BAB IV
IMPLEMENTASI DAN PENGUJIAN

IV.1. Implementasi

Implementasi merupakan tahap penciptaan perangkat lunak dan juga tahap kelanjutan dari perancangan aplikasi. Selanjutnya dilakukan uji kelayakan terhadap sistem aplikasi yang telah dibangun sehingga dapat memberikan masukan demi berkembangnya sistem aplikasi yang telah dibangun. Adapun sistem aplikasi yang telah dibangun adalah Implementasi Algoritma *Moving Target Search* Untuk Penyelamatan Korban Banjir. Selain itu, dibutuhkan perangkat keras dan perangkat lunak yang digunakan untuk melakukan implementasi dan pengujian terhadap aplikasi yaitu sebagai berikut:

IV.1.1. Implementasi Perangkat Keras

Implementasi perangkat keras yang digunakan untuk menjalankan *game* Penyelamatan Korban Banjir dapat dilihat pada spesifikasi tabel 4.1.

Tabel 4.1. Implementasi Perangkat Keras

No	Perangkat Keras	Spesifikasi
1	*Processor*	Intel Pentium 3.06 GHz
2	Monitor	Resolusi 1024 x 768
3	VGA	VGA *Card* 256 MB
4	RAM	1 Giga
6	Keyboard	Standar
7	*Mouse*	Standar

IV.1.2. Implementasi Perangkat Lunak

Adapun spesifikasi perangkat lunak yang dibutuhkan oleh pemain untuk menjalankan *game* Penyelamatan Korban Banjir dapat dilihat pada spesifikasi tabel 4.2.

Tabel 4.2. Implementasi Perangkat Lunak

No	Perangkat Lunak	Spesifikasi
1	*S*istem Operasi	Windows XP, dan Windows 7
2	Tool Compiler	Adobe Flash CS5

IV.2. Implementasi Aplikasi

Aplikasi *game* Penyelamatan Korban Banjir merupakan sebuah implementasi yang mengenalkan Algoritma *Moving Target Search* dan memberikan keterampilan mengenai simulasi penyelamatan korban banjir. *Game* Penyelamatan Koran Banjir dapat dijalankan pada sistem operasi windows.

IV.3. Implementasi *Class*

Class merupakan deskripsi sekelompok objek dari property (atribut), sifat (operasi), relasi antar objek dan semantik yang umum.

1. *Class Entity*

Tabel 4.3. Implementasi *Class Entity*

Nama *Class*	*Stereo type*	Keterangan
menu.as	*Entity*	Untuk menampilkan menu utama permaianan
Permainan.as	*Entity*	Untuk menampilkan Permainan
main.as	*Entity*	*Class* yang berfungsi awal ketika sistem dijalankan

2. *Class Main*

   Nama : main.as

   Stereo type : *main*

   Keterangan : *Class* yang memiliki fungsi awal dieksekusi ketika sistem dijalankan.

IV.4. Implementasi Antarmuka

Antarmuka merupakan sarana interaksi antara pengguna dengan sistem operasi. Berikut ini merupakan antarmuka yang ada pada *game* Penyelamatan Korban Banjir.

Tabel 4.4. Implementasi Antarmuka

No	Nama Antarmuka	Nama Movie Clip	Keterangan
1	Gambar 4.1. Tampilan Menu Utama	mc_menu	Tampilan awal pada saat memulai permainan.
2	Gambar 4.2. Tampilan Tentang	mc_TentangPembuat	Tampilan tentang identitas pengembang.
3	Gambar 4.3. Tampilan Permainan	mc_Permainan	Tampilan ketika simulasi berlangsung.
4	Gambar 4.4. Tampilan Keluar	mc_confirmForm	Tampilan konfirmasi untuk keluar dari program.

IV.5. Pengujian Sistem

Dalam melakukan pengujian terhadap aplikasi permainan *Penyelamatan Korban Banjir* dilakukan dengan satu macam teknik, yaitu pengujian *white box*. Adapun 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*.

IV.5.1. Pengujian Alpha

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

IV.5.1.1. Skenario Pengujian Aplikasi

Skenario pengujian menjelaskan pengujian terhadap sistem yang ada pada aplikasi permainan *Penyelamatan Korban Banjir*. Skenario pengujian dari aplikasi permainan *Penyelamatan Korban Banjir* dapat dilihat pada tabel 4.5.

