BAB 4
IMPLEMENTASI DAN PENGUJIAN

Pada bab ini akan dilakukan implementasi dan pengujian terhadap aplikasi permainan yang dibangun, yaitu *Monster Nest*. Hasil perancangan pada tahap sebelumnya kemudian diimplementasikan ke dalam bahasa pemrograman. Setelah tahap implementasi selesai, maka dilakukan tahap pengujian terhadap aplikasi permainan *Monster Nest*.

4.1 Implementasi

Tahap implementasi merupakan tahap penciptaan perangkat lunak dan juga tahap kelanjutan dari kegiatan perancangan aplikasi. Tahap ini merupakan tahap di mana aplikasi siap untuk dioperasikan, yang terdiri dari penjelasan mengenai lingkungan implementasi, baik itu lingkungan perangkat keras, maupun lingkungan perangkat lunak, serta implementasi program.

4.1.1 Implementasi Perangkat Keras

Perangkat keras yang digunakan untuk menjalankan aplikasi permainan *Monster Nest* tidak harus yang berspesifikasi tinggi. Spesifikasi minimum yang dapat digunakan untuk menjalankan aplikasi permainan *Monster Nest* adalah sebagai berikut :

1. *Chipset* Hummingbird
2. *CPU* 1 GHz *Cortex*-A8
3. *GPU PowerVR* SGX540
4. *Screen size* 600 x 1024 *pixels*, 7.0 *inches* (~170 ppi *pixel density*)
5. *Screen* type TFT *capacitive touchscreen*, 16M *colors*
6. *Multitouch*

4.1.2 Implementasi Perangkat Lunak

Adapun spesifikasi perangkat lunak yang digunakan untuk menjalankan aplikasi permainan *Monster Nest* adalah perangkat *mobile android* yang telah terinstal OS *froyo* v2.2 ke atas

4.1.3 Implementasi Aplikasi

Monster Nest merupakan aplikasi permainan *third person shooter*. Dalam permainan pemain ditugaskan untuk membunuh semua *monster* yang ada di arena, setiap kali membunuh *monster* pemain akan mendapatkan skor yang nantinya akan diakumulasikan dan disimpan di dalam *highscore* jika pemain mendapatkan skor di urutan delapan teratas.

Permainan dikatakan *game over* apabila jumlah darah pemain berkurang menjadi 0, dan sebaliknya permainan dikatakan berhasil apabila pemain mendapatkan skor yang tertinggi.

Aplikasi permainan ini berbasis *mobile android* yang dibangun dengan menggunakan bahasa pemrograman javascript/unityscript. Sehingga aplikasi permainan ini hanya dapat dijalankan pada perangkat *mobile* dengan OS *android*.

4.1.4 Implementasi Antarmuka

Adapun implementasi antarmuka pada aplikasi permainan *Monster Nest* adalah sebagai berikut :

1. Antarmuka menu utama

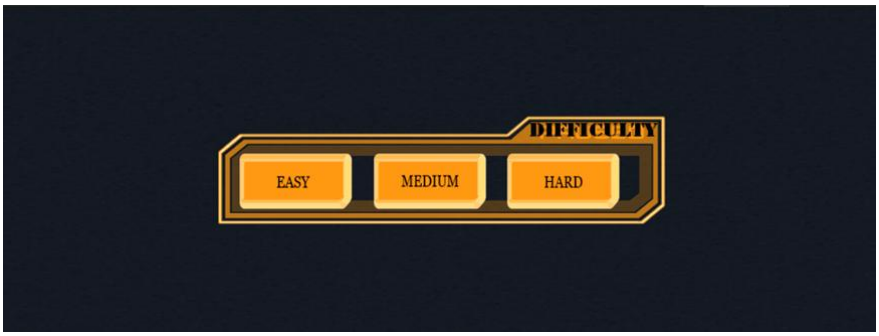
Antarmuka menu utama merupakan tampilan *menu* utama pada saat pertama kali pemain membuka aplikasi permainan. Tampilan dari antarmuka *menu* utama dapat dilihat pada gambar 4.1.



Gambar 4.1 Tampilan antarmuka *menu* utama

2. Antarmuka mengubah tingkat kesulitan

Antarmuka mengubah tingkat kesulitan merupakan tampilan ketika pemain menekan tombol *set level* sehingga langsung masuk ke pilihan tingkat kesulitan dalam permainan yang terdiri dari *easy*, *medium* dan *hard*. Tampilan dari antarmuka tingkat kesulitan dapat dilihat pada gambar 4.2.



Gambar 4.2 Tampilan antarmuka mengubah tingkat kesulitan

3. Antarmuka *highscore*

Antarmuka *highscore* merupakan tampilan ketika pemain menekan tombol *highscore* sehingga akan muncul tampilan *highscore* yang menampilkan peringkat, nama, dan skor dari pemain yang pernah bermain *game monster nest*. Tampilan dari antarmuka *highscore* dapat dilihat pada gambar 4.3.



Gambar 4.3 Tampilan antarmuka *highscore*

4. Antarmuka petunjuk

Antarmuka petunjuk merupakan tampilan ketika pemain menekan tombol *play game* kemudian yang pertama kali muncul adalah antarmuka petunjuk yang menampilkan keterangan dari layar yang akan digunakan pada saat bermain *game monster nest*. Tampilan dari antarmuka petunjuk dapat dilihat pada gambar 4.4.



Gambar 4.4 Tampilan antarmuka petunjuk

5. Antarmuka bermain

Antarmuka bermain merupakan tampilan ketika pemain berada dalam permainan *monster nest*, di dalam antarmuka ada tombol-tombol dan gambar yang sebelumnya sudah dijelaskan di antarmuka petunjuk gambar 4.5. Tampilan dari antarmuka bermain dapat dilihat pada gambar 4.5.



Gambar 4.5 Tampilan antarmuka bermain

6. Antarmuka menu *pause*

Antarmuka menu *pause* merupakan tampilan yang muncul ketika pemain menekan tombol *pause*, pada saat menu *pause* muncul permainan akan dihentikan untuk sementara. Tampilan dari antarmuka menu *pause* dapat dilihat pada gambar 4.6.



Gambar 4.6 Antarmuka menu *pause*

7. Antarmuka konfirmasi pada menu *pause*

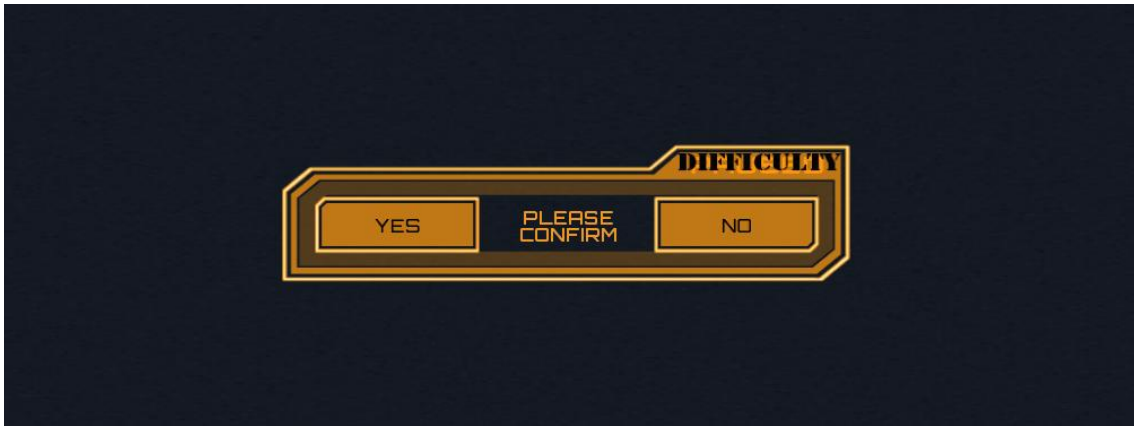
Antarmuka konfirmasi ini merupakan tampilan ketika yang muncul untuk konfirmasi pada menu *pause*. Tampilan dari antarmuka petunjuk dapat dilihat pada gambar 4.7.



Gambar 4.7 Antarmuka konfirmasi pada menu *pause*

8. Antarmuka konfirmasi pada tingkat kesulitan

Antarmuka konfirmasi ini merupakan tampilan ketika yang muncul untuk konfirmasi pada menu tingkat kesulitan. Tampilan dari antarmuka petunjuk dapat dilihat pada gambar 4.8.



