BAB IV
IMPLEMENTASI DAN PENGUJIAN

IV.1 Implementasi

Tahap implementasi merupakan tahap penciptaan perangkat lunak dan juga tahap kelanjutan dari kegiatan perancangan aplikasi, tahap ini merupakan tahap dimana aplikasi siap untuk dioperasikan. Aplikasi ini diimplementasikan dengan nama Pertempuran 10 November 1945 dengan *file extension*-nya *.exe.*

IV.1.1 Implementasi Perangkat Keras

Perangkat keras yang dapat digunakan untuk menjalankan aplikasi *game* ini harus memiliki spesifikasi yang cukup tinggi. Sebagai ilustrasi sistem minimum agar dapat menjalankan *game* Pertempuran 10 November 1945 Surabaya adalah sebagai berikut :

1. Processor Dual Core 1,80 Ghz
2. RAM 1 GB
3. VGA 512 MB
4. Harddisk 1 GB
5. *Mouse* dan *Keyboard*

IV.1.2 Implementasi Perangkat Lunak

Adapun spesifikasi perangkat lunak yang dibutuhkan untuk menjalankan aplikasi Pertempuran 10 November 1945 adalah :

1. Sistem operasi minimal Windows Xp
2. *Graphic card*
3. DirectX 9 keatas

IV.1.3 Implementasi Instalasi Aplikasi Game

Untuk dapat memainkan *game* Pertempuran 10 November 1945, pemain terlebih harus mempunyai *file* Pertempuran 10 November 1945.*exe* . Untuk mulai

memainkan *game*, pemain hanya perlu menjalankan *file* Pertempuran 10 November 1945.*exe* yang telah ada dalam perangkat keras.

IV.1.4 Implementasi Antarmuka

Antarmuka dapat membuat pengguna merasa nyaman dan bertahan lama berada dalam aplikasi atau sebaliknya. Berikut ini merupakan antarmuka yang ada pada aplikasi *game* Pertempuran 10 November 1945.

1. Antarmuka Menu Utama

Saat pengguna membuka aplikasi pertama kali, maka antarmuka yang ditampilkan oleh sistem adalah antarmuka menu utama. Berikut ini merupakan tampilan dari antarmuka menu utama :



Gambar IV.1 Antarmuka Menu Utama

Gambar IV.1 di atas merupakan menu utama pada *game* Pertempuran 10 November 1945. Terdapat 3 (tiga) sub menu di dalamnya, yaitu menu Mainkan yang berfungsi untuk memulai permainan, menu Petunjuk yang berfungsi untuk melihat informasi kontrol yang digunakan ketika memainkan *game*, dan menu Keluar yang berfungsi untuk keluar dari aplikasi.

2. Antarmuka Menu Mainkan

Saat pengguna memilih menu mainkan, maka antarmuka yang ditampilkan oleh sistem adalah; Pertama, antarmuka *scene video* berupa *video* cerita awal dari *game* pada level pertama. Selanjutnya akan masuk ke antarmuka petunjuk misi dan kemudian akan masuk pada antarmuka permainan. Jika seluruh misi telah terselesaikan maka akan masuk ke antarmuka *ending scene* beupa *video* akhir cerita *game*. Berikut ini merupakan tampilan dari antarmuka menu mainkan.



Gambar IV.2 Antarmuka *scene video level* 1

Gambar IV.2 merupakan antarmuka *scene video* pada *level* pertama yang terdapat pada *game* Pertempuran 10 November 1945.



Gambar IV.3 Antarmuka Petunjuk misi

Gambar IV.3 merupakan antarmuka petunjuk misi yang terdapat pada *game* Pertempuran 10 November 1945.



Gambar IV.4 Antarmuka Permainan di *level* 1-1

Gambar IV.4 merupakan antarmuka permainan yang terdapat pada *game* Pertempuran 10 November 1945. Gambar tersebut memperlihatkan posisi pemain ketika masuk ke dalam permainan.



Gambar IV.5 Antarmuka Permainan di *level* 1-2

Gambar IV.5 menunjukan posisi pemain ketika akan menghadapi senapan otomatis.



Gambar IV.6 Antarmuka Permainan di *level* 1-3

Gambar IV.6 menunjukan posisi pemain ketika telah menenmukan Hotel Yamato.

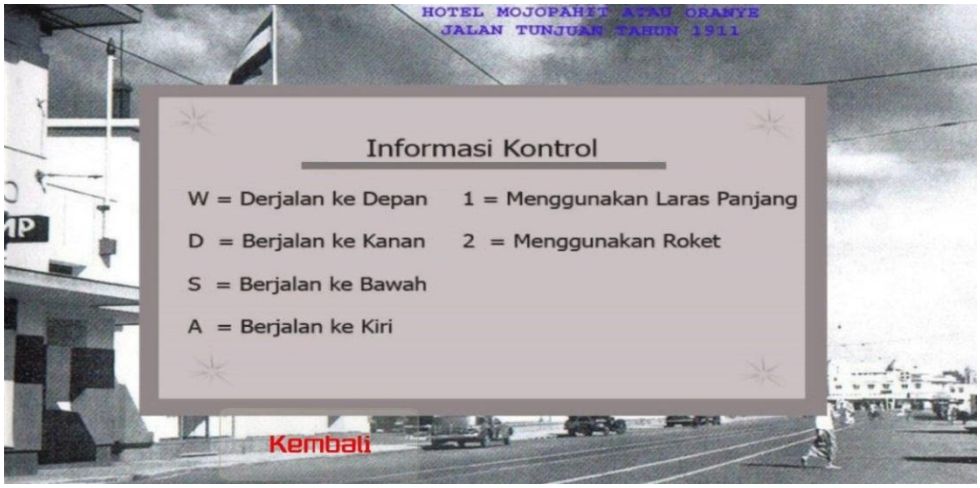


Gambar IV.7 Antarmuka *scene video level* 1 selesai.

Gambar IV.7 di atas merupakan antarmuka *scene video* ketika pemain berhasil menyelesaikan permainan pada level pertama.

3. Antarmuka Menu Petunjuk

Saat pengguna memilih menu petunjuk, maka antarmuka yang ditampilkan oleh sistem adalah antarmuka informasi kontrol yang digunakan pada *game* Pertempuran 10 November 1945.



Gambar IV.8 Tampilan dari antarmuka menu Petunjuk.

IV.2 Pengujian

Pada umunya ada 2 (dua) macam teknik pengujian yaitu *white box testing* dan *black box testing*. Pengujian yang menggunakan teknik *white box testing* lebih menitikberatkan pada struktur internal (*source code*) program, dimana setiap baris kode yang akan di*compile* atau dicek satu persatu untuk mengetahui apakah masih terjadi *error*. Pengujian pada *black box testing* tidak terlalu memperhatikan struktur internal (*source code*) dari program, tetapi lebih kepada menemukan kesalahan program. Pada *game* Pertempuran 10 November 1945 digunakan pengujian *black box testing*. Cara pengujian dilakukan melalui dua tahap, yaitu pengujian *alpha* (fungsional) dan pengujian *beta*.

