BAB 4
IMPLEMENTASI DAN PENGUJIAN SISTEM

4.1 Implementasi

Tahap implementasi dilakukan setelah analisis dan perancangan sistem selesai dilakukan, selanjutnya akan diimplementasikan kedalam bahasa pemograman dan diuji kedalam aplikasi. Aplikasi yang telah dibangun akan diimplementasikan untuk mengetahui sistem dapat berjalan sesuai tujuan atau tidak. Tujuan dari implementasi adalah untuk menerapkan perancangan yang telah dilakukan terhadap sistem, sehingga pemain dapat memberikan masukan untuk perkembangan sistem yang telah dibangun pada *game puzzle* kakuro dalam menerapkan algoritma *backtracking.*

4.1.1 Implementasi Perangkat Keras

Spesifikasi perangkat keras *(hardware)* yang digunakan dalam implementasi algoritma *backtracking* pada *game puzzle* kakuro dapat dilihat pada tabel 4.1.

Tabel 4.1 Spesifikasi perangkat keras

Komputer	Spesifikasi Perangkat Keras
Prosesor	Prosesor 1,8 Ghz
Resolusi Layar	1024 x 768 *pixel*
Memori	Memori 2 Giga
Hardisk	*Hardisk* 100 Gb
Keyboard dan *touchpad*	Standar

4.1.2 Implementasi Perangkat Lunak

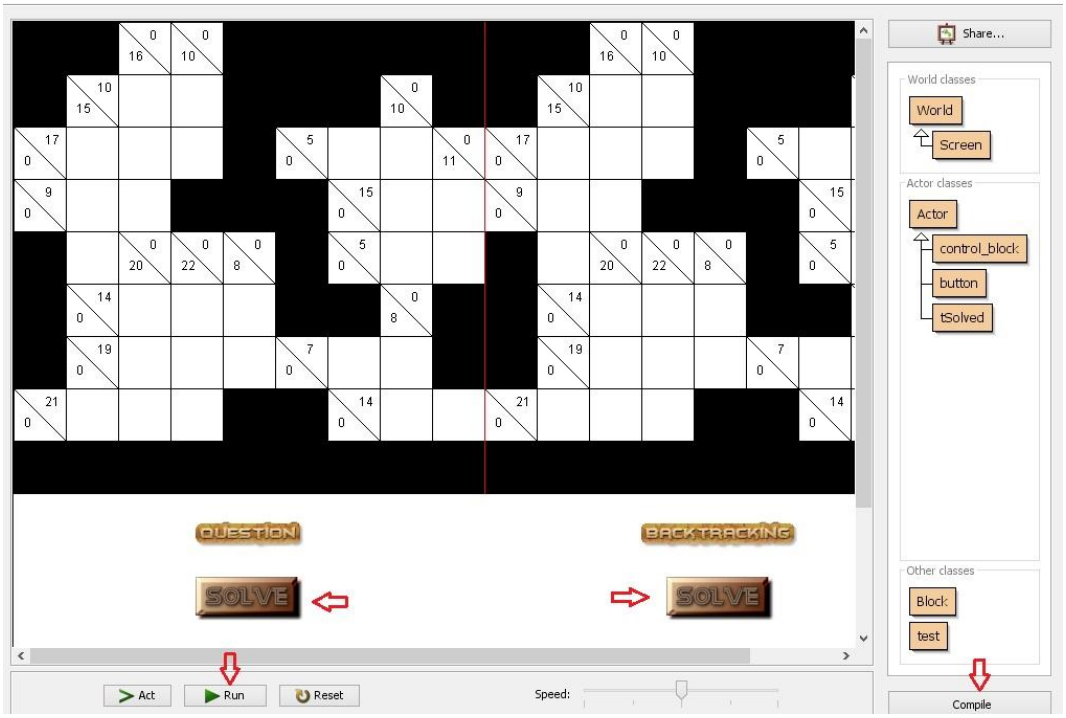
Spesifikasi perangkat lunak *(software)* yang digunakan dalam implementasi algoritma *backtracking* pada *game puzzle* kakuro dapat dilihat pada tabel 4.2.

