. Implementasi di dalam sistem dapat dilakukan dengan melakukan *sorting node* di dalam *open list* untuk mendapatkan nilai *f* terkecil.

Proses di atas terus di ulangi hingga *node* sekarang (*curNode*) merupakan *goal node*. Jika *goal* sudah ditemukan atau sudah tidak ada *node* di dalam *open list*, proses pencarian dilanjutkan dengan perangkaian jalur (*path*). Jalur ini terdiri dari *node-node* yang telah dijelajahi pada proses sebelumnya. Pada gambar 4.15 ditunjukkan bahwa suatu *node* akan memperbaharui *parent*-nya jika ditemukan nilai *f* yang lebih kecil, hal ini berfungsi agar jalur menuju *goal* dapat terbentuk. Penulis melakukan implementasi perangkaian jalur ini dengan mengurutkan *node-node* secara mundur dari *goal* menuju titik *start*. Penulis merancang node-node hanya memiliki *parent*, sehingga proses perangkaian jalur dilakukan dengan mengikuti hubungan antar *node* melalui *parent* tersebut. Potongan program yang menunjukkan implementasi perangkain kembali jalur pencarian ditunjukkan pada gambar 4.16.


```
var node:Object;
node = goalNode;
shortPath.push(node);
while(node.id != startNodeID)
{

    node = node.parent; //backward
    shortPath.unshift(node);

    if(ab > MAP_MAX_COL * MAP_MAX_ROW)
        break;
}
```

Gambar 4.16 Potongan Kode Untuk Perangkaian Jalur Terpendek

4.3. Analisis Kinerja Sistem

Setelah proses implementasi selesai, perlu diadakan evaluasi terhadap sistem yang telah dibangun. Evaluasi tersebut bertujuan untuk menjawab perumusan masalah dan hipotesis yang ada di awal penelitian.

4.3.1. Evaluasi Implementasi Algoritma A* Dalam Sistem

Setelah melakukan implementasi sistem, penulis mencoba menilai tingkat keberhasilan agen cerdas dalam menyelesaikan permainan *maze*. Agen cerdas dinyatakan berhasil menyelesaikan suatu *map maze* bila dapat memenuhi 2 syarat. Syarat yang pertama adalah jumlah langkah yang diperlukan oleh agen cerdas untuk menemukan *node goal* harus kurang dari atau sama dengan batas maksimum jumlah langkah pada 1 area *map maze*. Batas maksimum ini berbeda-beda untuk setiap *map maze* dan telah ditentukan oleh perancang *map maze*. Syarat yang kedua adalah jumlah koin yang didapatkan oleh agen cerdas harus memenuhi batas minimum jumlah koin pada 1 area *map maze*. Seperti halnya batas maksimum langkah, batas minimum koin juga ditentukan oleh perancang *map maze*. Untuk menentukan tingkat keberhasilan sistem, akan dilakukan beberapa kali ujicoba permainan dengan area *map maze* yang berbeda dan modifikasi pemain yang berbeda. Tingkat keberhasilan algoritma A* (B) untuk menyelesaikan 1 *map maze* (n) dihitung dengan rumus :

$$B(n) = \frac{Ujicoba\ yang\ Berhasil\ pada\ map\ maze\ n}{Jumlah\ Ujicoba\ pada\ map\ maze\ n}\ x\ 100\% \quad \textbf{[4.1]}$$

$B(n)$ = Tingkat keberhasilan algoritma A* dalam menyelesaikan *map maze-n*

Penulis menyediakan 3 desain *map maze* yang digunakan dalam uji coba sistem ini. Untuk uji coba pada setiap *map maze*, modifikasi *map maze* dan

pemilihan letak *node goal* dilakukan secara acak. Tabel 4.2, tabel 4.3, dan tabel 4.4 berikut menunjukkan hasil dari uji coba pada sistem yang dibangun.

Tabel 4.2

Hasil Ujicoba pada *Map Maze* 1

Uji coba	A*		A* improve		Keberhasilan				Batas *node* / koin
					A*		A* improve		
	node	koin	*node*	koin	p	k	p	k	
1	31	100	31	100	Y	N	Y	N	35 / 200
2	31	100	45	400	Y	N	Y	Y	50 / 400
3	51	300	69	500	Y	N	Y	Y	70 / 500
4	53	300	71	500	Y	N	Y	Y	71 / 500
5	65	400	83	600	Y	N	N	Y	74 / 500

Tabel 4.3

Hasil Ujicoba pada *Map Maze* 2

Uji coba	A*		A* improve		Keberhasilan				Batas *node* / koin
					A*		A* improve		
	node	koin	*node*	koin	p	k	p	k	
1	42	200	116	1200	Y	N	N	Y	76 / 600
2	132	1500	132	1800	Y	N	Y	Y	134 / 1800
3	116	900	116	1200	Y	N	Y	Y	118 / 1200
4	64	700	94	700	Y	N	Y	N	87 / 800
5	94	600	94	700	Y	N	Y	Y	96 / 700

Tabel 4.4

Hasil Ujicoba pada *Map Maze* 3

Uji coba	A*		A* improve		Keberhasilan				Batas *node* / koin
	node	koin	*node*	koin	A*		A* improve		
					p	k	p	k	
1	17	0	19	300	Y	N	Y	Y	20 / 300
2	59	0	71	600	Y	N	Y	Y	78 / 500
3	62	100	70	500	Y	N	Y	Y	70 / 500
4	77	0	83	500	Y	N	Y	Y	83 / 500
5	78	0	86	600	Y	N	Y	Y	86 / 500

Berdasarkan data tersebut, tingkat keberhasilan algoritma A* dalam mencari jalur terpendek (tanpa multi kriteria *goal*) pada *map maze* uji coba, adalah sebagai berikut :

Tabel 4.5

Tingkat Keberhasilan Algoritma A* Mencari Jalur Terpendek

Map	Tingkat Keberhasilan
Map Maze 1	$\frac{5}{5}\ x\ 100\% = 100\%$
Map Maze 2	$\frac{5}{5}\ x\ 100\% = 100\%$
Map Maze 3	$\frac{5}{5}\ x\ 100\% = 100\%$

Berdasarkan data tersebut, Algoritma A* yang diimplementasikan di dalam sistem dapat menemukan jalur terpendek (tanpa multi kriteria *goal*) dengan tingkat keberhasilan 100%. Sedangkan algoritma A* biasa tidak dapat menyelesaikan *map maze* dengan multi kriteria *goal*.

Tingkat keberhasilan algoritma A* yang dikembangkan dalam menyelesaikan *map maze* dengan multi kriteria *goal* adalah :

Tabel 4.6

Tingkat Keberhasilan Algoritma A* yang Dikembangkan

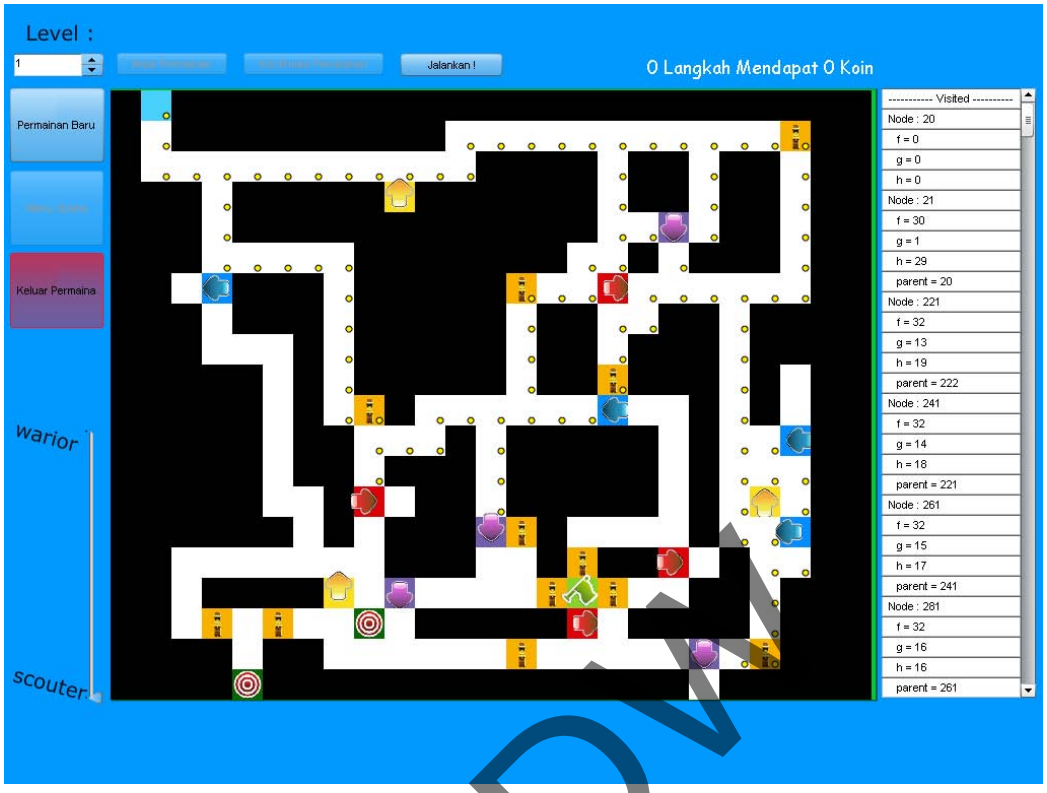
Map	Tingkat Keberhasilan
Map Maze 1	$\frac{3}{5}\ x\ 100\% = 60\%$
Map Maze 2	$\frac{3}{5}\ x\ 100\% = 60\%$
Map Maze 3	$\frac{5}{5}\ x\ 100\% = 100\%$

Algoritma A* yang dikembangkan dapat menyelesaikan *map maze* dengan tingkat keberhasilan (optimalitas) sebesar :

$$\frac{60 + 60 + 100}{3\ x\ 100}\ x\ 100\% = 73{,}33\% \qquad \textbf{[4.2]}$$

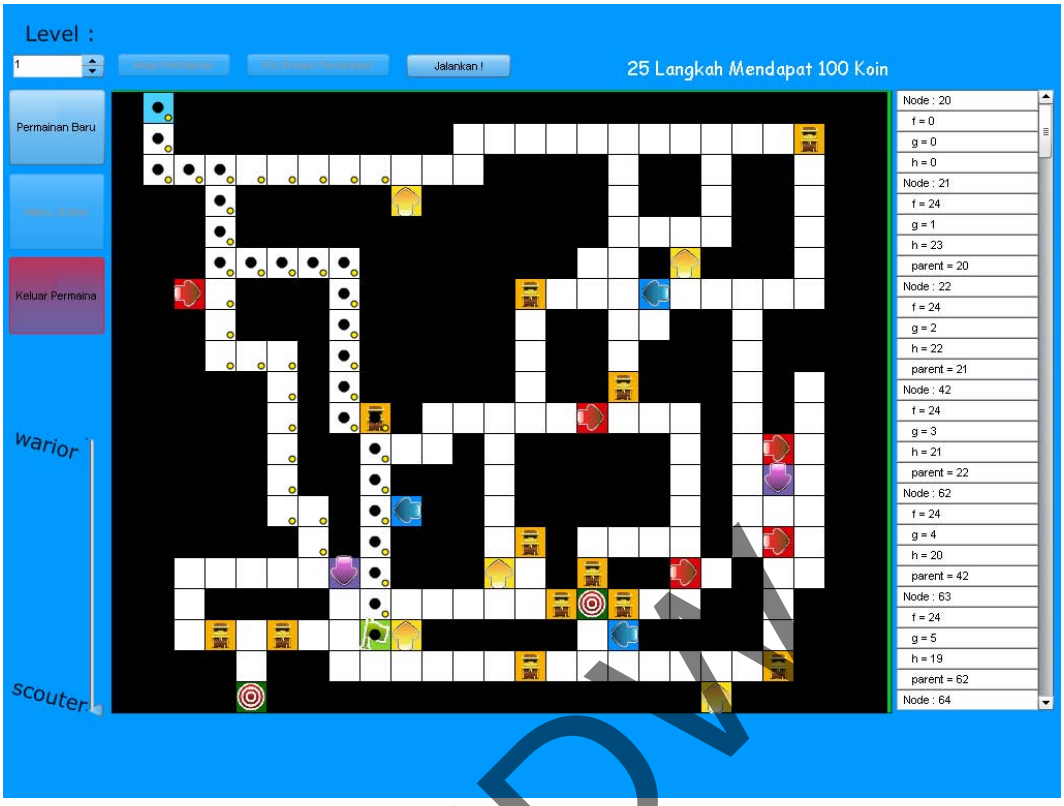
Berdasarkan data tersebut, penulis menyimpulkan bahwa tingkat keberhasilan agen cerdas dalam menyelesaikan permainan *maze* tergantung oleh modifikasi pemain dan batasan dari *map maze*. Bila modifikasi yang diberikan pemain dan aturan batas yang ditentukan sesuai, algoritma A* yang telah dikembangkan dapat menyelesaikan permainan *maze* dengan tingkat keberhasilan lebih dari 50%.

Keberhasilan agen cerdas dalam menemukan *node goal* juga ditentukan oleh rancangan dari *map maze*. Bila dalam perancangan *map maze* tidak dilakukan secara hati-hati, dapat menyebabkan *map maze* yang telah dimodifikasi oleh pemain tidak memiliki solusi (terjadi kebuntuan). Gambar 4.17 menunjukkan terjadinya kesalahan pada desain *map maze*, sehingga menyebabkan dalam *map maze* tersebut tidak tersedia solusi. Dalam penelitian ini, penulis tidak melakukan evaluasi pada desain *map maze*, sehingga bila solusi pada *map maze* tidak tersedia, hal tersebut tidak dihitung sebagai contoh kasus yang gagal.



