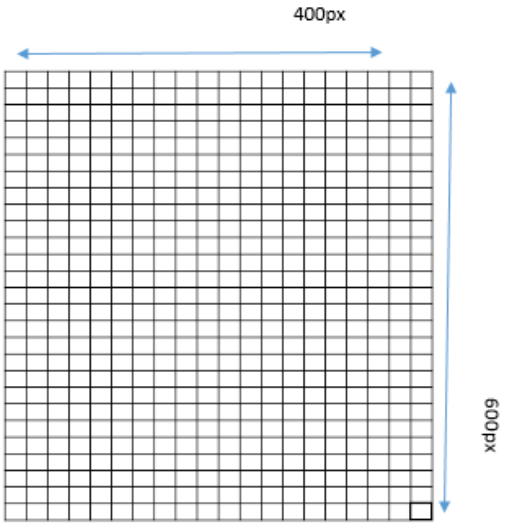
BAB 3
ANALISIS DAN KEBUTUHAN ALGORITMA

1.1. Analisis Masalah

Masalah yang diangkat dalam penulisan ini adalah penerapan algoritma WHCA* dengan *multi-agent* yang akan disimulasikan pada aplikasi dengan lingkungan yang diadaptasi dari gudang PT. INDOMARCO PRISTAMA. Pada gudang PT. INDOMARCO PRISTAMA terdapat banyak ***agent*** yang bertugas untuk menyusun dan mengantarkan barang dimana setiap ***agent*** memungkinkan terjadi tabrakan dalam mencapai tujuannya. Untuk itu dibutuhkan sebuah algoritma pencarian jalur terpendek yang dapat menghindari tabrakan antar ***agent***. Algoritma WHCA* merupakan pengembangan dari algoritma A* yang dapat mengatasi permasalahan pencarian jalur terpendek dengan *multi-agent* dengan menggunakan reservasi tabel sebagai tempat penyimpanan informasi *path* yang akan dilalui oleh ***agent*** hingga ke tujuan. Dengan adanya reservasi tabel, setiap ***agent*** akan kooperatif memberikan informasi jalur yang akan dilalui. Disisi lain algoritma WHCA* dibutuhkan kedalaman dalam pencarian dan jumlah path yang akan di reservasi, oleh karena itu yang akan diteliti untuk mengukur efektifitas penggunaaan algoritma WHCA* adalah jumlah kedalaman ketika proses pencarian dan waktu eksekusi untuk pencarian solusi.

1.2. Analisis Masukan

Analisis masukan menjelaskan data masukan yang akan diperlukan dalam penerapan algoritma WHCA* pada webots. Analisis masukan yang dibutuhkan pada algoritma WHCA* yaitu posisi awal ***agent***, posisi akhir ***agent***, posisi halangan dan kedalaman pencarian. Sebelum menghitung menggunakan algoritma WCHA* akan dihitung dulu biaya (*cost*) sebenarnya yang terdapat pada jalur itu sendiri. *Map* yang digunakan pada simulasi ini dengan skala *pixel* yang berukuran lebar 400 *pixel* dan tinggi 600 *pixel.* Satu *layer* tersebut disimpan dalam grid (suatu kotak-kotak kecil) yang dinamis dimana *user* dapat menentukan jumlah gridnya. Setiap grid memiliki ukuran dengan lebar 20 *pixel* dan tinggi 20 *pixel*, contoh *map* dengan ukuran lebar 20 *pixel* dan tinggi 20 *pixel* dapat dilihat pada gambar 3.1 berikut:

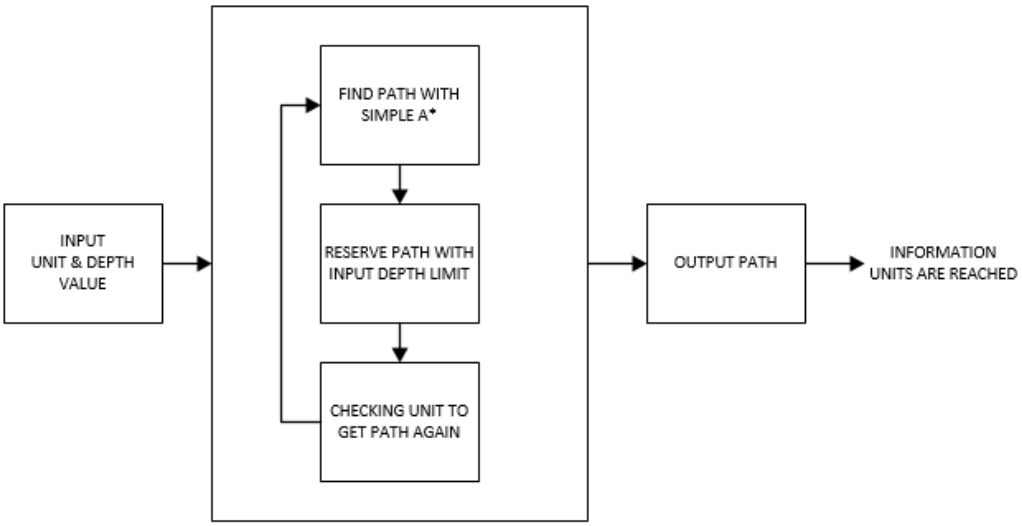


Gambar 3.1. Keterangan ukuran layar

Untuk mempermudah perhitungan maka ada beberapa nilai yang akan disederhanakan. Pertama nilai simpul yang tadinya 20x20 pixel diubah menjadi 1x1, artinya jarak simpul 1 ke simpul lainnya adalah 1. Jadi setiap simpul memiliki *cost* yang sama yaitu 1 secara horizontal maupun vertikal. Pergerakan yang dilakukan hanya bisa diagonal dan vertikal .

1.3. Analisis Algoritma

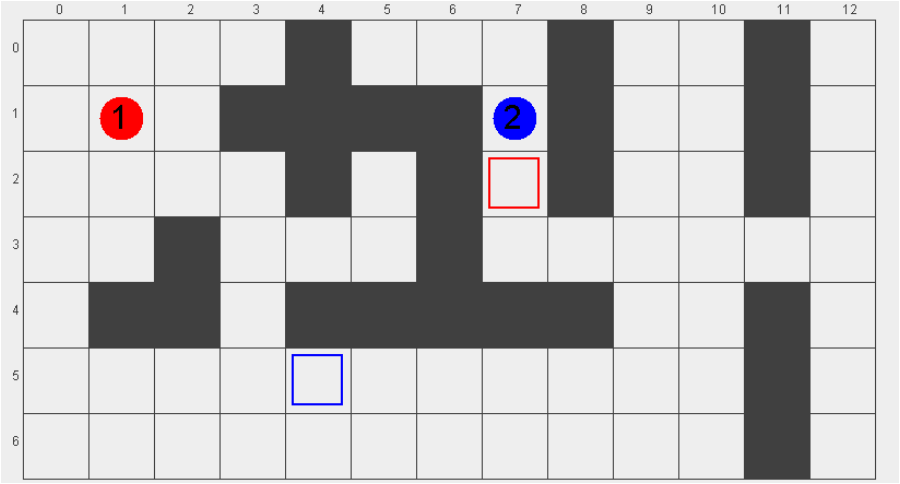
3.3.1. Block Diagram Algoritma WHCA



Gambar 3.2. Block Diagram algoritma WHCA*

3.3.2. Contoh kasus

Untuk lebih memahami algoritma WHCA*, bisa dilihat dari contoh kasus seperti terlihat pada Gambar 3.3. Diasumsikan dalam simulasi ada 2 mobile robot (warna biru dan merah) yang memiliki goal masing – masing, selain itu juga ada tembok penghalang.



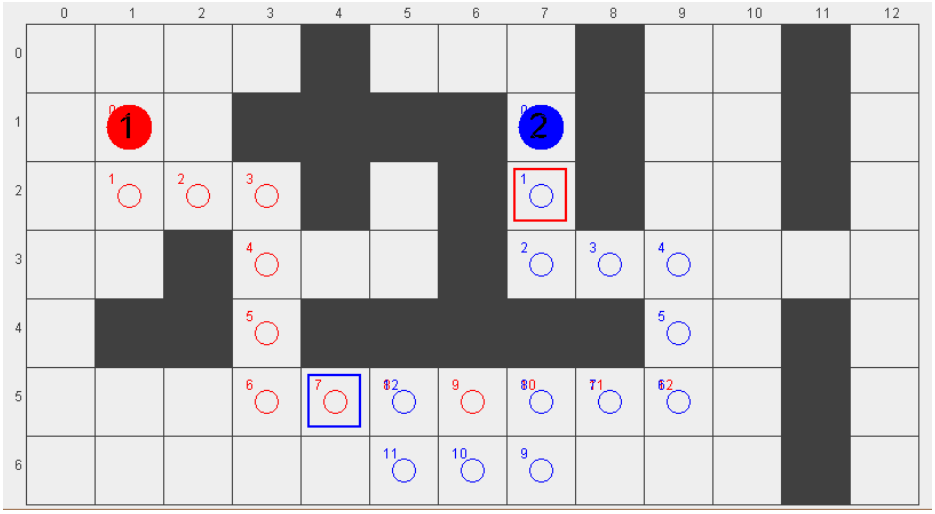
Gambar 3.3. Kondisi awal pencarian WHCA*

Keterangan:



Proses pertama yang dilakukan adalah setiap mencari jalur terpendek dari node awal, penyimpanan node menggunakan tiga dimensi ruang-waktu (x,y,time).

Setelah melakuan perhitungan awal, dan path awal telah ditemukan, setiap agen pindah ke tujuan node terdekat, seperti gambar 3.4 berikut ini:



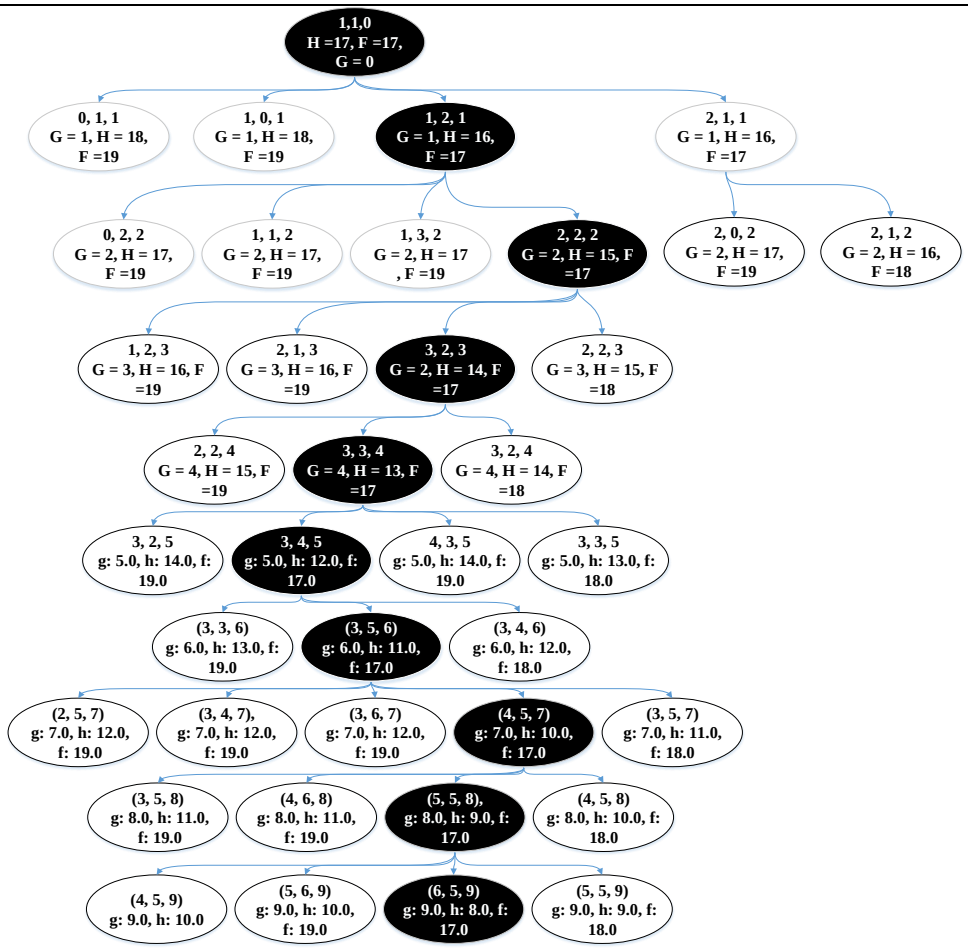
Gambar 3.4. Kondisi Awal dengan path yang akan dilalui