IV.2.1 Pengujian Alpha

Pengujian dilakukan terhadap aplikasi untuk memastikan bahwa aplikasi dapat berjalan dengan benar sesuai dengan kebutuhan dan tujuan yang diharapkan. Pengujian *alpha* berfokus pada persyaratan fungsional perangkat lunak.

IV.2.1.1 Skenario Pengujian Aplikasi

Skenario pengujian menjelaskan pengujian terhadap sistem yang ada pada aplikasi *game* Pertempuran 10 November 1945. Skenario pengujian yang akan dilakukan pada aplikasi ini selengkapnya dapat dilihat pada table IV.1 :

Tabel IV.1 Skenario Pengujian Aplikasi

No	Komponen yang di uji	Skenario Pengujian	Jenis Pengujian
1.	Menu Mainkan	Klik menu Mainkan	*Black box*
2.	Menu Petunjuk	Klik menu Mainkan	*Black box*
		Klik menu Kembali	*Black box*
3.	Menu Keluar	Klik menu Keluar	*Black box*

No	Komponen yang di uji	Skenario Pengujian	Jenis Pengujian
4.	Berjalan	Tekan tombol W *keyboard*	*Black box*
		Tekan tombol A *keyboard*	*Black box*
		Tekan tombol D *keyboard*	*Black box*
		Tekan tombol S *keyboard*	*Black box*
5.	Meloncat	Menekan spasi pada *keyboard*	*Black box*
6.	Mengambil darah	Mengarahkan karakter hingga menyentuh *icon* darah menggunakan *mouse* dan *keyboard*	*Black box*
7.	Mengambil Peluru Roket	Mengarahkan karakter hingga menyentuh *icon* peluru roket menggunakan *mouse* dan *keyboard*	*Black box*
8.	Menembak	Mengarahkan *pointer* / *cursor* senjata menggunakan *keyboard* dan *mouse* lalu menekan tombol kiri *mouse* untuk mengeluarkan peluru .	*Black box*
9.	Mengganti Senjata	Menekan tombol 1 pada *keyboard* untuk menggunakan senjata laras panjang dan menekan tombol 2 pada *keyboard* untuk menggunakan senjata roket	*Black box*
10.	Memberhentikan Permainan	Menekan tombol P pada *keyboard*	*Black box*
11.	Mengambil petunjuk	Mengarahkan karakter hingga menyentuh *icon*	*Black box*

No	Komponen yang di uji	Skenario Pengujian	Jenis Pengujian
		panah menggunakan *mouse* dan *keyboard*	
12.	Algortima A* (A *star*)	Pencarian rute atau jalan terbaik untuk NPC	*White Box*
13.	Algoritma *Fuzzy Logic*	Perilaku NPC	*White Box*

IV.2.1.2 Kasus dan Hasil Pengujian (*Black Box Testing*)

Berikut ini merupakan kasus untuk menguji perangkat lunak yang sudah dibangun dengan menggunakan metode *black box testing* berdasarkan skenario pengujian aplikasi yang terdapat pada tabel IV.1 yang sudah dijabarkan. Hasil pengujian yang akan dilakukan pada aplikasi ini selengkapnya dapat dilihat pada tabel IV.2

Tabel IV.2 Skenario Pengujian Black Box

No	Kasus yang akan di uji	Skenario dan Hasil Uji		
		Skenario Uji	Hasil yang Diharapkan	Kesimpulan
1.	Menu Mainkan	Klik Menu Mainkan	Sistem menampilkan halaman *Scene Video*	[√] Berhasil [ ] Tidak Berhasil
2.	Menu Petunjuk	Klik Menu Petunjuk	Sistem menampilkan petunjuk berupa informasi kontrol yang digunakan dalam *game*	[√] Berhasil [ ] Tidak Berhasil
		Klik Kembali	Sistem menampilkan menu utama	[√] Berhasil [ ] Tidak Berhasil

No	Kasus yang akan di uji	Skenario dan Hasil Uji		
		Skenario Uji	**Hasil yang Diharapkan**	**Kesimpulan**
3.	Menu Keluar	Klik Menu Keluar	Menutup atau Keluar dari aplikasi	[√] Berhasil [ ] Tidak Berhasil
4.	Berjalan	Tekan tombol W *keyboard*	Karakter pemain bergerak kedepan	[√] Berhasil [ ] Tidak Berhasil
		Tekan tombol A *keyboard*	Karakter pemain bergerak kekiri	[√] Berhasil [ ] Tidak Berhasil
		Tekan tombol S *keyboard*	Karakter pemain bergerak kebelakang	[√] Berhasil [ ] Tidak Berhasil
		Tekan tombol D *keyboard*	Karakter pemain bergerak kekanan	[√] Berhasil [ ] Tidak Berhasil
5.	Meloncat	Menekan tombol spasi pada *keyboard*	Karaker akan meloncat	[√] Berhasil [ ] Tidak Berhasil
6.	Mengambil darah	Mengarahkan karakter hingga menyentuh *icon* darah menggunakan *mouse* dan *keyboard*	*Bar* darah pada permainan akan bertambah jika darah pemain memang tidak penuh dan *icon* darah menghilang	[√] Berhasil [ ] Tidak Berhasil
7.	Mengambil Peluru Roket	Mengarahkan karakter hingga menyentuh *icon* peluru roket menggunakan *mouse* dan *keyboard*	*Bar* peluru roket pada permainan akan bertambah jika peluru roket pemain memang tidak penuh dan *icon* peluru roket menghilang	[√] Berhasil [ ] Tidak Berhasil
8.	Menembak	Mengarahkan *pointer* / *cursor* senjata menggunakan *keyboard* dan	Senjata yang dipegang karakter pada *game* akan mengeluarkan peluru dan suara	[√] Berhasil [ ] Tidak Berhasil

No	Kasus yang akan di uji	Skenario dan Hasil Uji		
		Skenario Uji	**Hasil yang Diharapkan**	**Kesimpulan**
		mouse lalu menekan tombol kiri *mouse* untuk mengeluarkan peluru / *mouse*	tembakan secara bersama-sama	
9.	Mengganti Senjata	Menekan tombol 1 pada *keyboard* untuk menggunakan senjata laras panjang dan menekan tombol 2 pada *keyboard* untuk menggunakan senjata roket	Senjata laras panjang akan digunakan apablia pemain menekan tombol 1 dan Senjata roket akan digunakan apablia pemain menekan tombol 2	[√] Berhasil [ ] Tidak Berhasil
10.	Memberhenti-kan permainan	Menekan tombol P pada *keyboard*	*Game* akan mengeluarkan *screen pause*	[√] Berhasil [ ] Tidak Berhasil
11.	Mengambil petunjuk	Mengarahkan karakter hingga menyentuh *icon* panah menggunakan *mouse* dan *keyboard*	*Game* akan menampilkan *pop up* berupa teks dan suara	[√] Berhasil [ ] Tidak Berhasil

