BAB 4
IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi

Tahap implementasi merupakan tahap penciptaan perangkat lunak kedalam bahasa pemograman yang dimengerti oleh komputer, serta merupakan tahap kelanjutan dari kegiatan perancangan sistem. Aplikasi ini diimplementasikan dengan nama Selamatkan orangutan dengan *file extension*-nya *.exe*.

4.1.1. Implementasi Perangkat Keras

Perangkat keras yang akan digunakan untuk menjalankan aplikasi *game* Selamatkan orangutan ini tidak harus komputer yang berspesifikasi tinggi. Adapun spesifikasi perangkat keras yang digunakan saat implementasi dan pengujian aplikasi *game* Selamatkan orangutan adalah sebagai berikiut :

1. Prosesor yang setara dengan kecepatan 2.0 GHz.
2. RAM 2 GB.
3. *Harddisk* dengan ruang kosong 3 GB.
4. VGA *Card* 512 MB.
5. Monitor dengan *resolusi* 1024 x 768 *pixel.*
6. *Speaker.*
7. *Keyboard* dan *mouse.*

4.1.2. Implementasi Perangkat Lunak

Spesifikasi perangkat lunak yang dibutuhkan untuk menjalankan *Game* Selamatkan orangutan yaitu sistem operasi *Windows* 7 dan *direct*X 9 untuk mendukung performa grafik untuk menjalankan aplikasi.

4.1.3. Implementasi Aplikasi

G*ame* Selamatkan orangutan merupakan sebuah *game* ber-*genre Third Person Shooter* (TPS) yang terdiri dari beberapa *Level* yang dimana pemain harus menyelesaikan misi di setiap *Level*-nya untuk dapat melanjutkan ke *Level*-*Level* berikutnya. G*ame* ini tidak melakukan proses instalasi, cukup dengan melakukan

double-klik pada *icon* aplikasi ini. Pemain terlebih harus mempunyai *file* Selamatkan Orangutan.*exe* . Untuk mulai memainkan *game*, pemain hanya perlu menjalankan *file* Selamatkan Orangutan.*exe* yang telah ada didalam perangkat keras.

4.1.4. Implementasi Antarmuka

Adapun implementasi antarmuka pada aplikasi *game* Selamatkan orangutan adalah sebagai berikut :

4.1.4.1. Antarmuka *Icon Dekstop*

Antaramuka *Icon Dekstop* merupakan tampilan *icon* pada *game* selamatkan orangutan, berikut ini tampilannya :



Gambar 4.1 *Icon Dekstop game* selamatkan orangutan

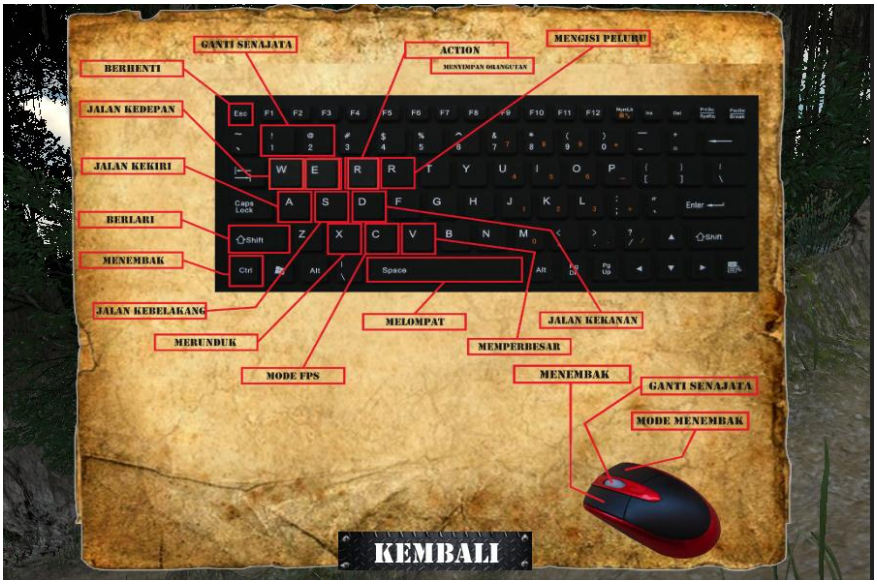
4.1.4.2. Antarmuka *Menu* Utama

Antarmuka *menu* utama merupakan tampilan *menu* utama aplikasi *game* setelah pengguna mengklik tampilan *icon* pada *dekstop*. Terdapat 3 *menu* dapat dilihat pada gambar berikut ini.



Gambar 4.2 Tampilan Antarmuka *Menu* Utama

Pada tampilan *menu* utama jika pengguna mengklik *menu* petunjuk maka akan tampil seperti pada gambar berikut ini.



Gambar 4.3 Tampilan Antarmuka Petunjuk

Antarmuka petunjuk merupakan tampilan petunjuk atau panduan bermain permainan. Pengguna bisa mengklik *mouse* pada setiap huruf untuk memperjelas *input* masukan yang ada pada permainan. Pada tampilan petunjuk pengguna dapat kembali ke *menu* utama dengan mengklik *menu* Kembali . Selanjunya pengguna

dapat memilih *menu* keluar pada pilihan *menu* utama. *Menu* keluar berfungsi untuk mengeluarkan pengguna dari layar permainan, berikut ini antarmuka untuk *menu* keluar dimana terdapat pesan yaitu Ya dan Tidak untuk dipilih pengguna.



Gambar 4.4 Tampilan Antarmuka *Menu* Keluar

4.1.4.3. Antarmuka Tampilan Petunjuk Misi

Antarmuka tampilan petunjuk misi merupakan tampilan yang akan ditampilkan pada saat pengguna mengklik *menu* main pada *menu* utama. Petunjuk misi ini berguna untuk petunjuk misi bagi pengguna dan menampilkan keterangan kondisi orangutan dimana pengguna dapat melihat dengan cara meng-klik gambar yang ada pada layar tampilan petunjuk. Dalam tampilan petunjuk misi terdapat menu mulai untuk memainkan permainan. Berikut ini antarmuka tampilan petunjuk misi pada setiap *level*-nya



Gambar 4.5 Tampilan Antarmuka Petunjuk Misi *Level* 1



Gambar 4.6 Tampilan Antarmuka Petunjuk Misi *Level* 2



Gambar 4.7 Tampilan Antarmuka Petunjuk Misi *Level* 3

4.1.4.4. Antarmuka Mulai Permainan

Antarmuka mulai permainan merupakan tampilan yang akan ditampilkan ketika pengguna mengklik tombol mulai pada tampilan petunjuk misi *level* 1, pengguna akan berada di *Level* 1 permainan yang ditempatkan disebuah hutan, dimana pengguna dapat mulai menggerakan karakter menggunakan *keyboard* dan *mouse* untuk menemukan perangkap dan menemukan sebuah desa. berikut adalah tampilan *level 1* permainan.



Gambar 4.8 Tampilan Antarmuka *Level* 1 Permainan

Setelah berhasil menyelesaikan misi pada *Level* 1 pengguna dapat melanjukan ke *Level* 2 dan dapat dilihat pada berikut ini.