Gambar 4.17 Jalur *Maze* Tidak Ditemukan (Buntu)

Hasil implementasi algoritma A* di dalam sistem, menunjukkan bahwa dalam memenuhi *goal* jalur terpendek, sistem yang dibangun dapat memberikan hasil yang selalu optimal. Dalam proses pencarian, jumlah *node* yang dibangkitkan tidak terlalu besar. Gambar 4.18 menunjukkan bahwa *node* yang dibangkitkan tidak terlalu banyak dan tidak terlalu jauh dari *goal* (ditunjukkan oleh titik kuning). Tingkat keberhasilan algoritma A* yang diterapkan dalam permainan dalam menemukan jalur terpendek ditunjukkan pada tabel 4.5. Dengan aturan batas yang sesuai yang diberikan pada *map maze*, agen cerdas dapat menemukan jalur terpendek dengan tingkat keberhasilan 100% (optimal). Dalam penelitian ini penulis tidak meneliti penentuan aturan batas untuk setiap *map maze* yang dimainkan, dalam melakukan uji coba, penentuan batas tersebut dilakukan melalui perhitungan rata-rata banyakanya *node* yang dilalui oleh jalur-jalur solusi.



Gambar 4.18 Pencarian Jalur dengan Algoritma A*

Gambar 4.19 Pencarian Jalur dengan Algoritma A*

Pengembangan Algoritma A* yang diimplementasikan untuk menyelesaikan permainan ini, terbukti dapat memenuhi *goal* permainan. Hasil pencarian dengan algoritma A* biasa dapat berbeda dengan hasil pencarian algoritma A* yanf dikembangkan. Gambar 4.19 menunjukkan perbedaan hasil pencarian jalur algoritma A* biasa dengan algoritma A* yang dikembangkan. Pencarian dengan algoritma A* yang dikembangkan memungkinkan agen cerdas untuk menemukan *goal*, serta memungkinkan agen cerdas untuk mendapatkan koin yang banyak.

Gambar 4.20 Jalur Algoritma A* (kiri) dengan Pengembangan Algoritma A* (kanan) Dalam Menyelesaikan Permainan *Maze*

Dari ujicoba yang dilakukan penulis terhadap sistem yang dibangun, dapat disimpulkan bahwa pengembangan algoritma A* yang diimplementasikan dalam menyelesaikan permainan *User Costumized Maze* memiliki kemampuan 100% dalam menemukan *node goal* selama solusi tersedia. Sedangkan optimalitas algoritma A* yang dikembangkan dalam menyelesaikan permainan *maze* dengan aturan batas yang sulit adalah sebesar 73,33%. Optimalitas atau tingkat keberhasilan ini dapat bertambah bila aturan batas yang diberlakukan mudah.

LAMPIRAN A

LISTING PROGRAM

1. MainVariable.as

```
package {
    import flash.display.MovieClip;
    import flash.display.Shape;

    //Pindahan dari .fla, biar rapi
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.text.TextFieldType;
    import flash.text.TextFieldAutoSize;
    import flash.events.Event;
    import flash.events.TimerEvent;
    import flash.events.MouseEvent;
    import flash.utils.Timer;
    import flash.system.*;

    public class MainVariable extends MovieClip {
        const FRAME_MAIN_MENU = 1;
        const FRAME_INGAME = 2;
        const FRAME_WARNING = 3;
        const FRAME_HELP = 4;

        //const
        //COL (X), ROW(Y)
        const MAP_MAX_X:Number = 25;
        const MAP_MAX_Y:Number = 20;
        const MAP_TILE_SIZE:Number = 30;

        //MAP_FLAG
        const MAP_TIPE_NORM = 0;
        const MAP_TIPE_WALL = 1;
        const MAP_TIPE_GOAL = 2;

        const MAP_TIPE_RIGHT = 3;
        const MAP_TIPE_LEFT = 4;
        const MAP_TIPE_UP = 5;
        const MAP_TIPE_DOWN = 6;

        const MAP_TIPE_OUT = 7;
        const MAP_TIPE_START = 8;

        const MAP_TIPE_COIN = 9;

        const HEURISTIC_FAST = 0;
        const HEURISTIC_MULTI = 1;
        const HEURISTIC_GREED = 2;

        //map variabel
        var grid:Grid;
        var mapArray:Array;
```

```
        var pathMarker:Array = [];

        var startID:uint;
        var goalID:uint;
        var coinArray:Array;

        //Hati2 konsep mapnya; di sini : map[row][col] -> row 0
([1,0,1,1,1]), punya element 5 col : 1,0,1,1,1

        var gameMapX:Number;
        var gameMapY:Number;
        var gameMap:Array;

        var AStarClass:AStar;

        //game variable
        public var currentLevel:Number;
        var bIsStarted:Boolean = false;
        var bIsConfirmed:Boolean = false;
        var bIsFinished:Boolean = false;
        var bInitializeFrame:Boolean = false;

        var warningMessage:String = "";
        var heuristicTipe:Number;

        var mapXmlLoader:XmlLoader;
        var maximumLevel:Number = 1;
    }
}
```

2. ModMaze.fla

a. Clip "Main Menu"

```
fscommand("allowscale","true");
fscommand("fullscreen","true");
stop();

buttonMulaiMenu.addEventListener(MouseEvent.CLICK, masukInGame);
function masukInGame(e:MouseEvent):void
{
    gotoAndPlay(FRAME_INGAME);
}

buttonBantuan.addEventListener(MouseEvent.CLICK, masukHelp);
function masukHelp(e:MouseEvent):void
{
    gotoAndPlay(FRAME_HELP);
}

buttonExitMenu.addEventListener(MouseEvent.CLICK, keluarGameMenu);
function keluarGameMenu(e:MouseEvent):void
{
    fscommand("quit");
}
```

