BAB 4 IMPLEMENTASI DAN PENGUJIAN

Pada bab ini akan dilakukan implementasi dan pengujian terhadap aplikasi yang dibangun. Tahapan ini dilakukan setelah analisis dan perancangan selesai dilakukan dan selanjutnya akan diimplementasikan kedalam bahasa pemograman. Setelah implementasi maka dilakukan pengujian terhadap aplikasi. Pengujian aplikasi yang telah dibangun dilakukan dengan metode pengujian Black box dan White Box untuk mengetahui fungsionalitas pada game yang dibangun, serta pengujian performansi untuk mengetahui kehandalan program saat dijalankan.

4.1 Implementasi

Tujuan implementasi adalah untuk menerapkan perancangan yang telah dilakukan terhadap aplikasi sehingga pengguna dapat melihat gambaran yang lebih jelas untuk hasil dari tujuan pembuatan aplikasi *game puzzle* ini. Berikut ini adalah spesifikasi perangkat keras dan perangkat lunak yang digunakan pada implementasi aplikasi ini.

4.1.1 Perangkat Keras

Perangkat keras yang digunakan untuk mengimplementasikan *game* dapat menggunakan sebuah komputer *desktop* atau perangkat *mobile* seperti *smartphone* dan *tablet PC* yang di dalamnya terdapat *web browser* modern yang dapat menjalankan semua fungsi atau *method* dengan kode *javascript*. Pada halaman berikut adalah spesifikasi perangkat keras untuk sebuah perangkat komputer *desktop* yang digunakan untuk pengimplementasian aplikasi ini.

Tabel 4. 1 Spesifikasi perangkat keras komputer untuk implementasi aplikasi

Perangkat Keras	Spesifikasi
Prosesor	Intel i5 2.26GHz
Monitor	LCD 11 inci
Memori	DDR3 4GB
Harddisk	SATA 320GB
Keyboard	*Onboard*
Trackpad	*Onboard*

4.1.2 Perangkat Lunak

Perangkat lunak yang digunakan dalam pengimplementasian perancangan aplikasi *game* ini adalah sebagai berikut :

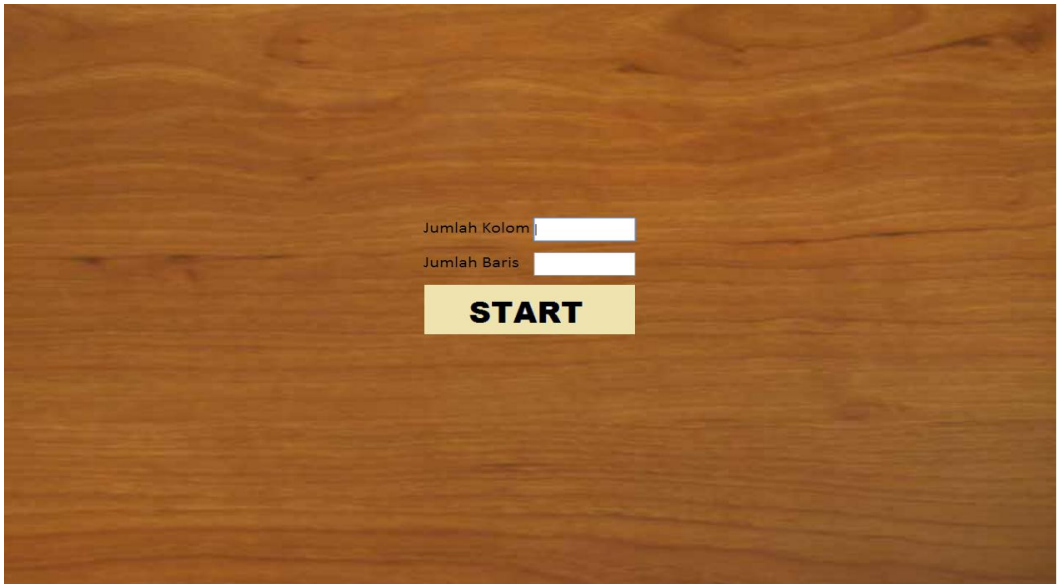
1. Sistem Operasi menggunakan *OSX Maverick*.
2. *Web Browser* menggunakan *Google Chrome* versi 35.0.1916.153 m.
3. *Text Editor* menggunakan *Sublime Text 3*.0.
4. *Framework Javascript* menggunakan *Enchant.JS* versi 0.8.1.