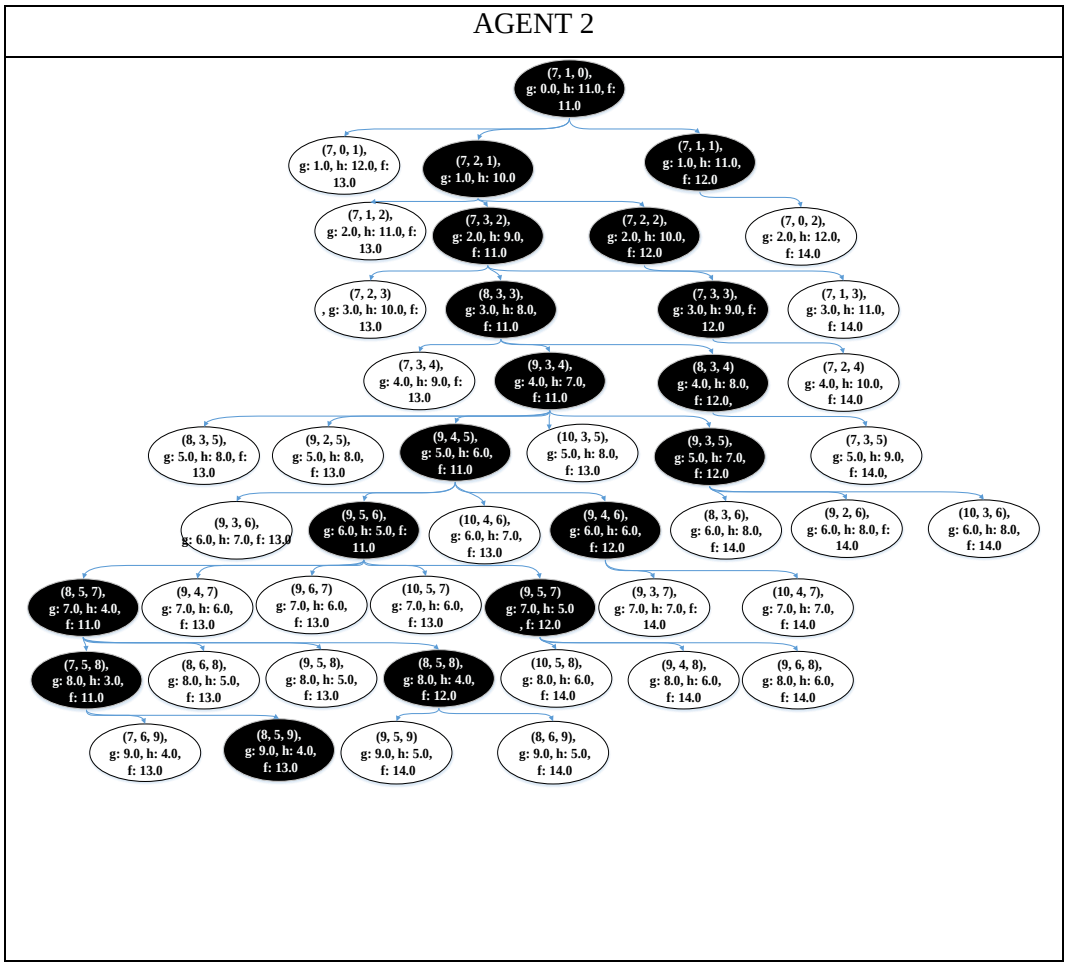
Pada iterasi ini algoritma whca* melakukan penambahan pada variable currentWindow menjadi currentWindow+1 dan currentTime menjadi currentTime +1, setelah itu langkah selanjutnya adalah mengecek ulang, apakah ada unit yang harus di kalkulasi ulang atau tidak, unit yang akan dicek diambil secara random berdasarkan currentWindow.

Tabel 3.1. Path Iterasi 0

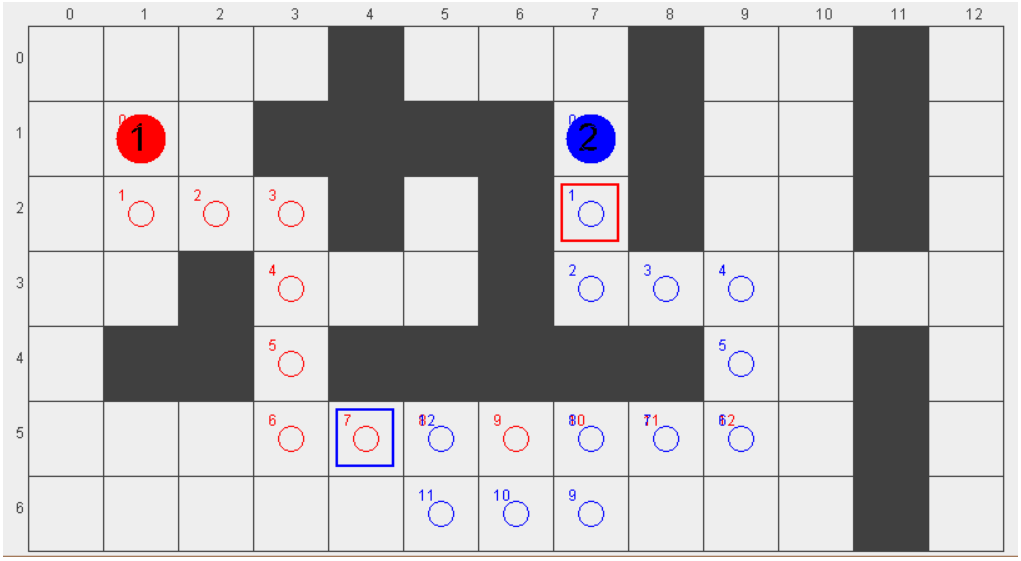
CurrentWindow = 0
currentTime = 0
depth = 10

AGENT 1





Langkah selanjutnya adalah memindahkan posisi awal semua unit ke posisi node pada path selanjutnya. seperti gambar 3.5 dibawah ini:



Gambar 3.5. Kondisi Iterasi 0

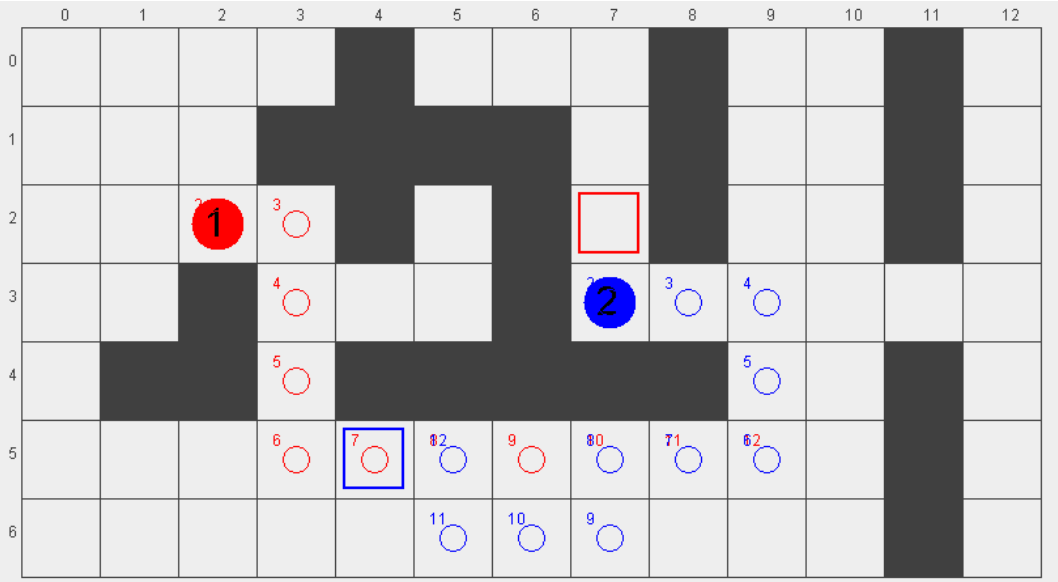
Langkah selanjutnya melakukan check nilai currentWindow pada saat ini, dan mengecek apakah ada unit yang perlu di kalkulasi, jika ada maka lakukan pencarian path kembali dari posisi node yang terakhir. Penentuan posisi unit pada window ditentukan dengan rumus nilai *depth*/2/jumlah unit.

Tabel 3.2. Path iterasi 1

CurrentWindow = 1 currentTime = 1 depth = 10	
AGENT 1	AGENT 2
Tidak ada unit yang perlu dikalkulasi, path sesuai dengan path sebelumnya	

Tabel 3.3. Path iterasi 2

CurrentWindow = 2 currentTime = 2 depth = 10	
AGENT 1	AGENT 2
Tidak ada unit yang perlu dikalkulasi, path sesuai dengan path sebelumnya	