b. Clip "In Game"

```
fscommand("allowscale","true");
fscommand("fullscreen","true");
stop();

stage.addEventListener(Event.ENTER_FRAME, gameMainLoop);
function gameMainLoop(evt:Event)
{
    if(!bInitializeFrame)
    {
        stepperLevel.maximum = maximumLevel;

        buttonMulai.enabled = true;
        buttonMulai.addEventListener(MouseEvent.CLICK, startGame);
        trace("\nInisialisasi");

        if(bIsStarted)
        {
            buttonMulai.enabled = false;
            buttonMulai.removeEventListener(MouseEvent.CLICK,
startGame);

            buttonConfirm.enabled = true;
            buttonConfirm.addEventListener(MouseEvent.CLICK,
confirmMaze); //bisa confirm

            buttonMainMenu.enabled = false;
            buttonMainMenu.removeEventListener(MouseEvent.CLICK,
toMainMenu);

            stepperLevel.enabled = true;
        }

        bInitializeFrame = true;
    }
}

buttonExit.addEventListener(MouseEvent.CLICK, keluarGame);
function keluarGame(e:MouseEvent):void
{
    fscommand("quit");
}

buttonMainMenu.addEventListener(MouseEvent.CLICK, toMainMenu);
function toMainMenu(e:MouseEvent):void
{
    bInitializeFrame = false;
    stage.removeEventListener(Event.ENTER_FRAME, gameMainLoop);
    gotoAndPlay(FRAME_MAIN_MENU);
}

function startGame(evt:MouseEvent)
{
    buttonMulai.enabled = false;
```

```
    buttonMulai.removeEventListener(MouseEvent.CLICK, startGame);
//gak bisa start

    var level:Number = 1;
    level = stepperLevel.value;
    trace("LEVEL = "+level);
    creatingMap(level);

    bIsStarted = true;
    bIsConfirmed = false;
    bIsFinished = false;

    //Controling Tombol
    buttonConfirm.enabled = true;
    buttonConfirm.addEventListener(MouseEvent.CLICK, confirmMaze);
//bisa confirm

    sliderTreshold.enabled = true;
    sliderTreshold.addEventListener(Event.CHANGE, moveSlider);

    buttonMainMenu.enabled = false;
    buttonMainMenu.removeEventListener(MouseEvent.CLICK,
toMainMenu);

    //Minimal pernah load xml
    stepperLevel.maximum = maximumLevel;
    stepperLevel.enabled = true;
}

sliderTreshold.minimum = 0;
sliderTreshold.maximum = 1;
sliderTreshold.value = 0;
sliderTreshold.tickInterval = 1;
sliderTreshold.snapInterval = 1;
sliderTreshold.liveDragging = true;
function moveSlider(e:Event)
{
    heuristicTipe = sliderTreshold.value;

}

stepperLevel.minimum = 1;
stepperLevel.maximum = 1;
stepperLevel.value = 1;
stepperLevel.stepSize = 1;
stepperLevel.enabled = false;

function creatingMap(level:Number)
{

    loadMapData(level);
}

function tileClick(evt:MouseEvent)
{
    if(bIsConfirmed || bIsFinished)
```

```
    {
        return; //hanya bisa klik kalau belum di confirm
    }

    //Optimasi draw
    if((evt.currentTarget.tipe != MAP_TIPE_RIGHT) &&
       (evt.currentTarget.tipe != MAP_TIPE_LEFT) &&
       (evt.currentTarget.tipe != MAP_TIPE_UP) &&
       (evt.currentTarget.tipe != MAP_TIPE_DOWN) &&
       (evt.currentTarget.tipe != MAP_TIPE_GOAL) &&
       (evt.currentTarget.tipe != MAP_TIPE_OUT))
    {
        return; //biar ga berat;
    }

    var switchTmp:Number;
    evt.currentTarget.removeEventListener(MouseEvent.CLICK,
tileClick);

    if(evt.currentTarget.tipe == MAP_TIPE_RIGHT)
    {
        switchTmp =
gameMap[evt.currentTarget.row][(evt.currentTarget.col)+1];
        gameMap[evt.currentTarget.row][(evt.currentTarget.col)+1]
= MAP_TIPE_LEFT;
        gameMap[evt.currentTarget.row][evt.currentTarget.col] =
switchTmp;

mapArray[evt.currentTarget.row][(evt.currentTarget.col)+1].tipe =
gameMap[evt.currentTarget.row][(evt.currentTarget.col)+1];

mapArray[evt.currentTarget.row][evt.currentTarget.col].tipe =
gameMap[evt.currentTarget.row][evt.currentTarget.col];
        //redraw seperlunya

mapArray[evt.currentTarget.row][(evt.currentTarget.col)+1].fungsiR
edraw();
    }

    if(evt.currentTarget.tipe == MAP_TIPE_LEFT)
    {
        switchTmp =
gameMap[evt.currentTarget.row][(evt.currentTarget.col)-1];
        gameMap[evt.currentTarget.row][(evt.currentTarget.col)-1]
= MAP_TIPE_RIGHT;
        gameMap[evt.currentTarget.row][evt.currentTarget.col] =
switchTmp;
        mapArray[evt.currentTarget.row][(evt.currentTarget.col)-
1].tipe = gameMap[evt.currentTarget.row][(evt.currentTarget.col)-
1];

mapArray[evt.currentTarget.row][evt.currentTarget.col].tipe =
gameMap[evt.currentTarget.row][evt.currentTarget.col];
        mapArray[evt.currentTarget.row][(evt.currentTarget.col)-
1].fungsiRedraw();
    }
```

```
    if(evt.currentTarget.tipe == MAP_TIPE_UP)
    {
        switchTmp = gameMap[(evt.currentTarget.row)-
1][evt.currentTarget.col];
        gameMap[(evt.currentTarget.row)-1][evt.currentTarget.col]
= MAP_TIPE_DOWN;
        gameMap[evt.currentTarget.row][evt.currentTarget.col] =
switchTmp;
        mapArray[(evt.currentTarget.row)-
1][evt.currentTarget.col].tipe = gameMap[(evt.currentTarget.row)-
1][evt.currentTarget.col];

mapArray[evt.currentTarget.row][evt.currentTarget.col].tipe =
gameMap[evt.currentTarget.row][evt.currentTarget.col];
        //redraw seperlunya
        mapArray[(evt.currentTarget.row)-
1][evt.currentTarget.col].fungsiRedraw();
    }

    if(evt.currentTarget.tipe == MAP_TIPE_DOWN)
    {
        switchTmp =
gameMap[(evt.currentTarget.row)+1][evt.currentTarget.col];
        gameMap[(evt.currentTarget.row)+1][evt.currentTarget.col]
= MAP_TIPE_UP;
        gameMap[evt.currentTarget.row][evt.currentTarget.col] =
switchTmp;

mapArray[(evt.currentTarget.row)+1][evt.currentTarget.col].tipe =
gameMap[(evt.currentTarget.row)+1][evt.currentTarget.col];

mapArray[evt.currentTarget.row][evt.currentTarget.col].tipe =
gameMap[evt.currentTarget.row][evt.currentTarget.col];
        //redraw seperlunya

mapArray[(evt.currentTarget.row)+1][evt.currentTarget.col].fungsiR
edraw();
    }

    if(evt.currentTarget.tipe == MAP_TIPE_GOAL)
    {
        gameMap[evt.currentTarget.row][evt.currentTarget.col] =
MAP_TIPE_OUT;

mapArray[evt.currentTarget.row][evt.currentTarget.col].tipe =
gameMap[evt.currentTarget.row][evt.currentTarget.col];
    }
    else if(evt.currentTarget.tipe == MAP_TIPE_OUT)
    {
        gameMap[evt.currentTarget.row][evt.currentTarget.col] =
MAP_TIPE_GOAL;

mapArray[evt.currentTarget.row][evt.currentTarget.col].tipe =
gameMap[evt.currentTarget.row][evt.currentTarget.col];
    }
```

```
    //seperlunya

mapArray[evt.currentTarget.row][evt.currentTarget.col].fungsiRedra
w();

    //Nic : Redraw, stupid way
    evt.currentTarget.addEventListener(MouseEvent.CLICK,
tileClick);
}

//buttonConfirm.addEventListener(MouseEvent.CLICK, confirmMaze);
function confirmMaze(e:MouseEvent):void
{
    trace("Confirm");
    var countGoal:Number = 0;
    //validasi start, goal, dll, split button and funct if needed
[done]
    for(var row:int = 0;row<gameMap.length;row++)
    {
        for(var col:int = 0;col<gameMap[0].length;col++)
        {
            if(gameMap[row][col] == MAP_TIPE_START)
            {
                startID = (col * gameMapY) + row;
                trace("found start id at "+startID);
            }

            if(gameMap[row][col] == MAP_TIPE_GOAL)
            {
                countGoal++;

                if(countGoal > 1)
                {
                    //goal > 1 -> Tidak valid !
                    break;
                }

                goalID = (col * gameMapY) + row;
                trace("found goal id at "+goalID);
            }
        }
    }

    if(countGoal > 1)
    {
        trace("Goal > 1");
        showWarningMessage("Goal lebih dari 1 !");
    }
    else if(countGoal < 1)
    {
        trace("Goal < 1");
        showWarningMessage("Goal tidak ditemukan !");
    }
    else
    {
        bIsConfirmed = true;
```

```
        //Controling tombol : biar aman disable tombol lain
        buttonConfirm.enabled = false;
        buttonConfirm.removeEventListener(MouseEvent.CLICK,
confirmMaze);
        buttonJalankan.enabled = true;
        buttonJalankan.addEventListener(MouseEvent.CLICK,
startWalk);

        heuristicTipe = sliderTreshold.value;
        sliderTreshold.enabled = false;
        sliderTreshold.removeEventListener(Event.CHANGE,
moveSlider);
    }
}

//buttonJalankan.addEventListener(MouseEvent.CLICK, startWalk);
function startWalk(evt:MouseEvent)
{
    //Controling tombol : disable tombol
    buttonJalankan.removeEventListener(MouseEvent.CLICK,
startWalk);
    trace("create path dengan heuristic "+heuristicTipe);

    if(bIsFinished) // clear lagi klw uda finish
    {
        textScore.text = "";

        tampilListNode.removeAll();

        while(pathMarker.length != 0)
        {
            stage.removeChild(pathMarker.shift());
        }

        heuristicTipe = sliderTreshold.value;
        sliderTreshold.enabled = false;
        sliderTreshold.removeEventListener(Event.CHANGE,
moveSlider);
        buttonNewGame.enabled = false;
        buttonNewGame.removeEventListener(MouseEvent.CLICK,
startNewGame);
    }

    bIsStarted = false;
    bIsConfirmed = false;
    bIsFinished = true;

    AStarClass = new AStar();
    var pathFinding:Array = AStarClass.findPath(startID, goalID,
gameMap, coinArray, heuristicTipe);
    var aStarNodeArray:Array = AStarClass.visitedNode;

    //textScore.label = pathFinding.length" Langkah membawa
"+poin;
```

```
    displayPath(pathFinding);
    displayScore(pathFinding.length, AStarClass.totalCoin);
    displayVisited(aStarNodeArray);

    //setelah finish
    buttonJalankan.addEventListener(MouseEvent.CLICK, startWalk);
    buttonNewGame.enabled = true;
    buttonNewGame.addEventListener(MouseEvent.CLICK,
startNewGame);
    sliderTreshold.enabled = true;
    sliderTreshold.addEventListener(Event.CHANGE, moveSlider);
}

function displayPath(path:Array):void
{
    trace("path marker length = "+pathMarker.length);

    for(var i:int=0; i<path.length; i++)
    {
        //trace("gambar langkah - "+i+" di "+path[i].id);
        var shape:Shape = new Shape();
        shape.graphics.lineStyle(1);
        shape.graphics.beginFill(0x000000);

shape.graphics.drawCircle(MAP_TILE_SIZE/2,MAP_TILE_SIZE/2,5);
        shape.graphics.endFill();
        stage.addChild(shape);

        shape.x /*col*/ = 105+(uint((path[i].id / gameMapY)) *
MAP_TILE_SIZE);
        shape.y /*row*/ = 85+(uint((path[i].id % gameMapY)) *
MAP_TILE_SIZE);

        //tampilListNode.addItem({label:"Node : "+path[i].id+"
f="+path[i].f+" g="+path[i].g+" h="+path[i].h});
        tampilListNode.addItem({label:"Node : "+path[i].id});
        tampilListNode.addItem({label:"\tf = "+path[i].f});
        tampilListNode.addItem({label:"\tg = "+path[i].g});
        tampilListNode.addItem({label:"\th = "+path[i].h});
        if(path[i].tipe != MAP_TIPE_START)
            tampilListNode.addItem({label:"\tparent =
"+path[i].parent.id});
        //tampilListNode.addEventListener(Event.CHANGE,
traceListData);
        pathMarker.push(shape);
    }
}

function displayScore(langkah:Number, coin:Number):void
{
    trace("langkah = "+langkah+" koin = "+coin);
    textScore.text = ""+langkah+" Langkah Mendapat "+coin+" Koin";
}

function displayVisited(nodeMap:Array):void
{
```

```
    tampilListNode.addItem({label:"---------- Visited ----------
"});
    for(var i:int=0; i<nodeMap.length; i++)
    {
        for(var j:int=0; j<nodeMap[0].length; j++)
        {
            if(nodeMap[i][j].visited)
            {
                //trace("gambar langkah - "+i+" di "+path[i].id);
                var shape:Shape = new Shape();
                shape.graphics.lineStyle(1);
                shape.graphics.beginFill(0xFFEF04);
                shape.graphics.drawCircle(MAP_TILE_SIZE -
5,MAP_TILE_SIZE - 5,3);
                shape.graphics.endFill();
                stage.addChild(shape);

                shape.x /*col*/ = 105+(uint((nodeMap[i][j].id /
gameMapY)) * MAP_TILE_SIZE);
                shape.y /*row*/ = 85+(uint((nodeMap[i][j].id %
gameMapY)) * MAP_TILE_SIZE);

                tampilListNode.addItem({label:"Node :
"+nodeMap[i][j].id});
                tampilListNode.addItem({label:"\tf =
"+nodeMap[i][j].f});
                tampilListNode.addItem({label:"\tg =
"+nodeMap[i][j].g});
                tampilListNode.addItem({label:"\th =
"+nodeMap[i][j].h});
                if(nodeMap[i][j].tipe != MAP_TIPE_START)
                    tampilListNode.addItem({label:"\tparent =
"+nodeMap[i][j].parent.id});
                pathMarker.push(shape);
            }
        }
    }
}

function showWarningMessage(s:String):void
{
    warningMessage = s;
    trace(warningMessage);
    stage.removeEventListener(Event.ENTER_FRAME, gameMainLoop);
    gotoAndPlay(FRAME_WARNING);
}

//Nic 1aug: Load dari file :)
function loadMapData(level:Number)
{
    trace(" ... LOAD MAP LEVEL "+level+"... ");
    var fileXmlName:String = "map.xml";
    mapXmlLoader = new XmlLoader(fileXmlName);
    mapXmlLoader.addEventListener(Event.COMPLETE,
generateMapFromData);
```

```
    currentLevel = level - 1;
}

function generateMapFromData(e:Event):void
{
    trace(" ... GENERATE MAP LEVEL "+currentLevel+"... ");
    gameMap = new Array();
    mapArray = new Array();
    coinArray = new Array();
    maximumLevel = mapXmlLoader.dataXML.level.length();
    //trace("max = "+maximumLevel);

    trace(" ... Parse From XML ...  ");
    var xmlRowArray:Array;
    var xmlRowLength:int =
mapXmlLoader.dataXML.level[currentLevel].tilemap.row.length();
    var xmlRowData:String;

    for(var i:int=0;i<xmlRowLength;i++)
    {
        xmlRowData =
mapXmlLoader.dataXML.level[currentLevel].tilemap.row[i].toString()
;

        xmlRowArray = xmlRowData.split(",");
        gameMap.push(xmlRowArray);
    }

    gameMapX = gameMap[0].length;
    gameMapY = gameMap.length;
    trace("length col = "+gameMapX);
    trace("length row = "+gameMapY);

    trace(" ... Creating Map ...  ");
    for(var row:int=0; row<gameMap.length; row++)
    {
        mapArray.push(new Array());
        for(var col:int=0; col<gameMap[0].length; col++)
        {
            // e.g : node 20 -> col = 1, row = 0. 20/20 = 1, 20%20
= 0 -> col 1, row 0 ; array 0,1
            grid = new
Grid(col,row,105+(col*MAP_TILE_SIZE),85+(row*MAP_TILE_SIZE),MAP_TI
LE_SIZE,gameMap[row][col],(col * gameMapY) + row);
            mapArray[row].push(grid);
            if(gameMap[row][col] == MAP_TIPE_COIN)
            {
                coinArray.push(grid);
            }

            grid.addEventListener(MouseEvent.CLICK, tileClick);
            stage.addChild(grid);
        }
    }
}
```

```
function startNewGame(e:MouseEvent):void
{
    trace("\n=------------- NEW GAME -------------\n");
    //trace("stage child = "+stage.numChildren);
    buttonJalankan.enabled = false;
    buttonJalankan.removeEventListener(MouseEvent.CLICK,
startWalk);

    buttonNewGame.enabled = false;
    buttonNewGame.removeEventListener(MouseEvent.CLICK,
startNewGame);

    buttonMainMenu.enabled = true;
    buttonMainMenu.addEventListener(MouseEvent.CLICK, toMainMenu);

    for(var row=0; row<mapArray.length; row++)
    {
        for(var col=0;col<mapArray[0].length;col++)
        {
            stage.removeChild(mapArray[row][col]);
        }
    }

    while(pathMarker.length != 0)
    {
        stage.removeChild(pathMarker.shift());
    }

    textScore.text = "";

    tampilListNode.removeAll();

    startID = 0;
    goalID = 0;

    gameMapX = 0;
    gameMapY = 0;
    gameMap = [];
    gameMap.length = 0;
    mapArray = [];
    mapArray.length = 0;
    coinArray = [];
    coinArray.length = 0;
    pathMarker = [];
    pathMarker.length = 0;

    bIsStarted = false;
    bIsConfirmed = false;
    bIsFinished = false;
    bInitializeFrame = false;

    warningMessage = "";
    heuristicTipe = 0;

    mapXmlLoader.dataXML = null;
```

```
    AStarClass.nodeMap = [];
    AStarClass.nodeMap.length = 0;
    AStarClass.visitedNode = [];
    AStarClass.visitedNode.length = 0;
    AStarClass.shortPath = [];
    AStarClass.shortPath.length = 0;

    AStarClass.startNodeID = 0;
    AStarClass.goalNodeID = 0;
    AStarClass.totalCoin = 0;

    buttonJalankan.addEventListener(MouseEvent.CLICK, startWalk);
    sliderTreshold.value = 1;
    sliderTreshold.enabled = true;
    sliderTreshold.addEventListener(Event.CHANGE, moveSlider);

    trace("stage child = "+stage.numChildren);
    gotoAndStop(FRAME_INGAME);
}
```

c. Clip "Warning"

```
fscommand("allowscale","true");
fscommand("fullscreen","true");
stop();

var warningTextFormat:TextFormat = new TextFormat();
warningTextFormat.bold = true;
warningTextFormat.font = "Courier New";
warningTextFormat.size = 16;
warningTextFormat.indent = "center";

var warningText:TextField = new TextField();
warningText.defaultTextFormat = warningTextFormat;
warningText.text = "Terjadi Kesahalahan : ";
warningText.appendText(warningMessage);
warningText.type = TextFieldType.DYNAMIC;
warningText.border = false;
warningText.selectable = false;
warningText.textColor = 0x000000;
warningText.autoSize = TextFieldAutoSize.CENTER;
warningText.x = (stage.stageWidth>>1) - (warningText.width>>1);
warningText.y = 272;

stage.addEventListener(Event.ENTER_FRAME, showWarning);
function showWarning(e:Event):void
{
    stage.addChild(warningText);

    for(var row=0; row<mapArray.length; row++)
    {
        for(var col=0;col<mapArray[0].length;col++)
        {
            stage.removeChild(mapArray[row][col]);
        }
```

```
    }

    stage.removeEventListener(Event.ENTER_FRAME, showWarning);
}

buttonOk.addEventListener(MouseEvent.CLICK, kembali);
function kembali(e:MouseEvent):void
{
    stage.removeChild(warningText);
    gotoAndPlay(FRAME_INGAME);

    warningMessage = "";
    for(var row=0; row<mapArray.length; row++)
    {
        for(var col=0;col<mapArray[0].length;col++)
        {
            stage.addChild(mapArray[row][col]);
        }
    }
    bIsStarted = true;
    bIsConfirmed = false;
    bIsFinished = false;

    bInitializeFrame = false;
}
```

d. Clip "Help"

```
fscommand("allowscale","true");
fscommand("fullscreen","true");
stop();

buttonExitHelp.addEventListener(MouseEvent.CLICK, helpKeMenu);
function helpKeMenu(e:MouseEvent):void
{
    gotoAndPlay(FRAME_MAIN_MENU);
}
```