4.1.3 Implementasi Antarmuka

Interaksi antarmuka merupakan media interaksi antara pengguna dengan aplikasi.

4.1.3.1 Tampilan Awal

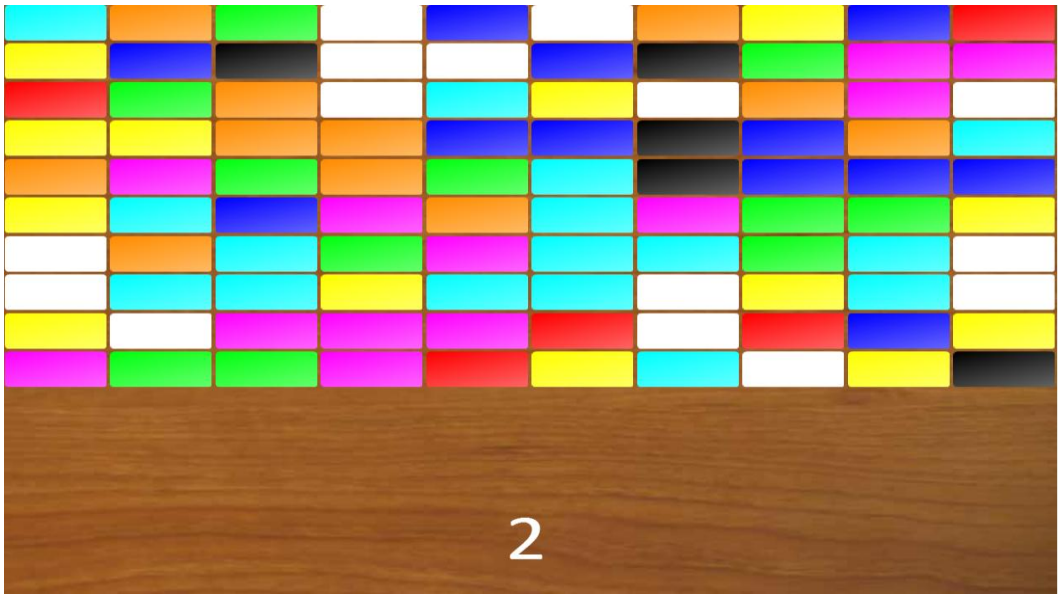
Tampilan awal adalah tampilan yang muncul ketika pengguna pertama kali mengakses aplikasi. Pada bagian ini, terdapat tombol *start* yang berfungsi sebagai inisialisasi ketika *game* dimulai. Sebelum pengguna memulai permainan dengan menekan tombol *start*, pengguna memasukkan nilai jumlah baris dan kolom pola utama yang akan di-*generate*. Pada halaman berikut merupakan tampilan awal aplikasi.