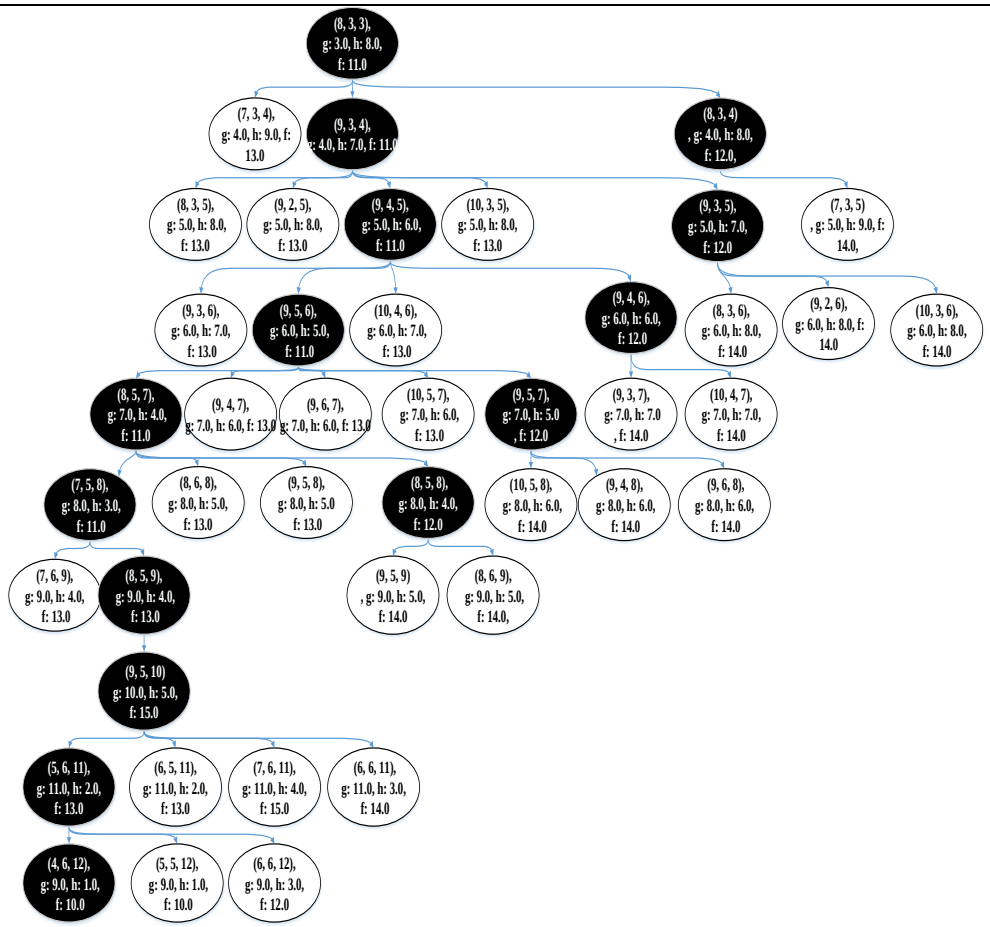


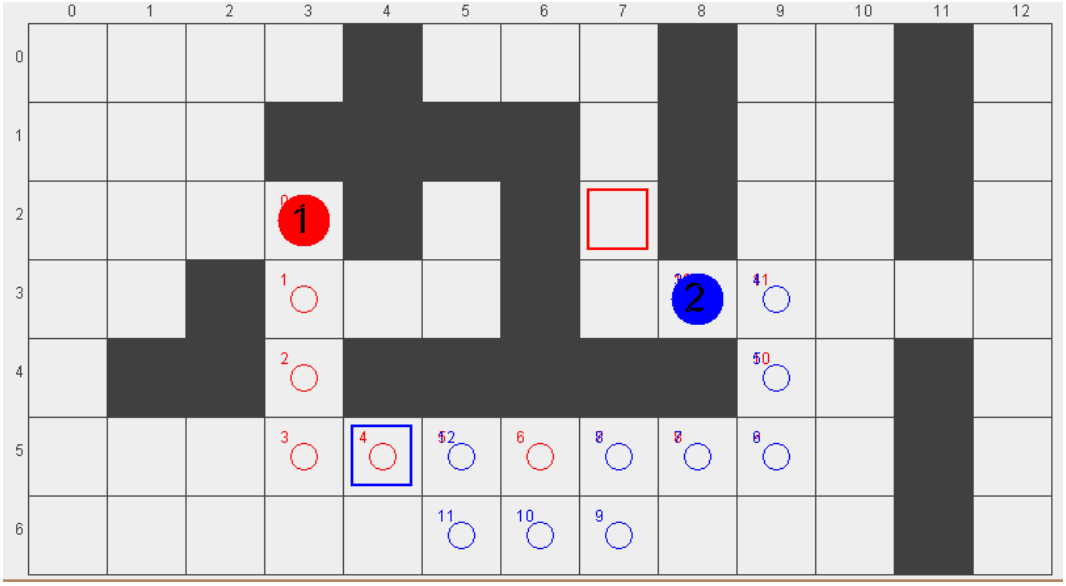
Gambar 3.6. kondisi *agents* iterasi 2

Tabel 3.4. Path iterasi ke 3

CurrentWindow = 3 currentTime = 3 depth = 10

AGENT 1
Path sesuai dengan jalur sebelumnya, karena *agent* 1 tidak melakukan *repath*.
AGENT 2

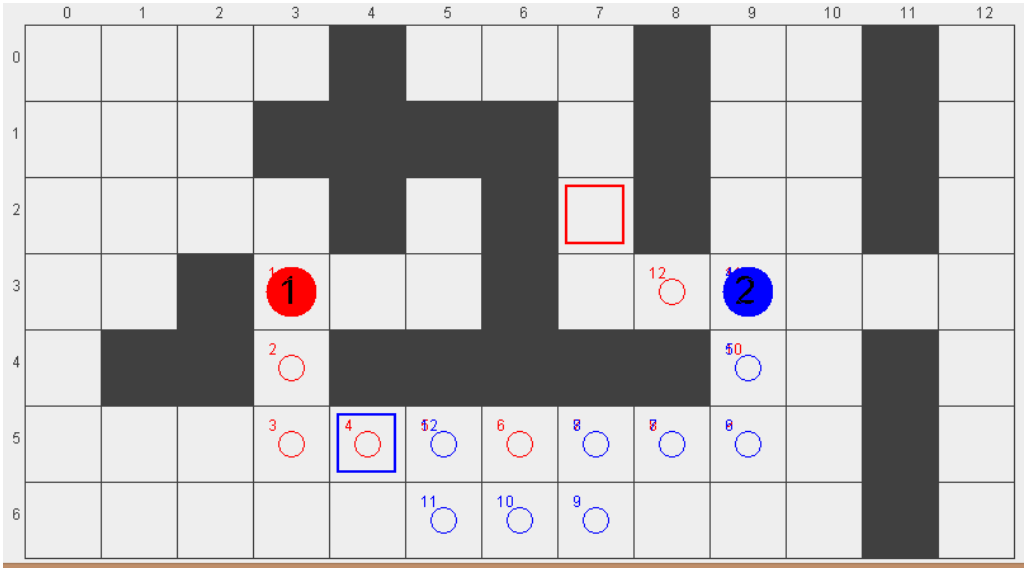




Gambar 3.7. kondisi *agents* iterasi 3

Tabel 3.5. Path iterasi 4

CurrentWindow = 4 currentTime = 4 depth = 10	
AGENT 1	AGENT 2
Tidak ada unit yang perlu dikalkulasi	

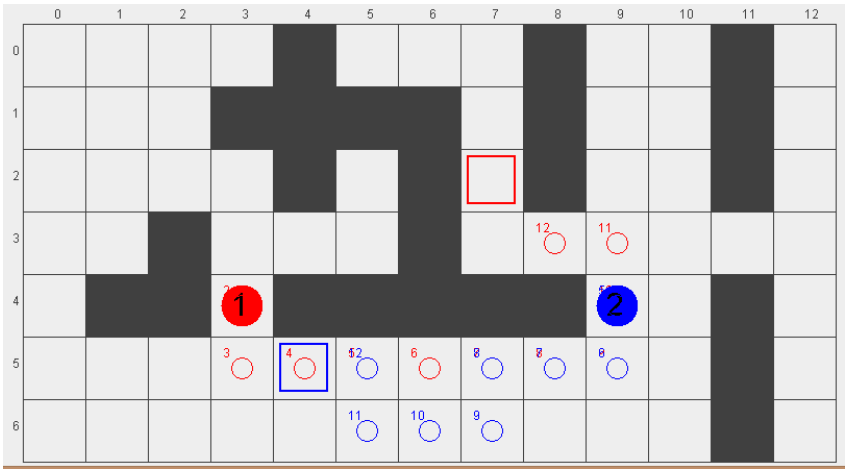


Gambar 3.8. Kondisi *agents* iterasi 4

Dikarenakan currentWindow = (depth/2) maka currentWindow kembali menjadi 0 dan *agent* 1 akan menjadi prioritas pencarian jalur terpendek karena agent 1 berada pada posisi array ke-0 pada window.

Tabel 3.6. Path iterasi 5

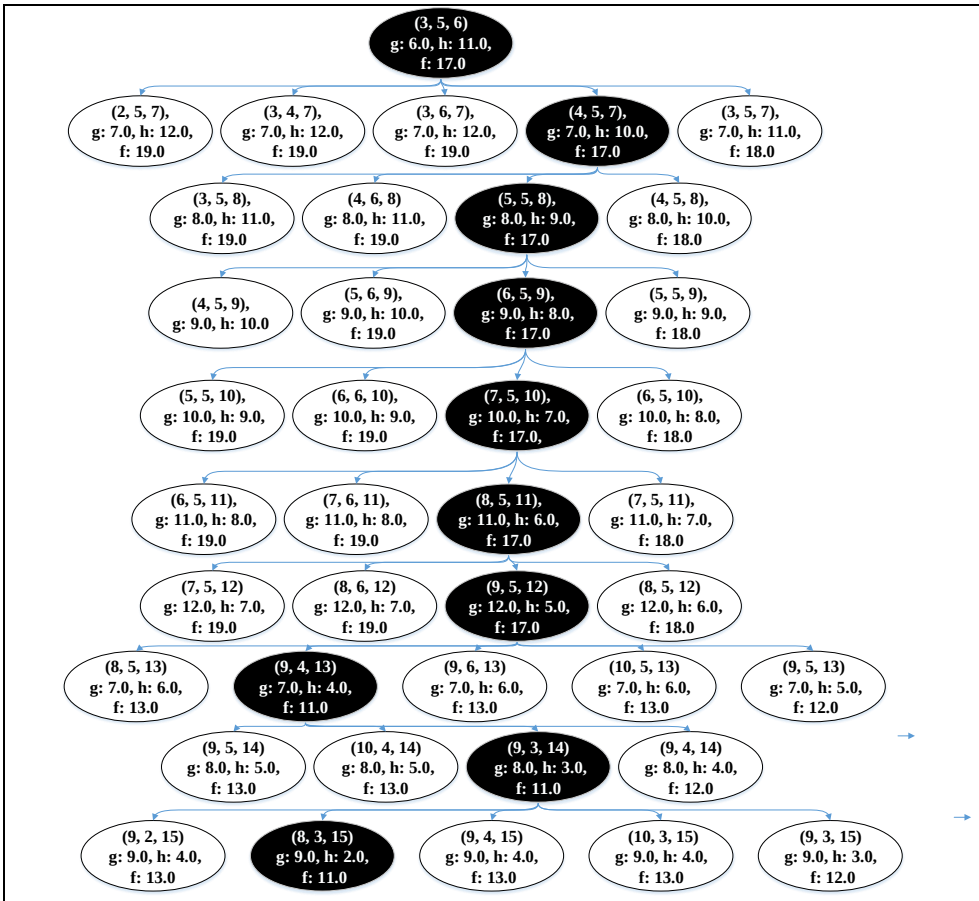
CurrentWindow = 0 currentTime = 5 depth = 10	
AGENT 1	AGENT 2
Tidak ada unit yang perlu dikalkulasi	



Gambar 3.9. Kondisi Agents Iterasi 5

Tabel 3.7. Path iterasi 6

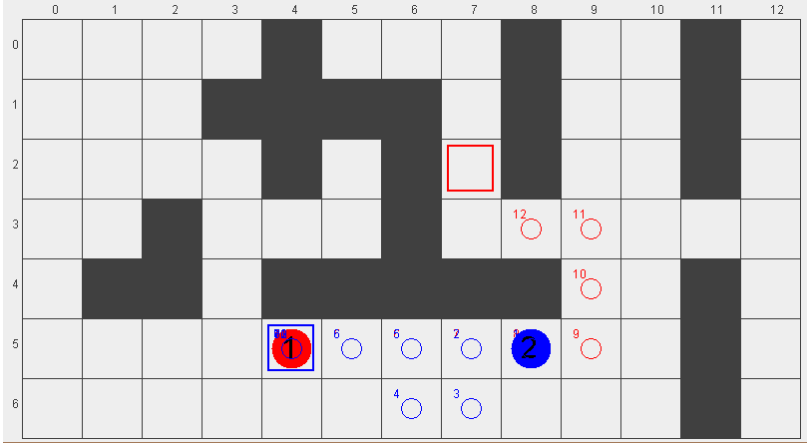
CurrentWindow = 1 currentTime = 6 depth = 10
AGENT 1