Tabel 4.2 Spesifikasi perangkat lunak

Spesifikasi Perangkat Lunak
Windows 8
Greenfoot

4.1.3 Implementasi Antarmuka

Algoritma yang telah dirancang pada sistem, diimplementasikan kedalam program dengan antarmuka sebagai berikut :



Gambar 4.1 Tampilan antarmuka

Pada tampilan ini, saat menjalankan aplikasi sudah menampilkan soal permainan dengan sistem *random* dan menampilkan tombol-tombol utama untuk memainkan *game puzzle* kakuro ini diantaranya terdapat tombol *run* yang berfungsi untuk menjalankan program apabila pemain ingin mengisikan nilai didalam kotak jawaban, tombol *compile* berfungsi untuk me-*restart* ulang soal permainan secara *random*, serta tombol *solve* (kanan) berfungsi untuk menampilkan hasil jawaban oleh menggunakan algoritma *backtracking* dan tombol *solve* (kiri) untuk mencocokan jawaban yang diisi oleh pemain dengan

jawaban yang telah dihasilkan oleh sistem dengan algoritma *backtracking.* Hasil dari tombol *solve* (kiri) akan menampilkan warna merah pada kotak jawaban yang tidak terisikan nilai maupun nilai jawaban yang diisikan tidak sesuai dengan nilai jawaban yang dihasilkan oleh sistem. Terdapat dua kondisi dalam memainkan permainan *puzzle* kakuro yang dibuat, yaitu :

1. Kondisi disaat kotak jawaban terisi.

   Pada kondisi ini, pemain dapat mengisikan nilai angka dari 1 hingga 9 dalam pengisian jawaban. Pengisian pada kotak jawaban, dapat diisi secara penuh menyeluruh atau diisi dengan keadaan beberapa kotak jawaban yang belum terisi. Tombol *solve* dalam kondisi ini, akan menampilkan kotak berwarna merah apabila terjadi kesalahan dalam pengisian jawaban dan akan menampilkan pesan sukses apabila pemain telah menjawab soal dengan benar.
2. Kondisi disaat kotak jawaban kosong.

   Pada kondisi ini, pemain dapat langsung melihat hasil jawaban dengan menekan tombol *solve* tanpa harus mengisi jawaban pada kotak permainan. Hasil jawaban yang didapat menggunakan algoritma *backtracking* dengan menerapkan aturan permainan *puzzle* kakuro, yaitu tidak boleh terdapat angka yang sama pada satu lajur serta jumlah nilai angka yang sudah di akumulasikan pada satu lajur harus sesuai dengan nilai soal.

4.2 Pengujian Sistem

Pengujian sistem dilakukan untuk menemukan kekurangan atau kesalahan pada perangkat lunak yang akan diuji. Pengujian ini bermaksud untuk mengetahui perangkat lunak yang dibuat sudah memenuhi kriteria yang sesuai dengan tujuan perancangan perangkat lunak tersebut. Pada pengujian perangkat lunak ini menggunakan metode pengujian *white box* dan *black box.* Pengujian dengan metode *white box* adalah pengujian dengan melihat kedalam modul untuk meneliti kode-kode program yang ada, dan menganalisis apakah terdapat kesalahan atau tidak. Pengujian dengan menggunakan metode *black box* berfokus pada persyaratan fungsional perangkat lunak.

4.2.1 Pengujian *White Box*

Pengujian *white box* merupakan metode perancangan *test case* yang menggunakan struktur *control* dari perancangan metode dalam mendapatkan *test case.* Adapun metode yang digunakan dalam pengujian *white box* ini adalah metode *basic path.* Metode *basic path* memungkinak perancang uji kasus untuk memperoleh ukuran kompleksitas logis dari sebuah perancangan prosedral dan menggunakan ukuran ini sebagai sebuah panduan dalam mendefinisikan sekumpulan dasar dari alur eksekusi.

Pengujian *white box* akan digunakan pada algoritma *backtracking* untuk mengetahui kinerja logika berdasarkan *flowchart* yang telah dibuat pada tahap analisis. Pengujian *white box* meliputi beberapa tahap, yaitu :

4.2.1.1 Pengujian Algoritma *Backtracking* Pada *Game puzzle* Kakuro

Pada pengujian ini, algoritma *backtracking* akan diuji dalam pembangkitan soal pada *game puzzle* kakuro. Berikut merupakan pengujian algoritma *backtracking* pada pembangkit soal dengan menggunakan *whitebox.*

1. Mengubah *source code* menjadi *flowchart,* kemudian diubah kedalam *flowgraph.* Berikut *source code* algoritma *backtracking* dalam membangkitkan soal permainan :