Gambar 4. 1 Tampilan awal aplikasi

4.1.3.2 Tampilan Susunan *Block* Warna Utama

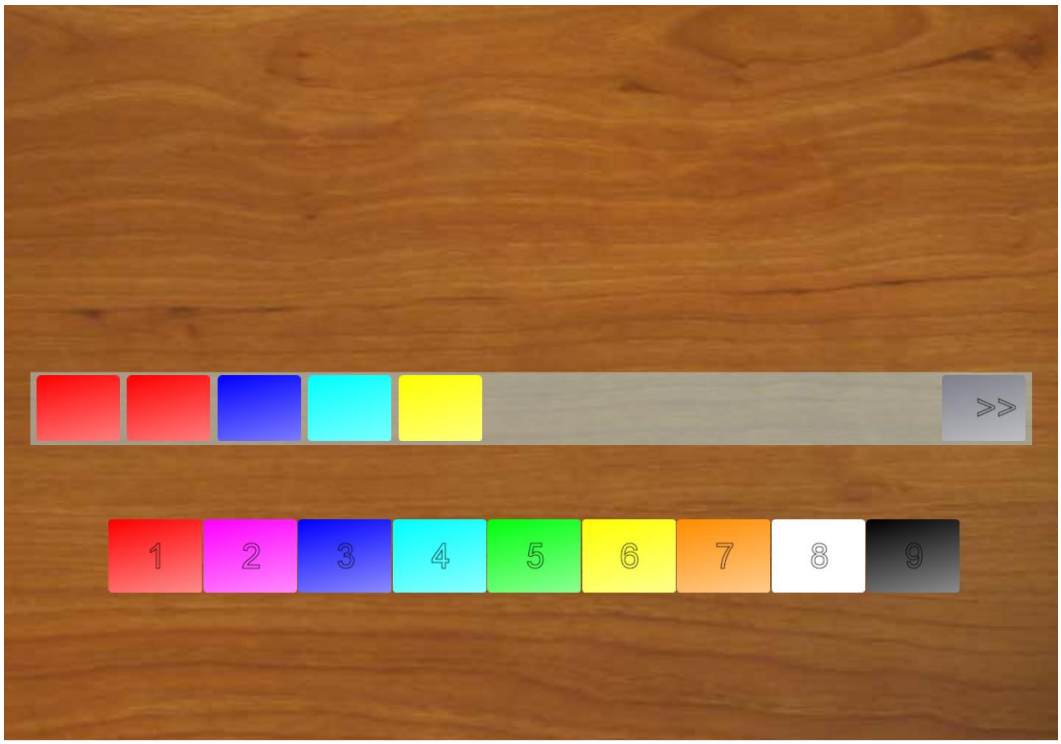
Tampilan ini muncul setelah penguna memasukkan nilai baris dan kolom dan menekan tombol *start*. Susunan warna tampil dengan warna acak dengan jumlah sesuai dengan baris dan kolom masukan pengguna. Dibawah susunan warna tersebut, terdapat sebuah indikator penghitung waktu dimana akan menghitung mundur sebelum susunan pola warna utama disembunyikan. Jadi, pengguna mempunyai waktu yang terbatas untuk mengingat susunan warna.



Gambar 4. 2 Tampilan awal Susunan Block Warna Utama

4.1.3.3 Tampilan Masukan Tebakan Pengguna

Tampilan masukan tebakan pengguna muncul setelah indicator penghitung waktu mundur menunjukan angka 0, yang artinya waktu habis dan susunan warna utama disembunyikan. Pada tampilan ini, pengguna dapat menebak urutan warna yang terdapat pada segmen susunan warna utama sebelumnya. Blok warna yang terdapat angka 1 sampai dengan 9 adalah tombol masukan pengguna. Apabila pengguna menekan salah satu tombol, maka blok dengan warna yang sama akan masuk dalam urutan tebakan. Setelah selesai dengan masukan data, pengguna menekan tombol *sumbit* yang berupa tombol abu-abu dengan tanda “>>”.



Gambar 4. 3 Tampilan Masukan Tebakan Pengguna

4.1.4 Implementasi Algoritma

Algoritma pencarian diimplementasikan dalam bahasa *javascript* sesuai dengan *pseudocode* yang telah dijelaskan pada bab sebelumnya.