Tabel 4.5. Skenario Pengujian Aplikasi

No	Komponen yang di uji	Jenis Pengujian
1	Menu Permainan	*Black Box*
2	Tentang Pembuat	*Black Box*
3	Algoritma *Moving Target Search* pada Tim Penyelamat	*White Box*

IV.5.1.2. Kasus dan Hasil Pengujian (*Black Box Testing*)

Pengujian ini dilakukan secara *black box* yaitu pengujian yang dilakukan dengan hanya memperhatikan masukan ke sistem dan respon sistem.

Tabel 4.6. Pengujian Menu Permainan

Kasus dan Hasil Uji			
No	**Data Masukan**	**Hasil Pengamatan**	**Hasl Pengujian**
1	Posisi Korban (x,y)	Bisa menempatkan korban dalam posisi yang telah ditetapkan	[√] Berhasil [ ] Tidak Berhasil
2	Posisi Tim Penyelamat (x,y)	Bisa menempatkan tim penyelamat dalam posisi tang telah ditetapkan	[√] Berhasil [ ] Tidak Berhasil

Tabel 4.7. Pengujian Menu Tentang Pembuat

Kasus dan Hasil Uji			
No	**Data Masukan**	**Hasil Pengamatan**	**Hasl Pengujian**
1	Tentang Pembuat	Menampilkan menu tentang pembuat	[√] Berhasil [ ] Tidak Berhasil

IV.5.1.3. Kasus dan Hasil Pengujian (*White Box Testing*)

Dibawah ini merupakan kasus algoritma *Moving Target Search* (MTS) untuk menguji perangkat lunak yang sudah dibangun dengan menggunakan metode *white box*, berdasarkan skenario pengujian aplikasi yang dijabarkan.