IV.2.1.3 Kasus dan Hasil Pengujian (White Box Testing*)*

1. Pengujian *White Box* Algoritma A*

Berikut ini merupakan algoritma A* untuk *level 2* yang penerapannya menggunakan bahasa pemrograman C# :

```
1.  public System.Type[] graphTypes{
2.  get {
```

```
3.    return astarData.graphTypes;}
4.    }
5.    public Matrix translate (float _X, float _Y, float _Z){
6.    m[12] = _X;
7.    m[13] = _Y;
8.    m[14] = _Z;
9.    return this;
10.   }
11.   public virtual int[] InitialOpen (BinaryHeapM open, Int3
      targetPosition, Int3 position, Path path, bool doOpen) {
12.   return BaseInitialOpen
13.   (open,targetPosition,position,path,doOpen);
14.   }
15.   public void Close () {
16.   if (readerStream != null) {
17.   readerStream.Close ();
18.   }
19.   if (writerStream != null) {
20.   if (anchors != null) {
21.   SerializeAnchors ();
22.   }
23.   writerStream.Close ();
24.   }
25.   }
26.   if (doOpen) {
27.   connectionCosts = costs;
28.   }
29.   if (connectionCosts == null) {
30.   return null;
31.   }
32.   int[] costs = connectionCosts;
33.   connectionCosts = new int[connectionCosts.Length];
34.   for (int i=0;i<connectionCosts.Length;i++) {
35.   connectionCosts[i] = (connections[i].position-
      position).costMagnitude;
36.   }
37.   if (!doOpen) {
38.   for (int i=0;i<connectionCosts.Length;i++) {
39.   Node other = connections[i];
40.   if (other.connections != null) {
41.   for (int q = 0;q < other.connections.Length;q++) {
42.   if (other.connections[q] == this) {
43.   other.connectionCosts[q] = connectionCosts[i];
44.   break;
45.   }
46.   }
47.   }
48.   }
49.   }
```

```
50.  if (connections[i].walkable && connections[i].area !=
     area) {
51.  stack.Push (connections[i]);
52.  connections[i].area = area;
53.  }
54.  if (doOpen) {
55.  nodeR.g = nodeR.parent.g+nodeR.cost+penalty +
     nodeRunData.path.GetTagPenalty(tags);
56.  public virtual void UpdateAllG (NodeRun nodeR, NodeRunData
     nodeRunData) {
57.  BaseUpdateAllG (nodeR, nodeRunData);
58.  }
59.  if (Open == null){
60.  return null;
61.  }
62.  return path;
```

a. *Flow Graph* Algoritma A*

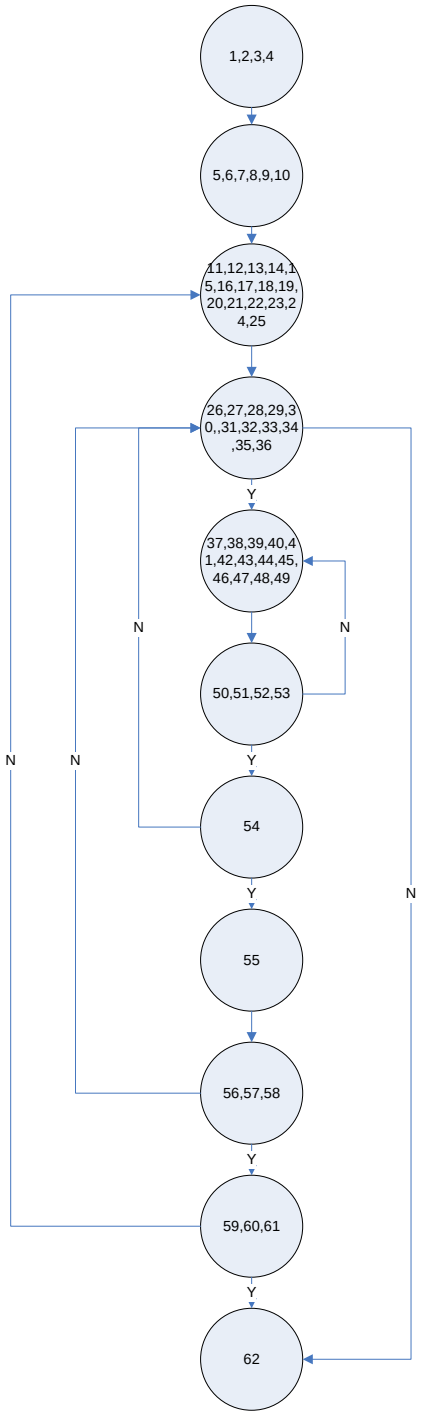
Berdasarkan *pseudocode* di atas maka *flow graph* algoritma A* adalah sebagai berikutt:

Keterangan:



= Menggambarkan kondisi

= Menggambarkan aksi



Gambar IV.9 *Flow Graph* Algoritma A*

Pada gambar IV.9 yaitu *Flow Graph* Algoritma A*, terdapat 11 simpul yang merepresentasikan alur dari baris kode program algoritma A*, pada simpul 1, terdapat 4 baris kode program berupa pemilihan tipe struktur data yang akan di gunakan pada program nantinya. Setelah eksekusi pemilihan struktur data telah tereksekusi maka Simpul ke 2 yaitu baris 5 sampai 10 akan di eksekusi, baris kode program ini bertujuan untuk membuat *node* atau simpul yang nantinya akan digunakan sebagai jalur dari pergerakan NPC (*Non Playbel Character*).

Simpul ke 3 pada *flow graph* di atas akan mengeksekusi baris kode program dari baris ke 10 sampai 25, kode program ini akan membuat 2 buah *method*. Kedua *method* tersebut merupakan *method* untuk menentukan atau menginisialisasi *open* dan *closed*. Fungsi dari *open* dan *closed* sendiri pada *game* yang dibangun adalah untuk menentukan *node* atau simpul mana saja yang harus dimasukan ke dalam *open* atau *closed*.

Simpul ke 4 yang akan mengeksekusi baris kode program dari baris ke 26 sampai 36 merupakan urutan untuk menentukan koneksi atau *pointer* dihitung dari costs (biaya) terhadap *Open*. Pada simpul ke 5, baris program yang dieksekusi adalah sebuah kondisi ketika *Open* yang berada pada simpul ke 4 tidak tereksekusi yaitu baris kode program ke 37 sampai 49.

Pada simpul ke 6 yaitu baris kode program dari 50 sampai 53 menentukan jalur yang *walkable* atau jalur yang dapat dilalui. Pada simpul ke 7 yaitu baris kode program ke 54 terdapat kondisi apabila *method doOpen* dieksekusi. Pada simpul ke 8 yaitu baris kode program ke 55 terdapat pembuatan suatu *parent* yang berfungsi untuk penentuan node-node atau simpul apa saja yang akan di telusuri berdasarkan parent terbaru.