AGENT 2
Path sesuai dengan jalur sebelumnya

Tabel 3.8. Path iterasi 7

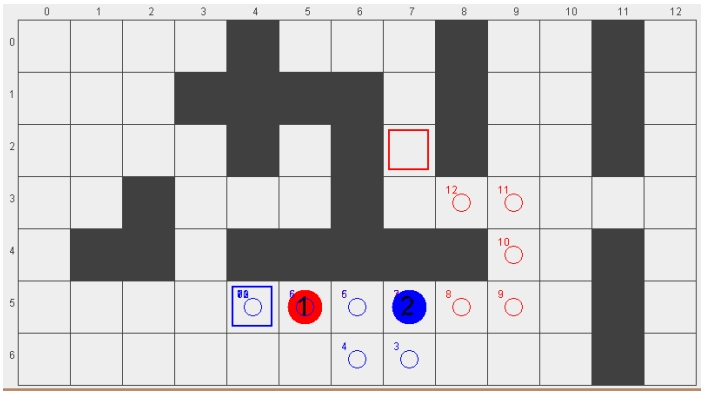
CurrentWindow = 2 currentTime = 7 depth = 10	
AGENT 1	AGENT 2
Tidak ada unit yang perlu dikalkulasi, path masih sesuai dengan path sebelumnya	



Gambar 3.10. kondisi *agents* iterasi 7

Tabel 3.9. Path iterasi 8

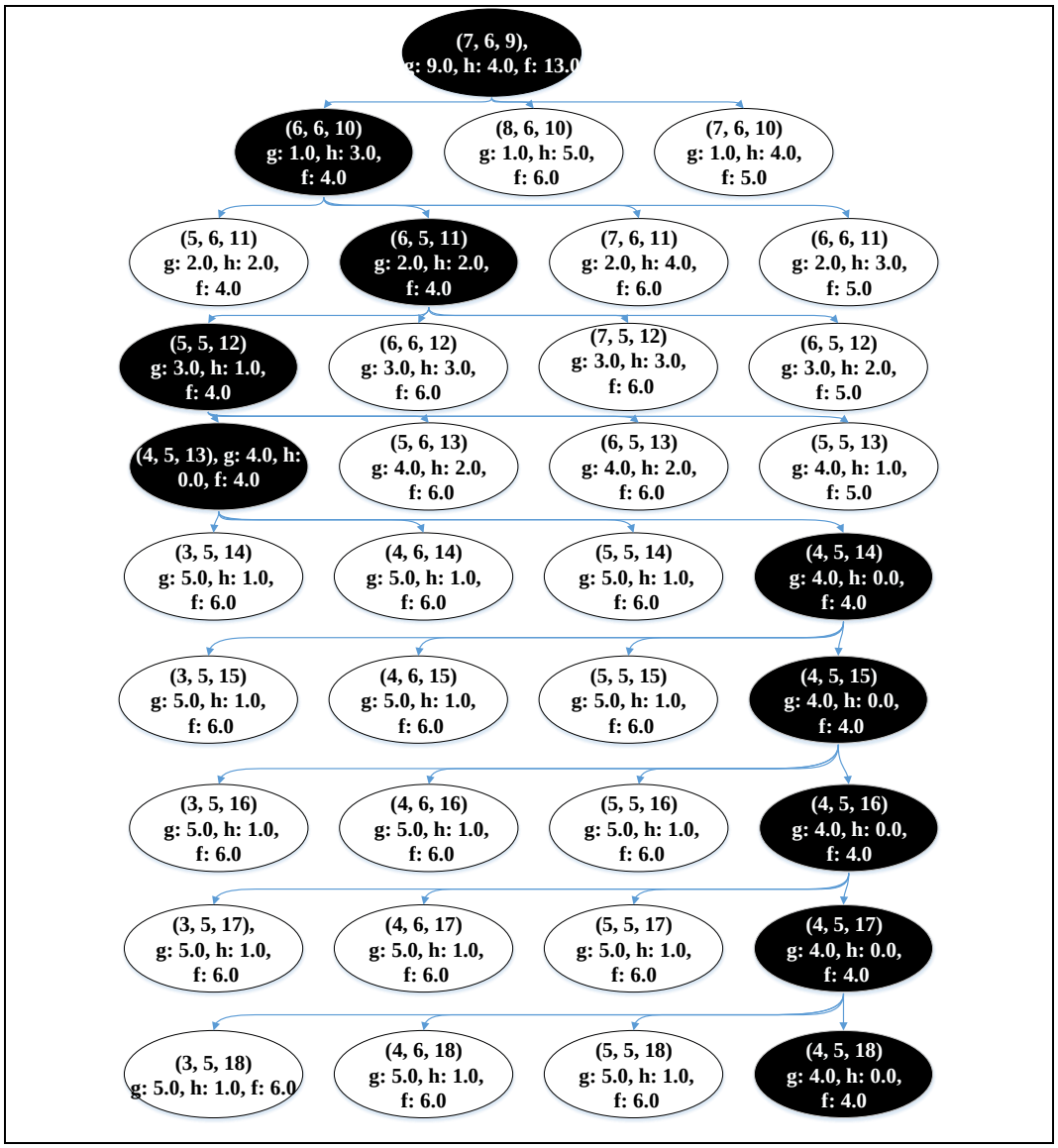
CurrentWindow = 3 currentTime = 8 depth = 10	
AGENT 1	AGENT 2
Tidak ada unit yang perlu dikalkulasi	

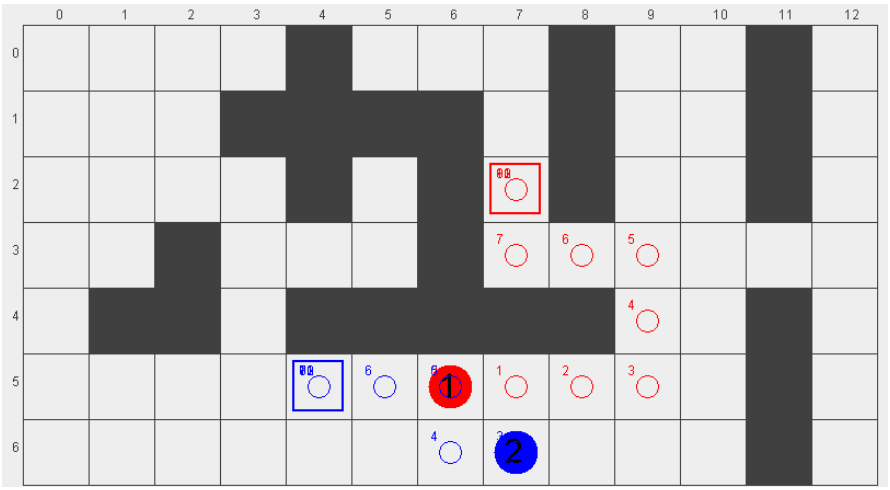


Gambar 3.11. kondisi *agents* iterasi 8

Tabel 3.10. Path iterasi 9

CurrentWindow = 4 currentTime = 9 depth = 10
AGENT 1
Unit tidak perlu dikalkulasi,karena prioritas pada iterasi ini adalah agent 2. Maka agent 2 melakukan kalkulasi ulang, path agen 1 masih sesuai dengan path sebelumnya.
AGENT 2

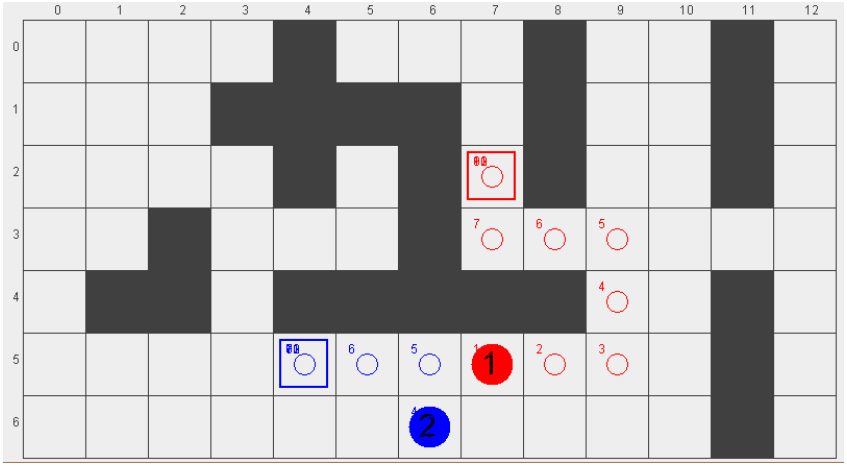




Gambar 3.12. Kondisi *agents* iterasi 9

Tabel 3.11. Path iterasi 10

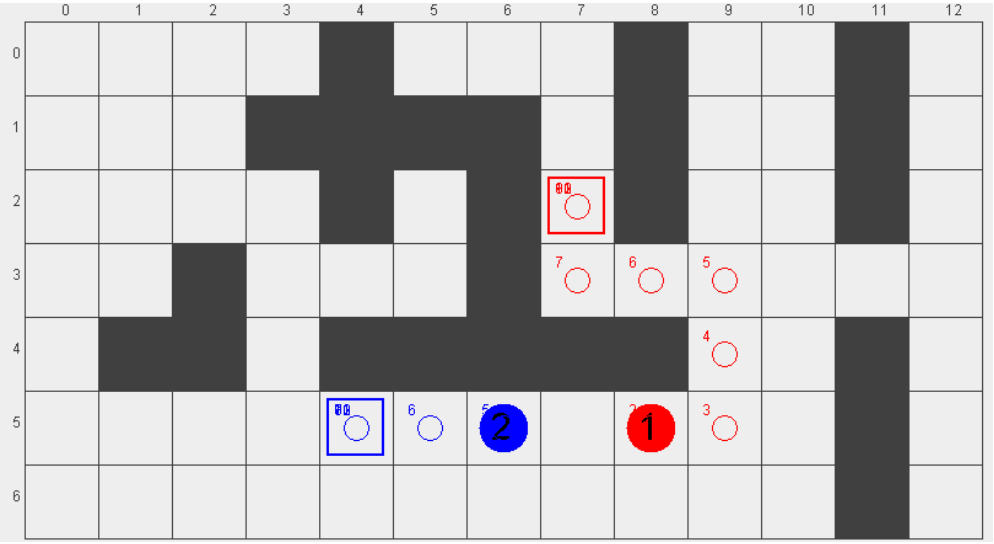
CurrentWindow = 0 currentTime = 10 depth = 10	
AGENT 1	AGENT 2
Tidak ada unit yang perlu dikalkulasi, path masih sesuai dengan path sebelumnya	



Gambar 3.13. Kondisi *agents* iterasi 10

Tabel 3.12. Path iterasi 11

CurrentWindow = 1 currentTime = 11 depth = 10	
AGENT 1	AGENT 2
Tidak ada unit yang perlu dikalkulasi, path masih sesuai dengan path sebelumnya	