Gambar 4.9 Tampilan Antarmuka *Level* 2 Permainan

Setelah berhasil menyelesaikan misi pada *Level* 2 pengguna dapat melanjukan ke *Level* 3.



Gambar 4.10 Tampilan Antarmuka *Level* 3 Permainan

4.2. Pengujian

Pengujian dilakukan betujuan untuk menemukan kesalahan kekurangan pada perangkat lunak yang diuji. Pengujian tersebut dimaksudkan untuk mengetahui perangkat lunak yang dibuat sudah memenuhi kriteria yang sesuai dengan tujuan perancangan perangkat lunak.

Pengujian dilakukan dengan dua macam teknik yaitu pengujian *white box* dan Pengujian *black box*. Adapun pengujian *black box* lebih berfokus pada menemukan kesalahan program, sedangkan pengujian *white box* berfokus pada struktur internal (*source code*) program, dimana setiap baris kode yang akan di-*compile* dicek satu persatu untuk mengetahui apakah masih terjadi *error*. Cara pengujian dilakukan melalui dua tahap, yaitu tahap pengujian *alpha* dan tahap pengujian *beta.*

4.2.1. Pengujian *Alpha*

Pada tahap *alpha*, pengujian dilakukan untuk memastikan *game* Selamatkan orangutan dapat berjalan sesuai dengan kebutuhan dan tujuan yang diharapkan. Pendekatan perancangan kasus uji menggunakan dua macam teknik yaitu *white box* dan *black box*.

4.2.1.1. Skenario Pengujian Aplikasi

Skenario pengujian menjelaskan pengujian terhadap sistem yang ada pada *game* Selamatkan orangutan. Berikut ini merupakan tabel skenario pengujian dari *game* Selamatkan orangutan.

Tabel 4.1 Skenario Pengujian

No	Komponen yang diuji	Skenario Pengujian	Jenis Pengujian
1	*Menu* Utama	Klik *Menu* Main	*Black Box*
		Klik *Menu* Petunjuk	*Black Box*
		Klik *Menu* Keluar	*Black Box*
2	*Menu* Mulai	Klik *Menu* Mulai	*Black Box*
3	*Pause Menu*	Tampil *Pause Menu*	*Black Box*
		Klik *Menu* Kembali	*Black Box*

		Klik *Menu* Ulangi	*Black Box*
		Klik *Menu Main Menu*	*Black Box*
		Klik *Menu* Petunjuk	*Black Box*
		Klik *Menu* Keluar	*Black Box*
5	Algoritma A*(A *Star*)	Implementasi Algoritma A* Pada musuh untuk Melakukan Pencarian Terhadap Karakter Pemain	*White Box*
6.	Algoritma *Fuzzy Logic*	Implementasi Algoritma *Fuzzy Logic* pada musuh untuk melakukan pergerakan keputusan	*White Box*

4.2.1.2. Kasus dan Hasil Pengujian (*Black Box*)

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

1. **Pengujian *Menu* Utama**

Pengujian menu utama merupakan proses dimana pengguna menyentuh menu-menu yang ada pada tampilan *menu* utama. Adapun hasil dari proses pengujian dapat dilihat pada tabel berikut ini.

Tabel 4.2 Pengujian *Menu* Utama

Kasus dan Hasil Uji			
Masukan	**Keluaran yang diharapkan**	**Pengamatan**	**Kesimpulan**
Klik *Menu* Main	Sistem melakukan proses untuk menampilkan tampilan misi	Sistem menampilkan tampilan misi permainan	[√] Berhasil [ ] Tidak Berhasil

	permainan		
Klik *Menu* Petunjuk	Sistem melakukan proses untuk menampilkan halam petunjuk	Sistem menampilkan halaman petunjuk	[√] Berhasil [ ] Tidak Berhasil
Klik *Menu* Keluar	Sistem melakukan proses untuk menutup aplikasi	Sistem menutup aplikasi	[√] Berhasil [ ] Tidak Berhasil

2. Pengujian *Menu* Main

Pengujian *menu* main merupakan proses setelah pengguna mengklik *menu* main pada *menu* utama yang selanjutkan akan menampilkan petunjuk misi. Adapun hasil dari proses pengujian dapat dilihat pada tabel berikut.

Tabel 4.3 Pengujian *Menu* Mulai Main

Kasus dan Hasil Uji			
Masukan	**Keluaran yang diharapkan**	**Pengamatan**	**Kesimpulan**
Klik *Menu* Mulai	Sistem melakukan proses untuk menampilkan halaman petunjuk misi	Sistem menampilkan halaman petunjuk misi	[√] Berhasil [ ] Tidak Berhasil

3. Pengujian *Menu* Petunjuk

Pengujian *menu* petunjuk merupakan proses dimana pengguna dapat menutup aplikasi. Adapun hasil dari proses pengujian dapat dilihat pada tabel berikut.

Tabel 4.4 Pengujian *Menu* Petunjuk

Kasus dan Hasil Uji			
Masukan	**Keluaran yang diharapkan**	**Pengamatan**	**Kesimpulan**
Klik *Menu* Petunjuk	Sistem melakukan proses untuk menampilkan halaman Petunjuk permainan	Sistem menampilkan halaman petunjuk permainan	[√] Berhasil [ ] Tidak Berhasil
Klik *Menu* Kembali	Sistem melakukan proses untuk menampilkan kembali ke *menu* utama	Sistem menampilkan *menu* utama	[√] Berhasil [ ] Tidak Berhasil

4. Pangujian *Menu* Keluar

Pengujian *menu* kaluar merupakan proses dimana pengguna dapat melihat petunjuk permainan. Adapun hasil dari proses pengujian dapat dilihat pada tabel berikut.

Tabel 4.5 Pengujian *Menu* Keluar

Kasus dan Hasil Uji			
Masukan	**Keluaran yang diharapkan**	**Pengamatan**	**Kesimpulan**
Klik *Menu* Kaluar	Sistem melakukan proses untuk menampilkan	Sistem menampilkan pesan keluar	[√] Berhasil [ ] Tidak Berhasil

	pesan keluar		
Klik *Menu* Ya	Sistem melakukan proses untuk menutup aplikasi	Sistem menutup aplikasi	[√] Berhasil [ ] Tidak Berhasil
Klik *Menu* Tidak	Sistem melakukan proses untuk menampilkan tampilan *menu* utama	Sistem menampilkan menu utama	[√] Berhasil [ ] Tidak Berhasil

5. Pangujian *Menu* Mulai

Pengujian *menu* mulai merupakan proses dimana pengguna dapat memulai permainan. Adapun hasil dari proses pengujian dapat dilihat pada tabel berikut.