Pada Simpul ke 9 yaitu baris kode program dari 56 sampai 58 terdapat method *UpdateAllG* yang berfungsi untuk meng*update* setiap nilai G yang diperoleh dari setiap pergerakan NPC. Apabila simpul ke 9 ini selesai di

eksekusi maka secara langsung simpul ke 9 ini akan *looping* terhadar baris kode program dari 26 sampai 36.

Pada simpul ke 10 yaitu baris kode program 59 sampai 61 adalah kondisi dimana nilai pada *Open* ini bernilai *null*, dan apabila baris ini dieksekusi maka simpul ke 3 yaitu baris ke 11 sampai ke 25 akan kembali di eksekusi.

Pada simpul terakhir yaitu baris kode program ke 62, apabila dari baris 1 sampai baris 61 telah dieksekusi dengan baik makan akan dikembalikan sebuah nilai berupa *path*. *Path* ini sendiri merupakan jalur dari pergerakan NPC untuk menemukan *player*.

b. *Cyclomatic Complexity* V(G)

Cyclomatic complexity akan digunakan dalam menguji *path* yang nantinya akan digunakan untuk mencari jumlah *path* dalam satu dan juga sebagai penentuan jalur atau *path* pencarian NPC terhadap *player*. Apabila menggunakan basis *path*, nilai yang dihitung untuk *Cyclomatic complexity* menentukan jalur indpenden dalam basis set suatu program dan memberi batas atas untuk jumlah uji coba yang harus dikerjakan bahwa seluruh perintah sekurang-kurangnya telah dikerjakan sekali.

Berikut ini adalah perhitungan *Cyclomatic complexity* yang nantinya akan digunakan sebagai jumlah aksi pada *independent path*.

E = 15, N = 11

V(G) = E – N + 2

V(G) = 15 – 11 + 2

V(G) = 4 + 2 = 6

Keterangan

E = Jumlah aksi

N = Jumlah Kondisi /*Node*

c. *Independent Path*

Independent Path adalah jalur yang melintasi atau melalui program dimanan sekurang-kurangnya terdapat proses perintah yang baru atau kondisi yang baru.

Path 1 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-54-55-56-57-58-59-60-61-62

Path 2 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-62

Path 3 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-54-55-56-57-58-59-60-61-62

Path 4 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-54-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-54-55-56-57-58-59-60-61-62

Path 5 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-54-55-56-57-58-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-54-55-56-57-58-59-60-61-62

Path 6 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-54-55-56-57-58-59-60-61-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-

30-31-32-33-34-35-36-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-54-55-56-57-58-59-60-61-62

d. *Graph* Matriks Algoritma A*

Graph metrik ini akan membantu uji coba *independent path*. Karena matrik persegi empat ini mempunyai ukuran yang sama dengan jumlah *node* pada *flowgraph*. Masing-masing baris dan kolom mempunyai hubugan dengan node yang telah ditentukan.

Tabel IV.3 *Graph Matriks* Algoritma A*

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

Keterangan:

1. Baris dan kolom merepresentasikan simpul
2. Nilai 1 merepresentasikan adanya keterhubungan antar simpul

e. Hasil Uji Algoritma A*

Tabel IV. 4 Hasil Uji Algoritma A*

Kasus dan Hasil Uji				
No. *Path*	**Butir uji algoritma A* *(Astar)***	***Node* (n), input kondisi**	**Pengamatan**	**Kesimpulan**
1	Rute = *null*, rute tidak ada	1(*void*) 2(*void*) 3(*void*) 4(*void*) 5(*void*) 6(*void*) 7(*void)* 8(*void*) 9(*void*) 10(*void*) 11(*void*)	Rute tidak ditemukan	*Path* dikerjakan
2	*Current Node = Goal*. Rute ditemukan	1,(*void*) 2(*void*) 3(*void*) 4(*void*) 11(void)	Rute ditemukan	*Path* dikerjakan
3	*Node* tidak dapat dilalui, pengujian *collision* atau	1(*void*) 2(*void*)	*Node* menjadi *unwalkable*	*Path* dikerjakan

Kasus dan Hasil Uji				
No. *Path*	**Butir uji algoritma A* *(Astar)***	***Node* (n), input kondisi**	**Pengamatan**	**Kesimpulan**
	unwalkable node	3(*void*) 4(*void*) 5(*void*) 6(*void*) 5(*void*) 6(*void*) 7(*void*) 8(*void*) 9(*void*) 10(*void*) 11(*void*)		
4	Masukkan semua *neighbor node* ke dalam *open* dan hitung nilai F,G,H. Set *current node*	1(*void*) 2(*void*) 3(*void*) 4(*void*) 5(*void*) 6(*void*) 7(*void*) 4(*void*) 5(*void*)	*Current node* dan nilai F,G,H masuk ke dalam *open*	*Path* dikerjakan

Kasus dan Hasil Uji				
No. *Path*	**Butir uji algoritma A* *(Astar)***	***Node* (n), input kondisi**	**Pengamatan**	**Kesimpulan**
		6(*void*) 7(*void)* 8(*void*) 9(*void*) 10(*void*) 11(*void*)		
5	Nilai G yang didapat lebih kecil dari nilai G sebelumnya. Kalkulasi ulang nilai F,G,H	1(*void*) 2(*void*) 3(*void*) 4(*void*) 5(*void*) 6(*void*) 7(*void)* 8(*void*) 9(*void*) 4(*void*) 5(*void*) 6(*void*) 7(*void)* 8(*void*)	Nilai F,G,H, dikalkulasi ulang	*Path* dikerjakan

Kasus dan Hasil Uji				
No. *Path*	**Butir uji algoritma A* *(Astar)***	***Node* (n), input kondisi**	**Pengamatan**	**Kesimpulan**
		9(*void*) 10(*void*) 11(*void*)		
6	*Open* = *null*, tidak ada *node* di dalam *open*	1(*void*) 2(*void*) 3(*void*) 4(*void*) 5(*void*) 6(*void*) 7(*void)* 8(*void*) 9(*void*) 10(*void*) 3(*void*) 4(*void*) 5(*void*) 6(*void*) 7(*void)* 8(*void*) 9(*void*)	Tidak ada *node* di dalam *open*	*Path* dikerjakan