3. AStar.as

```
package
{
    public class AStar
    {
        const MAP_TILE_SIZE:uint = 30;

        const MAP_TIPE_NORM = 0;
        const MAP_TIPE_WALL = 1;
        const MAP_TIPE_GOAL = 2;

        const MAP_TIPE_RIGHT = 3;
        const MAP_TIPE_LEFT = 4;
        const MAP_TIPE_UP = 5;
        const MAP_TIPE_DOWN = 6;
```

```
        const MAP_TIPE_OUT = 7;
        const MAP_TIPE_START = 8;

        const MAP_TIPE_COIN = 9;


        const HEURISTIC_FAST = 0;
        const HEURISTIC_MULTI = 1;
        const HEURISTIC_GREED = 2;

        private var MAP_MAX_COL:uint;
        private var MAP_MAX_ROW:uint;
        public var nodeMap:Array;
        public var visitedNode:Array;

        //stored shortest path
        public var shortPath:Array;

        public var startNodeID:uint;
        public var goalNodeID:uint;

        public var totalCoin:uint = 0;

        public function AStar() {
            trace("AStar class by Nico");
        }

        //load map lalu dijadikan node, untuk akses property map
(f,g,h,tipe)
        private function createNodeMap(loadedMap:Array):Array
        {
            //sementara, buat nyimpen semua node
            var map:Array = [];
            var col:int;
            var row:int;
            //NIC : 1-5-2013 --> UBAH SEMUA ROW dan COL di fix !!!
            visitedNode = new Array();
            for(row = 0;row < loadedMap.length; row++) //row map
e.g : 3
            {
                //Array 2D
                map[row] = new Array();
                visitedNode.push(new Array());

                //for(col = 0;col < loadedMap[0].length; col++)
//per col e.g : 5
                for(col = 0;col < loadedMap[0].length; col++)
//per col e.g : 5
                {
                    //var node:Array = [];
                    //Nic: 25-3-2013. Tiap node punya property -->
bkin object
                    var node:Object = new Object();

                    //Nic: 25-3-2013. Property A* yang dibutuhkan
```

```
                    node.f = 0;
                    node.g = 0;
                    node.h = 0;
                    node.parent = null;
                    // node.next = null;

                    // Nic: 25-3-2013. Row, Col, ID, Tipe/warna
                    // e.g : node 20 -> col = 1, row = 0. 20/20 =
1, 20%20 = 0 -> col 1, row 0 ; array 0,1
                    node.row = row;
                    node.col = col;
                    node.id = (col * /*Y*/ MAP_MAX_ROW) + row;
//Nic 27-Jul : MAP_MAX_COL->MAP_MAX_ROW
                    node.tipe = loadedMap[row][col]; //0 passable,
1 obstacle, 2 goal, 3 & 4 editable, 5 hadiah
                    // trace("node["+row+"]["+col+"] = "+node.id+"
tipe = "+node.tipe);

                    node.visited = false;
                    node.poin = 0;
                    if(node.tipe == MAP_TIPE_COIN)
                    {
                        node.poin = 100;
                    }

                    //row - node --> dibalik" dasarnya sama, biar
gampang saja (lihat bentuk array)
                    //tapi di sini karena bikin arraynya nyamping,
baca, nulis juga nyamping
                    map[row][col] = node;
                    visitedNode[row].push(node);
                }
            }
            return map;
        }

        //fungsi utama, mencari shortest path A*
        public function findPath(startNodeID:uint,
goalNodeID:uint, mazeMap:Array, coinArray:Array,
heuristic:Number):Array
        {
            MAP_MAX_COL = mazeMap[0].length;
            MAP_MAX_ROW = mazeMap.length;
            trace("AStar MAP_MAX_COL = "+MAP_MAX_COL);
            trace("AStar MAP_MAX_ROW = "+MAP_MAX_ROW);

            //Nic: 25-3-2013. Map dibikin object node
            nodeMap = createNodeMap(mazeMap);

            //hasil sorthest path, inisialisasi dulu, nanti di
akhir di return
            shortPath = [];
            //open close list
            var openList:Array = [];
            var closeList:Array = [];
```

```
            //inisialisasi start node
            this.startNodeID = startNodeID;
            this.goalNodeID = goalNodeID;

            //Nic: 25-3-2013. Untuk current node, pertama kali
masukan startNode
            var curNode:Object = nodeMap[uint(startNodeID %
MAP_MAX_ROW)][uint(startNodeID / MAP_MAX_ROW)];
            trace("startNode = "+uint(startNodeID %
MAP_MAX_ROW)+","+uint(startNodeID / MAP_MAX_ROW));
            var goalNode:Object = nodeMap[uint(goalNodeID %
MAP_MAX_ROW)][uint(goalNodeID / MAP_MAX_ROW)];
            trace("goalNode = "+uint(goalNodeID %
MAP_MAX_ROW)+","+uint(goalNodeID / MAP_MAX_ROW));

            var maxCoin:uint = coinArray.length;
            var count:int = 0;
            while(curNode.id != goalNodeID) //Nic: 10-4-2013.
Lebih optimal !!
            {
                trace("-----------------------------------------
------");
                trace("langkah - "+count);
                count++;
                curNode.visited = true;
                visitedNode[curNode.row][curNode.col] = curNode;
                for(var col:int = -1;col < 2;col++) //cek semua
arah (8 arah)
                {
                    for(var row:int = -1;row < 2;row++) //cek
semua arah (8 arah)
                    {
                        //koordinat grid yg di test
                        var tmpRow:int = curNode.row + row;
                        var tmpCol:int = curNode.col + col;

                        if(col == 0 || row == 0) //tanpa diagonal
(4 arah)
                        {
                            // trace("col = "+col+" row = "+row);
                            tmpRow = curNode.row + row;
                            tmpCol = curNode.col + col;

                            //cek batas grid -> cell harus di
dalam nodeMap
                            if((tmpRow > -1 && tmpRow <=
nodeMap.length)
                            && (tmpCol > -1 && tmpCol <=
nodeMap[0].length))
                            {

                                //pengecekan baru -> klw bukan cur
node / klw bukan obstacle
                                if((nodeMap[tmpRow][tmpCol].id !=
curNode.id)
```

```
                                        && (mazeMap[tmpRow][tmpCol] !=
MAP_TIPE_WALL)
                                        && (mazeMap[tmpRow][tmpCol] !=
MAP_TIPE_RIGHT)
                                        && (mazeMap[tmpRow][tmpCol] !=
MAP_TIPE_LEFT)
                                        && (mazeMap[tmpRow][tmpCol] !=
MAP_TIPE_UP)
                                        && (mazeMap[tmpRow][tmpCol] !=
MAP_TIPE_DOWN)
                                        && (mazeMap[tmpRow][tmpCol] !=
MAP_TIPE_OUT))
                                        {
                                            trace("curNode sekarang -
"+curNode.id);
                                            trace("cek node map
"+nodeMap[tmpRow][tmpCol].id);
                                            //kalau semua valid (dah di
cek), kandidat next node
                                            var tmpNextNode:Object =
nodeMap[tmpRow][tmpCol];

                                            //fungsi perhitungan A*
                                            var f:uint, g:uint, h1:uint=0,
h2:uint=0, h3:uint = 0, h:uint=0;

                                            //horizontal, vertical only
                                            var cost:uint = MAP_TILE_SIZE;

                                            //Nic Juli: nambah poin
                                            var alpha:Number = 0.75;
                                            if(heuristic ==
HEURISTIC_MULTI)
                                            {
                                                if(curNode.poin > 0)
                                                {
                                                    // alpha = 0;
                                                    // alpha = 0.5; //last
pake ini
                                                    alpha = 0.75;
                                                }

                                                var fG:Number;
                                                fG = alpha * (curNode.g -
MAP_TILE_SIZE);
                                                g = MAP_TILE_SIZE +
Math.ceil(fG);

                                                //g = cost + alpha *
(curNode.g - cost); //last pake ini
                                            }
                                            else
                                            {
                                                g = curNode.g + cost; //
costnya pake tile size, g : dari node sblumnya
                                            }
```

```
                                    //heuristic :  Manhattan ->
|dx-nx|+|dy-ny|
                                    h1 = (Math.abs(goalNode.col -
tmpNextNode.col)) * MAP_TILE_SIZE + (Math.abs(goalNode.row -
tmpNextNode.row)) * MAP_TILE_SIZE; //cost dgn tile size
                                    var nodeCoin:Object;
                                    var h2Temp:uint;
                                    for(var
coinIdx=0;coinIdx<coinArray.length; coinIdx++)
                                    {
                                        nodeCoin =
nodeMap[coinArray[coinIdx].row][coinArray[coinIdx].col];

                                        h2 =
(Math.abs(nodeCoin.col - tmpNextNode.col)) * MAP_TILE_SIZE +
(Math.abs(nodeCoin.row - tmpNextNode.row)) * MAP_TILE_SIZE;

                                        if(coinIdx == 0)
                                        {
                                            h2Temp = h2;

                                        }
                                        else
                                        {
                                            if(h2 < h2Temp)
                                            {
                                                h2Temp = h2;
                                            }
                                        }

                                    }

                                    if(heuristic ==
HEURISTIC_MULTI)

                                    {
                                        h2 = h2Temp;
                                        h = h2;
                                    }
                                    else
                                    {
                                        h = (Math.abs(goalNode.col
- tmpNextNode.col)) * MAP_TILE_SIZE + (Math.abs(goalNode.row -
tmpNextNode.row)) * MAP_TILE_SIZE;
                                    }

                                    f = g + h;

                                    trace("Node -
"+tmpNextNode.id+" g = "+g/cost+" h = "+h/cost+" f = "+f/cost);

                                    //indikator ada di open/close
list (daripada cek array 1 1)
                                    var bIsOnOpenList:Boolean =
false;
                                    var bIsOnCloseList:Boolean =
false;
```

```
                                        for(var
i:int=0;i<openList.length;i++) //cek di open list
                                        {

if(tmpNextNode == openList[i]) //cek smua isi openList
                                                bIsOnOpenList = true;
//klw ada, di mark open
                                        }

                                        for(var
j:int=0;j<closeList.length;j++) //cek di close list
                                        {
                                            if(tmpNextNode ==
closeList[j]) //cek smua isi closeList
                                                bIsOnCloseList = true;
//klw ada, di mark close
                                        }



                                        if(!(bIsOnCloseList ||
bIsOnOpenList)) //kalau belum masuk list -> masukkan
                                        {
                                            trace("masukkan
"+tmpNextNode.id+" ke openList\n");
                                            //masukkan node skarang ke
dalam list
                                            //nilai di assign ke node
ini
                                            tmpNextNode.f = f;
//masukkan f,g,h, dan node sebelumnya;
                                            tmpNextNode.g = g;
                                            tmpNextNode.h = h;
                                            tmpNextNode.parent =
curNode; //di link dgn node sebelumnya (u/ backtrack)

openList.push(tmpNextNode); // masukan ke openList
                                        }
                                        else // kalau di dalam list,
tmpNextNode nya -> nodeMap[x][y], dah ada f,g,h
                                        {
                                            trace(tmpNextNode.id+" ada
di open/close List\n");
                                            if(tmpNextNode.f > f)
//kalau ada calonNode yg sudah di dalam list,
                                            {
//sudah punya nilai f -> dibandingkan dgn f yg baru
                                                trace("node next
("+tmpNextNode.id+" punya f = "+tmpNextNode.f+" >  f = "+f);
                                                //klw teryata node
                                                tmpNextNode.f = f;
                                                tmpNextNode.g = g;
                                                tmpNextNode.h = h;
                                                tmpNextNode.parent =
curNode;
                                            }
```

```
                                    } //if(!(bIsOnCloseList ||
bIsOnOpenList)) ...
                                } //if((nodeMap[tmpCol][tmpRow].id
!= curNode.id) ...
                            } //if((tmpRow > -1 && tmpRow <
nodeMap.length) ...
                        }//if(col == 0 || row == 0)
                    } //for(var row:int = -1;row < 2;row++) ...

                } //for(var col:int = -1;col < 2;col++)

                // msh dalam while(curNode.id != goalNodeID)
                //klw nextNode sudah dapat dan dimasukkan
openList, node lama masukan closeList
                trace("Ke close list !");
                closeList.push(curNode);

                //Nic Todo : pengecekan klw openList / closeList
error, mis openList kosong -> return
                if(openList.length == 0)
                    return shortPath;


                    openList.sortOn('f', Array.NUMERIC); //bwt
sort array di AS3
                    //setelah di sort, nilai yg terkecil ambil
jadi curNode baru. Todo : cari fungsi di AS3

                    curNode = openList.shift();
                    trace("curNode dari openlist (next node) =
"+curNode.id+" parent = "+curNode.parent.id)

            } //while(curNode.id != goalNodeID)
            trace("\nSelesai Looping !!!\n");
            trace("xxxx==xxxxx==xxxx===xxxxx==");
            trace("x=====x===x==x===x====x====");
            trace("xxxx==x===x==xxxx=====x====");
            trace("===x==x===x==x=x======x====");
            trace("xxxx==xxxxx==x==x=====x====\n");

            // kalau sudah punya open list -> short path
            var jumKoin:int = 0;
            // if(openList.length != 0)
            {

                var node:Object;
                //dibalik, soalnya pake parent, biar pake unshift
nanti
                node = goalNode;
                shortPath.push(node);
                trace("node skarang = "+node.id+" shortPath
pertama = "+shortPath[0].id);

                var ab:int = 0;
                while(node.id != startNodeID)
                {
```

```
                    totalCoin += node.poin;

                    trace("totalCoin = "+totalCoin);

                    ab++;
                    trace("short path ke - "+ab);
                    trace("node skarang sebelum backward -
"+node.id+" parent = "+node.parent.id);
                    node = node.parent; //backward
                    trace("node skarang setelah backward -
"+node.id);
                    shortPath.unshift(node);

                    if(ab > MAP_MAX_COL * MAP_MAX_ROW)
                        break;
                }

            }

            return shortPath;
        }
    }
}
```