Gambar 4.8 Antarmuka konfirmasi pada tingkat kesulitan

4.2 Pengujian

Dalam melakukan pengujian terhadap aplikasi permainan *Monster Nest* dilakukan dengan dua macam teknik, yaitu pengujian *black-box* dan pengujian *white-box*. Adapun pengujian *black box* berfokus pada menemukan kesalahan program, sedangkan pengujian *white box* berfokus kepada struktur internal (*source* code) program untuk mengetahui apakah masih terjadi *error*. Pengujian dilakukan dengan dua tahap, yaitu tahap pengujian *alpha* dan tahap pengujian *bet*a.

4.2.1 Pengujian Alpha

Pada tahap ini, pengujian dilakukan untuk memastikan aplikasi permainan *Monster Nest* telah berjalan dengan benar dan sesuai dengan kebutuhan yang diinginkan. Pengujian *alpha* menitikberatkan kepada persyaratan fungsional perangkat lunak.

4.2.1.1 Skenario Pengujian Aplikasi

Skenario pengujian menjelaskan pengujian terhadap sistem yang ada pada aplikasi permainan *Monster Nest.* Skenario pengujian dari aplikasi permainan *Monster Nest* dapat dilihat pada tabel 4.1.

Tabel 4.1 Skenario pengujian aplikasi permainan *monster nest*

No	Komponen yang di uji	Skenario Pengujian	Jenis Pengujian
1	*Menu* Utama	Menekan *menu Play game*	*Black Box*
		Menekan *menu set level*	*Black Box*
		Menekan *menu highscore*	*Black Box*
2	*Set level*	Menekan tombol *easy*	*Black Box*
		Menekan tombol *medium*	*Black Box*
		Menekan tombol *hard*	*Black Box*
3	*Highscore*	Menekan *menu highscore*	*Black Box*
		Mengisi nama pemain	*Black Box*
		Menekan tombol *back*	*Black Box*
4	Layar permainan	Menekan tombol *pause*	*Black Box*
		Menekan tombol ganti senjata	*Black Box*
		Menekan tombol *reload*	*Black Box*
		Menekan tombol tembak	*Black Box*
		Menggerakkan *touchpad* kiri	*Black Box*
		Menggerakkan *touchpad* kanan	*Black Box*
5	Menu *pause*	Menekan tombol *resume*	*Black Box*
		Menekan tombol *start over*	*Black Box*
		Menekan tombol *main menu*	*Black Box*
6	Bermain	Menyentuh *ammo pack*	*Black Box*
		Menyentuh *health pack*	*Black Box*
		Menembak musuh	*Black Box*
		Berpindah *ronde*	*Black Box*
		Player mati	*Black Box*
7	Algoritma SMA* (*Simplified Memory Bounded A star*)	Pencarian jalan	*White Box*
		Pengecekan *node* terdekat	*White Box*

4.2.1.2 Kasus dan Hasil Pengujian (*Black Box*)

Kasus dan hasil uji perangkat lunak yang sudah dibangun dengan menggunakan metode *black box* berdasarkan pengamatan yang dilakukan dan menghasilkan kesimpulan dari pengujian tersebut, di antaranya sebagai berikut :

1. Pengujian *Menu* Utama

Pengujian *menu* utama dengan kasus dan hasil uji, kemudian diamati dan menghasilkan kesimpulan dapat dilihat pada tabel 4.2.

Tabel 4.2 Pengujian menu utama(data benar)

Kasus dan Hasil Uji(Data benar)			
Masukan	Keluaran yang diharapkan	Pengamatan	Kesimpulan
Menekan *menu play game*	Menampilkan petunjuk permainan dan masuk ke dalam permainan	Dapat menampilkan petunjuk permainan dan kemudian masuk ke dalam permainan	[√] Diterima [ ] Ditolak
Menekan *menu set level*	Menampilkan *menu set level*	Dapat menampilkan *menu set level*	[√] Diterima [ ] Ditolak
Menekan *menu highscore*	Menampilkan *menu highscore*	Dapat menampilkan *highscore*	[√] Diterima [ ] Ditolak

Tabel 4.3 Pengujian menu utama (data salah)

Kasus dan Hasil Uji(Data Salah)			
Masukan	Keluaran yang diharapkan	Pengamatan	Kesimpulan
Menekan layar di sembarang tempat	Tidak ada pesan yang muncul	Tidak ada respon	[√] Diterima [ ] Ditolak

2. Pengujian *Set Level*

Pengujian *set level* dengan kasus dan hasil uji, kemudian diamati dan menghasilkan kesimpulan dapat dilihat pada tabel 4.4.

Tabel 4.4 Pengujian *set level* (data benar)

Kasus dan Hasil Uji(Data benar)			
Masukan	Keluaran yang diharapkan	Pengamatan	Kesimpulan
Menekan tombol *easy*	Tingkat kesulitan permainan menjadi mudah	Menampilkan *dialog* konfirmasi dan apabila ditekan tombol *yes* maka kembali ke menu utama, dan tingkat kesulitan permainan menjadi mudah	[√] Diterima [ ] Ditolak
Menekan tombol *medium*	Tingkat kesulitan permainan menjadi menengah	Menampilkan *dialog* konfirmasi dan apabila ditekan tombol *yes* maka kembali ke menu utama, dan tingkat kesulitan permainan menjadi sedang	[√] Diterima [ ] Ditolak
Menekan tombol *hard*	Tingkat kesulitan permainan menjadi sulit	Menampilkan *dialog* konfirmasi dan apabila ditekan tombol *yes* maka kembali ke menu utama, dan tingkat kesulitan permainan menjadi sulit	[√] Diterima [ ] Ditolak

Tabel 4.5 Pengujian *set level* (data salah)

Kasus dan Hasil Uji(Data Salah)			
Masukan	Keluaran yang diharapkan	Pengamatan	Kesimpulan
Menekan layar di sembarang tempat	Tidak ada pesan yang muncul	Tidak ada respon	[√] Diterima [ ] Ditolak

3. Pengujian *Highscore*

Pengujian *highscore* dengan kasus dan hasil uji, kemudian diamati dan menghasilkan kesimpulan dapat dilihat pada tabel 4.6.

Tabel 4.6 Pengujian *highscore* (data benar)

Kasus dan Hasil Uji(Data benar)			
Masukan	Keluaran yang diharapkan	Pengamatan	Kesimpulan
Menekan tombol *highscore*	Menampilkan daftar delapan skor tertinggi	Dapat menampilkan daftar delapan skor tertinggi	[√] Diterima [ ] Ditolak
Memasukkan nama pemain	Nama pemain yang masuk dalam delapan skor tertinggi masuk ke dalam *highscore*	Urutan skor pemain ditampilkan, pemain memasukkan nama, skor pemain masuk ke dalam delapan skor tertinggi	[√] Diterima [ ] Ditolak
Menekan tombol *back*	Kembali ke menu utama	Dapat kembali ke menu utama	[√] Diterima [ ] Ditolak

Tabel 4.7 Pengujian *highscore* (data salah)

Kasus dan Hasil Uji(Data Salah)			
Masukan	Keluaran yang diharapkan	Pengamatan	Kesimpulan
Menekan layar di sembarang tempat	Tidak ada pesan yang muncul	Tidak ada respon	[√] Diterima [ ] Ditolak
Player tidak meng-*input*-kan nama	Nama tidak kosong	Nama terisi dengan nama *default*	[√] Diterima [ ] Ditolak

4. Pengujian Layar Permainan

Pengujian layar permainan dengan kasus dan hasil uji, kemudian diamati dan menghasilkan kesimpulan dapat dilihat pada tabel 4.8.