Kasus dan Hasil Uji				
No. *Path*	**Butir uji algoritma A* *(Astar)***	***Node* (n), input kondisi**	**Pengamatan**	**Kesimpulan**
		10(*void*) 11(*void*)		

f. 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

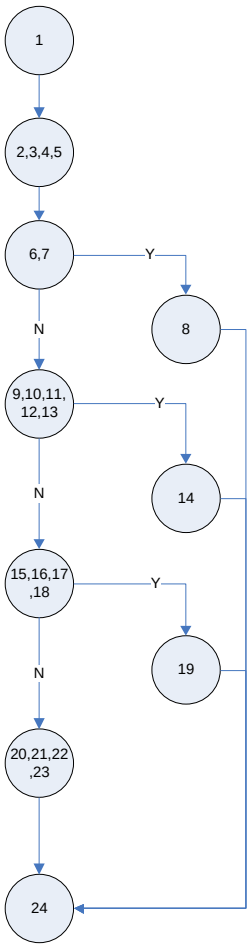
Berikut ini merupakan algoritma *fuzzy logic* yang penerapannya menggunakan bahasa pemrograman C#:

```
1.  jarak = (target.position -
    myTransform.position).magnitude;
2.  rules rule = new rules(jarak,hitPoints);
3.  defuzzyfikasi def = new defuzzyfikasi();
4.  rule.periksaRule();
5.  def.hasilDefuzzy();
6.  if((balikKeMarkas.position -
    myTransform.position).magnitude < 1.0){
7.  StartCoroutine(tambahDarah());
8.  }
9.  else if(def.hasilDefuzzyfikasi <= 10.0f) {
10. balikKeMarkas();
11. myTransform.position += myTransform.forward * moveSpeed *
    Time.deltaTime;
12. anim.animation.CrossFade(walkAnim);
13. playWalkSounds();
14. }
15. else if(def.hasilDefuzzyfikasi > 10.0f &&
    def.hasilDefuzzyfikasi <= 20.0f){
16. bertahan();
```

```
17.  anim.animation.CrossFade(attackAnim2);
18.  audio.Stop();
19.  }
20.  else if(def.hasilDefuzzyfikasi > 20.0f &&
     def.hasilDefuzzyfikasi <= 30.0f){
21.  menyerang();
22.  damage = Random.Range(20, 30);
23.  anim.animation.CrossFade(walkAnim);
24.  }
```

a. *Flow Graph* Algoritma *Fuzzy Logic*

Berdasarkan *pseudocode* di atas maka *flow graph* algoritma *fuzzy logic* adalah sebagai berikut:



Gambar IV.10 *Flow Graph* Algoritma *Fuzzy Logic*

Pada simpul 1 yaitu baris kode program ke-1, merupakan penentunan jarak antara *player* dengan musuh (NPC). Pada simpul ke-2, baris kode program 2 sampai 5 ada pembuatan objek dari *rule* dan *deffuzifikasi*, fungsi *rule* sendiri adalah untuk menentukan aturan *fuzzy logic*nya sendiri seperti, jika jarak dekat dan kekuatan kuat maka keputusannya adalah menyerang. Sedangkan *deffuzifikasi* sendiri digunakan untuk menentukan nilai rata-rata dari hasil *fuzzyfikasi* yang nantinya digunakan untuk menentukan keputusannya.

Pada simpul ke 3 baris kode program 6 sampai 7, ada kondisi dari *deffuzifikasi* jika nilai *deffuzifikasi* <10 maka musuh (NPC) akan menambahkan darah jika memang darah yang dimiliki musuh tidak penuh. Lalu pada simpul ke 4 baris kode program dari 9 sampai 13, musuh akan melarikan diri ke markas apabila nilai *deffuzifikasi* ≤10 dan pada simpul ke 4 ini akan dieksekusi juga *method walkAnim* yang berfungsi sebagai anmiasi berjalan musuh.

Pada simpul ke 5 yaitu baris kode program dari 15 sampai 18, musuh (NPC) akan mempunyai keputusan berupa bertahan dalam artian musuh akan diam ditempat dan tetap menyerang apabila player mendekat karena nilai *deffuzifikasi*nya berada diantara 10 dan 20 atau >10 dan ≤20. Pada simpul ke 5 juga akan dieksekusi *method attackAnim2* berfungsi sebagai animasi menyerang musuh.

Pada simpul ke 6 yaitu baris kode program dari 20 sampai 23, musuh (NPC) akan mempunyai keputusan berupa menyerang karena nilai *deffuzifika*sinya berada diantara 20 dan 30 atau >20 dan ≤30. Pada simpul ke 6 juga akan dieksekusi *method* menyerang().

Keterangan:



= Menggambarkan kondisi



= Menggambarkan aksi

b. *Cyclomatic Complexity* V(G)

E = 12, N = 10

V(G) = E – N + 2

V(G) = 12 – 10 + 2

V(G) = 2 + 2 = 4

Keterangan

E = Jumlah aksi

N = Jumlah Kondisi

c. *Independent Path*

Independent Path adalah jalur yang melintasi atau melalui program dimanan sekurang-kurangnya terdapat proses perintah yang baru atau kondisi yang baru.

Path 1 = 1-2-3-4-5-6-7-9-10-11-12-13-15-16-17-18-20-21-22-23-24

Path 2 = 1-2-3-4-5-6-7-8-24

Path 3 = 1-2-3-4-5-6-7-9-10-11-12-13-14-24

Path 4 = 1-2-3-4-5-6-7-9-10-11-12-13-15-16-17-18-19-24

d. *Graph Matriks* Algoritma *Fuzzy Logic*

Tabel IV.5 *Graph Matriks* Algoritma *Fuzzy Logic*

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

Keterangan:

1. Baris dan kolom merepresentasikan simpul
2. Nilai 1 merepresentasikan adanya keterhubungan antar simpul

e. Hasil Uji Algoritma *Fuzzy Logic*

Tabel IV.6 Hasil Uji Algoritma *Fuzzy Logic*

Kasus dan Hasil Uji				
No. *Path*	**Butir uji algoritma *fuzzy logic***	***Node* (n), input kondisi**	**Pengamatan**	**Kesimpulan**
1	NPC menambah *hitpoints* saat kembali ke markas	1,(*void*) 2(*void*) 3(*void*) 4(*void*) 12(*void*)	NPC menambah *hitpoints*	*Path* dikerjakan
2	Hasil *defuzzy* <= 10 NPC melarikan diri ke markas	1,(*void*) 2(*void*) 3(*void*) 5(*void*) 6(*void*) 12(*void*)	NPC melarikan diri ke markas	*Path* dikerjakan
3	Hasil *defuzzy* > 10 dan <= 20 NPC bertahan	1,(*void*) 2(*void*) 3(*void*) 5(*void*) 7(*void*) 8(*void*) 12(*void*)	NPC bertahan	*Path* dikerjakan

Kasus dan Hasil Uji				
No. *Path*	**Butir uji algoritma *fuzzy logic***	***Node* (n), input kondisi**	**Pengamatan**	**Kesimpulan**
4	Hasil *defuzzy* > 20 dan <= 30 NPC menyerang	1,(*void*) 2(*void*) 3(*void*) 5(*void*) 7(*void*) 9(*void*) 10(*void*) 12(*void*)	NPC menyerang	*Path* dikerjakan

f. 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.

IV.2.1.4 Kesimpulan Pengujian Alpha

Dari hasil pengujian yang dilakukan, dapat disimpulkan bahwa sistem sudah dapat berjalan sesuai dengan yang diharapkan beserta proses penanganan kesalahannya. Jadi, secara fungsional sistem sudah dapat menghasilkan keluaran yang diharapkan.