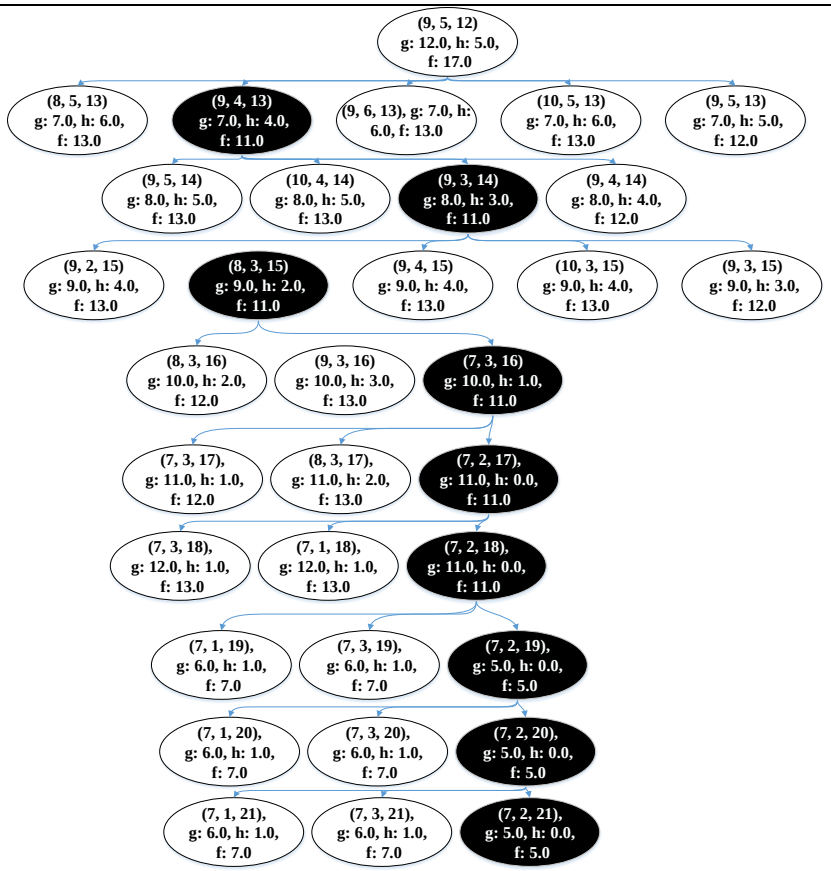


Gambar 3.14. Kondisi *agents* iterasi 11

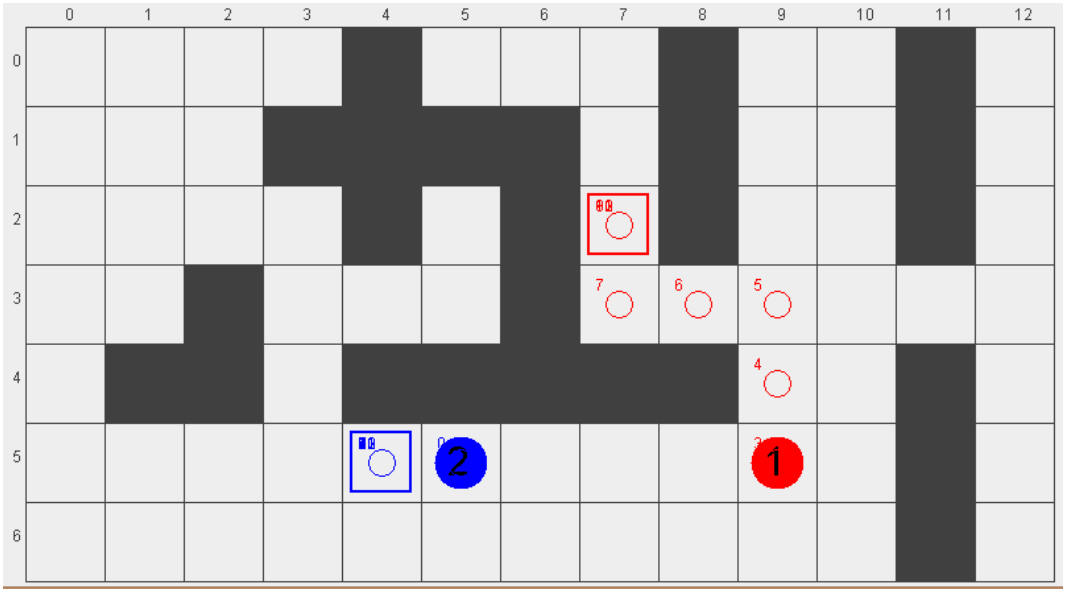
Tabel 3.13. Path iterasi 12

CurrentWindow = 2
currentTime = 12
depth = 10

AGENT 1



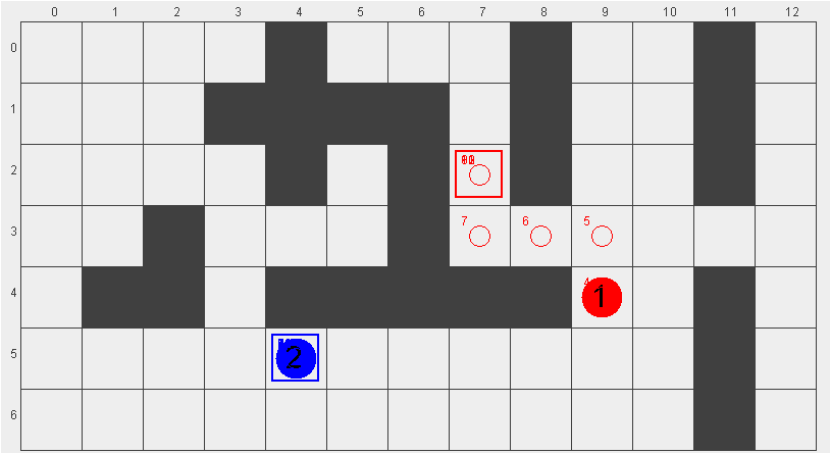
AGENT 2
Unit tidak perlu dikalkulasi,karena prioritas pada iterasi ini adalah agent 1. Maka agent 1 melakukan kalkulasi ulang, path agen 2 masih sesuai dengan path sebelumnya



Gambar 3.15. Kondisi *agents* iterasi 12

Tabel 3.14. Path iterasi 13

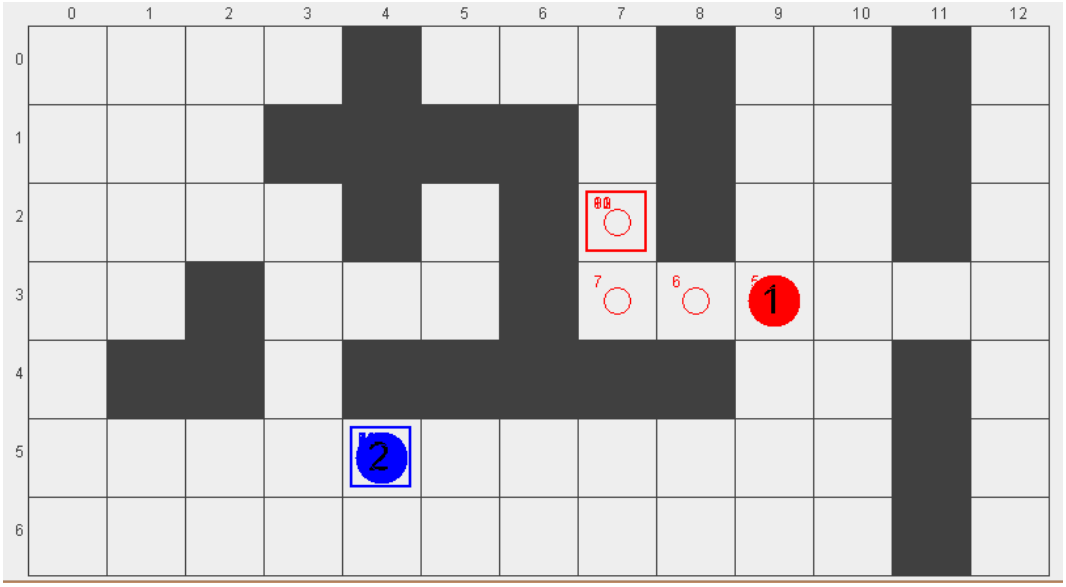
CurrentWindow = 3 currentTime = 13 depth = 10	
AGENT 1	AGENT 2
Tidak ada unit yang perlu dikalkulasi, path masih sesuai dengan path sebelumnya	



Gambar 3.16. Kondisi *agents* iterasi 13

Tabel 3.15. Path iterasi 14

CurrentWindow = 4 currentTime = 14 depth = 10	
AGENT 1	AGENT 2
Tidak ada unit yang perlu dikalkulasi, path masih sesuai dengan path sebelumnya	



Gambar 3.17. Kondisi Agents iterasi 14

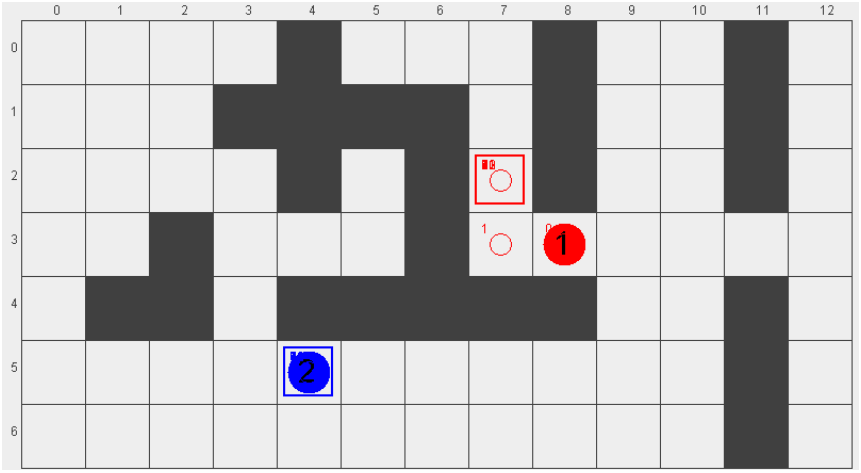
Tabel 3.16. Path iterasi 15

CurrentWindow = 0 currentTime = 15 depth = 10
AGENT 1



AGENT 2

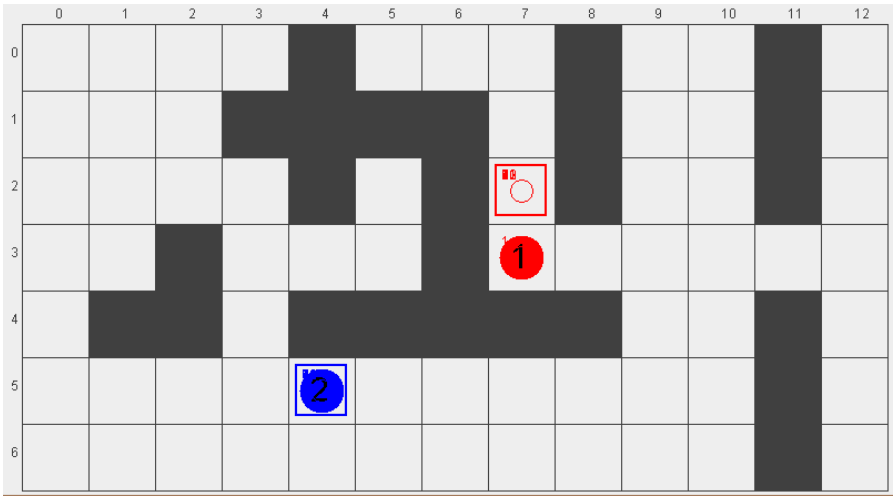
Unit tidak perlu dikalkulasi,karena prioritas pada iterasi ini adalah *agent* 1. Maka *agent* 1 melakukan kalkulasi ulang, *path agent* 2 masih sesuai dengan path sebelumnya



Gambar 3.18. Kondisi *agents* iterasi 15

Tabel 3.17. Path iterasi 16

CurrentWindow = 1 currentTime = 16 depth = 10	
AGENT 1	AGENT 2
Tidak ada unit yang perlu dikalkulasi, path masih sesuai dengan path sebelumnya	

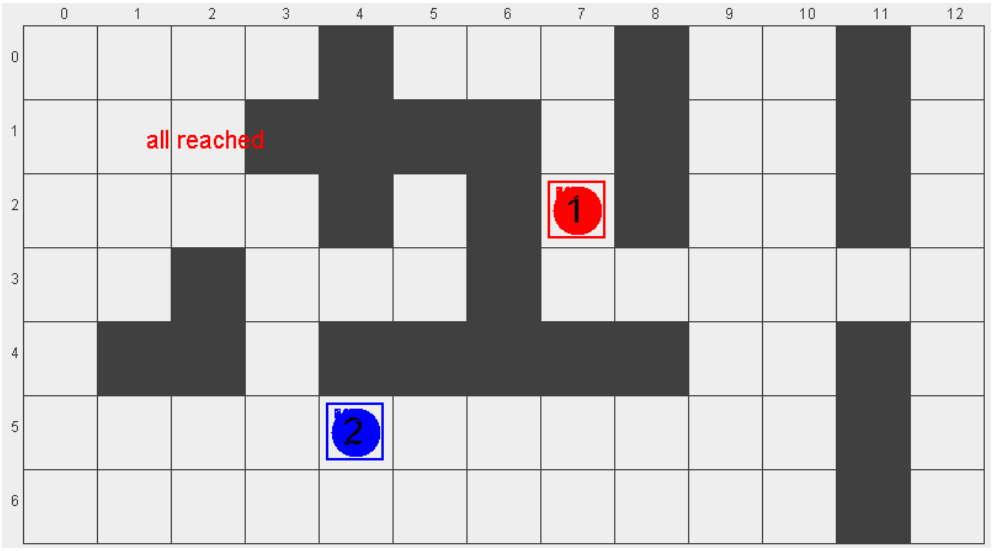


Gambar 3.19. Kondisi *agents* iterasi 16

Tabel 3.18. Path iterasi 17

CurrentWindow = 2 currentTime = 17 depth = 10	
AGENT 1	AGENT 2

Semua unit sudah mencapai tujuannya masing-masing



Gambar 3.20. Kondisi *agents* iterasi 17

3.3 Analisis Perangkat Keras

Spesifikasi perangkat keras (*hardware*) yang dibutuhkan untuk menguji pengimplementasian algoritma WHCA* pada webots ini dapat dilihat pada Tabel 3.19.