Tabel 4.6 Pengujian *Menu* Main

Kasus dan Hasil Uji			
Masukan	**Keluaran yang diharapkan**	**Pengamatan**	**Kesimpulan**
Klik *Menu* Main	Sistem melakukan proses untuk menampilkan halaman Permainan	Sistem menampilkan halaman permainan	[√] Berhasil [ ] Tidak Berhasil

6. Pengujian *Pause Menu*

Pengujian *pause menu* merupakan proses dimana pengguna dapat menghentikan sementara jalannya permainan sewaktu-waktu atau ingin keluar ditengah-tengah jalannya permainan. Adapun hasil dari proses pengujian dapat dilihat pada tabel berikut ini

Tabel 4.7 Pengujian *Pause Menu*

Kasus dan Hasil Uji			
Masukan	**Keluaran yang diharapkan**	**Pengamatan**	**Kesimpulan**
Tekan Tombol Esc pada *keyboard*	Sistem melakukan proses untuk menampilkan halaman *pause menu*	Sistem menampilkan halaman *pause menu*	[√] Berhasil [ ] Tidak Berhasil
Klik *Menu* Kembali	Sistem melakukan proses untuk menampilkan kembali permainan yang telah dimainkan	Sistem menampilkan area permainan yang dimainkan	[√] Berhasil [ ] Tidak Berhasil
Klik *Menu* Ulangi	Sistem melakukan proses untuk mengulangi *level* yang dimainkan	Sistem menampilkan area permainan yang dimainkan	[√] Berhasil [ ] Tidak Berhasil
Klik *Menu* Main *Menu*	Sistem melakukan proses untuk kembali ke *menu* utama	Sistem menampilkan *menu* utama	[√] Berhasil [ ] Tidak Berhasil
Klik *Menu* Petunjuk	Sistem melakukan proses untuk	Sistem menampilkan	[√] Berhasil

	menampilkan Petunjuk permainan	petunjuk permainan	[ ] Tidak Berhasil
Tekan *Menu* Keluar	Sistem melakukan proses untuk menutup aplikasi	Sistem menutup aplikasi	[√] Berhasil [ ] Tidak Berhasil

4.2.1.3. Kasus dan Hasil Pengujian (*White Box Testing)*

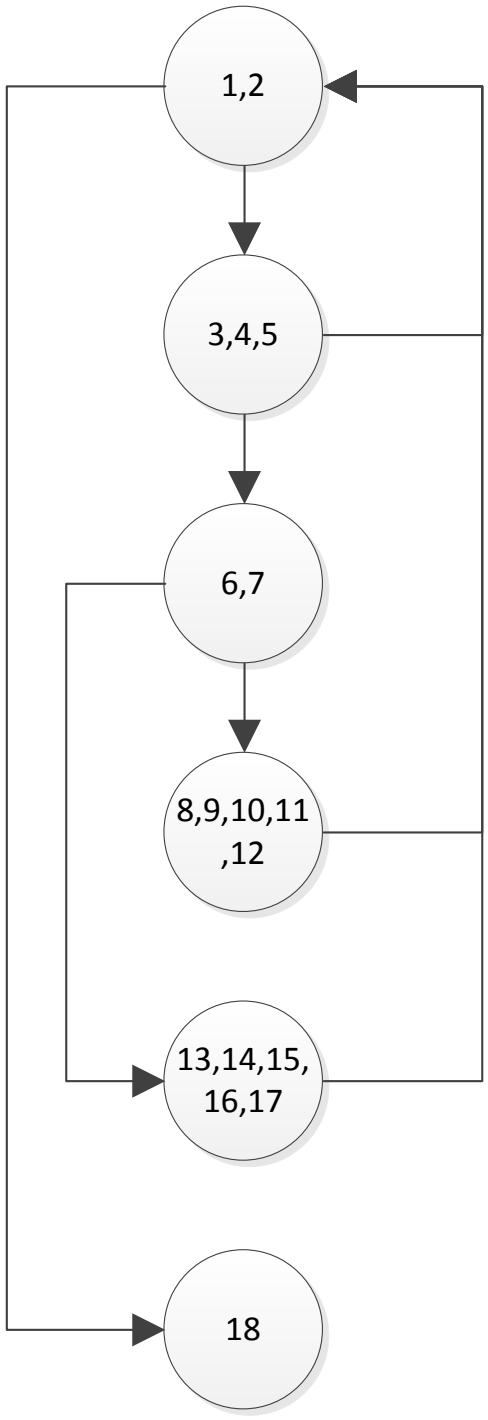
1. Pengujian *White Box* Algoritma A* (A *Star*)

Berikut ini merupakan algoritma A* untuk *level* 2 yang penerapannya menggunakan bahasa pemrograman *Javascipt*.

a. *Flow Graph* Algoritma A* Prosedur Pencarian Jalan

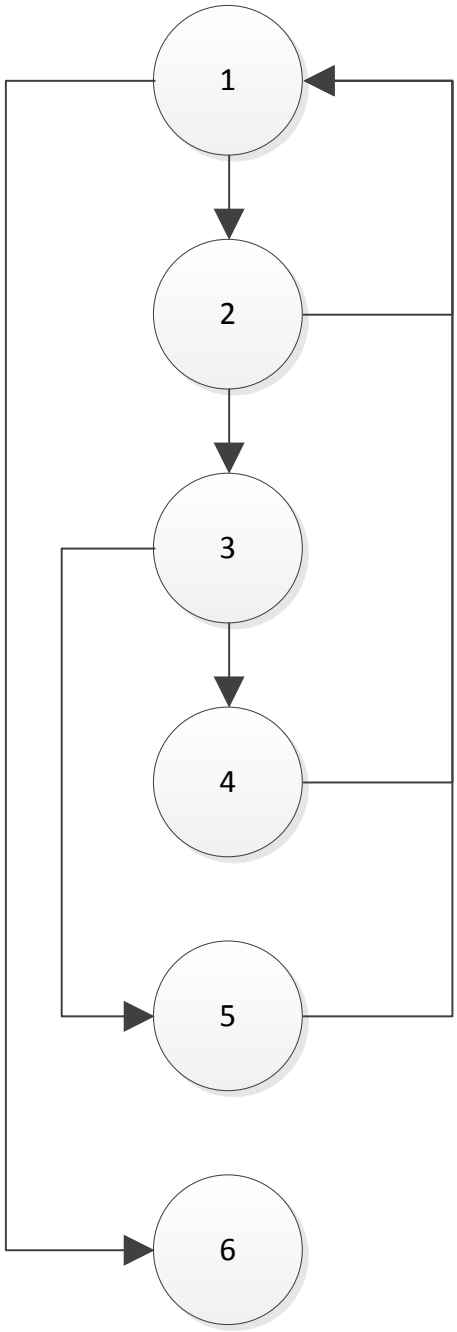
```
1. while(foundTarget=false){
2. checkingNode = GetSmallestFValueNode();
3. if (checkingNode==null){
4. GetBackup();
5. }
6. AddToClosedList(checkingNode);
7. RemoveFromOpenList(checkingNode);
8. if(checkingNode == targetNode)
9. {
10. foundTarget = true;
11. TraceBackPath (musuh);
12. }
13. else{
14. for (var node : PathfindNode in checkingNode.dekat){
15. DetermineNodeValues (checkingNode, node);
16. }
17. }
18. }
```

Berdasarkan *Sourcecode* prosedur pencarian jalan, maka dibuat *flow graph* yang bisa dilihat pada gambar berikut ini.