4. Grid.as

```
package  {
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.text.TextFieldType;

    public class Grid extends Sprite
    {
        private var sisi:Number;
        public var tipe:uint;
        public var id:Number;

        public var posX:Number;
        public var posY:Number;
        public var col:Number;
        public var row:Number;

        public var gridText:String;

        private var inputSisi:Number;
        private var inputCol:Number;
        private var inputRow:Number;
        private var inputX:Number;
        private var inputY:Number;
        private var gridType:Number;
        private var gridId:Number;

        public function Grid(inputCol, inputRow, inputX, inputY,
inputSisi, gridType, gridId):void
```

```
        {
            this.row = inputRow;
            this.col = inputCol;
            this.posX = inputX;
            this.posY = inputY;
            this.sisi = inputSisi;
            //this.warna = 0x000000;
            this.tipe = gridType;
            this.id = gridId;
            this.gridText = gridId;

            var gridColor:uint;
            switch (tipe) { //Nic 26-08-2012
                case 0 : gridColor = 0xFFFFFF; break;
                case 1 : gridColor = 0x000000; break;
                //todo : dibikin color untuk yg pindah kanan,
kanan kiri, atas, atas bawah
                case 2 : gridColor = 0x88D42A; break;

                case 3 : gridColor = 0xDE0209; break;
                case 4 : gridColor = 0x038DFF; break;
                case 5 : gridColor = 0xFBDE1D; break;
                case 6 : gridColor = 0x745EAE; break;

                case 7 : gridColor = 0x0A6701; break; //out
                case 8 : gridColor = 0x49D1FE; break; //start
                case 9 : gridColor = 0xF8B001; break; //koin
                default : gridColor = 0xC9C9C9;
                // default : gridColor = 0xFFFFFF;
            }

            fungsiGambar(gridColor);
        }

        public function fungsiGambar(warna:uint):void
        {
            //trace("Draw grid - "+gridText);
            graphics.beginFill(warna);
            graphics.drawRect(posX, posY, sisi, sisi);
            graphics.endFill();

            var gridLabelTextFormat:TextFormat = new TextFormat();
            gridLabelTextFormat.bold = true;
            gridLabelTextFormat.font = "Comic Sans MS";
            gridLabelTextFormat.size = 32;
            gridLabelTextFormat.indent = "center";

            var gridLabelText:TextField = new TextField();
            gridLabelText.text = "";
            gridLabelText.type = TextFieldType.DYNAMIC;
            gridLabelText.defaultTextFormat = gridLabelTextFormat;
            gridLabelText.x = posX;
            gridLabelText.y = posY;
            gridLabelText.width = sisi;
            gridLabelText.height = sisi;
            gridLabelText.border = true;
```

```
            gridLabelText.selectable = false;
            if(warna == 0x000000 || warna == 0x745EAE || warna ==
0x0A6701 || warna == 0xDE0209 || warna == 0x038DFF)
            {
                gridLabelText.textColor = 0xFFFFFF;
            }
            else
            {
                gridLabelText.textColor = 0x000000;
            }
            addChild(gridLabelText);

            if(tipe == 3) //right
            {
                var rightImage:MapTipeRightAnimated = new
MapTipeRightAnimated();
                rightImage.x = posX + (sisi>>1);
                rightImage.y = posY + (sisi>>1);
                addChild(rightImage);
            }

            if(tipe == 4) //left
            {
                var leftImage:MapTipeLeftAnimated = new
MapTipeLeftAnimated();
                leftImage.x = posX + (sisi>>1);
                leftImage.y = posY + (sisi>>1);
                addChild(leftImage);
            }

            if(tipe == 5) //up
            {
                var upImage:MapTipeUpAnimated = new
MapTipeUpAnimated();
                upImage.x = posX + (sisi>>1);
                upImage.y = posY + (sisi>>1);
                addChild(upImage);
            }

            if(tipe == 6) //down
            {
                var downImage:MapTipeDownAnimated = new
MapTipeDownAnimated();
                downImage.x = posX + (sisi>>1);
                downImage.y = posY + (sisi>>1);
                addChild(downImage);
            }

            if(tipe == 9)
            {
                var coinImage:AnimatedCoin = new AnimatedCoin();
                coinImage.x = posX + (sisi>>1);
                coinImage.y = posY + (sisi>>1);
                addChild(coinImage);
            }
```

```
            if(tipe == 7)
            {
                var targetImage:AnimatedTarget = new
AnimatedTarget();
                targetImage.x = posX + (sisi>>1);
                targetImage.y = posY + (sisi>>1);
                addChild(targetImage);
            }

            if(tipe == 2)
            {
                var goalImage:AnimatedGoal = new AnimatedGoal();
                goalImage.x = posX + (sisi>>1);
                goalImage.y = posY + (sisi>>1);
                addChild(goalImage);
            }

        }

        public function fungsiRedraw():void
        {
            //remove semua, lalu gambar lagi
            //trace("numChild redraw "+numChildren);
            for(var child=numChildren-1;child>=0;child--)
            {
                //trace("remove - "+child+" =
"+getChildAt(child).toString());
                removeChildAt(child);
            }

            var warna:uint;
            switch (tipe) { //Nic 26-08-2012
                case 0 : warna = 0xFFFFFF; break;
                case 1 : warna = 0x000000; break;
                case 2 : warna = 0x88D42A; break; //goal

                //todo : dibikin color untuk yg pindah kanan,
kanan kiri, atas, atas bawah
                case 3 : warna = 0xDE0209; break;
                case 4 : warna = 0x038DFF; break;
                case 5 : warna = 0xFBDE1D; break;
                case 6 : warna = 0x745EAE; break;

                case 7 : warna = 0x0A6701; break; //out
                case 8 : warna = 0x49D1FE; break; //start
                case 9 : warna = 0xF8B001; break; //koin
                default : warna = 0xC9C9C9;
                // default : warna = 0xFFFFFF;
            }

            // trace("REDRAW "+gridText+" = "+tipe+" !!!");
            graphics.beginFill(warna);
            graphics.drawRect(posX, posY, sisi, sisi);
            graphics.endFill();

            var gridLabelTextFormat:TextFormat = new TextFormat();
```

```
            gridLabelTextFormat.bold = true;
            gridLabelTextFormat.font = "Comic Sans MS";
            gridLabelTextFormat.size = 32;
            gridLabelTextFormat.indent = "center";

            var gridLabelText:TextField = new TextField();
            gridLabelText.text = "";
            gridLabelText.type = TextFieldType.DYNAMIC;
            gridLabelText.defaultTextFormat = gridLabelTextFormat;
            gridLabelText.x = posX;
            gridLabelText.y = posY;
            gridLabelText.width = sisi;
            gridLabelText.height = sisi;
            gridLabelText.border = true;
            gridLabelText.selectable = false;
            if(warna == 0x000000 || warna == 0x745EAE || warna ==
0x0A6701 || warna == 0xDE0209 || warna == 0x038DFF)
            {
                gridLabelText.textColor = 0xFFFFFF;
            }
            else
            {
                gridLabelText.textColor = 0x000000;
            }
            addChild(gridLabelText);

            if(tipe == 3) //right
            {
                var rightImage:MapTipeRightAnimated = new
MapTipeRightAnimated();
                rightImage.x = posX + (sisi>>1);
                rightImage.y = posY + (sisi>>1);
                addChild(rightImage);
            }

            if(tipe == 4) //left
            {
                var leftImage:MapTipeLeftAnimated = new
MapTipeLeftAnimated();
                leftImage.x = posX + (sisi>>1);
                leftImage.y = posY + (sisi>>1);
                addChild(leftImage);
            }

            if(tipe == 5) //up
            {
                var upImage:MapTipeUpAnimated = new
MapTipeUpAnimated();
                upImage.x = posX + (sisi>>1);
                upImage.y = posY + (sisi>>1);
                addChild(upImage);
            }

            if(tipe == 6) //down
            {
```

```
                var downImage:MapTipeDownAnimated = new
MapTipeDownAnimated();
                downImage.x = posX + (sisi>>1);
                downImage.y = posY + (sisi>>1);
                addChild(downImage);
            }

            if(tipe == 2)
            {

                var goalImage:AnimatedGoal = new AnimatedGoal();
                goalImage.name = "AnimatedGoal";
                goalImage.x = posX + (sisi>>1);
                goalImage.y = posY + (sisi>>1);
                addChild(goalImage);
            }

            if(tipe == 7)
            {
                var targetImage:AnimatedTarget = new
AnimatedTarget();
                targetImage.name = "AnimatedTarget";
                targetImage.x = posX + (sisi>>1);
                targetImage.y = posY + (sisi>>1);
                addChild(targetImage);
            }
        }

    }
}
```

5. XmlLoader.as

```
package {
    import flash.events.EventDispatcher;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import flash.events.Event;
    import flash.events.IOErrorEvent;

    public class XmlLoader extends EventDispatcher
    {
        public var dataXML:XML;
        private var urlReqXML:URLRequest;
        private var urlLoadXML:URLLoader;
        private var fileNameXML:String;

        public function XmlLoader(url:String)
        {
            dataXML = new XML();
            fileNameXML = url;
            urlReqXML = new URLRequest(fileNameXML);
            urlLoadXML = new URLLoader(urlReqXML);
```

```
            urlLoadXML.addEventListener(Event.COMPLETE,
loadCompleteXML); //bwt gantiin urlLoadXML.load
            urlLoadXML.addEventListener(IOErrorEvent.IO_ERROR,
loadError);
        }

        private function loadCompleteXML(evt:Event):void
        {
            dataXML = XML(evt.target.data);
            dispatchEvent(new Event(Event.COMPLETE));
            trace(" ... DATA LOADED ... ");
        }

        private function loadError(evt:IOErrorEvent):void
        {
            trace("Error loading XML "+evt.text);
        }
    }
}
```

LAMPIRAN B

GAME MAP

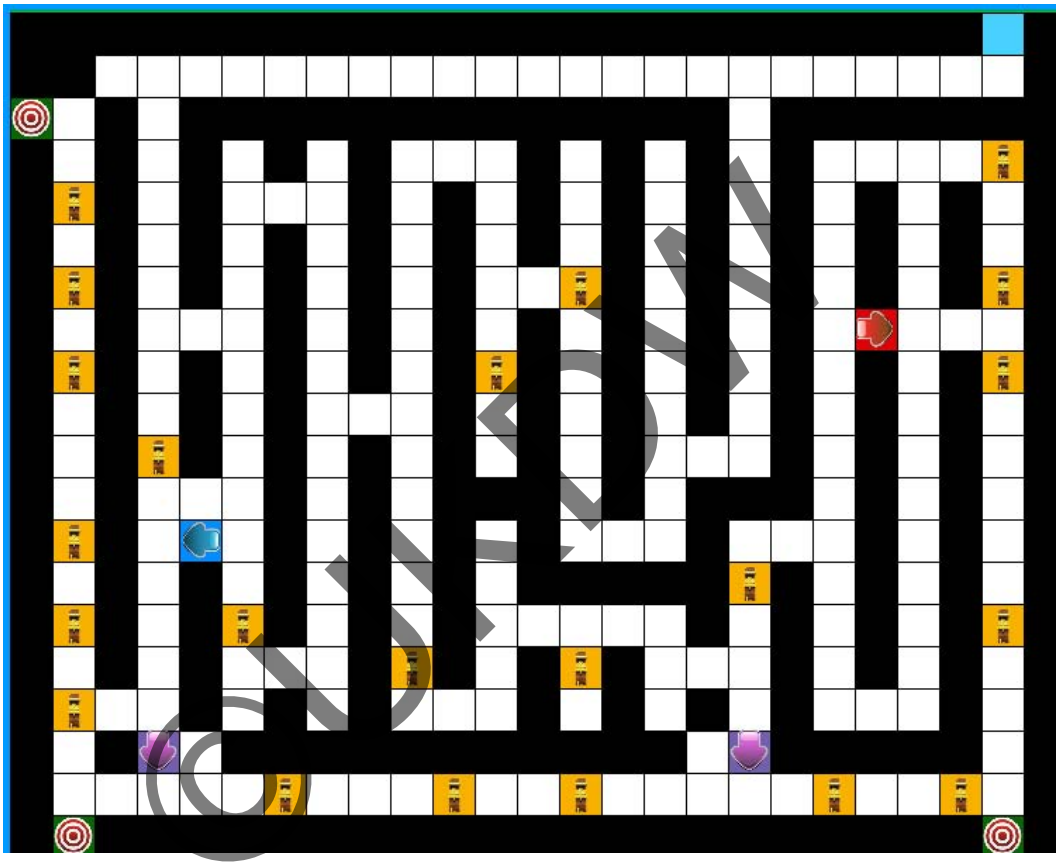
1. Map 1

```
<level id="1">
    <langkah>
        <biasa>60</biasa>
        <multi>70</multi>
    </langkah>
    <koin>
        <biasa>0</biasa>
        <multi>300</multi>
    </koin>
    <tilemap>
<row>1,8,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1</row>
<row>1,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,9,1,1</row>
<row>1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,1,1,0,1,1,0,1,1</row>
<row>1,1,1,0,1,1,1,1,1,5,1,1,1,1,1,1,0,1,1,0,1,1,0,1,1</row>
<row>1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,1,0,1,1</row>
<row>1,1,1,0,0,0,0,0,1,1,1,1,1,1,1,0,0,1,5,1,1,1,0,1,1</row>
<row>1,1,3,0,1,1,1,0,1,1,1,1,1,9,0,0,0,4,0,0,0,0,0,1,1</row>
<row>1,1,1,0,1,1,1,0,1,1,1,1,1,1,0,1,1,0,0,1,1,0,1,1,1,1</row>
<row>1,1,1,0,0,0,1,0,1,1,1,1,1,1,0,1,1,0,1,1,1,0,1,1,1,1</row>
<row>1,1,1,1,1,1,0,1,0,1,1,1,1,1,1,0,1,1,9,1,1,1,0,1,0,1,1</row>
<row>1,1,1,1,1,1,0,1,0,9,1,0,0,0,0,0,0,3,0,0,0,1,0,1,0,1,1</row>
<row>1,1,1,1,1,1,0,1,1,0,0,0,1,0,1,1,1,1,1,1,0,1,0,3,0,1,1</row>
<row>1,1,1,1,1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,0,1,0,6,0,1,1</row>
<row>1,1,1,1,1,1,0,0,1,0,4,1,1,0,1,1,1,1,1,1,0,1,0,0,0,1,1</row>
<row>1,1,1,1,1,1,1,0,1,0,1,1,1,0,9,1,0,0,0,0,1,0,3,0,1,1</row>
<row>1,1,0,0,0,0,0,6,0,1,1,1,5,0,1,9,1,1,3,0,1,0,0,1,1</row>
<row>1,1,0,1,1,1,1,1,0,0,0,0,0,0,0,9,7,9,0,0,1,1,0,1,1,1</row>
<row>1,1,0,9,0,9,0,0,7,5,1,1,1,1,1,0,4,1,1,1,1,0,1,1,1</row>
<row>1,1,1,1,0,1,1,0,0,0,0,0,0,9,0,0,0,0,0,0,0,9,1,1,1</row>
<row>1,1,1,1,7,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1</row>
    </tilemap>
</level>
```