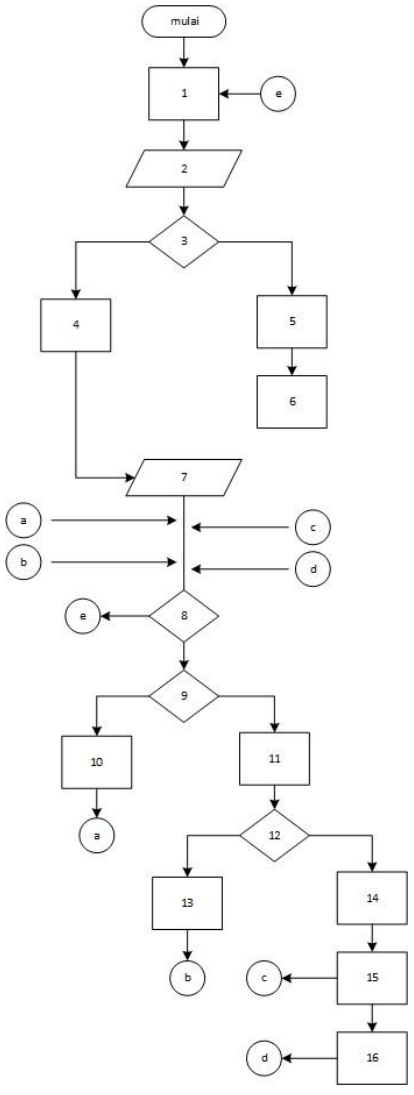
Tabel 4.3 *Source code* algoritma *backtracking* pada pembangkit soal

```
    public void backtrack(int posX, int posY, int tBlock){
1      cekRow(posX, posY, tBlock);
2      int nIterasi = getIteration(posX, posY, tBlock);
       int nBlock = tBlock - nIterasi;
       posX = blocks[0].length - tBlock;
3        if(nBlock>0){
4           randomx(posX,posY, nIterasi);
            backtrack(posX+nIterasi, posY, nBlock);
5        }else if(nBlock≤0){
            randomx(posX,posY, nIterasi);
6        }else if(nBlock≤0){
     }
    }

    public void randomx(int posX, int posY, int nIterasi){
7      int posAkhir = posX + nIterasi;
8      for(int x=posX; x<posAkhir; x++){
9         if(nIterasi <= 2){
```

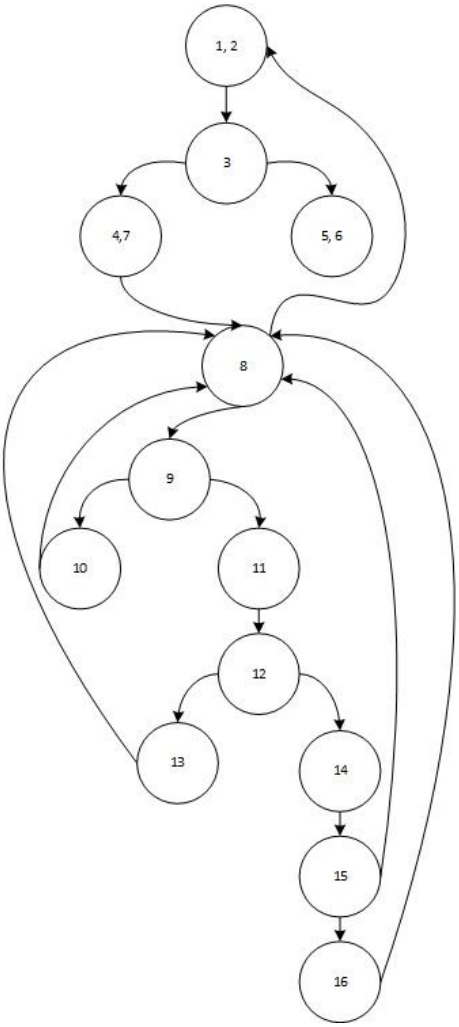
```
10              blocks[x][posY].setBlok(true);
11          }else if(nIterasi > 2){
12              if(x == posX){
13                  blocks[x][posY].setBlok(true);
14              }else{
15                  if(blocks[x][posY].getBlok()==false){
16                      int nilai = Greenfoot.getRandomNumber(9)+1;
                        blocks[x][posY].setMiddle(nilai);
                        cekDuplikat(posX, posY, x, posY);
                        cekTotal(posX, posY, x, posY);
                    }
                }
            }
        }
    }
```

2. Mengubah *source code* menjadi *flowchart*, dapat dilihat pada gambar 4.2.



Gambar 4.2 *Source code* menjadi *flowchart*

3. Mengubah *flowchart* menjadi *flowgraph*, dapat dilihat pada gambar 4.3.



Gambar 4.3 *Flowchart* menjadi *flowgraph*

4. *Cyclomatic complexity*

   Dari diagram alir pada gambar 4.3 dapat dihitung *cyclomatic complexity*, yakni :