Tabel 3.19. Kebutuhan Perangkat Keras

Perangkat Keras	**Spesifikasi**
Processor	*Intel Core*-i5-3230M 2.6 Ghz
Resolusi layar	1024 x 768 pixel
Memori	2 GB
Harddisk	500GB

3.4 Analisis Perangkat Lunak

Spesifikasi perangkat lunak (*software*) yang dibutuhkan untuk menguji pengimplementasian algoritma WHCA* pada webots ini dapat dilihat pada Tabel 3.20.

Tabel 3.20. Kebutuhan Perangkat Lunak

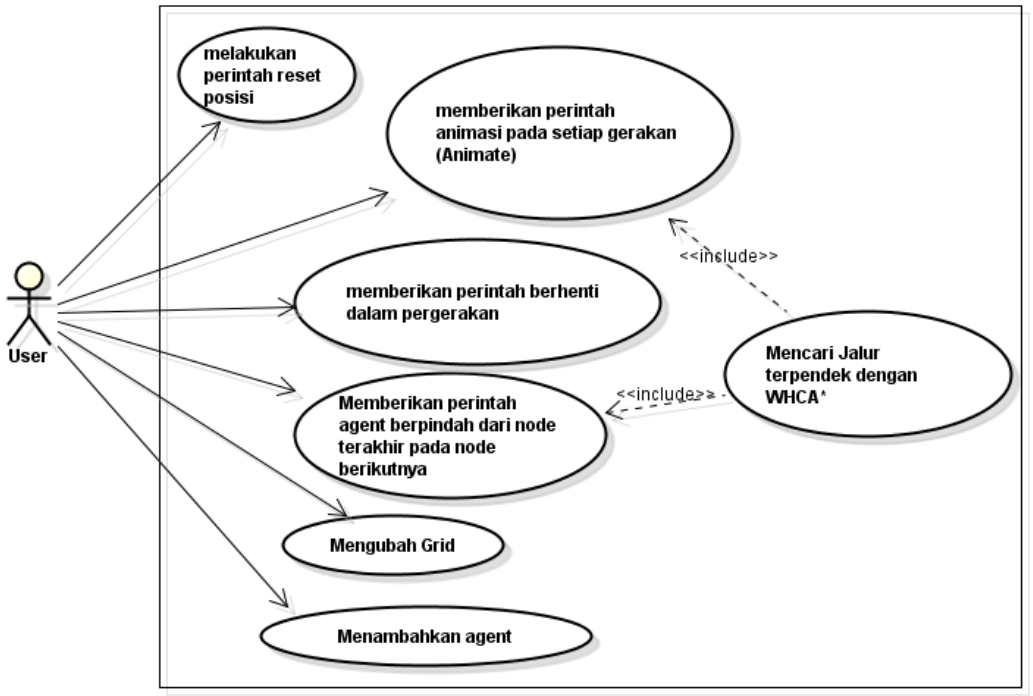
Komputer	**Spesifikasi**
Sistem Operasi	Windows 8
Bahasa Pemograman	Java
IDE	Eclipse

3.5 Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional menggambarkan proses kegiatan yang akan diterapkan dalam sebuah sistem dan menjelaskan kebutuhan yang diperlukan sistem agar sistem dapat berjalan dengan baik sesuai kebutuhan. Analisis yang dilakukan dimodelkan dengan menggunakan UML (*Unified Modeling Language*). Tahapan pemodelan dalam analisis tersebut antara lain pembuatan *use case diagram*, definisi aktor, definisi *use case*, skenario *use case*, *activity diagram*, *sequence diagram*, dan *class diagram*.

1.5.1. Usecase Diagram

Use case diagram digunakan untuk memodelkan proses bisnis dan merepresentasikan sebuah interaksi antara aktor dengan sistem. Aktor disini adalah pengguna, yang mana pengguna dapat menginput *starting point* dan *end point* dengan unit yang banyak.



Gambar 3.21. Usecase Diagram WHCA*

3.5.2. Skenario Use Case

Skenario setiap bagian pada *use case* menunjukkan proses apa yang terjadi pada setiap bagian didalam *use case* tersebut, dimana pengguna memberikan perintah pada setiap bagian dan respon apa yang diberikan oleh sistem kepada

pengguna setelah pengguna memberikan perintah pada setiap bagian-bagian *use case.*

Tabel 3.21. Skenario Use Case Menambah *Agent*

Identifikasi	
Nama	Menambah *Agent*
Tujuan	Menginstrusikan program untuk menambah agent
Deskripsi	Inisiasi agent untuk proses pencarian
Aktor	Pengguna
Skenario Utama	
Kondisi Awal	Pengguna berada di menu utama pada program
Aksi actor	Reaksi Sistem
1. User memilih koordinat 2. User Menekan tombol tambah agent	3. Unit melakukan inisialisasi path 4. Sistem menampilkan unit pada koordinat yang diinputkan
Kondisi Akhir	Sistem menampilkan agent pada koordinat yang diinputkan

Tabel 3.22. Skenario Use Case Mencari jalur terpendek dengan WCHA*

Identifikasi	
Nama	Mencari jalur terpendek dengan WHCA*
Tujuan	Menginstrusikan program untuk pencarian jalur terpendek menggunakan WHCA*
Deskripsi	Proses pencarian jalur terpendek
Aktor	Sistem
Skenario Utama	
Kondisi Awal	Unit memiliki koordinat awal dan akhir
Aksi -	Reaksi Sistem
-	1. Inisialisasi node awal dan node akhir 2. Sistem menyimpan node awal ke *list open* 3. Melakukan iterasi tetangga dan melakukan validasi node tujuan 4. Sistem menampilkan *path* setiap *agent*
Kondisi Akhir	*Agent* memiliki path sampai tujuan dengan jalur yang unik, tanpa adanya kemungkinan tabrakan

Tabel 3.23. Skenario Use Case *Reset* agent

Identifikasi	
Nama	Reset unit
Tujuan	Menginstrusikan program untuk me-*reset* ulang posisi *agent*
Deskripsi	Proses *reset agent*
Aktor	Sistem
Skenario Utama	
Kondisi Awal	*Agent* berada pada posisi pencarian
Aksi User	Reaksi Sistem
1. Tekan tombol Reset	2. Sistem menampilkan *agent* pada posisi awal
Kondisi Akhir	*Agent* kembali ke posisi awal

Tabel 3.24. Skenario Use Case *Animate agent*

Identifikasi	
Nama	*Animate* unit
Tujuan	Menginstrusikan program untuk melakukan pencarian jalur terpendek dengan adanya animasi pada setiap pergerakan
Deskripsi	Proses animasi pergerakan pencarian jalur terpendek
Aktor	Sistem
Skenario Utama	
Kondisi Awal	*Agent* berada pada posisi awal
Aksi User	Reaksi Sistem
1. Tekan tombol *Animate*	2. *Agent* melakukan pencarian dengan animasi pada setiap pergerakan
Kondisi Akhir	*Agent* bergerak mencari jalur

Tabel 3.25. Skenario Use Case *Stop Agent*

Identifikasi	
Nama	*Stop agent*
Tujuan	Menginstrusikan program untuk berhenti dalam pergerakan mencari jalur terpendek
Deskripsi	Proses melakukan berhenti dalam pergerakan pencarian
Aktor	Sistem
Skenario Utama	
Kondisi Awal	Agent berada pada koordinat tertentu
Aksi User	Reaksi Sistem
1. Tekan tombol *Stop*	2. Menampilkan *agent* pada posisi diam dan koordinat terakhir

Kondisi Akhir	*Agent* berhenti pada posisi terakhir

Tabel 3.26. Skenario Use Case *Step Agent*

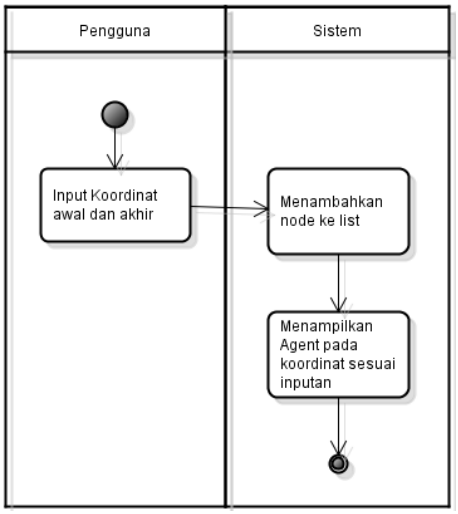
Identifikasi	
Nama	*Step Agent*
Tujuan	Menginstrusikan program untuk maju selangkah menuju *node* selanjutnya
Deskripsi	Proses melakukan perpindahan *agent* dari node awal ke *node* selanjutnya
Aktor	Sistem
Skenario Utama	
Kondisi Awal	*Agent* berada pada posisi awal
Aksi User	Reaksi Sistem
1. Tekan tombol *Step*	2. Sistem menampilkan *agent* pada posisi selanjutnya
Kondisi Akhir	*Agent* berada pada posisi *node* selanjutnya

Tabel 3.27. Skenario Use Case mengubah *Grid*

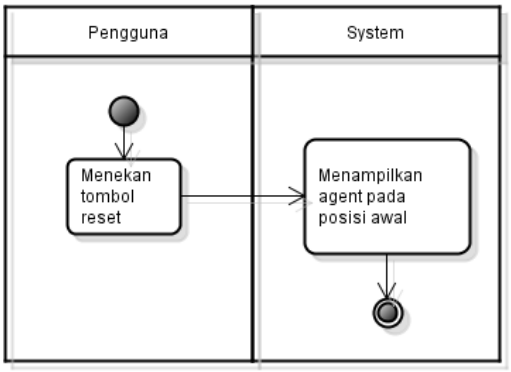
Identifikasi	
Nama	Mengubah grid
Tujuan	Menginstrusikan program untuk mengubah jumlah kolom dan baris grid
Deskripsi	Proses mengubah baris dan kolom grid
Aktor	Sistem
Skenario Utama	
Kondisi Awal	Jumlah grid sesuai *default*
Aksi User	Reaksi Sistem
1. mengisi data nilai kolom 2. Mengisi data nilai baris 3. Menekan tombol Grid Baru	4. Sistem menampilkan *grid*
Kondisi Akhir	Jumlah kolom dan baris grid sesuai dengan yang diinputkan