IV.2.2 Pengujian Beta

Pengujian *beta* merupakan pengujian yang dilakukan secara objektif, dimana pengujian langsung melibatkan para pengguna aplikasi permainan secara langsung dengan menggunakan kuisioner mengenai kepuasan para pengguna terhadap aplikasi

permainan yang dibangun. Pengujian dilakukan terhadap 30 responden dengan usia minimal 16 tahun dan setiap responden diberikan 8 pernyataan untuk kemudian dilakukan penilaian dengan memilih 4 pilihan yang telah disediakan untuk setiap pernyataannya.

IV.2.3 Kuesioner

Data hasil kuisioner yang telah diberikan dapat dicari presentasinya dari masing-masing jawaban dengan menggunakan rumus :

$$P = \frac{S}{skorideal} x100\%$$

Keterangan :

P = Nilai persentase yang dicari

S = Jumlah frekuensi dikalikan dengan nilai yang di tetapkan jawaban

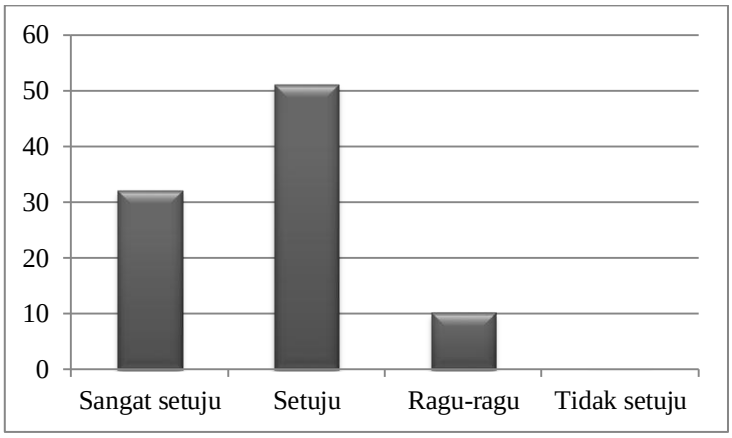
Skor ideal = Nilai tertinggi dikalikan dengan jumlah sampel

Berikut ini adalah hasil persentase masing-masing jawaban yang sudah dihitung nilainya dengan menggunakan rumus diatas.

1. Apakah anda setuju cerita sejarah pertempuran 10 November 1945 telah sesuai dengan *game*?

Tabel IV.7 Perhitungan persentase pertanyaan ke-1

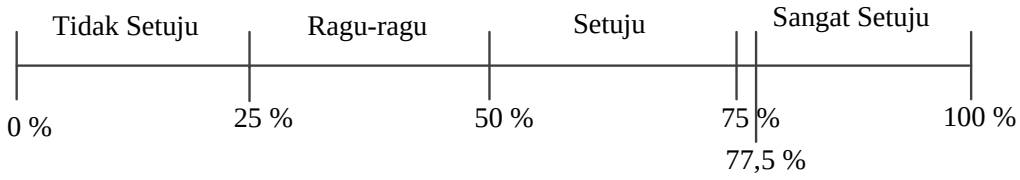
Pertanyaan	Keterangan	Skor	Frekuensi	S
1.	Sangat setuju	4	8	32
	Setuju	3	17	51
	Ragu-ragu	2	5	10
	Tidak setuju	1	0	0
Jumlah			30	93



Gambar IV.11 Grafik presentase skor pertanyaan ke-1

$$P = \frac{93}{120} x100 = 77{,}5\%$$

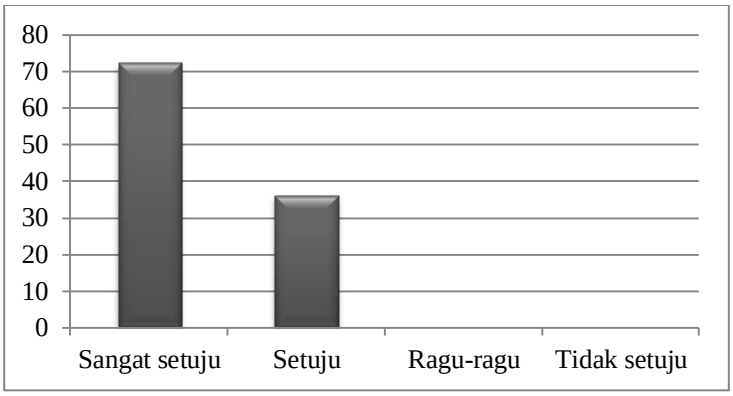
Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap sesuai tidaknya cerita Pertempuran 10 November 1945 dengan cerita pada *game* adalah 77,5% dari yang diharapkan 100%, atau dapat dikategorikan sebagai sangat setuju. Berikut kriteria interprestasi skor :



2. Apakah anda setuju setelah memainkan *game* ini, anda menjadi lebih tahu dan mengenal sejarah Pertempuran 10 November 1945?

Tabel IV.8 Tabel perhitungan persentase pertanyaan ke-2

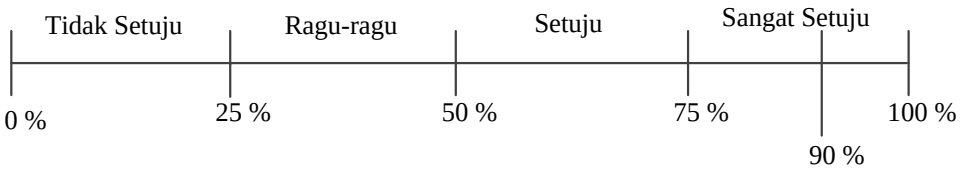
Pertanyaan	Keterangan	Skor	Frekuensi	S
2.	Sangat setuju	4	18	72
	Setuju	3	12	36
	Ragu-ragu	2	0	0
	Tidak setuju	1	0	0
Jumlah			30	108



Gambar IV.12 Grafik presentase skor pertanyaan ke-2

$$P = \frac{108}{120} x100 = 90\%$$

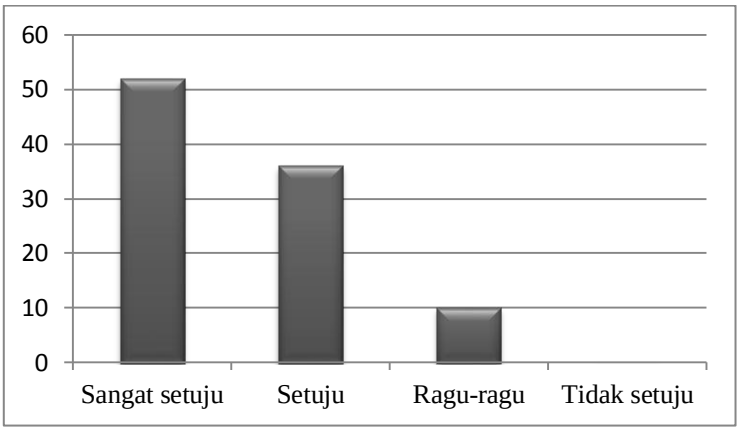
Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap pertanyaan tentang setuju tidakkah setelah memainkan *game* ini anda menjadi lebih tahu dan mengenal sejarah Pertempuran 10 November 1945 adalah 90% dari yang diharapkan 100%, atau dapat dikategorikan sangat setuju. Berikut kriteria interprestasi skor :



3. Apakah anda setuju materi atau informasi tentang sejarah Pertempuran 10 November 1945 yang terdapat pada *game* ini mudah untuk diingat?