   V(G) = 17 bsusur + 13 simpul + 2

   V(G) = 6

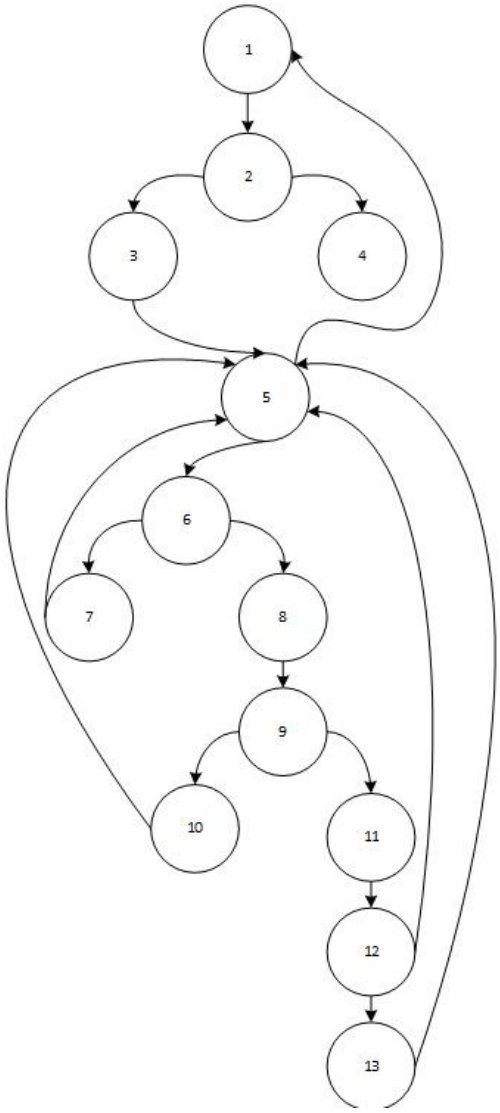
5. *Predicate node*

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

   V(G) = 5 *predicate node* + 1

   V(G) = 6

6. Penomoran ulang *flowgraph*, dapat dilihat pada gambar 4.4.



Gambar 4.4 Penomoran ulang *flowgraph*

7. *Graph matrix*, dapat dilihat pada tabel 4.4.

Tabel 4.4 *Graph matrix*

Node	1	2	3	4	5	6	7	8	9	10	11	12	13	Perhitungan
1		1												0
2			1	1										1
3					1	1								1
4														0
5						1								0
6							1	1						1
7					1									0

Node	1	2	3	4	5	6	7	8	9	10	11	12	13	Perhitungan
8									1					0
9										1	1			1
10					1									0
11												1		0
12					1								1	1
13					1									0
Jumlah (x)														5

V(G) = x + 1

V(G) = 5 + 1

V(G) = 6

4.2.2 Pengujian *Black Box*

Pengujian *black box* adalah pengujian yang mengatakan benar bila *input* yang diberikan akan menghasilkan *output* yang sesuai dengan spesifikasi program tanpa memperhatikan struktur logika dalam program.

4.2.2.1 Rencana Pengujian

Rencana pengujian yang akan dilakukan pada aplikasi ini dapat dilihat pada tabel 4.5.

Tabel 4.5 Rencana Pengujian

Komponen yang diuji	Skenario pengujian	Jenis pengujian
Menu Utama	Memilih kotak	*Black box*
	Memasukan angka	*Black box*
	Menekan tombol *solve*	*Black box*

4.2.2.2 Kasus Dan Hasil Pengujian *(Black Box)*

Kasus dan hasil uji aplikasi yang sudah dibangun dengan menggunakan metode *black box* berdasarkan pengamatan yang dilakukan dan menghasilkan kesimpulan dari pengujian tersebut. Kasus dan hasil pengujian dapat dilihat pada tabel 4.6

Tabel 4.6 Kasus dan hasil pengujian

Kasus yang diuji	Skenario pengujian	Hasil yang diharapkan	Kesimpulan
Menu Utama	Memilih kotak	*Pointer* dapat menunjuk kotak yang dipilih.	[√] Berhasil [ ] Tidak berhasil
	Memasukan angka	Sistem dapat meng-*input*-kan nilai yang dimasukan pemain.	[√] Berhasil [ ] Tidak berhasil
	Menekan tombol *solve*	Sistem memberikan jawaban penyelesaianan atau menunjukan kesalahan dengan menampilkan warna merah pada kotak yang terjadi kesalahan dalam pengisian nilai.	[√] Berhasil [ ] Tidak berhasil

4.3 Kesimpulan

Dari hasil pengujian dapat disimpulkan bahwa algoritma *backtracking* dapat berfungsi dalam pembangkitan soal pada *game puzzle* kakuro. Pada pembangkitan soal, algoritma *backtracking* dapat bekerja dengan baik dalam membangkitkan nilai soal permainan maupun dalam menyelesaikan jawaban pada *game puzzle* kakuro ini. Nilai soal dan nilai jawaban yang dihasilkan menggunakan algoritma *backtracking* disesuaikan dengan syarat permainan *puzzle* kakuro yaitu nilai soal dan nilai jawaban akan saling terkait, baik dari nilai soal satu ke nilai soal lainnya maupun dari nilai jawaban satu ke nilai jawaban lainnya.