Gambar 4.11 *Flow Graph* Prosedur Pencarian Jalan

Atau disederhanakan berdasarkan kondisi atau simpul sebagai berikut :



Gambar 4.12 Penyederhanaan *Flow Graph* Prosedur Pencarian Jalan



= Menggambarkan Kondisi

= Menggambarkan Aksi

a. *Cyclomatic Complexity* V(G)

E = 8, N = 6

V(G) = E – N + 2

V(G) = 8 – 6 + 2

V(G) = 2 + 2 = 4

Keterangan :

E = Jumlah Aksi

N = Jumlah Kondisi

b. *Independent Path*

Path 1 = 1-6

Path 2 = 1-2-1-6

Path 3 = 1-2-3-4-1-6

Path 4 = 1-2-3-5-1-6

c. *Graph Matriks* Algoritma A* Prosedur Pencarian Jalan

Penomoran ulang *flowgraph* atau penyerderhanaan berdasarkan kondisi atau simpul dapat dilihat pada gambar berikut ini.

N	1	2	3	4	5	6	N(E) -1
1		1				1	1
2	1		1				1
3				1	1		1
4	1						0
5	1						0
6							0
						Sum(E)+1	4

Keterangan :

1. Baris dan Kolom merepsentasikan simpul
2. Nilai 1 merepsentasikan adanya keterhubungan antara simpul

d. Hasil Uji Algoritma Prosedur Pencarian Jalan

Tabel 4.8 Hasil Uji Algoritma A* Prosedur Pencarian Jalan

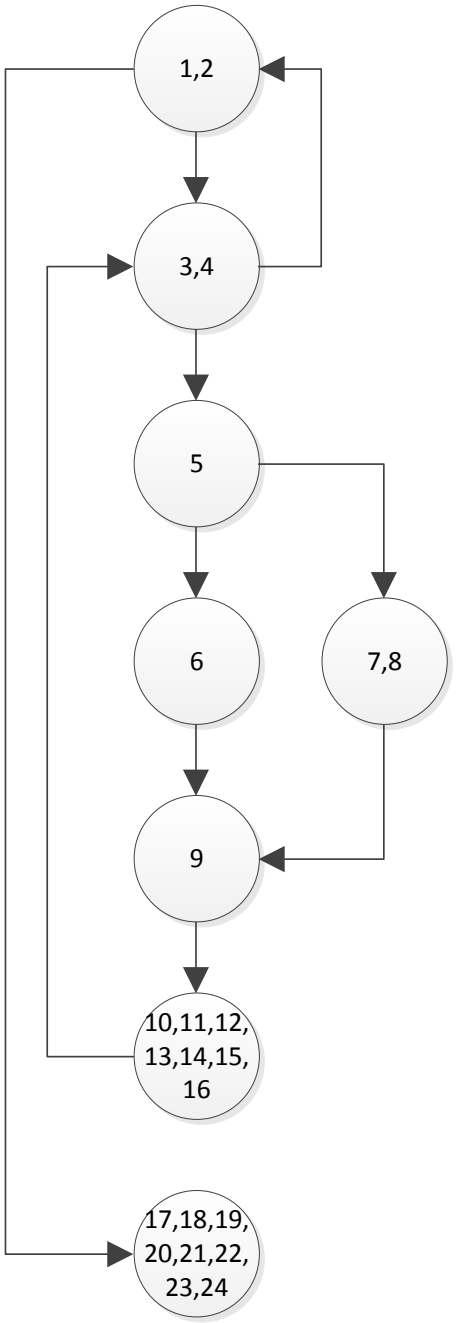
No	Kasus Uji	Yang diharapkan	Pengamatan	Keterangan
1	`While (foundTarget = false){ checkingNode = GetSmallestFValue() ;`	Pencarian *node* yang dicari belum ada	Pencarian dilakukan sampai mendapatkan nilai F terkecil	[√] Alur terlewati [ ] Alur tidak terlewati
2	`While (foundTarget = false){ checkingNode = GetSmallestFValue() ;` `if (checkingNode ==null){ GetBackup();`	Mengambil Nilai *Node* terkecil dari Node yang dicek dengan menggunakan fungsi GetBackup	Pencarian dilakukan dengan mengambil nilai *node* terkecil	[√] Alur terlewati [ ] Alur tidak terlewati
3	`While (foundTarget = false){ checkingNode = GetSmallestFValue() ;` `if (checkingNode =true; TraceBackPath(musuh )`	Node yang dicari sudah ada maka Musuh akan mengikuti *Node* sampai *Node* akhir dengan fungsi TraceBackPath	Pencarian jalan dilakuakn dan fungsi TraceBackPath dilakukan	[√] Alur terlewati [ ] Alur tidak terlewati
4	`While (foundTarget = false){ checkingNode =`	Pengecekan *node*	Pengecekan *node*	[√] Alur terlewati

	`GetSmallestFValue(); for (var node : pathfindNode in checkingNode. Dekat){ DetermineNodeValues (checkingNode, Node);`	dilakukan lagi sebanyak *node* dekat	dilakukan sebanyak node dekat	[ ] Alur tidak terlewati

b. *Flow Graph* Algoritma A* Prosedur Pengecekan *Node* terdekat

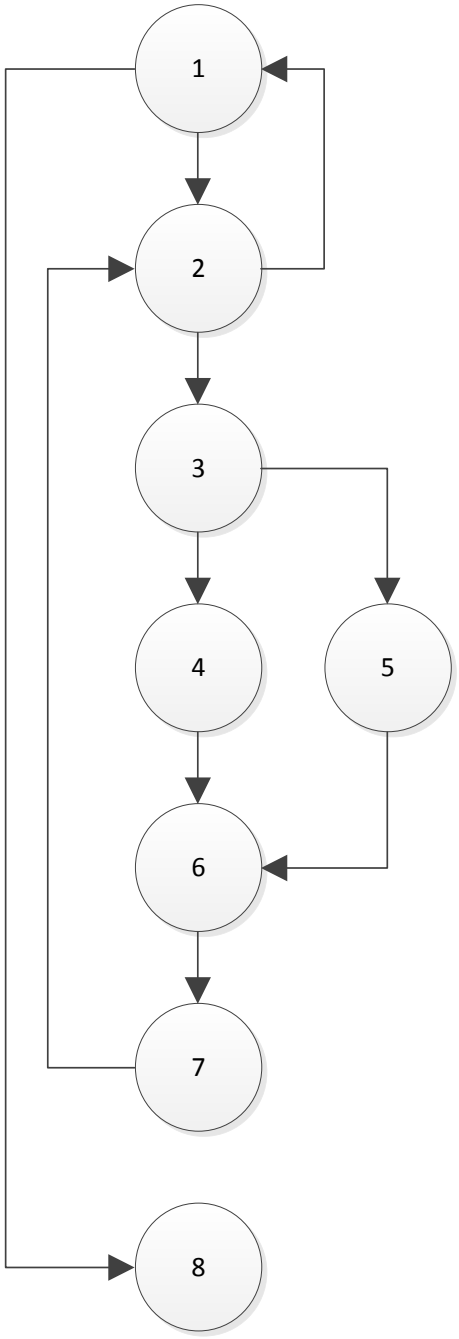
```
1. if (closedList.Contains(testing) == false)
2. {
3. if (openList.Contains(testing) == true)
4. {
5. if (Vector3.Distance(currentNode.tranform.position,
   testing.transfrom.position))
6. baseMovementCost = 14;
7. else
8. baseMovementCost = 10;
9. Var newGCost : int = currentNode.getGValue()+ basemovementCost;
10. if (newGCost < testing.getGValue())
11.{
12. testing.setParent (currentNode);
13. testing.setGValue (newGCost);
14. testing.CalculateFValue();
15. }
16. }
17. else
18. {
19. testing.setParent (currentNode);
20. testing.setGValue (currentNode.getGValue()+ basemovementCost);
21. testing.CalculateFValue();
22. AddToOpenList(testing);
23. }
24. }
```