3.5.3. Activity Diagram

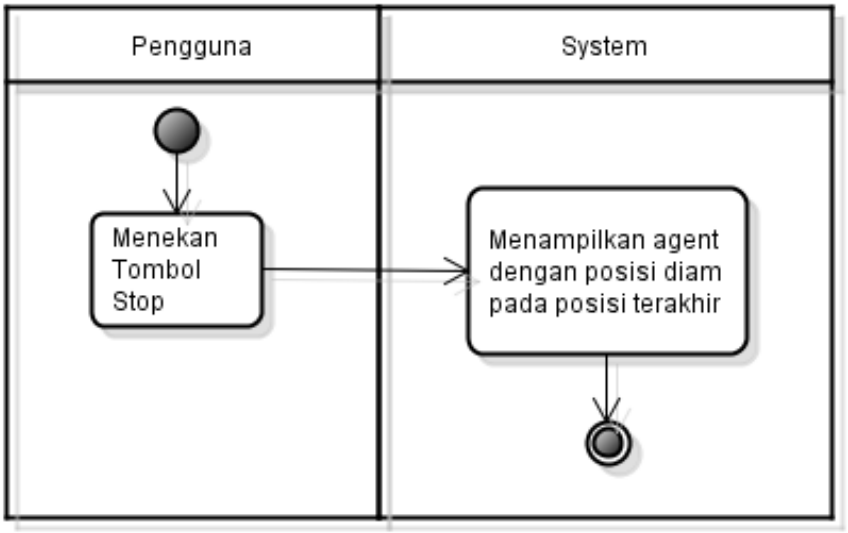
Diagram aktivitas atau *Activity Diagram* menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Penggambaran *activity* diagram memiliki kemiripan dengan *flowchart diagram. Activity diagram* memodelkan *event-event* yang terjadi pada *Use Case* dan digunakan untuk pemodelan aspek dinamis dari sistem.



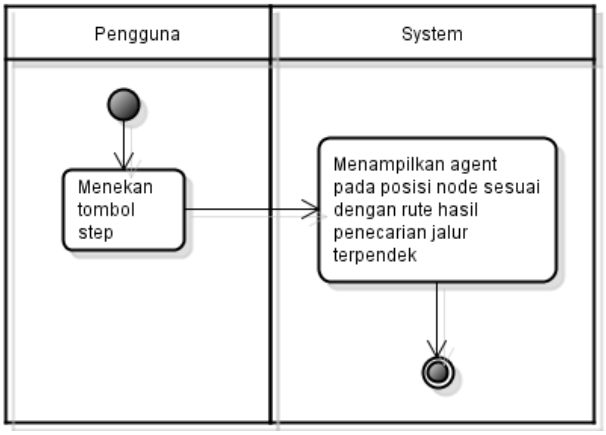
Gambar 3.22. *Activity* Diagram Tambah Agent



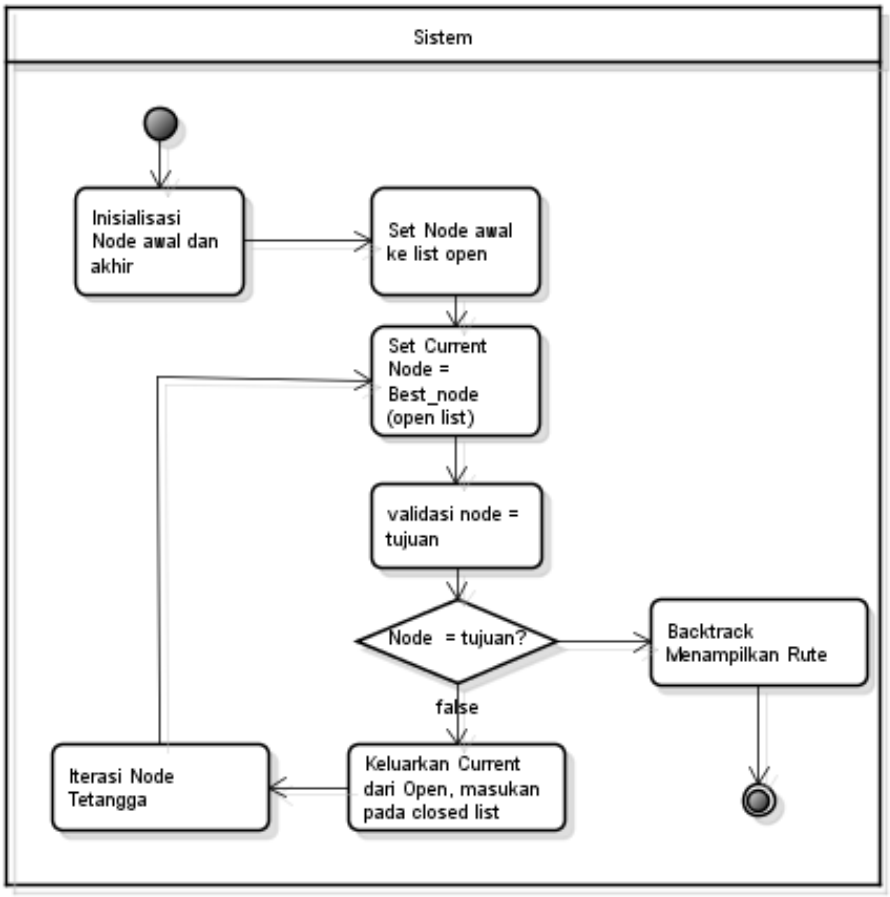
Gambar 3.23. *Activity Diagram* Tombol *Reset*



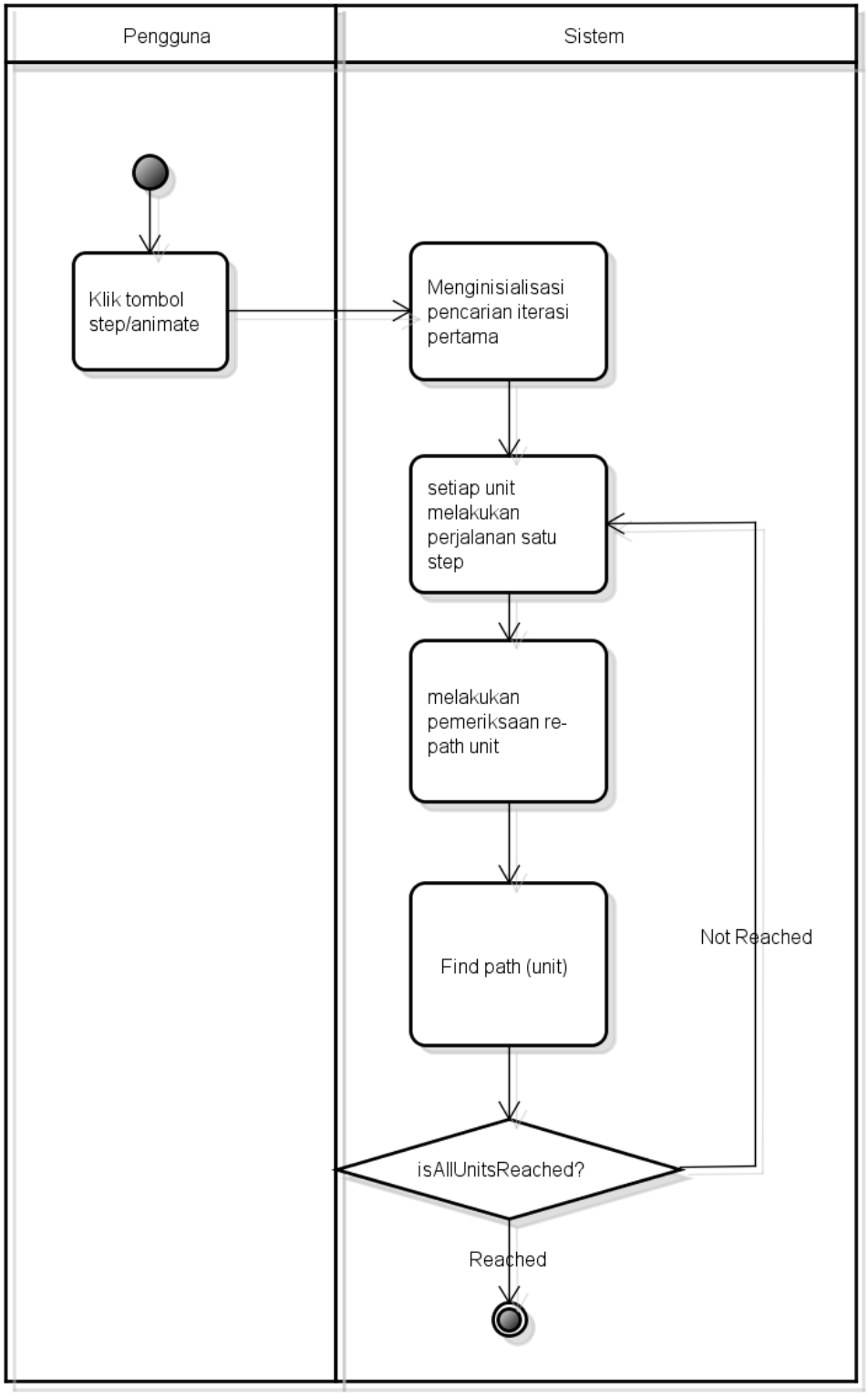
Gambar 3.24. *Activity Diagram* Tombol *Stop*