Tabel 4.8 Pengujian layar permainan (data benar)

Kasus dan Hasil Uji(Data benar)			
Masukan	Keluaran yang diharapkan	Pengamatan	Kesimpulan
Menekan tombol *pause*	Permainan berhenti dan masuk ke menu *pause*	Permainan dapat berhenti dan menu *pause* dimunculkan	[√] Diterima [ ] Ditolak
Menekan tombol ganti senjata	Senjata *player* dalam permainan diganti dengan senjata yang berbeda	Bentuk senjata berubah dan indikator senjata di kanan atas layar berubah	[√] Diterima [ ] Ditolak
Menekan tombol *reload*	Jumlah peluru menjadi penuh kembali	Jumlah peluru pda indikator senjata di kanan atas menjadi penuh	[√] Diterima [ ] Ditolak
Menekan tombol tembak	*Player* dalam permain melakukan tembakan	Terdengar suara tembakan lalu ada animasi percikan api yang muncul dan peluru berkurang	[√] Diterima [ ] Ditolak
Menggerakkan *touchpad kiri*	Player bergerak sesuai dengan pergerakan *touchpad*	*Player* bergerak ke kanan apabila *touchpad* digerakkan ke kanan, *player* bergerak ke kiri apabila *touchpad* digerakkan ke kiri, *player* bergerak ke depan jika *touchpad* digerakkan ke	[√] Diterima [ ] Ditolak

Kasus dan Hasil Uji(Data benar)			
Masukan	Keluaran yang diharapkan	Pengamatan	Kesimpulan
		atas, *player* bergerak ke belakang jika *touchpad* digerakkan ke bawah	
Menggerakkan *touchpad* kanan	Layar bergerak sesuai dengan pergerakan *touchpad*	Layar bergerak ke kanan apabila *touchpad* digerakkan ke kanan, *player* bergerak ke kiri apabila *touchpad* digerakkan ke kiri, *player* bergerak ke depan jika *touchpad* digerakkan ke atas, *player* bergerak ke belakang jika *touchpad* digerakkan ke bawah	[√] Diterima [ ] Ditolak

Tabel 4.9 Pengujian layar permainan (data salah)

Kasus dan Hasil Uji(Data Salah)			
Masukan	Keluaran yang diharapkan	Pengamatan	Kesimpulan
Menekan layar di sembarang tempat	Tidak ada pesan yang muncul	Tidak ada respon	[√] Diterima [ ] Ditolak

5. Pengujian Menu *Pause*

Pengujian menu *pause* dengan kasus dan hasil uji, kemudian diamati dan menghasilkan kesimpulan dapat dilihat pada tabel 4.10.

Tabel 4.10 Pengujian menu *pause*

Kasus dan Hasil Uji(Data benar)			
Masukan	Keluaran yang diharapkan	Pengamatan	Kesimpulan
Menekan tombol *resume*	Permainan kembali seperti sebelum menekan tombol *pause*	Menu *pause* hilang dan permainan kembali seperti sebelumnya	[√] Diterima [ ] Ditolak
Menekan tombol *start over*	Permainan mengulang dari awal	Menu *pause* hilang dan mengulang dari awal dengan jumlah skor nol	[√] Diterima [ ] Ditolak
Menekan tombol *main menu*	Kembali ke *main menu*	Keluar dari permainan dan kembali ke *main menu*	[√] Diterima [ ] Ditolak

Tabel 4.11 Pengujian *set level* (data salah)

Kasus dan Hasil Uji(Data Salah)			
Masukan	Keluaran yang diharapkan	Pengamatan	Kesimpulan
Menekan layar di sembarang tempat selain tombol	Tidak ada pesan yang muncul	Tidak ada respon	[√] Diterima [ ] Ditolak

6. Pengujian Bermain

Pengujian bermain dengan kasus dan hasil uji, kemudian diamati dan menghasilkan kesimpulan dapat dilihat pada tabel 4.12.

Tabel 4.12 Pengujian bermain

Kasus dan Hasil Uji(Data benar)			
Masukan	Keluaran yang diharapkan	Pengamatan	Kesimpulan
Player menyentuh *ammo pack*	Jumlah amunisi senjata menjadi bertambah	Jumlah amunisi pada indikator senjata di kanan atas bertambah satu	[√] Diterima [ ] Ditolak
Player menyentuh *health pack*	Jumlah darah *player* terisi kembali	Darah pada indikator darah terisi penuh	[√] Diterima [ ] Ditolak
Waktu habis	*Ronde* bertambah	Ditampilkan tulisan *get ready* lalu karakter dipindahkan ke tempat lain dan jumlah *ronde* bertambah	[√] Diterima [ ] Ditolak
Darah *player* habis	Keluar dari permainan	karakter melakukan animasi mati kemudian keluar dari permainan dan menuju *highscore*	[√] Diterima [ ] Ditolak

4.2.1.3 Kasus dan Pengujian (*White Box*)

Di bawah ini merupakan kasus algoritma SMA* (*Simplified Memory Bounded* A *star*) untuk menguji perangkat lunak yang sudah dibangun dengan menggunakan metode *white box*, berdasarkan skenario pengujian aplikasi yang sudah dijabarkan.

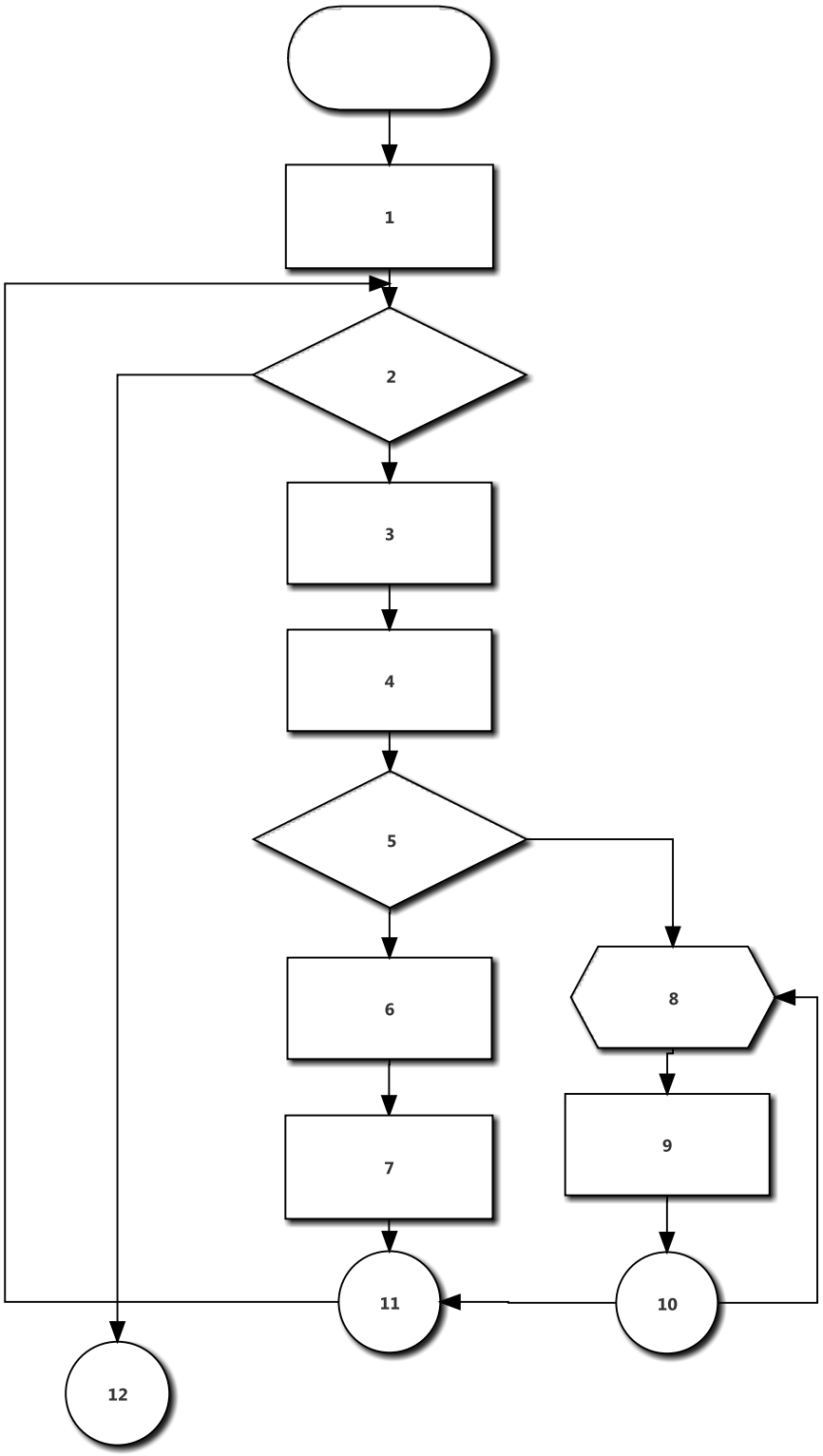
Pengujian Algoritma SMA* merupakan proses di mana akan diuji terhadap musuh dalam melakukan pencarian rute terpendek yang terdapat di dalam permainan *Monster Nest*.

1. Prosedur pencarian jalan

Berikut adalah *psudeocode* prosedur pencarian jalan.