Berdasarkan *Sourcecode* prosedur pengecekan *node* terdekat, maka dibuat *flow graph* yang bisa dilihat pada gambar berikut ini.



Gambar 4.13 *Flow Graph* Prosedur Pengecekan *Node* Terdekat

Atau disederhanakan berdasarkan kondisi atau simpul sebagai berikut :



Gambar 4.14 Penyederhanaan *Flow Graph* Pengecekan *Node* Terdekat

= Menggambarkan Kondisi

= Menggambarkan Aksi

a. *Cyclomatic Complexity* V(G)

E = 10, N = 8

V(G) = E – N + 2

V(G) = 10 – 8 + 2

V(G) = 2 + 2 = 5

Keterangan :

E = Jumlah Aksi

N = Jumlah Kondisi

b. *Independent Path*

Path 1 = 1-8

Path 2 = 1-2-1-8

Path 3 = 1-2-3-4-6-7-2-1-8

Path 4 = 1-2-3-5-6-7-2-1-8

c. *Graph Matriks* Algoritma A* Prosedur pengecekan *node* terdekat Penomoran ulang *flowgraph* atau penyerderhanaan berdasarkan kondisi atau simpul dapat dilihat pada gambar berikut ini.

N	1	2	3	4	5	6	7	8	N(E) -1
1		1						1	1
2	1		1						1
3				1	1				1
4						1			0
5						1			0
6							1		0
7		1							0
8									0
								Sum(E)+1	4

Keterangan :

1. Baris dan Kolom merepsentasikan simpul
2. Nilai 1 merepsentasikan adanya keterhubungan antara simpul

d. Hasil Uji Algoritma Prosedur pengecekan *node* terdekat

Tabel 4.9 Hasil Uji Algoritma A* Prosedur Pengecekan *Node* Terdekat

No	Kasus Uji	Yang diharapkan	Pengamatan	Keterangan
1	`if (closedList.Contains(testing) == false)`	Pengecekan *Node* tidak dilakukan	Pengecekan *node* gagal dilakukan	[√] Alur terlewati [ ] Alur tidak terlewati
2	`if (closedList.Contains(testing) == false) else { testing.setParent(currentNode); testing.setGValue(currentNode.getGValue() + baseMovementCost); testing.CalculateFValue(); AddToOpenList(testing); } }`	Dilakukan perhitungan nilai G pada node yang dicek dan menghitung nilai F lalu memindahkan menjadi openlist	Terjadi perpindahan node ke openlist dengan perhitungan nilai G dan F	[√] Alur terlewati [ ] Alur tidak terlewati
3	`if (closedList.Contains(testing) == false) if (openList.Contains(testing) == true) { if(Vector3.Dista`	Dilakukan perhitungan pada nilai G secara diagonal dan dilakukan pergantian parent dengan nilai G yang	Perhitungan nilai G yang baru dilakukan dab pergantian parent dengan nilai G yang lebih baik	[√] Alur terlewati [ ] Alur tidak terlewati

	nce(currentNode.transform.position,testing.transform.position))
	baseMovementCost=14;
	var newGCost : int = currentNode.getGValue() + baseMovementCos
	if (newGCost < testing.getGValue())
	{
	testing.setParent(currentNode);
	testing.setGValue(newGCost);
	testing.CalculateFValue();
	}
	}
		lebih baik		

4	if (closedList.Contains(testing) == false)
	if (openList.Contains(testing) == true) {
	if(Vector3.Distance(currentNode.transform.position,testing.transform.position))
	baseMovementCost
		Dilakukan perhitungan pada nilai G secara horizontal atau vertical dan dilakukan pergantian parent dengan nilai G yang lebih baik	Perhitungan nilai G yang baru dilakukan dab pergantian parent dengan nilai G yang lebih baik	[√] Alur terlewati
				[ ] Alur tidak terlewati

	=10;
	var newGCost :
	int =
	currentNode.getG
	Value() +
	baseMovementCos
	if (newGCost <
	testing.getGValu
	e())
	{
	testing.setParen
	t(currentNode);
	testing.setGValu
	e(newGCost);
	testing.Calculat
	eFValue();
	}
	}			

c. Kesimpulan

Dari hasil pengujian algoritma A* yang dilakukan, maka didapat kesimpulan bahwa seluruh *node* yang ada pada setiap *independent path* telah dikerjakan, serta tidak terjadi pengulangan tak terhingga