2. Map 2

```
<level id="2">
    <langkah>
        <biasa>100</biasa>
        <multi>120</multi>
    </langkah>
    <koin>
        <biasa>700</biasa>
        <multi>900</multi>
    </koin>
    <tilemap>
<row>1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,8,1</row>
<row>1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1</row>
<row>7,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1</row>
<row>1,0,1,0,1,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,0,0,0,9,1</row>
<row>1,9,1,0,1,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1</row>
<row>1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1</row>
<row>1,9,1,0,1,0,1,0,1,0,1,0,0,9,1,0,1,0,1,0,1,0,1,9,1</row>
<row>1,0,1,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,3,0,0,0,1</row>
<row>1,9,1,0,1,0,1,0,1,0,1,9,1,0,1,0,1,0,1,0,1,0,1,9,1</row>
<row>1,0,1,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1</row>
<row>1,0,1,9,1,0,1,0,1,0,1,0,1,0,1,0,0,0,1,0,1,0,1,0,1</row>
<row>1,0,1,0,0,0,1,0,1,0,1,1,1,0,1,0,1,1,1,0,1,0,1,0,1</row>
```

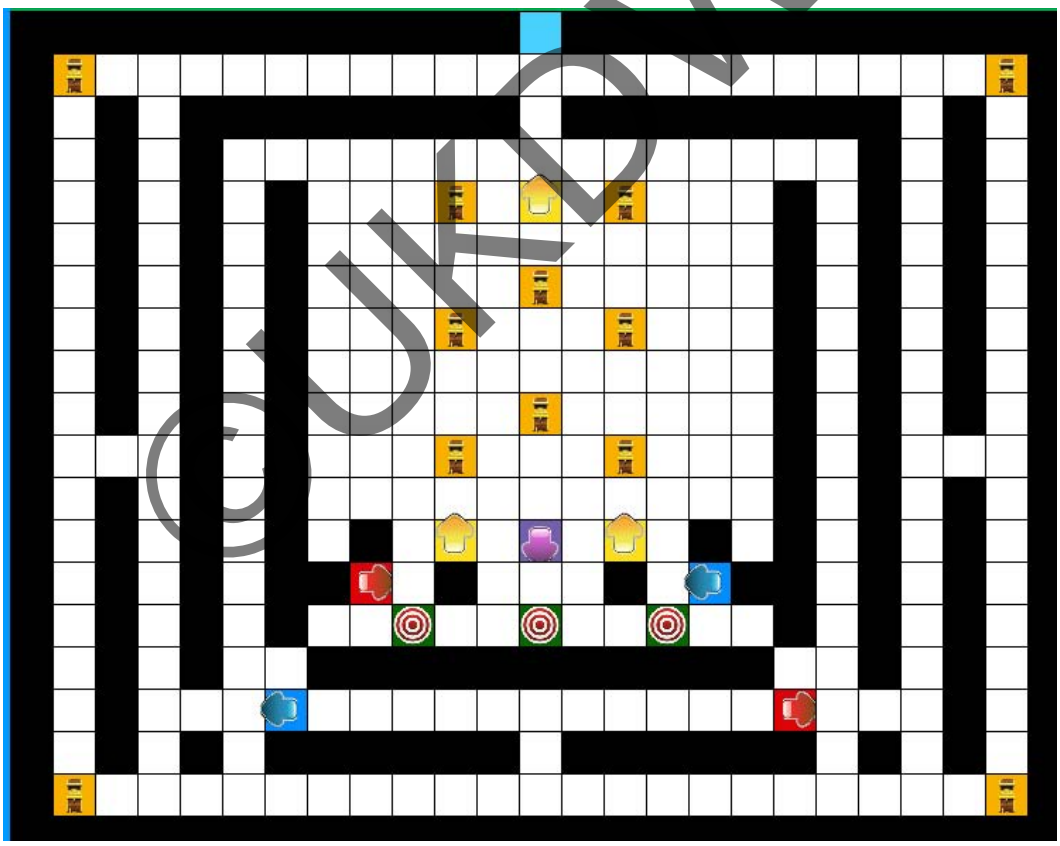
```
<row>1,9,1,0,4,0,1,0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,1,0,1</row>
<row>1,0,1,0,1,0,1,0,1,0,1,0,1,1,1,1,1,9,1,0,1,0,1,0,1</row>
<row>1,9,1,0,1,9,1,0,1,0,1,0,0,0,0,0,1,0,1,0,1,0,1,9,1</row>
<row>1,0,1,0,1,0,0,0,1,9,1,0,1,9,1,0,0,0,1,0,1,0,1,0,1</row>
<row>1,9,0,0,1,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,0,0,1,0,1</row>
<row>1,0,1,6,0,1,1,1,1,1,1,1,1,1,1,1,1,0,6,1,1,1,1,1,0,1</row>
<row>1,0,0,0,0,0,9,0,0,0,9,0,0,9,0,0,0,0,0,9,0,0,9,0,1</row>
<row>1,7,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,7,1</row>
    </tilemap>
</level>
```



3. Map 3

```
<level id="3">
    <langkah>
        <biasa>100</biasa>
        <multi>120</multi>
    </langkah>
    <koin>
        <biasa>700</biasa>
        <multi>900</multi>
    </koin>
    <tilemap>
<row>1,1,1,1,1,1,1,1,1,1,1,1,8,1,1,1,1,1,1,1,1,1,1,1,1</row>
<row>1,9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,9,1</row>
```

```
<row>1,0,1,0,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,1,0,1</row>
<row>1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1</row>
<row>1,0,1,0,1,0,1,0,0,0,9,0,5,0,9,0,0,0,1,0,1,0,1,0,1</row>
<row>1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1</row>
<row>1,0,1,0,1,0,1,0,0,0,0,0,9,0,0,0,0,0,1,0,1,0,1,0,1</row>
<row>1,0,1,0,1,0,1,0,0,0,9,0,0,0,9,0,0,0,1,0,1,0,1,0,1</row>
<row>1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1</row>
<row>1,0,1,0,1,0,1,0,0,0,0,0,9,0,0,0,0,0,1,0,1,0,1,0,1</row>
<row>1,0,0,0,1,0,1,0,0,0,9,0,0,0,9,0,0,0,1,0,1,0,0,0,1</row>
<row>1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1</row>
<row>1,0,1,0,1,0,1,0,1,0,5,0,6,0,5,0,1,0,1,0,1,0,1,0,1</row>
<row>1,0,1,0,1,0,1,1,3,0,1,0,0,0,1,0,4,1,1,0,1,0,1,0,1</row>
<row>1,0,1,0,1,0,1,0,0,7,0,0,7,0,0,7,0,0,1,0,1,0,1,0,1</row>
<row>1,0,1,0,1,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,1,0,1,0,1</row>
<row>1,0,1,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,1,0,1</row>
<row>1,0,1,0,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,0,1,0,1</row>
<row>1,9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,9,1</row>
<row>1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1</row>
    </tilemap>
</level>
```