Tabel IV.9 Tabel perhitungan persentase pertanyaan ke-3

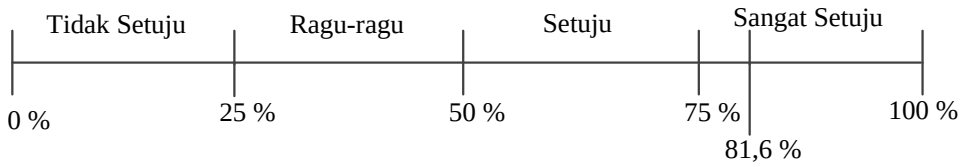
Pertanyaan	Keterangan	Skor	Frekuensi	S
3.	Sangat setuju	4	13	52
	Setuju	3	12	36
	Ragu-ragu	2	5	10
	Tidak setuju	1	0	0
Jumlah			30	98



Gambar IV.13 Grafik presentase skor pertanyaan ke-3

$$P = \frac{98}{120} x100 = 81{,}6\%$$

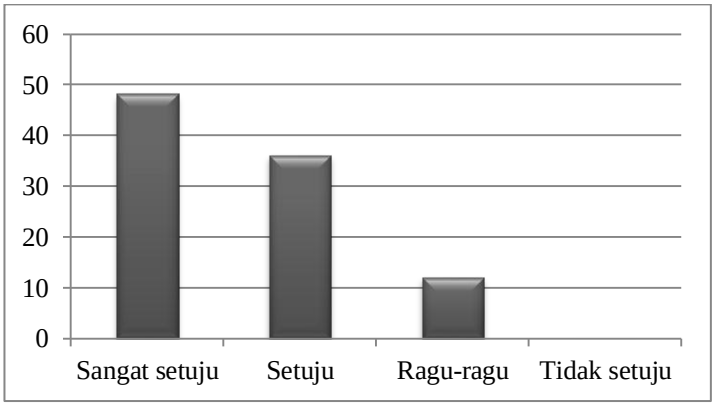
Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa materi atau informasi yang diberikan pada *game* Pertempuran 10 November 1945 mudah untuk diingat adalah 81,6% dari yang diharapkan 100%, atau dapat dikategorikan sebagai sangat setuju. Berikut kriteria interprestasi skor :



4. Apakah anda setuju *game* ini merupakan salah satu media pembelajaran yang menarik selain media pembelajaran dari buku atau pembelajaran konvensional di kelas?

Tabel IV.10 Tabel perhitungan persentase pertanyaan ke-4

Pertanyaan	Keterangan	Skor	Frekuensi	S
4.	Sangat setuju	4	12	48
	Setuju	3	12	36
	Ragu-ragu	2	6	12
	Tidak setuju	1	0	0
Jumlah			30	96

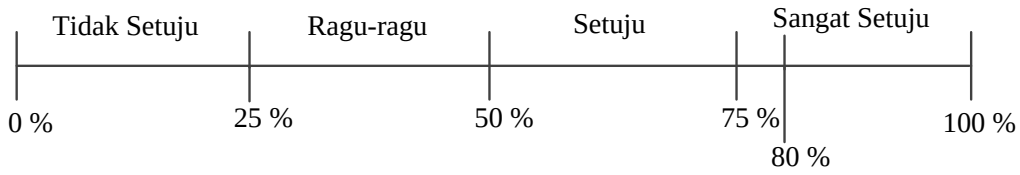


Gambar IV.14 Grafik presentase skor pertanyaan ke-4

$$P = \frac{96}{120} x100 = 80\%$$

Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap setuju atau ridaknya *game* ini sebagai salah satu media pembelajaran yang menarik adalah 80% dari yang diharapkan 100%,

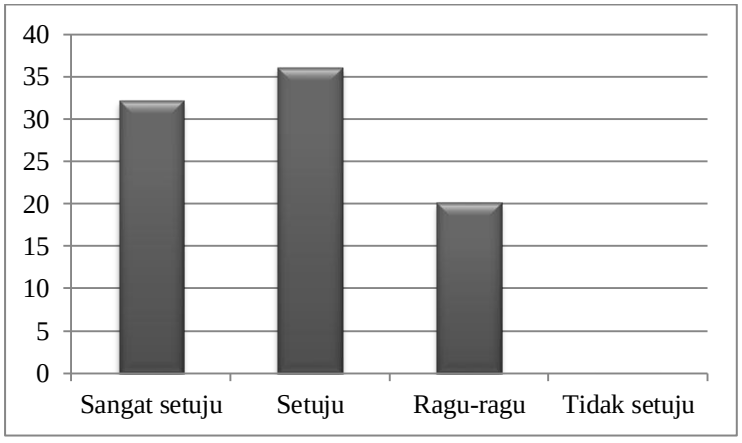
atau dapat dikategorikan sebagai sangat setuju. Berikut kriteria interprestasi skor :



5. Apakah anda setuju game sejarah Pertempuran 10 November mudah untuk dimainkan?

Tabel IV.11 Perhitungan persentase pertanyaan ke-5

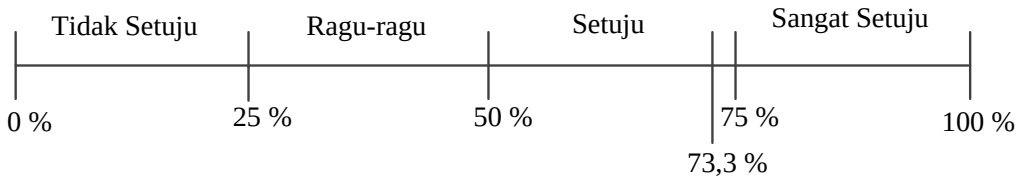
Pertanyaan	Keterangan	Skor	Frekuensi	S
5.	Sangat setuju	4	8	32
	Setuju	3	12	36
	Ragu-ragu	2	10	20
	Tidak setuju	1	0	0
Jumlah			30	88