2. Pengujian *White Box* Algoritma *Fuzzy Logic*

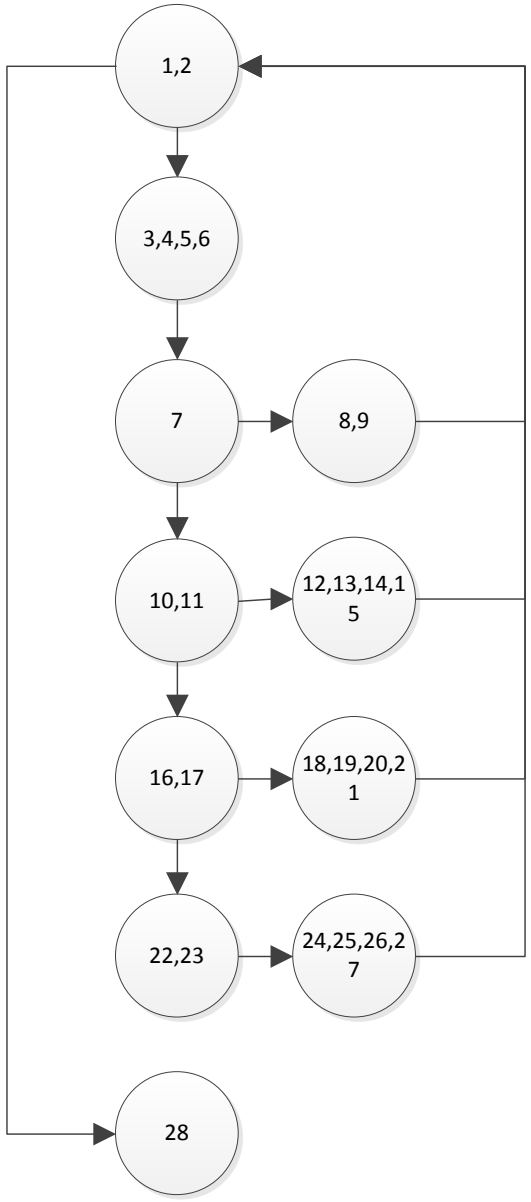
Berikut ini merupakan algoritma *Fuzzy Logic* untuk *level* 3 yang penerapannya menggunakan bahasa pemrograman C#.

```
1.  if (target) {
2.  jarak = (target.position – myTranform.position).magnitude;
3.  rules rule = new rules (jarak,hitPoint);
4.  defuzzyfikasi def = new defuzzyfikasi ();
5.  rule.periksaRule ();
6.  def.hasilDefuzzy();
7.  if((healingpoint.position – myTransform.position)
    .magnitude<=2.0)
8.  StartCoroutine(tambahDarah());
9.  }
10. if(def.hasilDefuzzyfikasi <=30.0f)
11. {
12. Debug.Log ("Mengambil Darah")
13. myTransform.rotation = Quaternation.Slerp
   (myTransform.rotation, Quaternation.LookRotation(healingpoint.
14. anim.animation.CrossFade(runAnim);
15. }
16. else if(def.hasilDefuzzyfikasi > 30f && def.hasilDefuzzyfikasi
    <= 60.0f)
17. {
18. Debug.Log("menghindar");
19. myTransform.rotation = Quaternion.Slerp(myTransform.rotation,
    Quaternion.LookRotation(target.position - myTransform.
    position), rotationSpeed*Time.deltaTime);
20. anim.animation.CrossFade(attackAnim);
21. }
22. else if(def.hasilDefuzzyfikasi > 60.0f && def.
    hasilDefuzzyfikasi <= 90.0f)
23. {
24. Debug.Log("menyerang");
25. myTransform.rotation = Quaternion.Slerp(myTransform.rotation,
    Quaternion.LookRotation(target.position - myTransform.
    position), rotationSpeed*Time.deltaTime);
```

```
26. anim.animation.CrossFade(walkAnim);
27. }
28. }
```

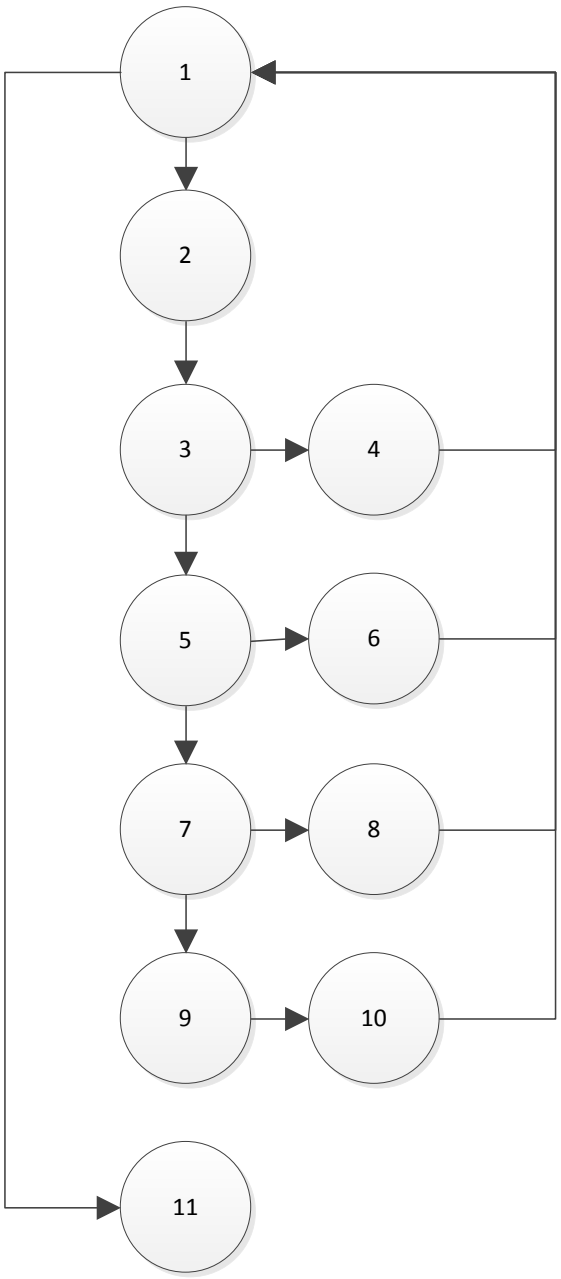
a. *Flow Graph* Algoritma *Fuzzy Logic*

Berdasarkan *Source code* diatas maka *flow graph* algoritma *fuzzy logic* adalah sebagia berikut.



Gambar 4.15 *Flow Graph* Algoritma *Fuzzy Logic*

Atau disederhanakan berdasarkan kondisi atau simpul sebagai berikut :



Gambar 4.16 Penyederhanaan *Flow Graph* Algoritma *Fuzzy Logic*

= Menggambarkan Kondisi

= Menggambarkan Aksi

a. *Cyclomatic Complexity* V(G)

E = 14, N = 11

V(G) = E – N + 2

V(G) = 14 – 11 + 2

V(G) = 3 + 2 = 5

Keterangan :

E = Jumlah Aksi

N = Jumlah Kondisi

b. *Independent Path*

Path 1 = 1-11

Path 2 = 1-2-3-4-1-11

Path 3 = 1-2-3-5-6-1-11

Path 4 = 1-2-3-5-7-8-1-11

Path 5 = 1-2-3-5-7-9-10-1-11

c. *Graph Matriks* Algoritma *Fuzzy Logic* Penomoran ulang *flowgraph* atau penyerderhanaan berdasarkan kondisi atau simpul dapat dilihat pada gambar berikut ini.

N	1	2	3	4	5	6	7	8	9	10	11	N(E) -1
1		1									1	1
2			1									0
3				1	1							1
4	1											0
5						1	1					1
6	1											0
7								1	1			1
8	1											0
9										1		0
10	1											0
11												0
											Sum(E)+1	5

Keterangan :

1. Baris dan Kolom merepsentasikan simpul
2. Nilai 1 merepsentasikan adanya keterhubungan antara simpul

d. Hasil Uji Algoritma *Fuzzy Logic*

Tabel 4.10 Hasil Uji Algoritma *Fuzzy Logic*

No	Kasus Uji	Yang diharapkan	Pengamatan	Keterangan
1	`if (target) { jarak = (target.position – myTranform.position).magnitude;`	Posisi musuh dan posisi player telah didapatkan	Jarak didapat dari posisi musuh dan player	[√] Alur terlewati [ ] Alur tidak terlewati
2	`if (target) { jarak = (target.position – myTranform.position).magnitude; rules rule = new rules (jarak,hitPoint); defuzzyfikasi def = new defuzzyfikasi (); rule.periksaRule (); def.hasilDefuzzy (); if((healingpoint .position – myTransform.position)`	Hasil dari rule akan di hitung dari jarak dan darah untuk menentukan defuzzyfikasi dan NPC menambah darah saat kondisi mengambil darah	Hasil dari defuzzyfikasi dan NPC menambah darah saat kondisi mnagambil darah	[√] Alur terlewati [ ] Alur tidak terlewati