4.1.4.1 Implementasi Metode Pencarian *Brute Force*

Jenis pencarian dibagi menjadi tiga bagian berdasarkan arah pencariannya, yaitu horizontal, vertikal, diagonal naik dan diagonal turun.

a) Horizontal

```
1. var i = 0;
2. var j = 0;
3. var k = 0;
4. var m = 0;
5. while (m <= P.length-1){
6.    if(P[m][k+i] == p[i].color){
7.          if((i == p.length-1) && (k+i <= P[m].length-1)){
8.                resultArray.horizontal[j] = { row : m, column :
k };
9.                i = 0;
10.               j++;
11.               k++;
12.         } else if (k+i+1 > P[m].length-1){
13.               m++;
14.               k = 0;
15.               i = 0;
16.               if (m > P.length-1){
17.                     break;}
18.         } else {
19.               i++;}
20.   } else {
21.         k++;
22.         i = 0;
23.         if (k > P[m].length-1){
24.               m++;
25.               k = 0;
26.               i = 0;
27.               if (m > P.length-1){
28.                     break;}}}}
```

Gambar 4. 4 Implementasi metode pencarian *brute force* dengan arah pencarian horizontal

b) Vertikal

```
1. var i = 0;
2. var j = 0;
3. var k = 0;
4. var m = 0;
5. while (m <= P[0].length-1){
6.    if(P[k+i][m] == p[i].color){
7.          if((i == p.length-1) && (k+i <= P.length-1)){
8.                resultArray.vertikal[j] = {row:k,column:m};9.
9.                i = 0;
10.               j++;
11.               k++;
12.         } else if (k+i+1 > P.length-1){
13.               m++;
14.               k = 0;
15.               i = 0;
16.               if (m > P[0].length-1){
17.                     break;}
18.         } else {
19.               i++;}
20.   } else {
21.         k++;
22.         i = 0;
23.         if (k > P.length-1){
24.               m++;
25.               k = 0;
26.               i = 0;
27.               if (m > P[0].length-1){
28.                     break;}}}}
```

Gambar 4. 5 Implementasi metode pencarian *brute force* dengan arah pencarian vertikal

4.1.4.2 Implementasi Metode Pencarian *Knuth-Morris-Pratt*

a) Fase Pra-pencarian

```
1. var k = 0;
2. var c = "";
3. var v = 0;
4. var overlap = new Array();
5. overlap[0] = 0;
6. for(k=0; k <= p.length-2; k++){
7.    c = p[k+1].color;
8.    v = overlap[k];
9.    while((p[v].color != c) && (v != 0)){
10.         v = overlap[v-1];
11.   }
12.   if(p[v].color == c){
13.         overlap[k+1] = v+1;
14.   } else {
15.         overlap[k+1] = 0;}
16.}
```

Gambar 4. 6 Implementasi metode pencarian *Knuth-Morris-Pratt* pada fase *precomputation*

```
1. var i = 1;
2. var j = 1;
3. var k = 1;
4. var n = P.length;
5. var m = p.length;
6. var H = new Array();
7. while(n-k >= m){
8.    while((j <= m) && (P[i-1].color == p[j-1].color)){
9.          i++;
10.         j++;
11.   }
12.   if(j>m){
13.         H[H.length] = k-1;}
14.   if(overlap[j-2] > 0){
15.         k = i - overlap[j-2];
16.   } else {
17.         if(i==k){
18.              i++;}
19.         k = i;}
20.   if(j>1){
21.         j = overlap[j-2]+1;}
22.}
```

Gambar 4.7 Implementasi metode pencarian *Knuth-Morris-Pratt* pada fase pencarian