Gambar 3.25. *Activity Diagram* Tombol *Step*

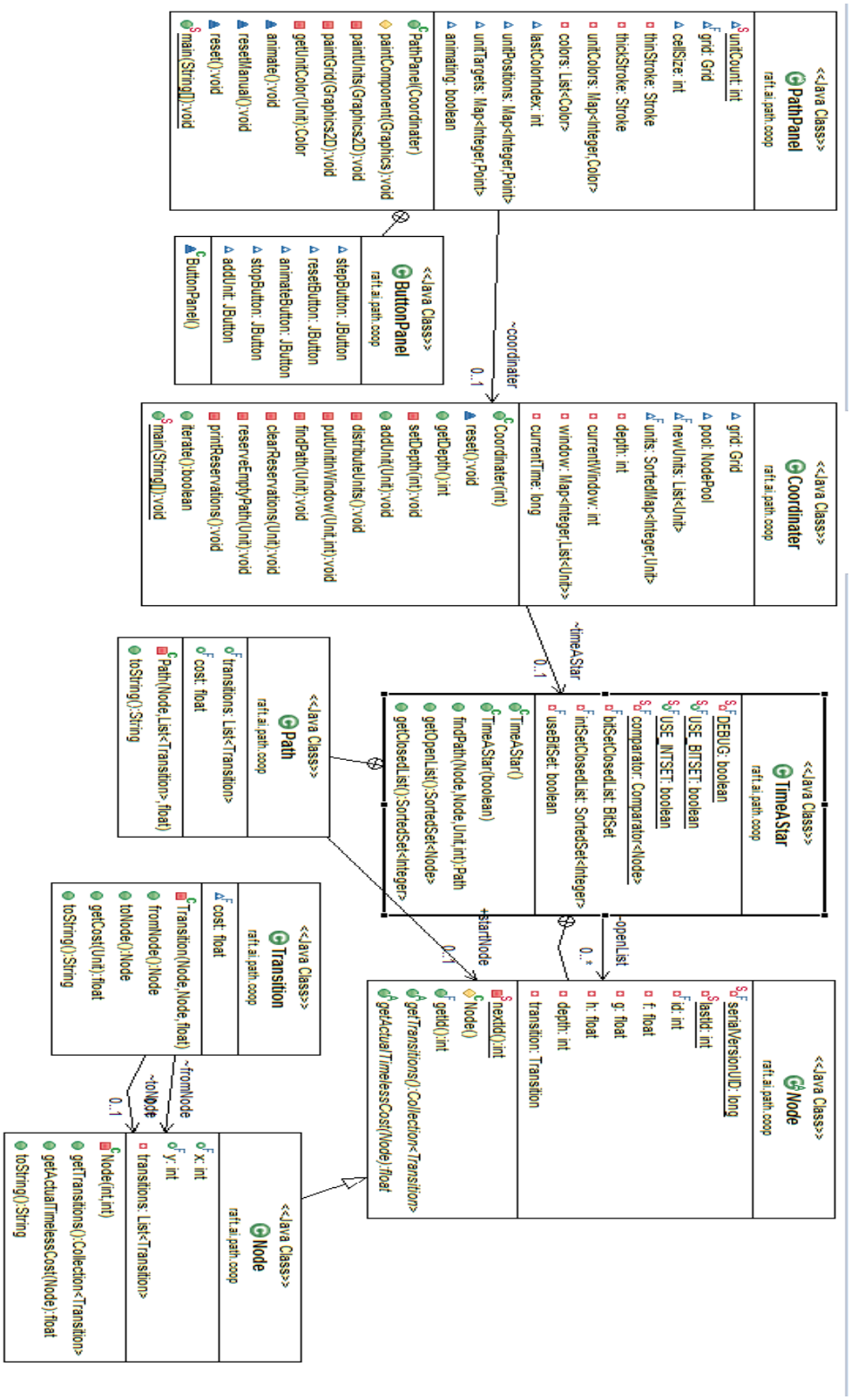


Gambar 3.26. Activity Diagram Pencarian WHCA*



Gambar 3.27. *Activity Diagram* Tombol *Animate*

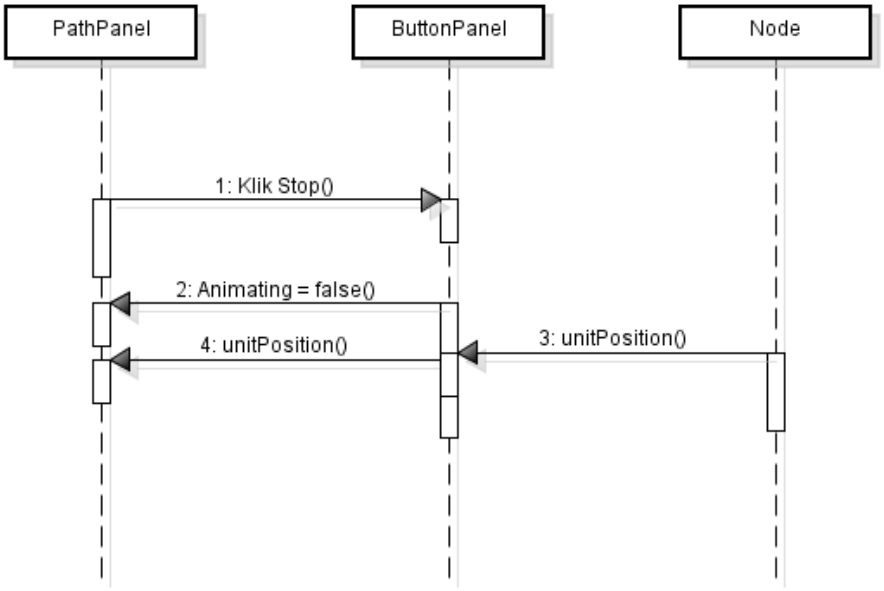
3.5.4. Class Diagram



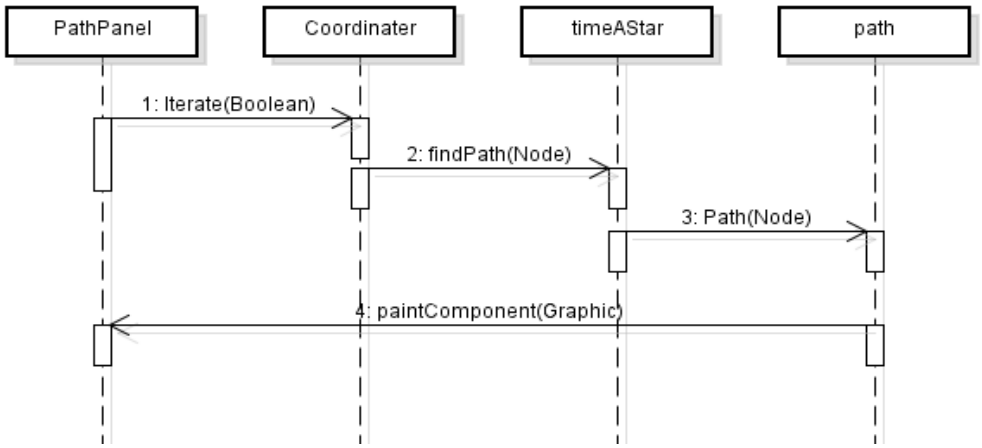
Gambar 3.26. Class Diagram

3.5.5. Sequence Diagram

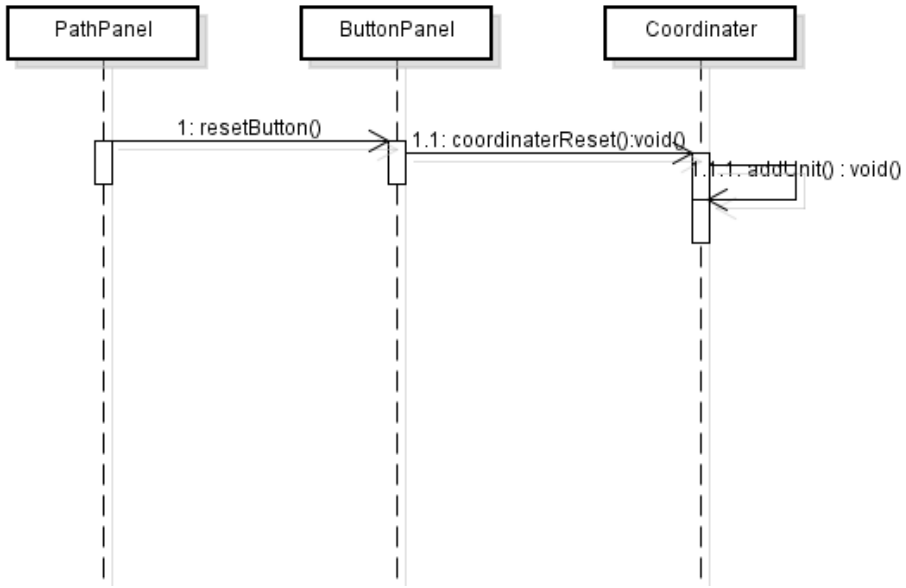
Diagram *sequence* atau *Sequence Diagram* menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek.



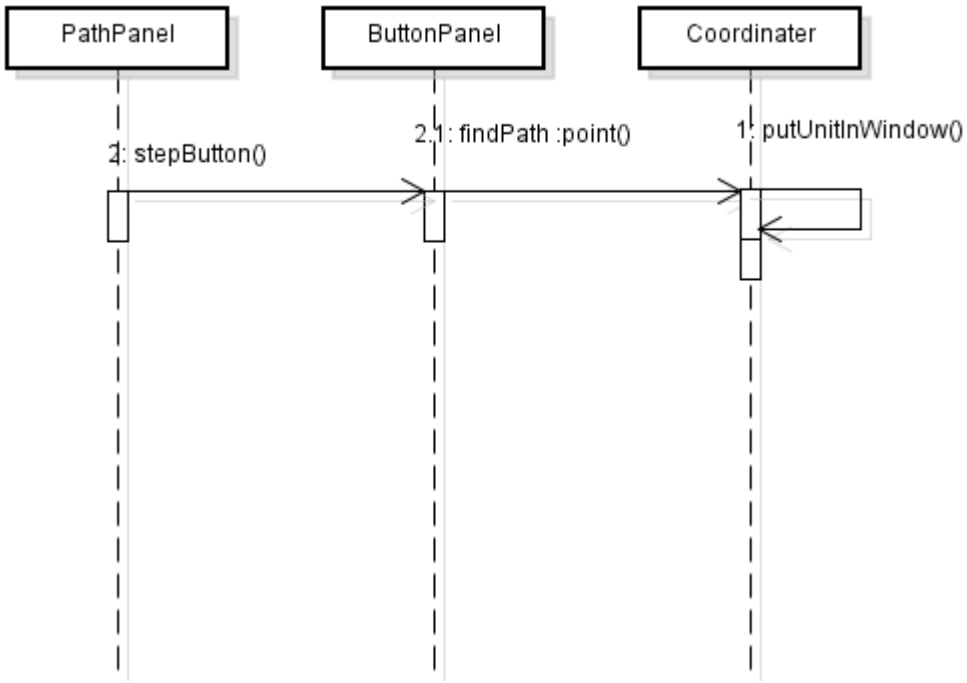
Gambar 3.27. Sequence Diagram Tombol *Stop*



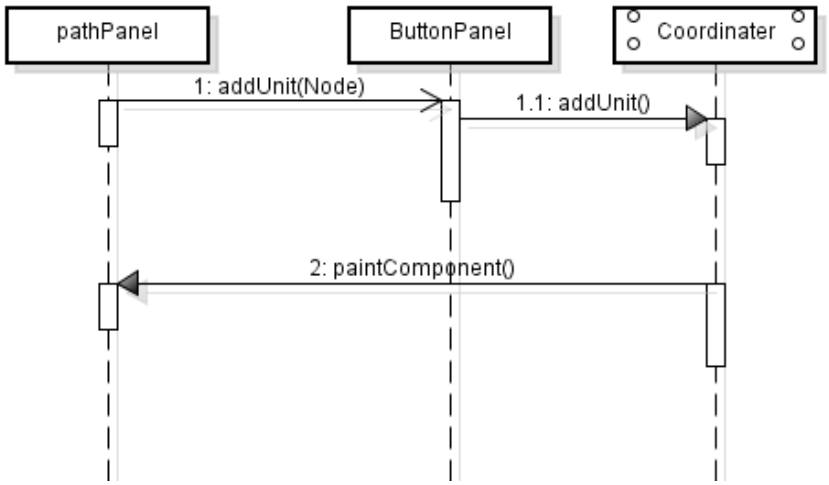
Gambar 3.28. Sequence Diagram Tombol *Animate*



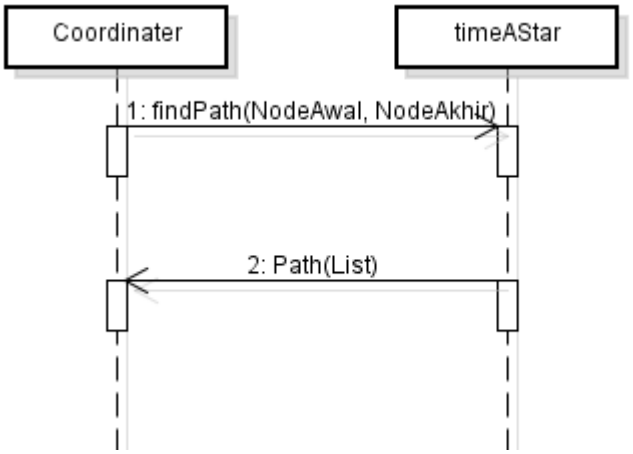
Gambar 3.29. Sequence Diagram Tombol Reset



Gambar 3.30. Sequence Diagram Tombol Step



Gambar 3.31. Sequence Diagram Menambah *Agent*



Gambar 3.32. Sequence Diagram WHCA*