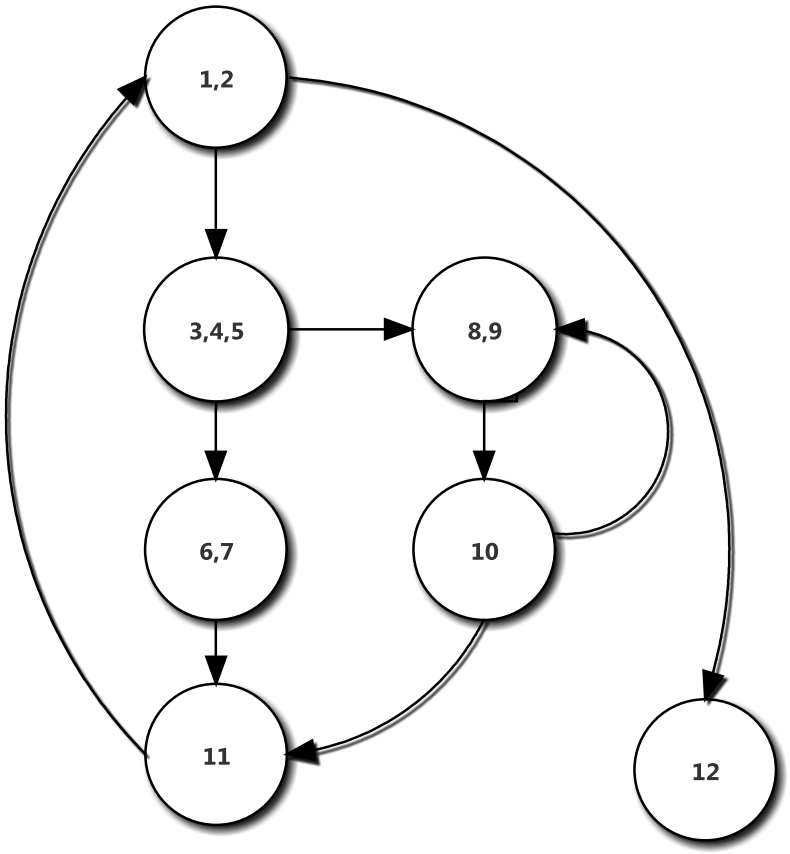
```
1 :   openList.Add(nodeAwal);
2 :   while(foundTarget==false){
3 :         checkingNode = GetSmallestFValueNode();
4 :         pindah checkingNode ke closedList
5 :         if (checkingNode == targetNode)
6 :         then foundTarget = true;
7 :               TraceBackPath(musuh);
8 :         else for (var node : PathfindNode in checkingNode.dekat)
9 :               cekNode(checkingNode, node);
10:          endfor
11:         endif
12:   endwhile
```

Berdasarkan *psudeocode* prosedur pencarian jalan, dibuat *flowchart* yang bisa dilihat pada gambar 4.9.



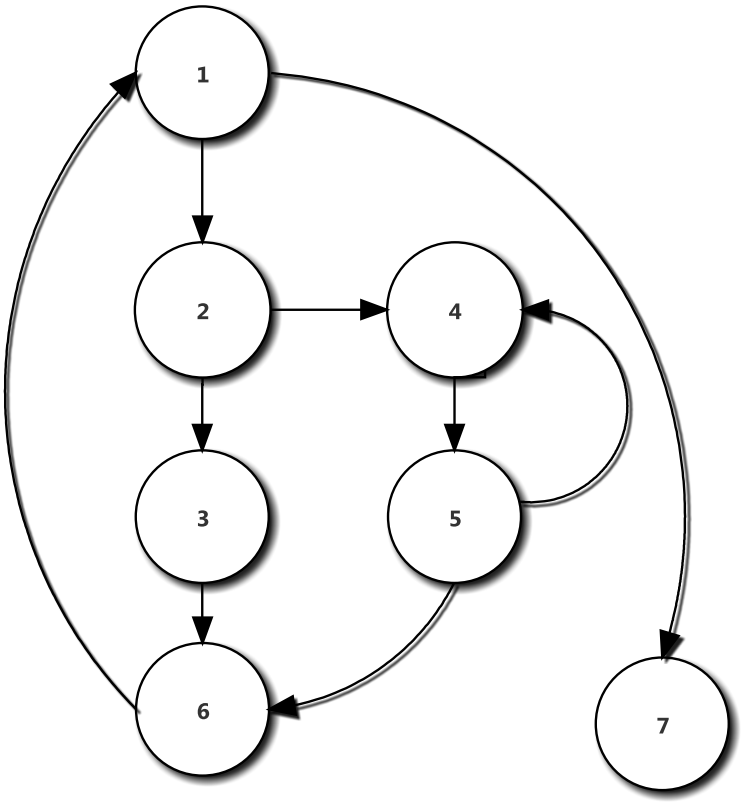
Gambar 4.9 *Flowchart* prosedur pencarian jalan

Flowgraph yang merupakan hasil dari perubahan *flowchart* gambar IV.9 dapat dilihat pada gambar 4.10.



Gambar 4.10 *Flowgraph* prosedur pencarian jalan

Atau disederhanakan berdasarkan kondisi atau simpul sebagai berikut:



Gambar 4.11 Penyederhanaan *Flow Graph* pencarian jalan

Jumlah dari *cyclomatix complexity* atau V(G) dari flowgraph pada gambar IV.10 adalah sebagai berikut :

Edge = 10, *Node* = 8

V(G) = *Edge – Node* + 2

V(G) = 10 – 8 + 2

V(G) = 2 + 2

V(G) = 4

Dari gambar IV.10 didapat *independent path* sebagai berikut :

Path 1 = 1, 2, 12

Path 2 = 1, 2, 3, 4, 5, 6, 7, 11, 2, 12

Path 3 = 1, 2, 3, 4, 5, 8, 9, 10, 11, 2, 12

Path 4 = 1, 2, 3, 4, 5, 8, 9, 10, 8, 9, 11, 2, 12

Dengan menggunakan rumus *predicate node* didapatkan hasil sebagai berikut:

V(G) = Jumlah *node* yang memiliki lebih dari 1 jalur + 1

V(G) = 3 + 1

V(G) = 4

Graph Matriks pengujian algoritma SMA* prosedur pencarian jalan dapat dilihat pada tabel 4.13.

Tabel 4.13 *Graph* matriks algoritma SMA* pencarian jalan

	1	2	3	4	5	6	7	
1		1					1	1
2			1	1				1
3						1		
4					1			
5				1		1		1
6	1							
7								
Hasil								3

V(G) = Jumlah *Graph Matriks* + 1

V(G) = 3 + 1

V(G) = 4

Kesimpulan yang didapatkan berdasarkan hasil pengujian dihasilkan nilai *Cyclomatic Complexity* yang sama yaitu 4, maka dapat disimpulkan bahwa pada proses pencarian dengan algoritma SMA* pada prosedur pencarian jalan berjalan dengan baik, karena setiap pengujian menghasilkan nilai yang sama.

Hasil Uji Algoritma SMA* untuk pencarian jalan:

Tabel 4.15 Hasil Uji Algoritma SMA* pencarian jalan

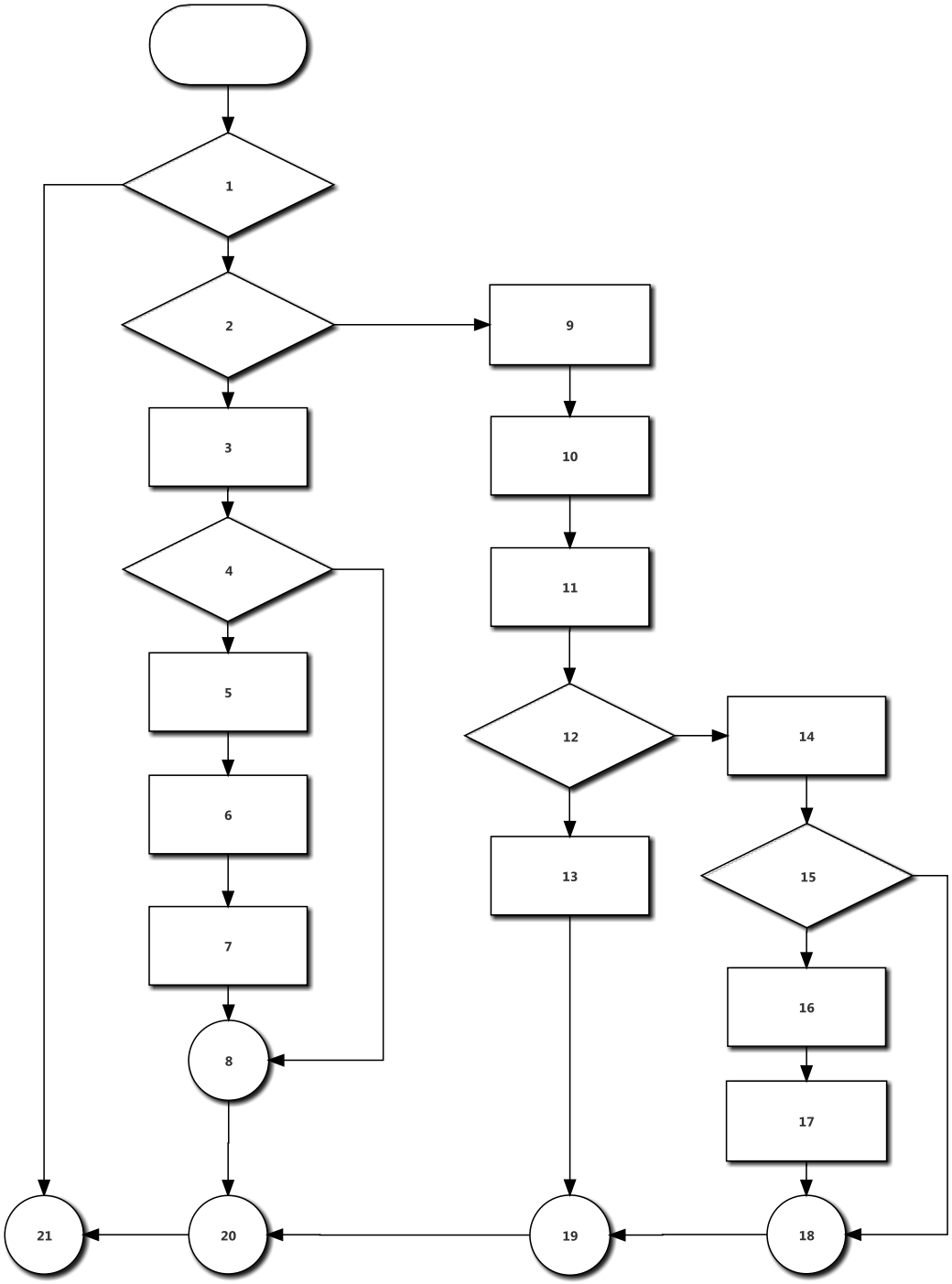
No	Kasus uji	Yang diharapkan	Pengamatan	Keterangan
1	foundTarget = true	Pencarian jalan gagal dilakukan	Pencarian jalan tidak dilakukan	[√] Diterima [ ] Ditolak
2	• foundTarget = false • CheckingNode = target node	Pencarian jalan dilakukan dan fungsi TraceBackPath dilakukan	Pencarian jalan dilakukan dan fungsi TraceBackPath dilakukan	[√] Diterima [ ] Ditolak
3	• foundTarget = false • Checking node != target node	Pencarian jalan dilakukan dan pengecekan *node* dilakukan	Pencarian jalan dilakukan dan pengecekan *node* dilakukan	[√] Diterima [ ] Ditolak
4	• foundTarget = false • Checking node != target node • *Node* di checkingNode .dekat masih ada	Pengecekan *node* dilakukan lagi sebanyak jumlah *node* dekat	Pengecekan *node* dilakukan lagi sebanyak *node* dekat	[√] Diterima [ ] Ditolak

2. Prosedur pengecekan *node* terdekat

Berikut adalah *psudeocode* prosedur pengecekan *node* terdekat.

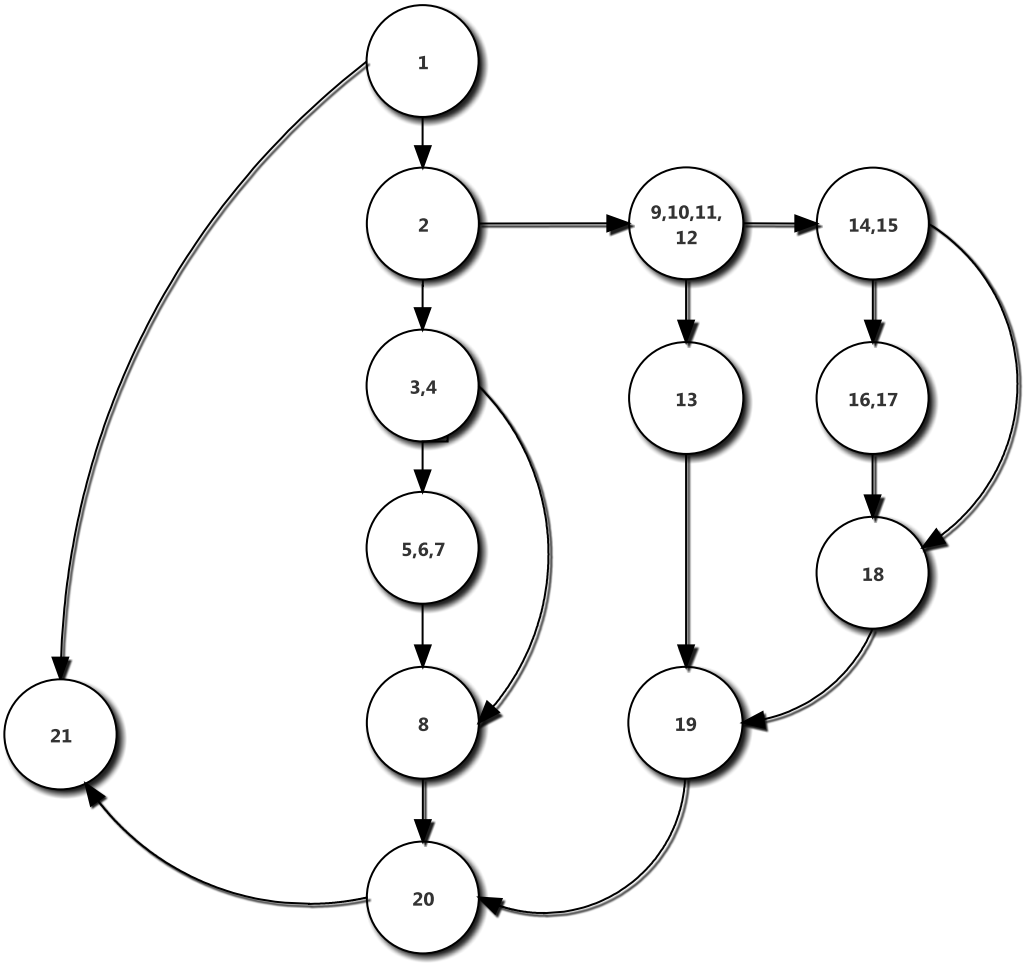
```
1 :   if (closedList.Contains(testing) == false)
2 :   then if (openList.Contains(testing) == true)
3 :       then hitung G cost baru
4 :           if (newGCost < testing.getGValue())
5 :           then testing.setParent(currentNode);
6 :                   testing.setGValue(newGCost);
7 :                   testing.CalculateFValue();
8 :           endif
9 :       else testing.setParent(currentNode);
10:           testing.setGValue(currentNode.getGValue()+baseMovementCost);
11:           testing.CalculateFValue();
12:           if(openList.Count<limit)
13:           then AddToOpenList(testing);
14:           else var test : PathfindNode =GetBiggestFValueNode();
15:                   if(testing.getFValue()<=test.getFValue())
16:                   then openList.Remove(test);
17:                           AddToOpenList(testing);
18:                   endif
19:           endif
20:       endif
21:   endif
```

Berdasarkan *psudeocode* prosedur pengecekan *node* terdekat, dibuat *flowchart* yang bisa dilihat pada gambar 4.12