1. ***Pseudocode* Algoritma *Moving Target Search***

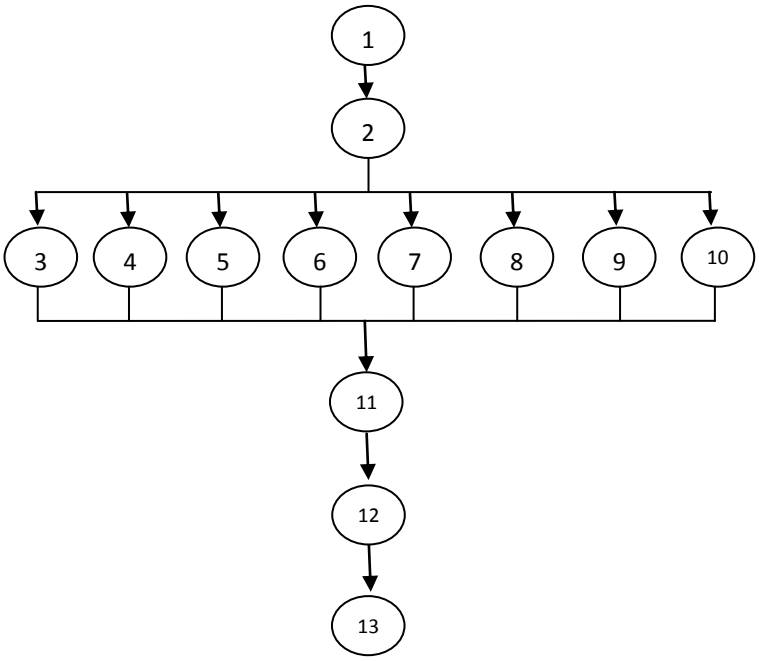
   Pseudocode algoritma *Moving Target Search* pada tim penyelamat yang penerapannya menggunakan bahasa pemrograman *Action Script 3.0.* Berikut *Pseudocode Moving Target Search*:

```
1.  for (var i:int = 0; i < nTT.length; i++)
    {
2.  if (nTT[i] == Math.min(nTT[0],nTT[1],nTT[2],nTT[3],nTT[4],nTT[5],nTT[6],nTT[7]))
    then
3.  if (i == 0)
    then
    penyelamat.x = pPenyelamatX - 1; // [kiri]
       pPenyelamatX = penyelamat.x;
       tes.splice(0,1,true);
       cekStatus();
       trace("kiri");
       //trace(nTT[i]);
       nTT.splice(0,nTT.length);
    else
4.  if (i == 1)
    then
    penyelamat.y = pPenyelamatY - 1; // [Atas]
       pPenyelamatY = penyelamat.y;
       tes.splice(0,1,true);
       cekStatus();
       trace("atas");
       //trace(nTT[i]);
       nTT.splice(0,nTT.length);
    else
5.  if (i == 2)
    then
    penyelamat.x = pPenyelamatX + 1; // [Kanan]
       pPenyelamatX = penyelamat.x;
       tes.splice(0,1,true);
       cekStatus();
       trace("kanan");
       //trace(nTT[i]);
       nTT.splice(0,nTT.length);
    else
6.  if (i == 3)
    then
    penyelamat.y = pPenyelamatY + 1; // [Bawah]
       pPenyelamatY = penyelamat.y;
       tes.splice(0,1,true);
       cekStatus();
       trace("bawah");
       //trace(nTT[i]);
       nTT.splice(0,nTT.length);
    else
7.  if (i == 4)
    then
    penyelamat.x = pPenyelamatX - 1; // [KiriAtas]
    penyelamat.y = pPenyelamatY - 1; // [KiriAtas]
       pPenyelamatX = penyelamat.x;
       pPenyelamatY = penyelamat.y;
       tes.splice(0,1,true);
          cekStatus();
          trace("kiriAtas");
          nTT.splice(0,nTT.length);
       trace(nTT.length);
       //trace(nTT[i]);
    else
8.  if (i == 5)
    then
    penyelamat.x = pPenyelamatX + 1; // [KananAtas]
    penyelamat.y = pPenyelamatY - 1; // [KananAtas]
       pPenyelamatX = penyelamat.x;
       pPenyelamatY = penyelamat.y;
       tes.splice(0,1,true);
       cekStatus();
       trace("kananAtas");
       nTT.splice(0,nTT.length);
       ///trace(nTT[i]);
    else
9.  if (i == 6)
    then
    penyelamat.x = pPenyelamatX + 1; // [KananBawah]
    penyelamat.y = pPenyelamatY + 1; // [KananBawah]
       pPenyelamatX = penyelamat.x;
       pPenyelamatY = penyelamat.y;
       tes.splice(0,1,true);
       cekStatus();
       trace("kananBawah");
       //trace(nTT[i]);
       nTT.splice(0,nTT.length);
    else
10. if (i == 7)
    then
    penyelamat.x = pPenyelamatX - 1; // [KiriBawah]
    penyelamat.y = pPenyelamatY + 1; // [KiriBawah]
       pPenyelamatX = penyelamat.x;
       pPenyelamatY = penyelamat.y;
       tes.splice(0,1,true);
       cekStatus();
       trace("kiriBawah");
       //trace(nTT[i]);
        nTT.splice(0,nTT.length);
```

```
11. private function pers(tx:int,py:int,gx:int,gy:int)
    hDiagonal = (Math.abs(tx-gx)+Math.abs(py-gy))*-1;
    hOrthogonal = (Math.abs(tx-gx)+Math.abs(py-gy));
    hn = hDiagonal + (hOrthogonal-(2*hDiagonal));
12. f = hn - 1;
    //trace(f);
    return f;
13. end
```

2. *Flow Graph*

Berdasarkan *pseudocode* di atas maka *flow graph* algoritma *Moving Target Search* dapat dilihat pada gambar 4.1.



Gambar 4.1. *Flow Graph Moving Target Search*

Keterangan:

= Menggambarkan kondisi.

= Menggambarkan aksi.

3. *Cyclomatic complexity* V(G)

E = 19, N = 13

V(G) = (E – N) + 2

V(G) = (19 – 13) + 2

V(G) = 8

Keterangan :

E = Jumlah Aksi

N = Jumlah Kondisi

4. Grap Matrix *Moving Target Search*

	1	2	3	4	5	6	7	8	9	10	11	12	13	CC
1		1												0
2			1	1	1	1	1	1	1	1				7
3											1			0
4											1			0
5											1			0
6											1			0
7											1			0
8											1			0
9											1			0
10											1			0
11												1		0
12													1	0
13														0
													CC =	7 + 1 = 8

Gambar 4.2. Grap Matrix *Moving Target Search*

Keterangan :

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

Dari hasil pengujian *white box* dapat disimpulkan bahwa jumlah CC pada *graph matrix* sama dengan jumlah *Cyclomatic complexity* V(G) yaitu 8, sehingga tidak terjadi *error* pada setiap baris *source code*.