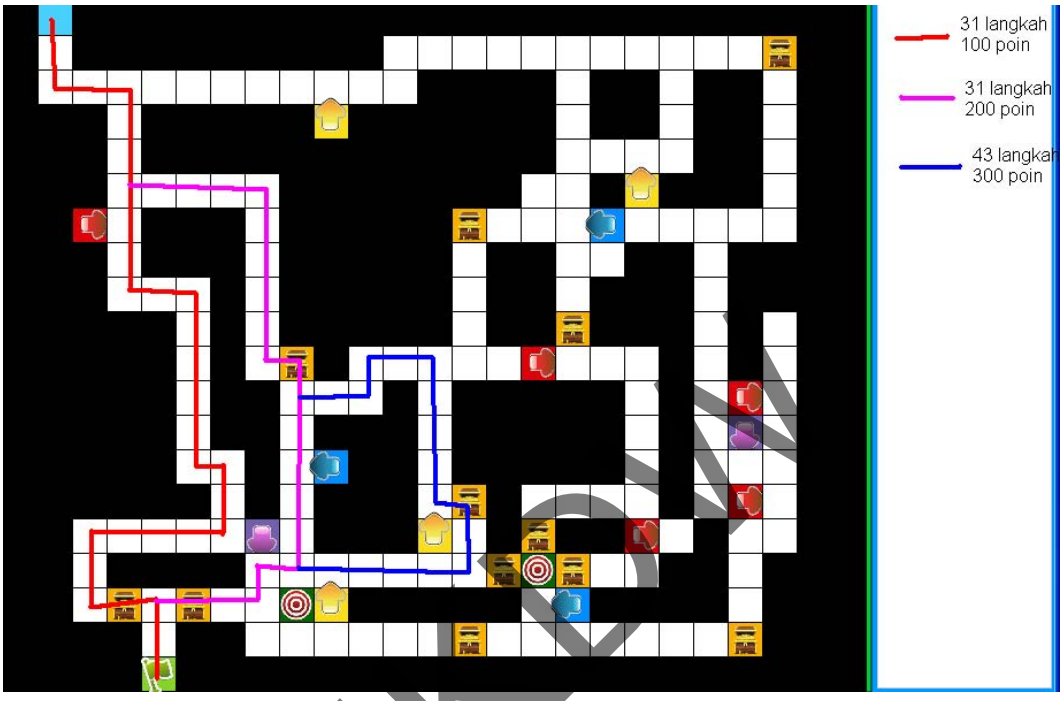


LAMPIRAN C

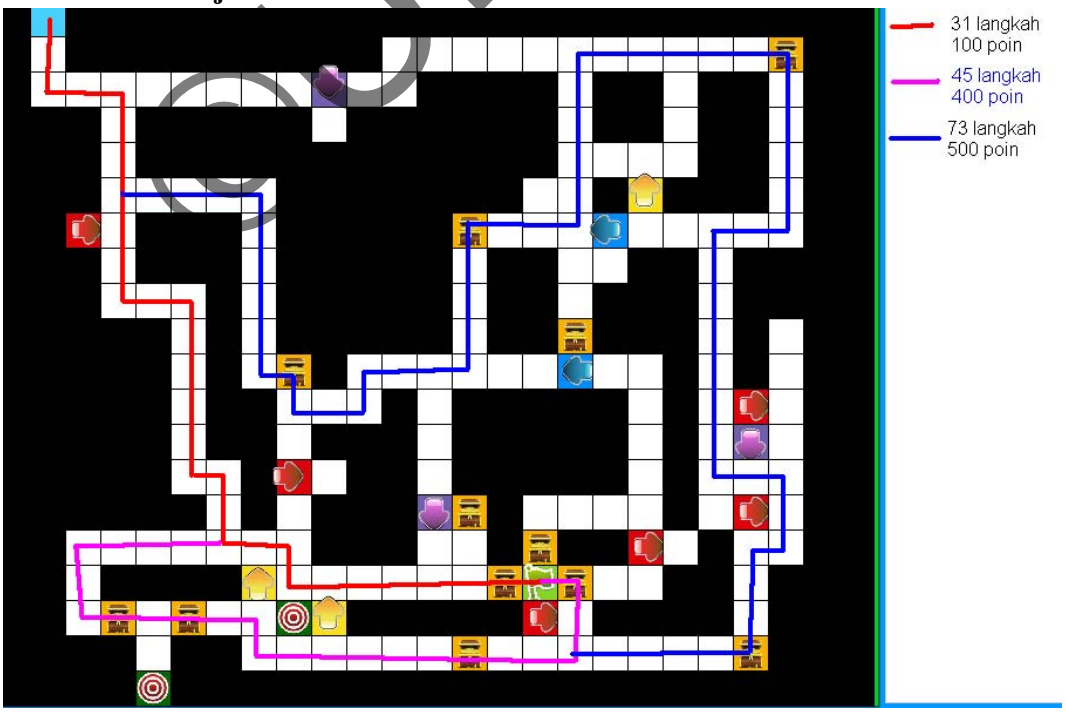
PERHITUNGAN BATASAN MAKSIMUM *MAP MAZE*

1. Map 1

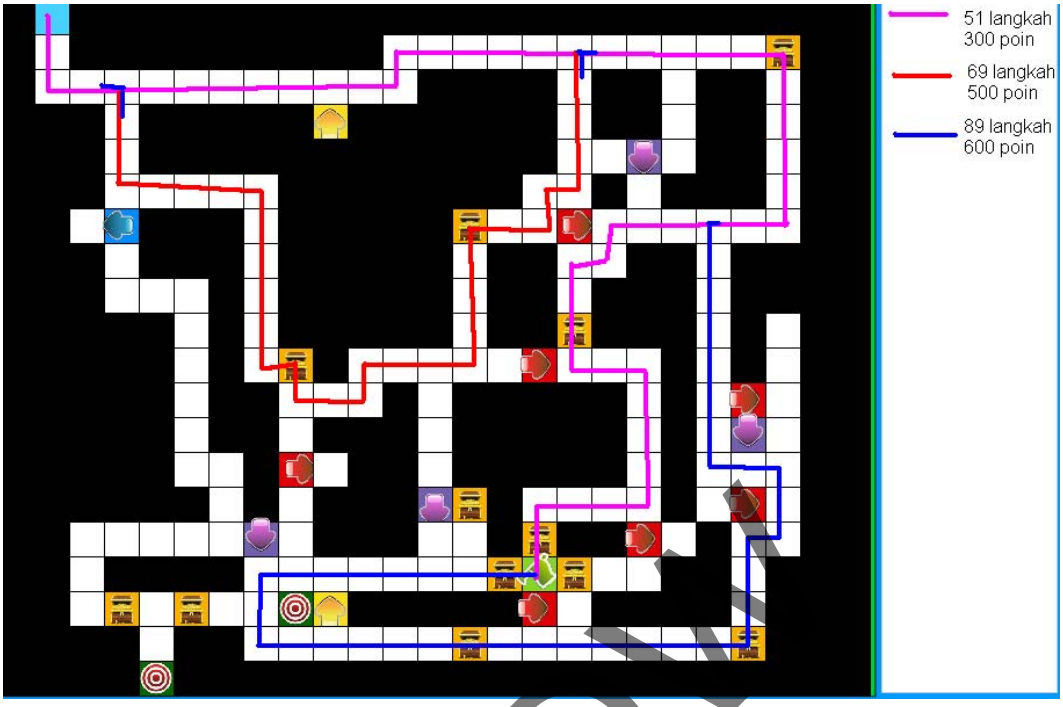
a. Uji coba 1



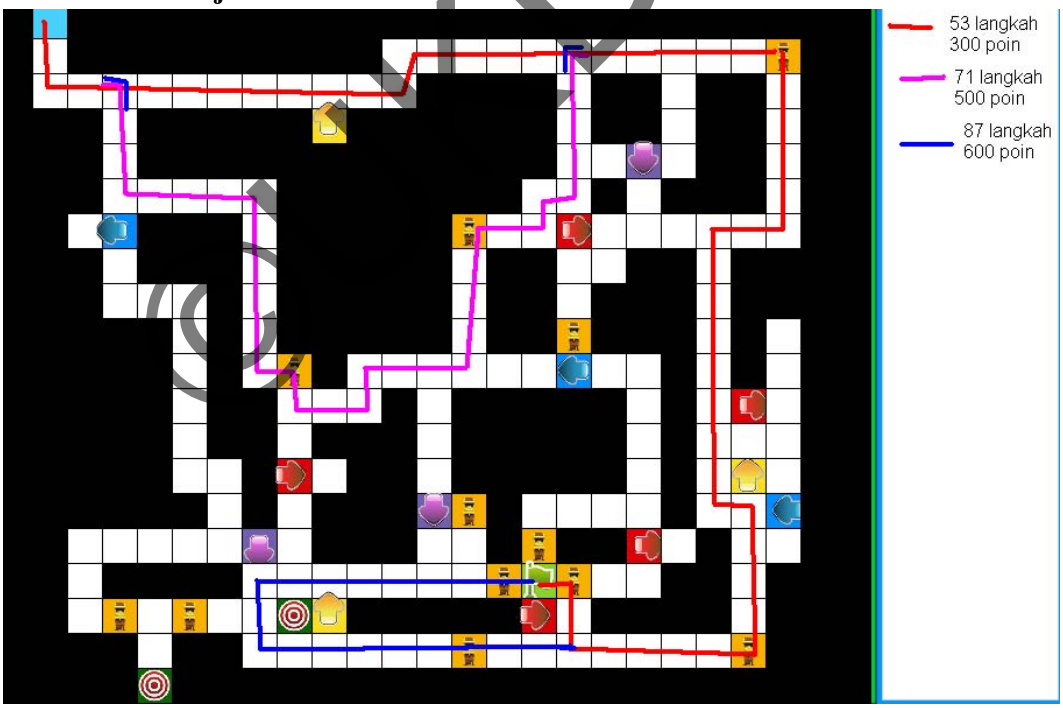
b. Uji coba 2



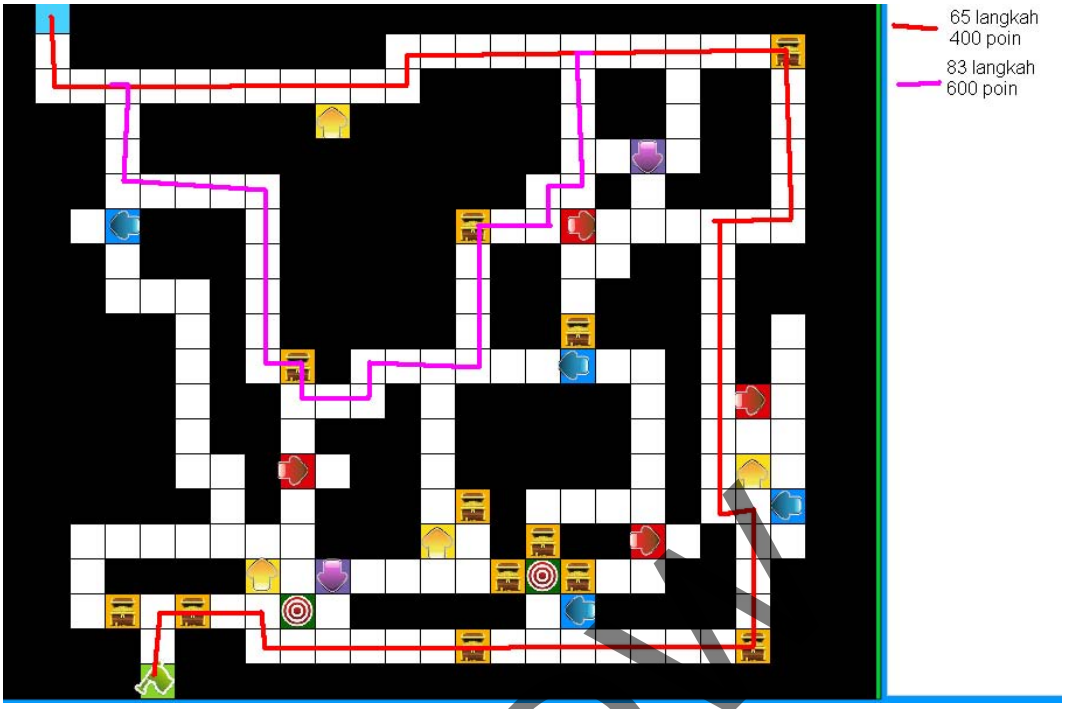
c. **Uji coba 3**



d. **Uji coba 4**

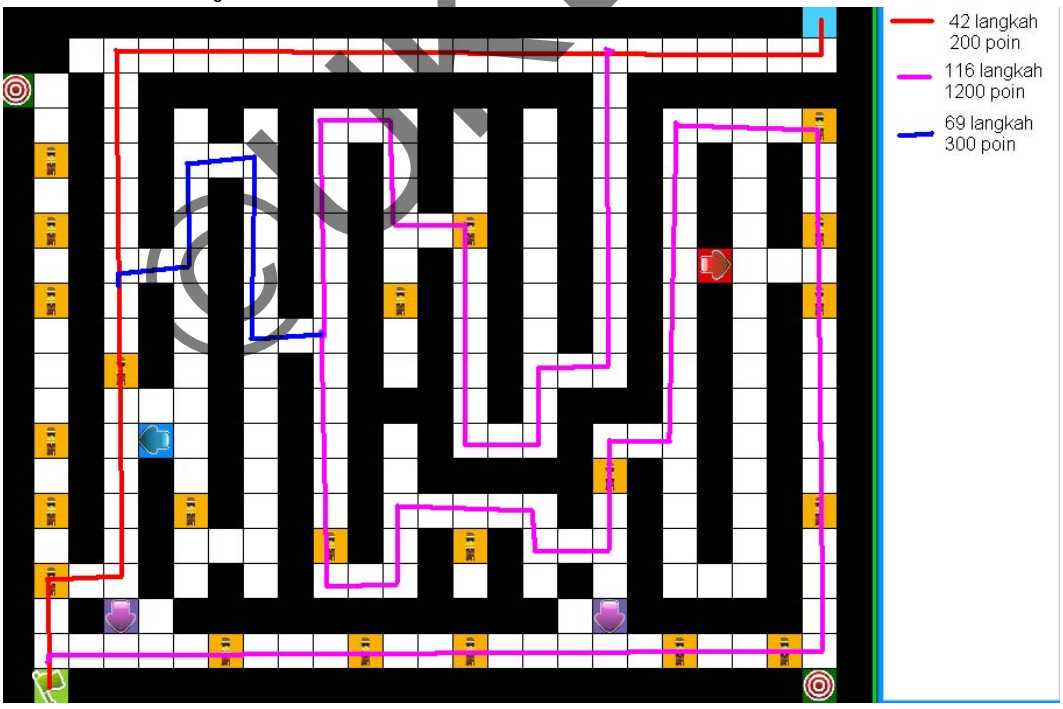


e. **Uji coba 5**

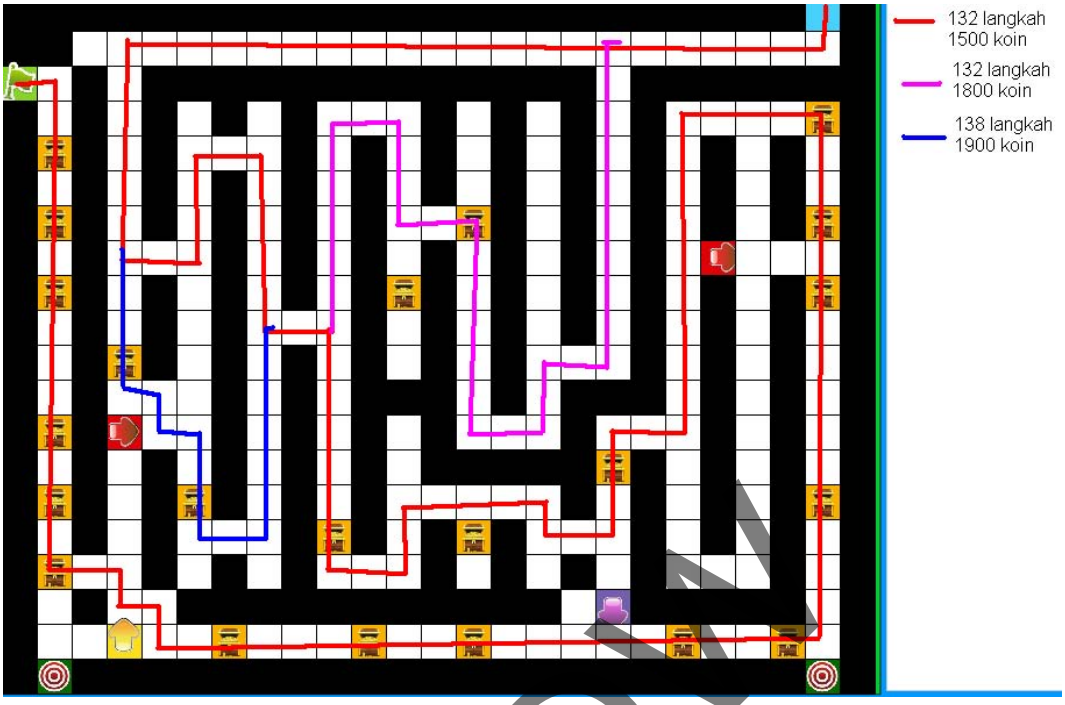


2. **Map 2**