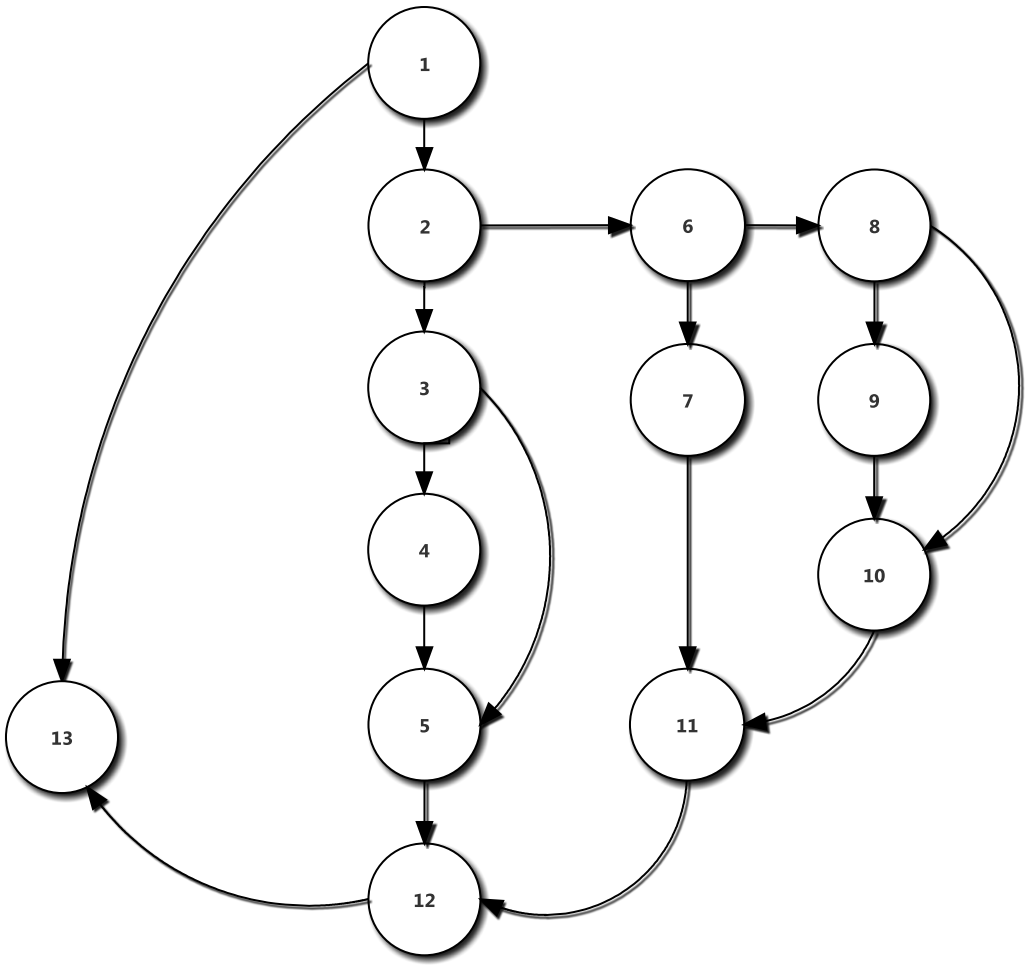


Gambar 4.12 *Flowchart* prosedur pengecekan *node* terdekat

Flowgraph yang merupakan hasil dari perubahan *flowchart* gambar 4.12 dapat dilihat pada gambar 4.13.



Gambar 4.13 *Flowgraph* prosedur pengecekan *node* terdekat



Gambar 4.14 Penyederhanaan *Flow Graph* Pengecekan *node*

Jumlah dari *cyclomatix complexity* atau V(G) dari *flowgraph* pada gambar 4.13 adalah sebagai berikut :

Edge = 16, *Node* = 13

V(G) = *Edge – Node* + 2

V(G) = 17 – 13 + 2

V(G) = 4 + 2

V(G) = 6

Dari gambar 4.13 didapat *independent path* sebagai berikut :

Path 1 = 1, 20

Path 2 = 1, 2, 3, 4, 8, 20, 21

Path 3 = 1, 2, 3, 4, 5, 6, 7, 8, 20, 21

Path 4 = 1, 2, 9, 10, 11, 12, 13, 19, 20, 21

Path 5 = 1, 2, 9, 10, 11, 12, 14, 15, 18, 19, 20, 21

Path 6 = 1, 2, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21

Dengan menggunakan rumus *predicate node* didapatkan hasil sebagai berikut:

V(G) = Jumlah *node* yang memiliki lebih dari 1 jalur + 1

V(G) = 5 + 1

V(G) = 6

Graph Matriks pengujian algoritma SMA* prosedur pengecekan *node* terdekat dapat dilihat pada tabel 4.14.

Tabel 4.14 *Graph matriks* algoritma SMA* pengecekan *node* terdekat

	1	2	3	4	5	6	7	8	9	10	11	12	13	
1		1												
2			1			1								1
3				1	1									1
4					1			1						1
5												1		
6							1	1						1
7											1			
8									1	1				1
9										1				
10											1			
11												1		
12													1	
13														
hasil														5

V(G) = Jumlah *Graph Matriks* + 1

V(G) = 5 + 1

V(G) = 6

Kesimpulan yang didapatkan berdasarkan hasil pengujian dihasilkan nilai *Cyclomatic Complexity* yang sama yaitu 6, maka dapat disimpulkan bahwa pada proses pencarian dengan algoritma SMA* pada prosedur pengecekan *node* terdekat berjalan dengan baik, karena setiap pengujian menghasilkan nilai yang sama.

Hasil Uji Algoritma SMA* untuk pengecekan node terdekat:

Tabel 4.15 Hasil Uji Algoritma SMA* pengecekan *node* terdekat

No	Kasus uji	Yang diharapkan	Pengamatan	Keterangan
1	closedList.Contains(testing) = true	Pengecekan *node* tidak dilakukan	Pengecekan *node* gagal dilakukan	[√] Diterima [ ] Ditolak
2	• closedList.Contains(testing) = false • openList.Contains(testing) = true • newGCost > testing.getGValue()	Hanya dilakukan perhitungan nilai g *cost* baru	Hanya dilakukan perhitungan nilai g *cost* baru	[√] Diterima [ ] Ditolak
3	• closedList.Contains(testing) = false • openList.Contains(testing) = true • newGCost < testing.getGValue()	Dilakukan perhitungan nilai g *cost* baru dan dilakukan penggantian *parent* dengan g *cost* yang lebih baik	Perhitungan nilai g *cost* baru dilakukan dan penggantian *parent* dengan g *cost* yang lebih baik	[√] Diterima [ ] Ditolak

No	Kasus uji	Yang diharapkan	Pengamatan	Keterangan
4	• closedList.Contains(testing) = false • openList.Contains(testing) = false • openList.Count < limit	Dilakukan perhitungan nilai g dan f serta pemindahan *node* ke *open list*	Terjadi pemindahan *node* ke *open list* dengan perhitungan nilai g dan f-nya	[√] Diterima [ ] Ditolak
5	• closedList.Contains(testing) = false • openList.Contains(testing) = false • openList.Count > limit • testing.getFValue() > test.getFValue()	Dilakukan perhitungan nilai g dan f serta mengambil *node* dengan nilai terburuk di *open list*	Dilakukan perhitungan nilai g dan f serta mengambil *node* dengan nilai terburuk di *open list*	[√] Diterima [ ] Ditolak
6	• closedList.Contains(testing) = false • openList.Contains(testing) = false • openList.Count > limit • testing.getFValue() <= test.getFValue()	Dilakukan perhitungan nilai g dan f serta mengambil *node* dengan nilai terburuk di *open list* lalu dilakukan penghapusan *node* dengan nilai terburuk	Dilakukan perhitungan nilai g dan f serta mengambil *node* dengan nilai terburuk di *open list* lalu dilakukan penghapusan *node* dengan nilai terburuk	[√] Diterima [ ] Ditolak

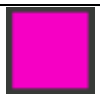
4.2.1.4 Kesimpulan Pengujian *Alpha*

Dari hasil pengujian yang dilakukan, dapat disimpulkan bahwa sistem sudah berjalan seperti yang diharapkan dan secara fungsional sudah dapat menghasilkan keluaran yang diharapkan.

4.2.2 Pengujian SMA* dan A* dengan Program Simulasi

Pengujian dilakukan dengan membuat program simulasi yang terpisah dari *game*, program ini terdiri dari *node* - *node* dengan keterangan yang bisa dilihat pada tabel 4.15:

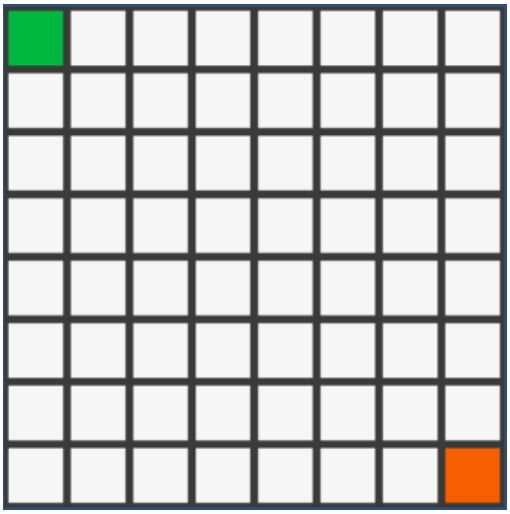
Tabel 4.15 Keterangan *node* program simulasi

No	Node	Keterangan
1		titik awal
2		titik tujuan
3		*node* yang bisa dilalui
4		penghalang yang tidak bisa dilalui
5		*node* yang berada di *closed list*
6		*node* yang berada di *open list*
7		*Node* yang berada di *closed list* dan jalan yang berhasil ditemukan melalui hasil pencarian

Program ini digunakan untuk melihat penggunaan memori pada SMA* dan A* dengan melihat jumlah dari *open list* dan *closed list* yang dihasilkan pada pencarian jalan. Pengujian yang akan dilakukan terdiri dari beberapa kasus, yaitu:

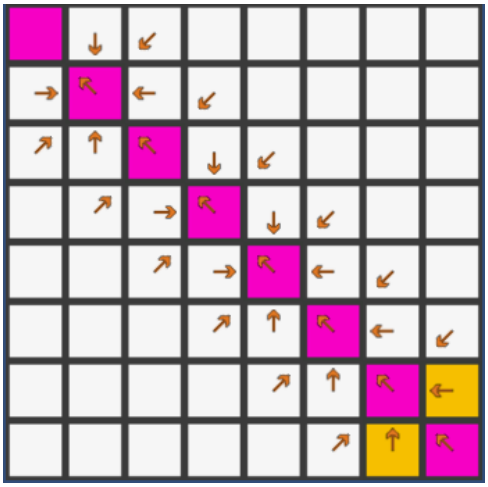
a. Kasus 1

Ordo 8x8 tanpa penghalang



Gambar 4.15 kasus 1

Dengan algoritma SMA* dengan *limit* 3

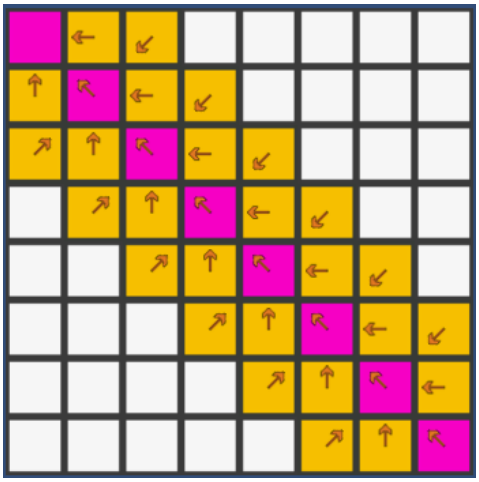


Gambar 4.16 kasus 1 dengan algoritma SMA* *limit* 3

Didapatkan:

- *Open list* = 2
- *Closed list* = 8

Dengan algoritma A*



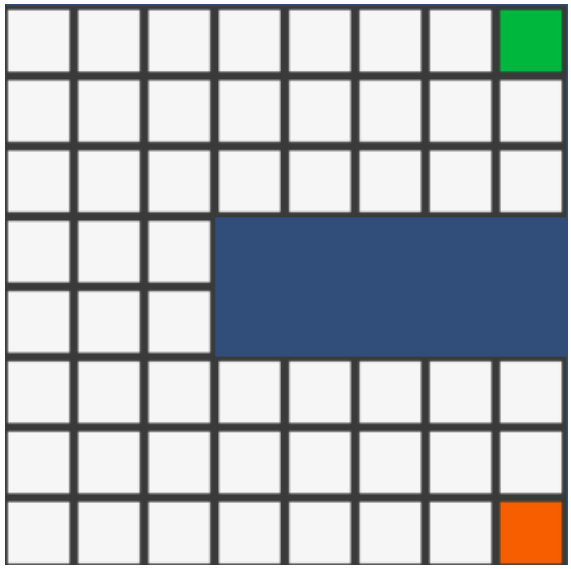
Gambar 4.17 kasus 1 dengan A*

Didapatkan:

- *Open list* = 26
- *Closed list* = 8

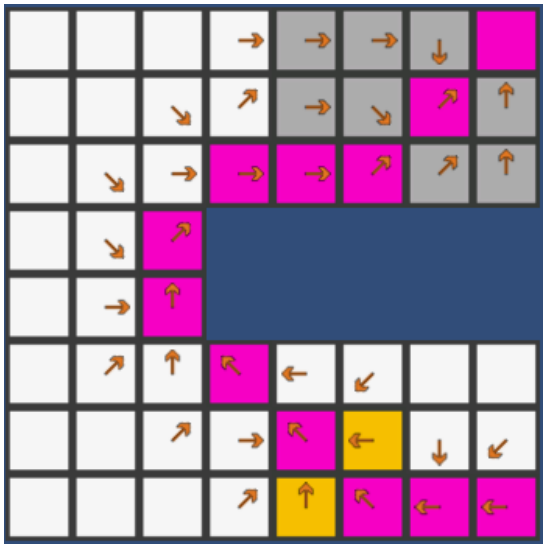
b. Kasus 2

Ordo 8x8 dengan penghalang



Gambar 4.18 kasus 2

Dengan algoritma SMA* dengan limit 3

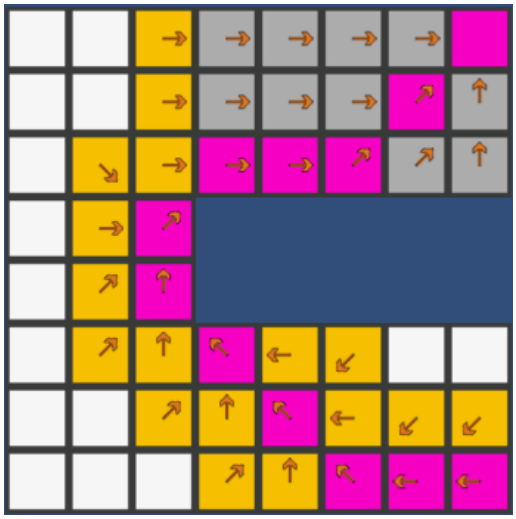


Gambar 4.19 kasus 2 dengan SMA* *limit* 3

Didapatkan

- *Open list* = 2
- *Closed list* = 20

Dengan algoritma A*



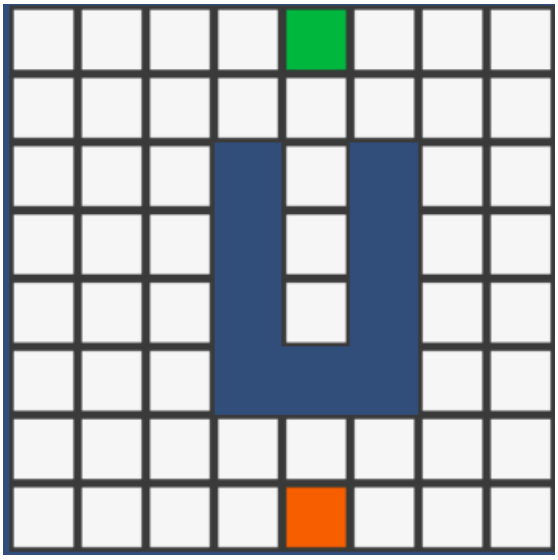
Gambar 4.20 kasus 2 dengan A*

Didapatkan

- *Open list* = 17
- *Closed list* = 22

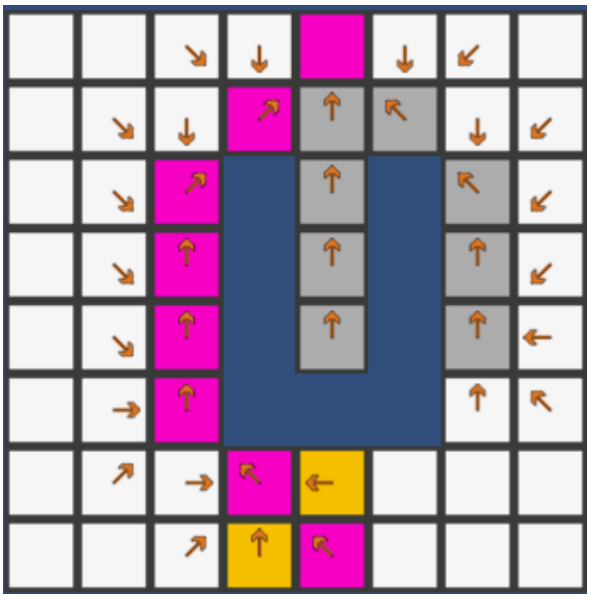
c. Kasus 3

Ordo 8x8 dengan jebakan lebar 1 *node*



Gambar 4.21 kasus 3

Dengan algoritma SMA*

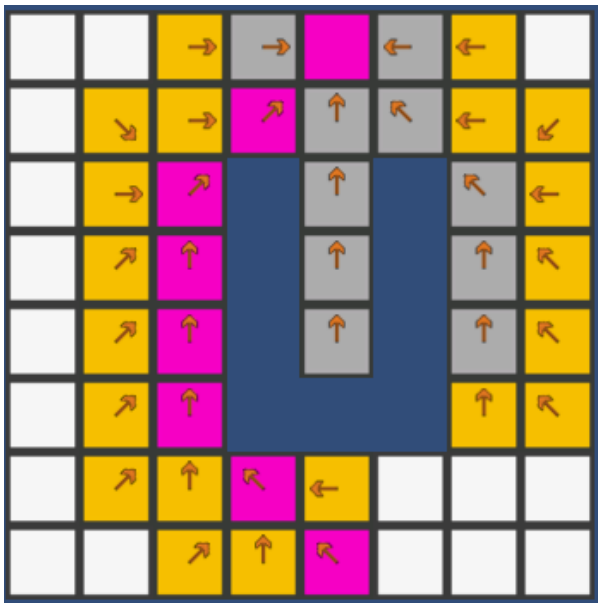


Gambar 4.22 kasus 3 dengan SMA* *limit* 3

Didapatkan

- *Open list* = 2
- *Closed list* =16

Dengan algoritma A*



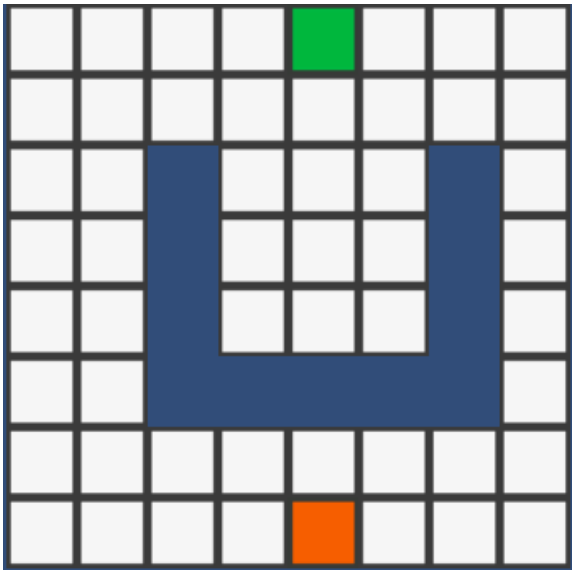
Gambar 4.23 kasus 3 dengan A*

Didapatkan

- *Open list* = 20
- *Closed list* = 18

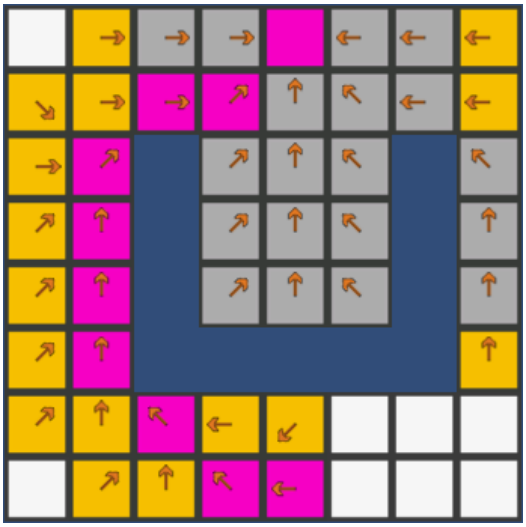
d. Kasus 4

Ordo 8x8 dengan jebakan lebar 3 *node*



Gambar 4.24 kasus 4

Dengan algoritma A*

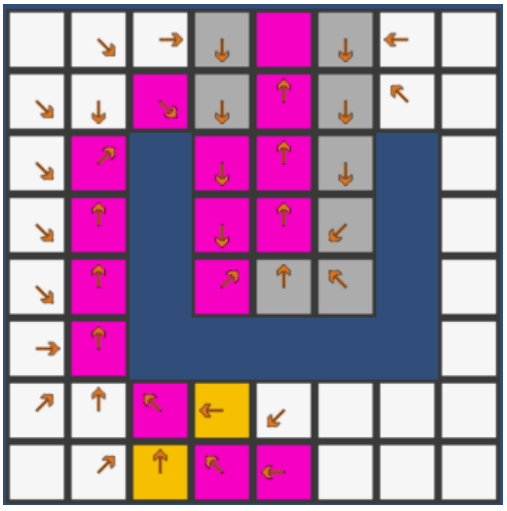


Gambar 4.25 kasus 4 dengan A*

Didapatkan

- *Open list* = 16
- *Closed list* = 29

Dengan algoritma SMA* *limit* 3

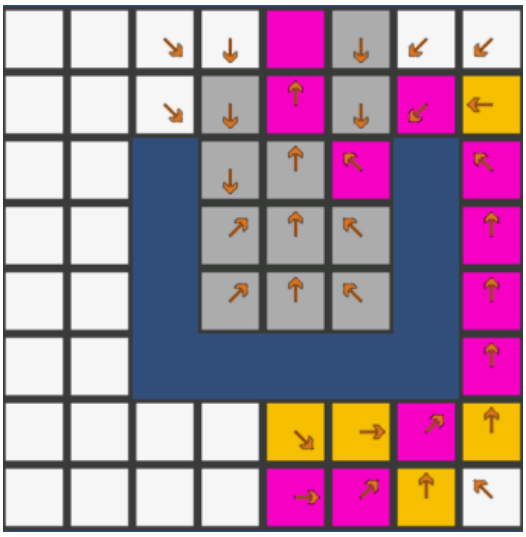


Gambar 4.26 kasus 4 dengan SMA* *limit* 3

Didapatkan:

- *Open list* = 2
- *Closed list* =23

Dengan algoritma SMA* *limit* 6

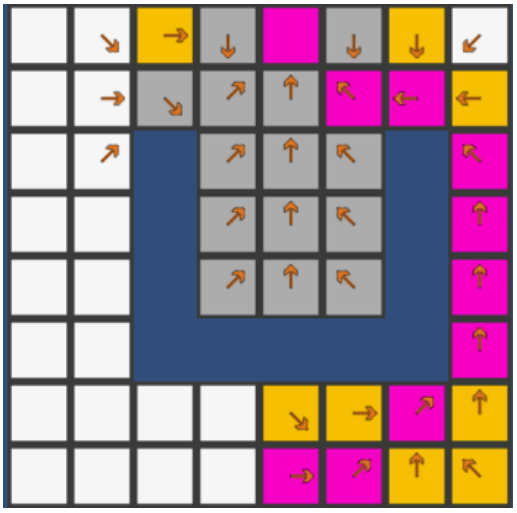


Gambar 4.27 kasus 4 dengan SMA* *limit* 6

Didapatkan:

- *Open list* = 5
- *Closed list* = 22

Dengan algoritma SMA* dengan *limit* 9

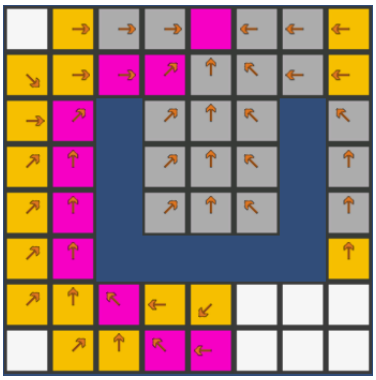


Gambar 4.28 kasus 4 dengan SMA* *limit* 9

Didapatkan:

- *Open list* = 8
- *Closed list* = 24

Dengan algoritma SMA* dengan *limit* 17



Gambar 4.29 Kasus 4 dengan SMA* *limit* 17

Didapatkan:

- *Open list* = 16
- *Closed list* = 29

Hasil pengujian dari kasus 1 sampai 4 dapat dilihat pada tabel 4.16.

Tabel 4.16 Hasil pengujian program simulasi

Kasus	**Algoritma**	***Open list***	***Closed list***	**Jumlah**	**F-*Cost***
1	SMA* *limit* 3	2	8	10	98
	A*	26	8	34	98
2	SMA* *limit* 3	2	20	22	134
	A*	17	22	39	134
3	SMA* *limit* 3	2	16	18	86
	A*	20	18	38	86
4	A*	16	29	44	106
	SMA* *limit* 3	2	23	25	160
	SMA* *limit* 6	5	22	27	120
	SMA* *limit* 9	8	24	32	106
	SMA* *limit* 17	16	29	45	106

4.2.2.1 Kesimpulan Pengujian SMA* dan A* dengan Program Simulasi

Dari pengujian yang sudah dilakukan dapat dilihat bahwa algoritma SMA* menggunakan memori lebih sedikit dibandingkan dengan A*, ini dapat dilihat dari jumlah *open list* dan *closed list* hasil algoritma SMA*, hal ini terlihat jelas pada kasus 1. Namun algoritma SMA* tidak selalu optimal dalam pencarian jika *limit* yang diberikan pada *open list* tidak mencukupi, terlihat pada kasus 4 algoritma SMA* dengan *limit* 3, 6, dan 9 yang menghasilkan nilai F-*Cost* lebih besar dibandingkan A*.