	.magnitude<=2.0) StartCoroutine(tambahDarah()); }			

3	if(def.hasilDefuzzyfikasi
	<=30.0f)
	{
	Debug.Log
	(“Mengambil Darah”)
	myTransform.rotation =
	Quaternation.Slerp
	(myTransform.rotation,
	Quaternation.LookRotation(healingpoint.
	anim.animation.CrossFade(runAnim);
	}
	else
		Hasil dari defuzzyfikasi <= 30 NPC akan mengambil darah	NPC akan mengambil darah	[√] Alur terlewati
				[ ] Alur tidak terlewati

4	if(def.hasilDefuzzyfikasi > 30f &&def.hasilDefuzzyfikasi
	<=60.0f){
	Debug.Log("menghindar");
	myTransform.rotation =
	Quaternion.Slerp
	(myTransform.rotation,
		Hasil dari defuzzyfikasi > 30 dan < =60 NPC akan menghidar	NPC akan menghidar	[√] Alur terlewati
				[ ] Alur tidak terlewati

	Quaternion.LookRotation(target.position - myTransform.position),rotationSpeed*Time.deltaTime);anim.animation.CrossFade(attackAnim);}			
5	else if(def.hasilDefuzzyfikasi > 60.0f && def.hasilDefuzzyfikasi <= 90.0f) {Debug.Log("menyerang"); myTransform.rotation = Quaternion.Slerp (myTransform.rotation, Quaternion.LookRotation(target.position - myTransform. position), rotationSpeed*Time.deltaTime); anim.animation.CrossFade(walkAnim); }	Hasil dari defuzzyfikasi 60 > dan <= 90 NPC akan menyerang	NPC akan menyerang	[√] Alur terlewati [ ] Alur tidak terlewati

c. Kesimpulan

Dari hasil pengujian algoritma *fuzzy logic* yang dilakukan, maka didapat kesimpulan bahwa seluruh *node* yang ada pada setiap *independent path* telah dikerjakan, serta tidak terjadi pengulangan tak terhingga

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, akan tetapi prosesnnya masih memungkinkan untuk terjadi kesalahan, namun frekuensi kesalahannya relatif kecil.

4.2.2. Pengujian *Beta*

Pengujian *beta* merupakan pengujian yang dilakukan secara objektif, dimana dilakukan pengujian secara langsung terhadap pengguna dengan menggunakan kuesioner mengenai kepuasan para pengguna terhadap *game* yang dibangun. Metode penilaian pengujian yang digunakan adalah metode kuantitatif berdasarkan data dari pengguna.

4.2.2.1. Kuesioner

Kuesioner disebarkan kepada 30 orang responden minimal dari umur 13 tahun, dari hasil kuesioner tersebut akan dilakukan perhitungan agar dapat diambil kesimpulan terhadap penilaian penerapan sistem yang telah dibangun. Kuesioner terdiri dari 10 pernyataan dengan menggunakan sistem penskoran dengan skala pengukuran yang digunakan adalah skala *likert*, Berikut skor untuk jawaban kuesioner.

Tabel 4.11 Skor Jawaban Pada Kuesioner

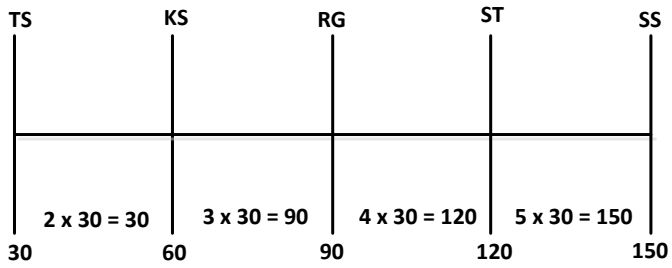
Skala Jawaban	Keterangan	Skor
SS	Sangat Setuju	5
ST	Setuju	4
RG	Ragu-ragu	3
KS	Kurang Setuju	2

TS	Tidak Setuju	1

Data yang diperoleh dari pemberian kuesioner kepada responden dapat dianalisis dengan menghitung rata-rata jawaban berdasarkan *scoring* setiap jawaban dari responden. Analisis yang dapat dilakukan yaitu sebagai berikut.

a. Jumlah skor ideal (kriterium) untuk seluruh item = 5 × jumlah responden
b. Rata-rata skor dalam persentase = (jumlah skor : jumlah skor ideal) × 100%
c. Jumlah skor ideal pada penelitian ini yaitu : Jumlah skor ideal = 5×30 = 150

Secara kontinum dapat digambarkan seperti berikut:



Gambar 4.17 Penilaian

Adapun pertanyaan yang diajukan dalam kuesioner untuk pengguna aplikasi ini adalah sebagai berikut :

1. Apakah *game* ini bisa memberikan unsur sosialisasi tentang kondisi habitat orangutan pada saat ini?
2. Apakah pemanfaatan *game* ini secara posistif sudah baik?
3. Apakah *game* ini berguna bagi masyarakat untuk mendukung kelestarian habitat hewan khas Indonesia yaitu orangutan?
4. Apakah kemampuan dalam memilih keputusan pergerakan yang dimiliki oleh musuh pada *game* ini berjalan dengan baik?
5. Apakah karena musuh dalam *game* ini memiliki kemampuan memilih keputusan pergerakan dapat membuat *game* lebih menantang untuk dimainkan?

Berikut adalah hasil kuisioner masing-masing jawaban yang sudah dihitung dengan menggunakan persamaan skala kontinum (persamaan 2.7). Kuisioner ini ditujukan kepada 30 responden.

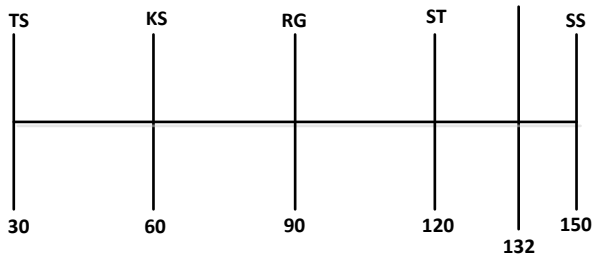
1. Apakah *game* ini bisa memberikan unsur sosialisasi tentang kondisi habitat orangutan pada saat ini?

   Hasil kuesioner untuk pertanyaan ke-1 dapat dilihat pada tabel 4.12.

Tabel 4.12 Hasil Kuesioner Pertanyaan 1

Kategori Jawaban	Skor	Responden	S
Sangat Setuju	5	12	60
Setuju	4	18	72
Ragu-ragu	3	0	0
Kurang Setuju	2	0	0
Tidak Setuju	1	0	0
Jumlah		**30**	**132**

Berdasarkan hasil skor nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap *game* ini adalah 132 dari yang diharapkan 150, atau dapat dikategorikan sangat setuju dan dapat disimpulkan bahwa *game* ini bisa memberikan unsur sosialisasi tentang kondisi orangutan pada saat ini.