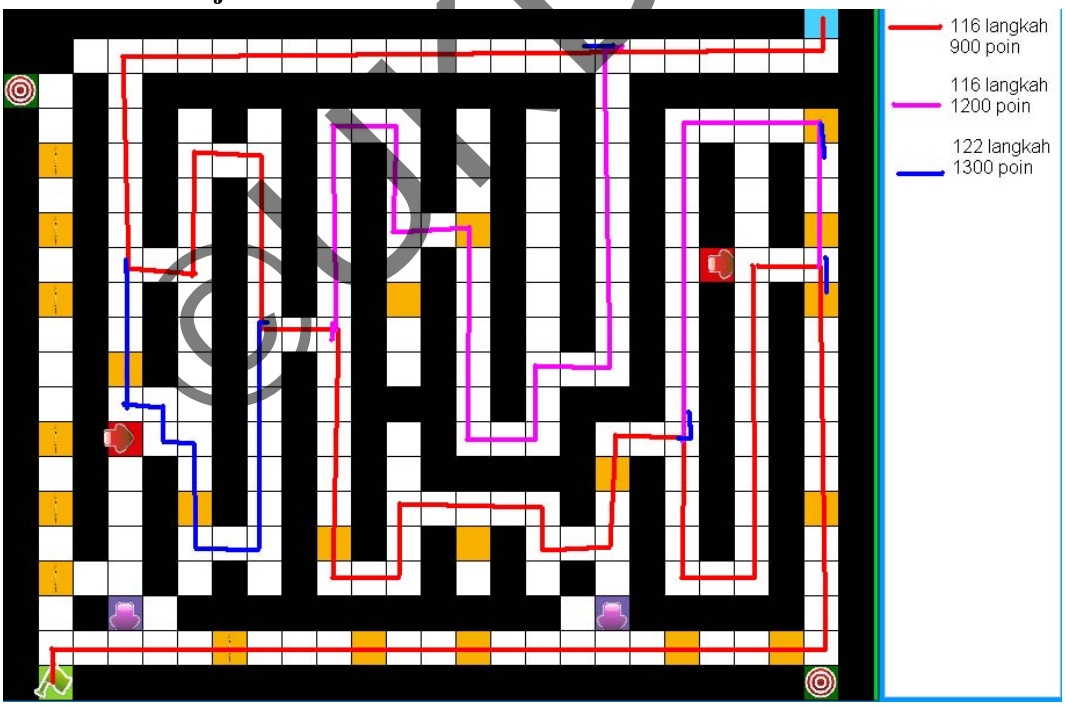
a. **Uji coba 1**



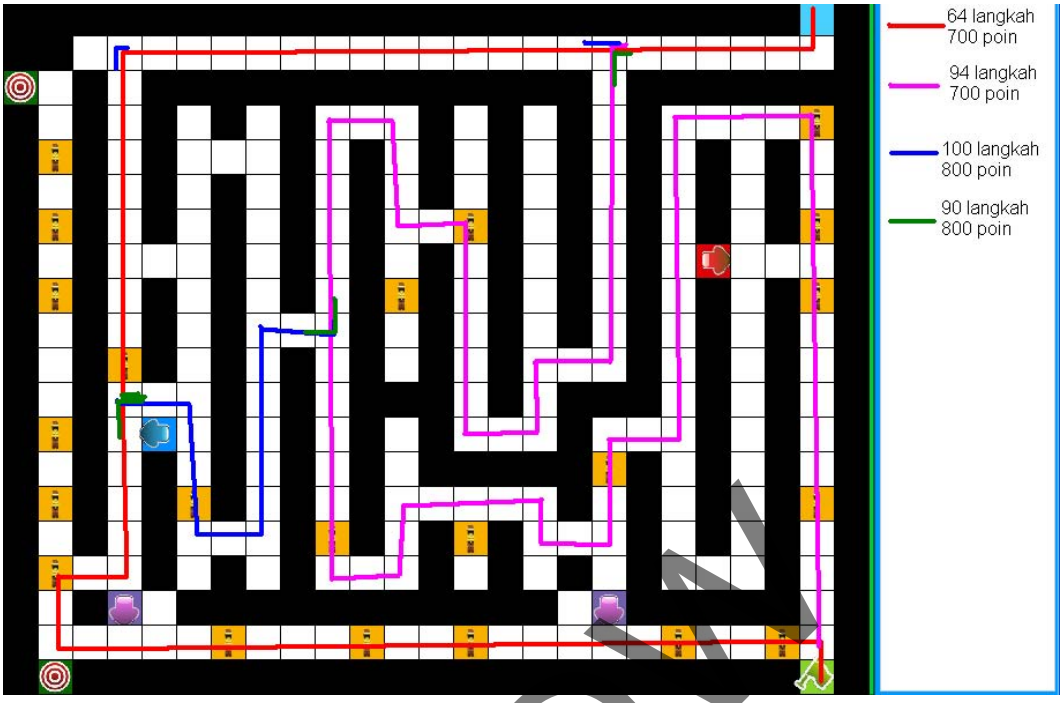
b. Uji coba 2



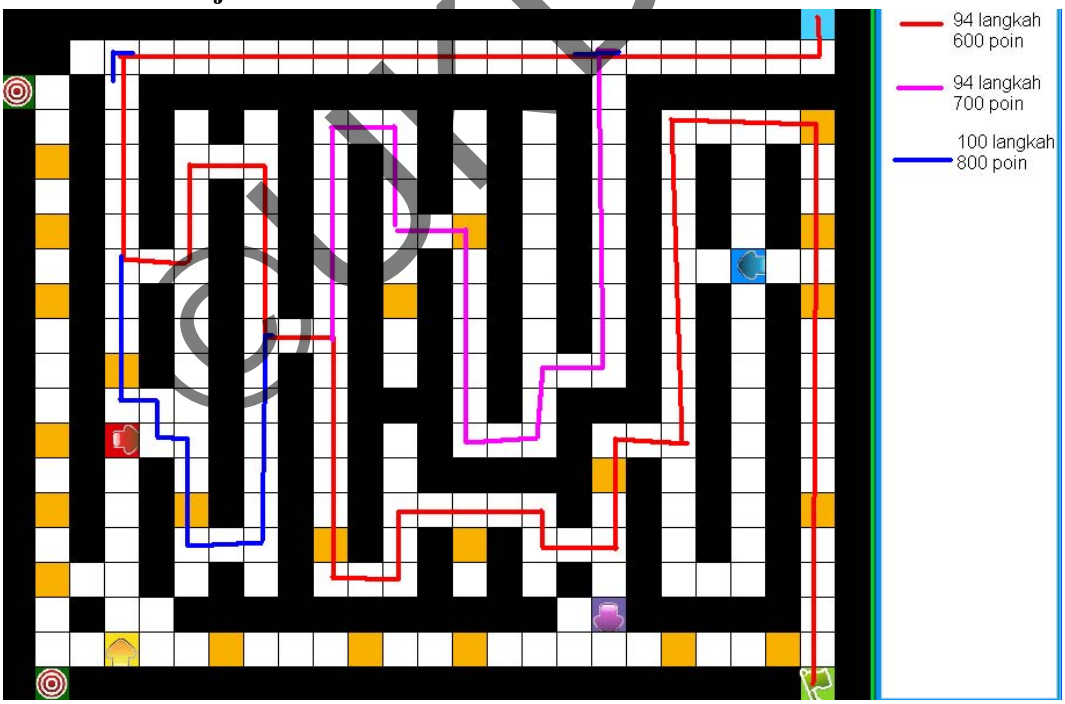
c. Uji coba 3



d. Uji coba 4

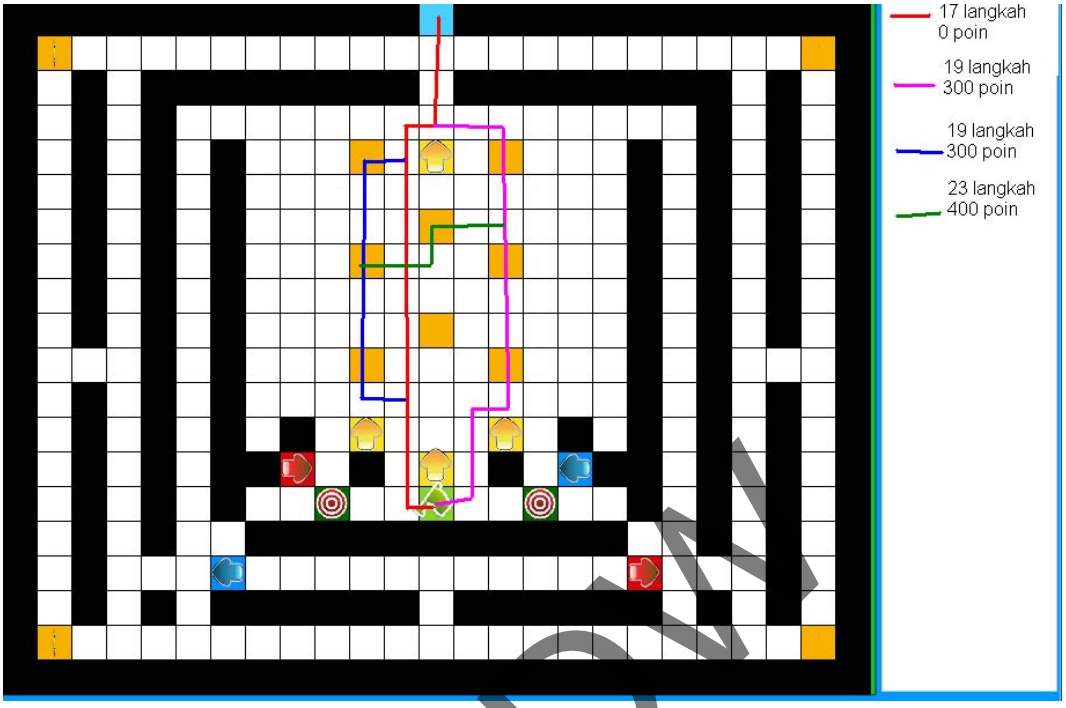


e. Uji coba 5

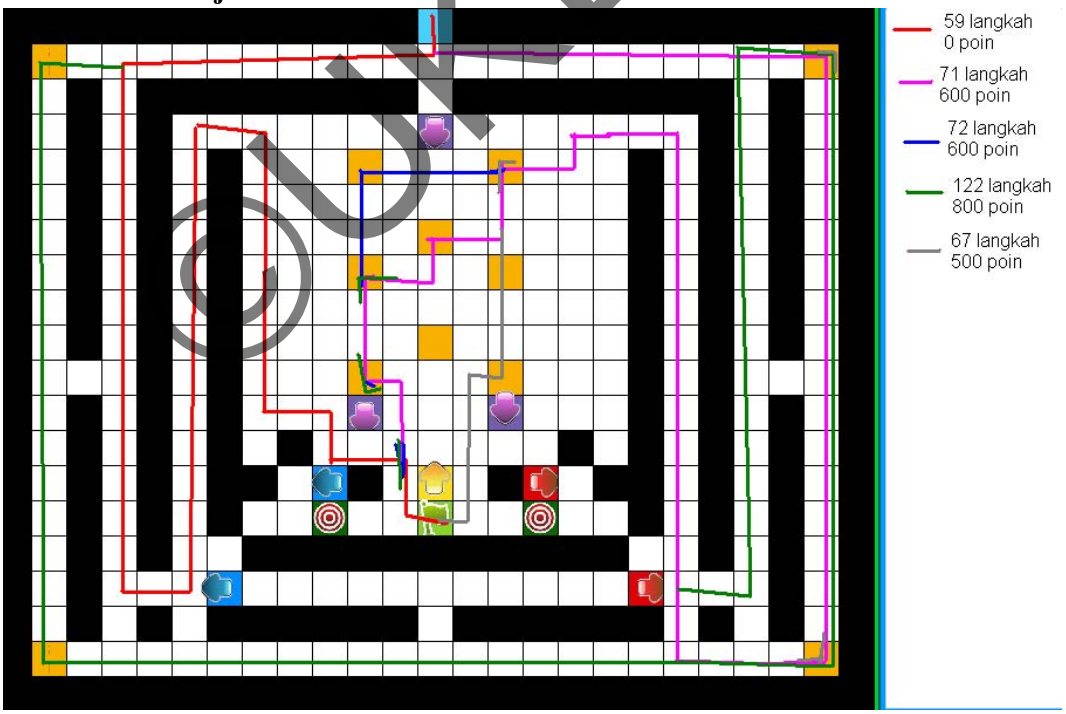


3. **Map 3**

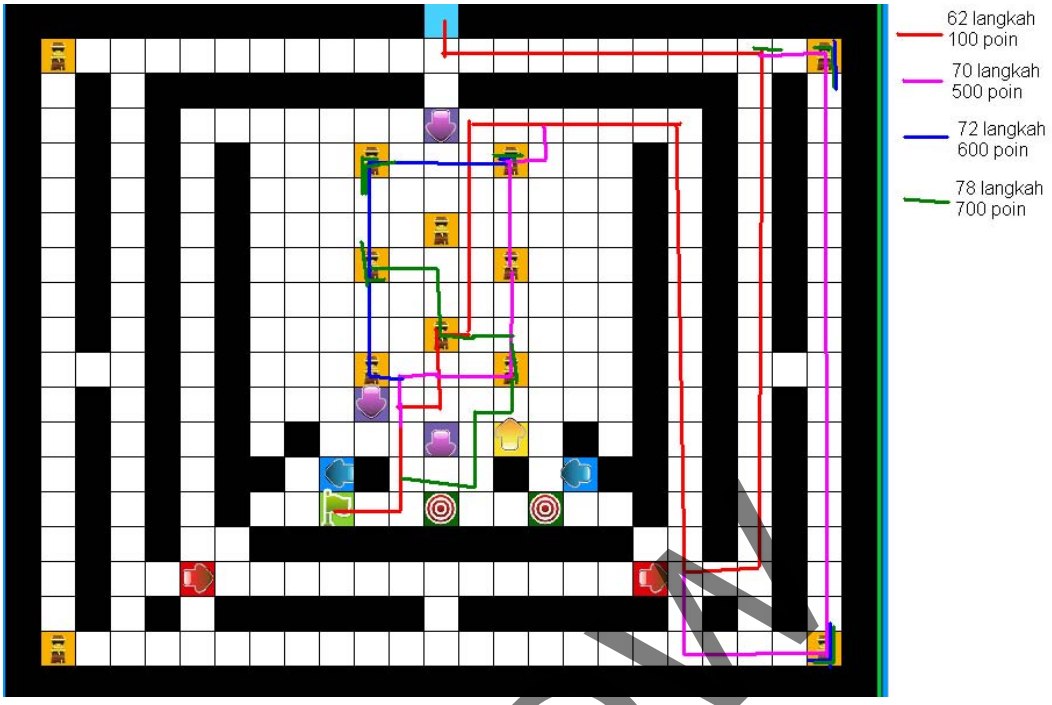
a. **Uji coba 1**



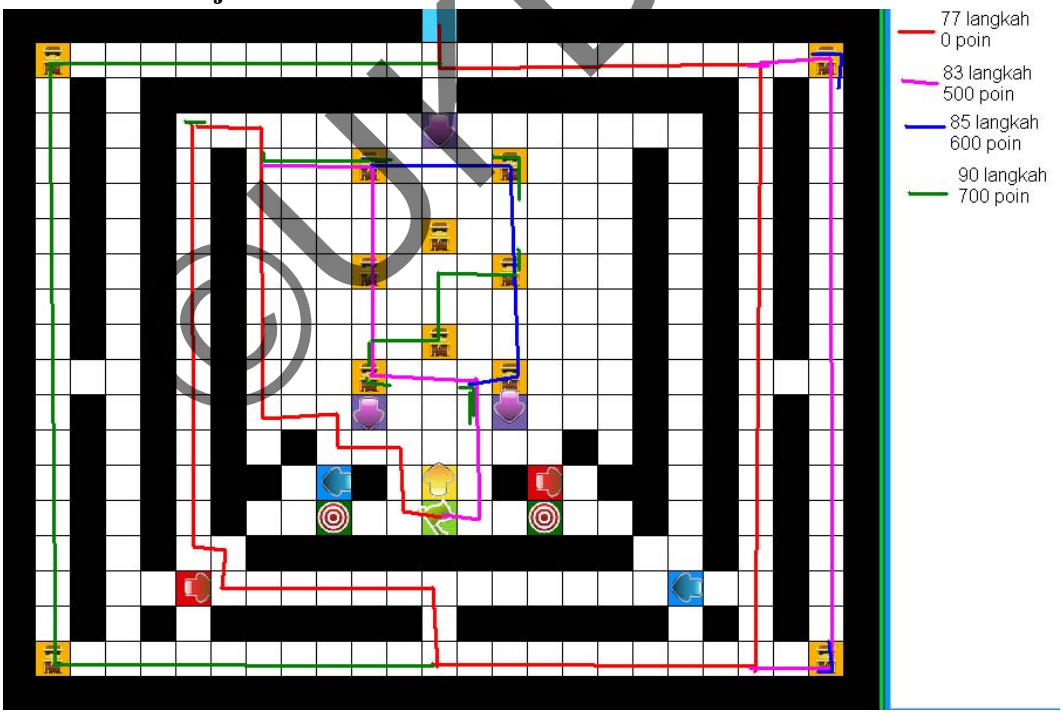
b. **Uji coba 2**



c. **Uji coba 3**



d. **Uji coba 4**



e. Uji coba 5