Gambar IV.15 Grafik presentase skor pertanyaan ke-5

$$P = \frac{88}{120} x100 = 73{,}3\%$$

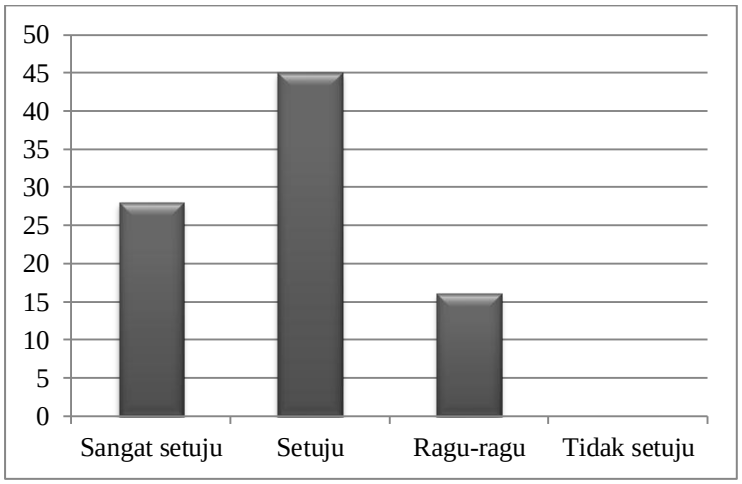
Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap mudah tidaknya *game* Pertempuran 10 November 1945 dimainkan adalah 73,3% dari yang diharapkan 100%, atau dapat dikategorikan sebagai setuju. Berikut kriteria interprestasi skor :



6. Apakah anda setuju musuh dalam permainan game ini sulit untuk dikalahkan?

Tabel IV.12 Perhitungan persentase pertanyaan ke-6

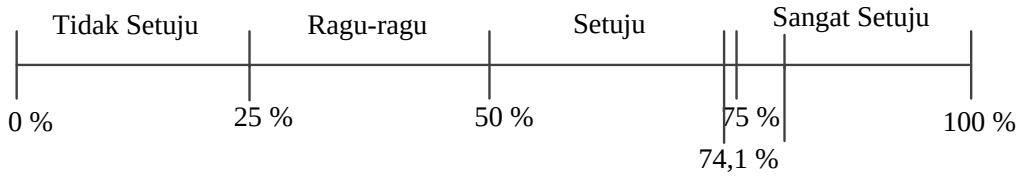
Pertanyaan	Keterangan	Skor	Frekuensi	S
6.	Sangat setuju	4	7	28
	Setuju	3	15	45
	Ragu-ragu	2	8	16
	Tidak setuju	1	0	0
Jumlah			30	89



Gambar IV.16 Grafik presentase pertanyaan ke-6

$$P = \frac{89}{120} x100 = 74{,}1\%$$

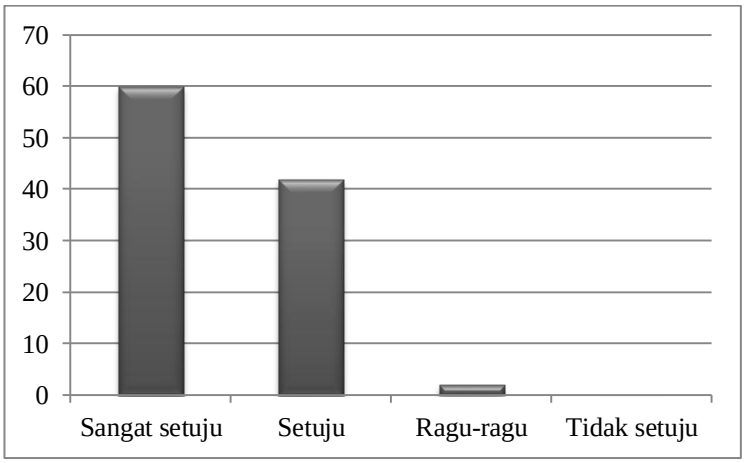
Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap apakah musuh dalam *game* ini sulit untuk dikalahkan adalah 74,1% dari yang diharapkan 100%, atau dapat dikategorikan sebagai setuju. Berikut kriteria interprestasi skor :



7. Apakah anda setuju grafis 3D yang terdapat pada *game* merupakan faktor kenyamanan dalam memainkan *game* ini

Tabel IV.13 Perhitungan persentase pertanyaan ke-7

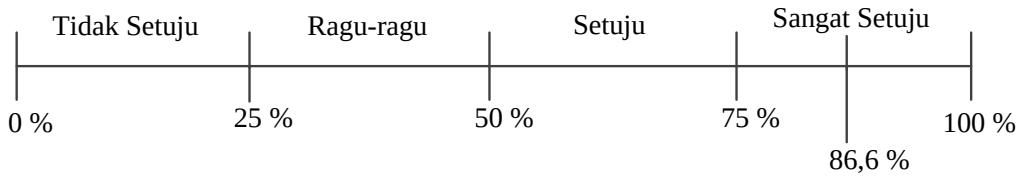
Pertanyaan	Keterangan	Skor	Frekuensi	S
7.	Sangat setuju	4	15	60
	Setuju	3	14	42
	Ragu-ragu	2	1	2
	Tidak setuju	1	0	0
Jumlah			30	104



Gambar IV.17 Grafik presentase pertanyaan ke-7

$$P = \frac{104}{120} x100 = 86{,}6\%$$

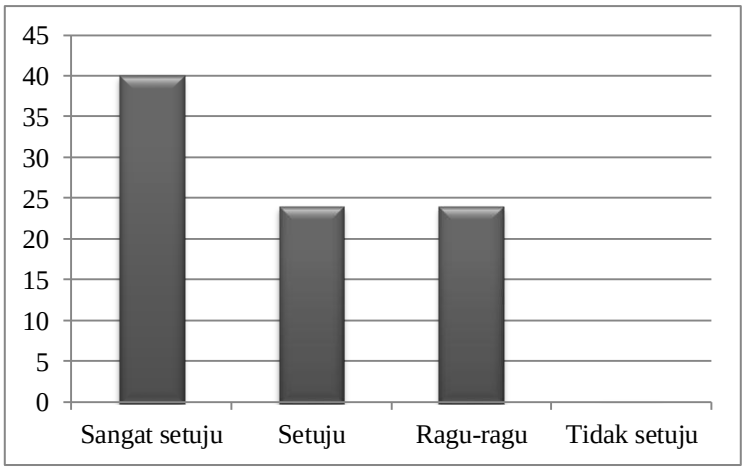
Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap grafis 3D yang terdapat pada *game* merupakan faktor kenyamanan dalam memainkan *game* adalah 86,6% dari yang diharapkan 100%, atau dapat dikategorikan sangat setuju. Berikut kriteria interprestasi skor :



8. Apakah anda setuju setelah memainkan *game* ini anda merasa terhibur?

Tabel IV.14 Perhitungan persentase pertanyaan ke-8

Pertanyaan	Keterangan	Skor	Frekuensi	S
8.	Sangat setuju	4	10	40
	Setuju	3	8	24
	Ragu-ragu	2	12	24
	Tidak setuju	1	0	0
Jumlah			30	88



Gambar IV.18 Grafik presentase pertanyaan ke-8

$$P = \frac{88}{120} x100 = 73{,}3\%$$

Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap terhiburkah anda setelah memainkan *game* ini adalah 73,3% dari yang diharapkan 100%, atau dapat dikategorikan setuju. Berikut kriteria interprestasi skor :