Gambar 4.18 *Interprestasi* Skor Pertanyaan 1

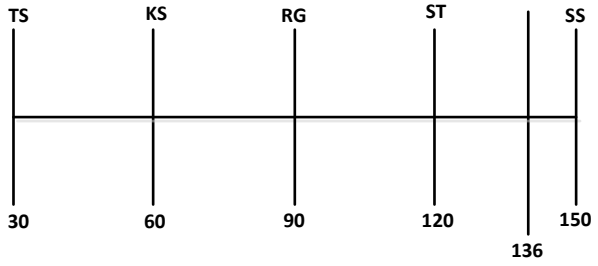
2. Apakah pemanfaatan *game* ini secara posistif sudah baik?

   Hasil kuesioner untuk pertanyaan ke-2 dapat dilihat pada tabel 4.13.

Tabel 4.13 Hasil Kuesioner Pertanyaan 2

Kategori Jawaban	Skor	Responden	S
Sangat Setuju	5	16	80
Setuju	4	14	56
Ragu-ragu	3	0	0
Kurang Setuju	2	0	0
Tidak Setuju	1	0	0
Jumlah		**30**	**136**

Berdasarkan hasil skor nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap *game* ini adalah 136 dari yang diharapkan 150, atau dapat dikategorikan sangat setuju dan dapat disimpulkan bahwa pemenfaatan *game* ini secara positif sudah baik.



Gambar 4.19 *Interprestasi* Skor Pertanyaan 2

3. Apakah *game* ini berguna bagi masyarakat untuk mendukung kelestarian habitat hewan khas Indonesia yaitu orangutan?

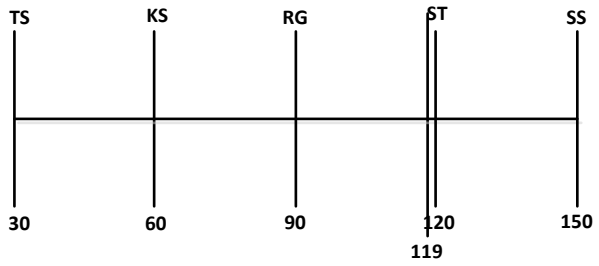
   Hasil kuesioner untuk pertanyaan ke-3 dapat dilihat pada tabel 4.14.

Tabel 4.14 Hasil Kuesioner Pertanyaan 3

Kategori Jawaban	Skor	Responden	S
Sangat Setuju	5	5	25
Setuju	4	19	76
Ragu-ragu	3	6	18
Kurang Setuju	2	0	0

Tidak Setuju	1	0	0
Jumlah		**30**	**119**

Berdasarkan hasil skor nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap *game* ini adalah 119 dari yang diharapkan 150, atau dapat dikategorikan setuju dan dapat disimpulkan bahwa *game* ini berguna bagi masyarakat untuk mendukung kelestarian habitat hewan khas Indoensia yaitu orangutan.



Gambar 4.20 *Interprestasi* Skor Pertanyaan 3

4. Apakah kemampuan dalam memilih keputusan pergerakan yang dimiliki oleh musuh pada *game* ini berjalan dengan baik?

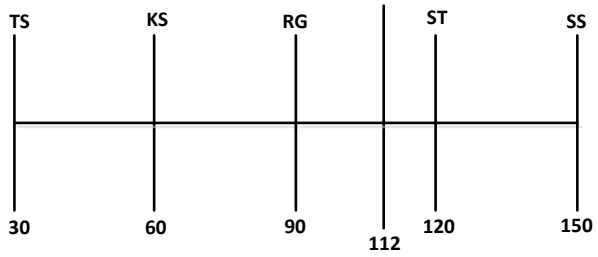
   Hasil kuesioner untuk pertanyaan ke-4 dapat dilihat pada tabel 4.15.

Tabel 4.15 Hasil Kuesioner Pertanyaan 4

Kategori Jawaban	Skor	Responden	S
Sangat Setuju	5	3	15
Setuju	4	18	72
Ragu-ragu	3	7	21
Kurang Setuju	2	2	4
Tidak Setuju	1	0	0
Jumlah		**30**	**112**

Berdasarkan hasil skor nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap *game* ini adalah 125 dari yang diharapkan 150, atau dapat dikategorikan sangat setuju dan dapat disimpulkan bahwa kemampuan dalam

memeilih keputusan pergerakan yang dimiliki oleh musuh pada *game* ini berjalan dengan baik.



Gambar 4.21 *Interprestasi* Skor Pertanyaan 4

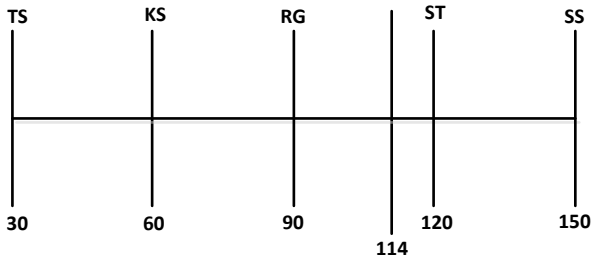
5. Apakah karena musuh dalam *game* ini memiliki kemampuan memilih keputusan pergerakan dapat membuat *game* lebih menantang untuk dimainkan?

   Hasil kuesioner untuk pertanyaan ke-5 dapat dilihat pada tabel 4.16.

Tabel 4.16 Hasil Kuesioner Pertanyaan 5

Kategori Jawaban	**Skor**	**Responden**	**S**
Sangat Setuju	5	2	10
Setuju	4	21	84
Ragu-ragu	3	6	18
Kurang Setuju	2	1	2
Tidak Setuju	1	0	0
Jumlah		**30**	**114**

Berdasarkan hasil skor nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap *game* ini adalah 114 dari yang diharapkan 150, atau dapat dikategorikan setuju dan dapat disimpulkan bahwa karena musuh dalam *game* ini memiliki kemampuan memilih keputusan pergerakan dapat membuat *game* lebih menanantang untuk dimainkan.



Gambar 4.22 *Interprestasi* Skor Pertanyaan 5

4.2.2.2. Kesimpulan Pengujian *Beta*

Berdasarkan dari jawaban yang diberikan oleh responden, bisa ditarik kesimpulan bahwa "*Game Third Person Shooter* 3D Selamatkan Orangutan" dinyatakan bisa memberikan unsur sosialisasi tentang kondisi orangutan pada saat ini, pemanfaatan *game* secara positif sudah baik dan dapat berguna bagi masyarakat untuk mendukung kelestarian habitat hewan khas Indonesia yaitu orangutan, kemampuan dalam memilih keputusan pergerakan yang dimiliki oleh musuh berjalan dengan baik dan karena musuh memiliki kemampuan memilih keputusan pergerakan dapat membuat *game* lebih menantang untuk dimainkan.