4.2 Pengujian

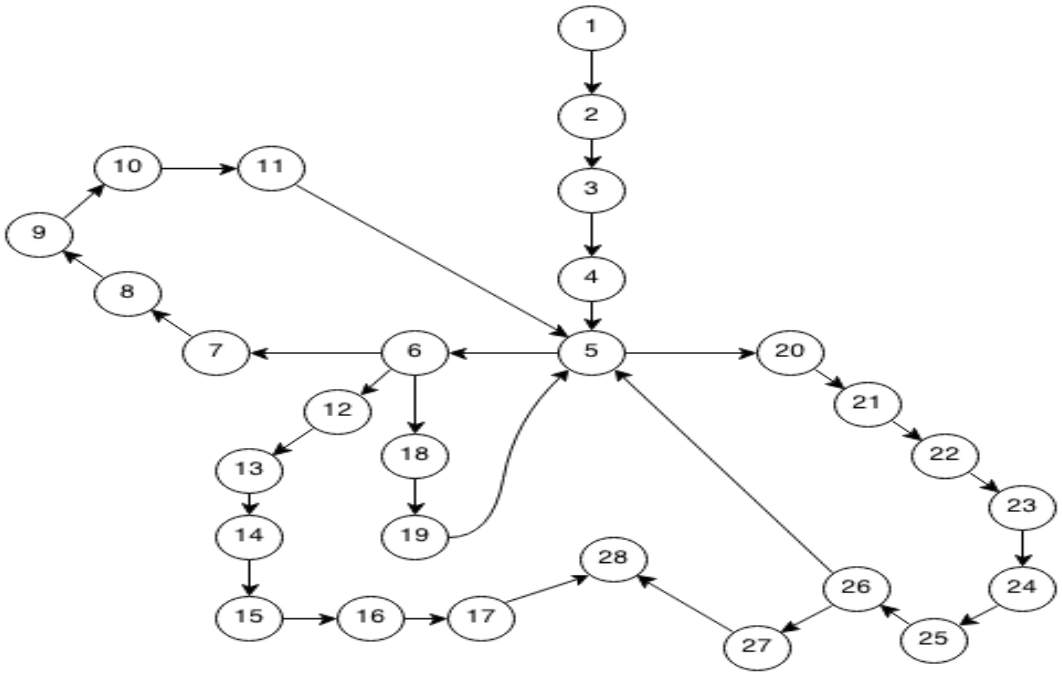
Pengujian pada aplikasi yang dibangun dilakukan melalui tahap pengujian *white box*, pengujian *black box* dan pengujian performansi.

4.2.1 Pengujian *White Box*

Pengujian *white box* atau *white-box testing* merupakan metode perancangan *test case* yang menggunakan struktur kontrol dari perancangan prosedural dalam mendapatkan *test case*. Adapun metode yang digunakan dalam pengujian *white box* ini adalah metode Basis *Path*. Metode Basis *Path* mengijinkan pendesain kasus uji untuk membuat perkiraan lojik yang kompleks dari desain prosedural dan menggunakan perkiraan ini untuk mendefinisikan aliran eksekusi.

4.2.1.1 Metode Pencarian *Brute Force*

Berikut ini adalah diagram alir metode pencarian *brute force* dengan *nodes* berdasarkan implementasi algoritma pada gambar 4.4.



Gambar 4.8 Diagram alir metode pencarian *Brute Force*

Dari diagram alir pada halaman sebelumnya, dapat dihitung *cyclomatic complexity* dengan rumus :

V(G) = *Edge – Node* + (2P)

V(G) = 31 – 28 + 2 = 5

Dari hasil perhitungan *cyclomatic complexity* terdapat 5 *independent path* yaitu:

Path 1 : 1-2-3-4-5-20-21-22-23-24-25-26-27-28

Path 2 : 1-2-3-4-5-20-21-22-23-24-25-26-5-20-21-22-23-24-25-26-27-28

Path 3 : 1-2-3-4-5-6-18-19-5-20-21-22-23-24-25-26-27-28

Path 4 : 1-2-3-4-5-6-12-13-14-15-16-17-28

Path 5 : 1-2-3-4-5-6-7-8-9-10-11-5-20-21-22-23-24-25-26-27-28

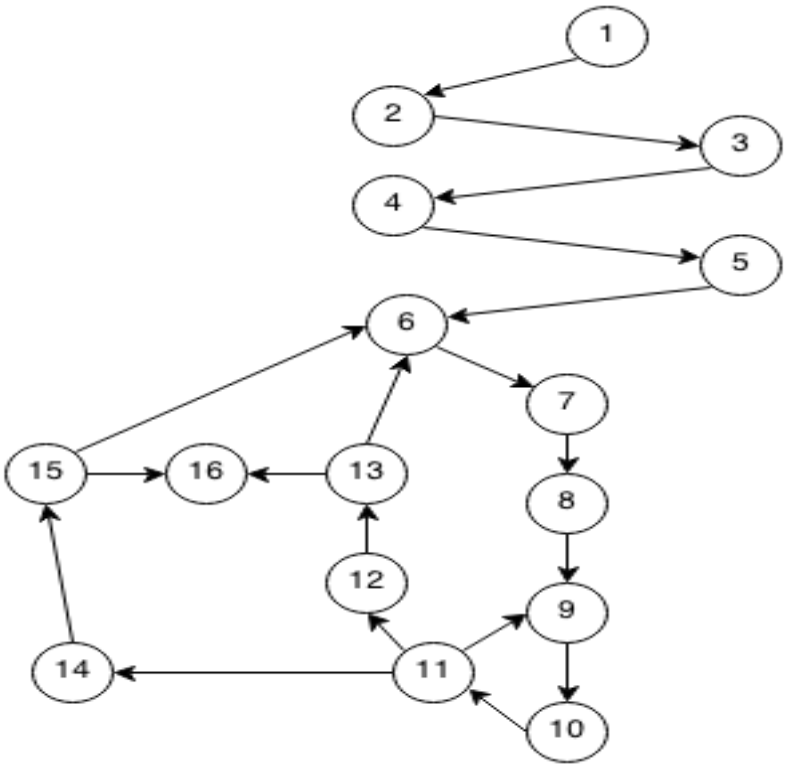
Untuk melakukan uji coba basis *path* di atas digunakan *graph* matrik. *Graph* matrik merupakan matrik empat persegi yang mempunyai ukuran yang sama dengan jumlah *node* pada diagram alir. Adapun *graph* matrik pada diagram alir di atas adalah sebagai berikut:

Tabel 4.2 Tabel *graph* matriks untuk diagram alir metode *Brute Force*

Nodes	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	
1		1																											0
2			1																										0
3				1																									0
4					1																								0
5						1														1									1
6							1					1						1											2
7								1																					0
8									1																				0
9										1																			0
10											1																		0
11					1																								0
12													1																0
13														1															0
14															1														0
15																1													0
16																	1												0
17																												1	0
18																			1										0
19					1																								0
20																					1								0
21																						1							0
22																							1						0
23																								1					0
24																									1				0
25																										1			0
26					1																						1		1
27																												1	0
28																													0
																							SUM(E)+1		4+1 = 5				

4.2.1.2 Metode Pencarian *Knuth-Morris-Pratt*

Berikut ini adalah diagram alir metode pencarian *Knuth-Morris-Pratt* pada tahap *precomputation* dengan *nodes* berdasarkan implementasi algoritma pada gambar 4.6.



Gambar 4.9 Diagram alir metode pencarian *Knuth-Morris-Pratt* pada fase *pre-computation*

Dari diagram alir diatas, dapat dihitung *cyclomatic complexity* dengan rumus :

V(G) = *Edge – Node* + (2P)

V(G) = 19 – 16 + 2 = 5

Dari hasil perhitungan *cyclomatic complexity* terdapat 5 *independent path* yaitu:

Path 1 : 1-2-3-4-5-6-7-8-9-10-11-12-13-16

Path 2 : 1-2-3-4-5-6-7-8-9-10-11-9-10-11-12-13-16

Path 3 : 1-2-3-4-5-6-7-8-9-10-11-12-13-6-7-8-9-10-11-12-13-16

Path 4 : 1-2-3-4-5-6-7-8-9-10-11-14-15-16

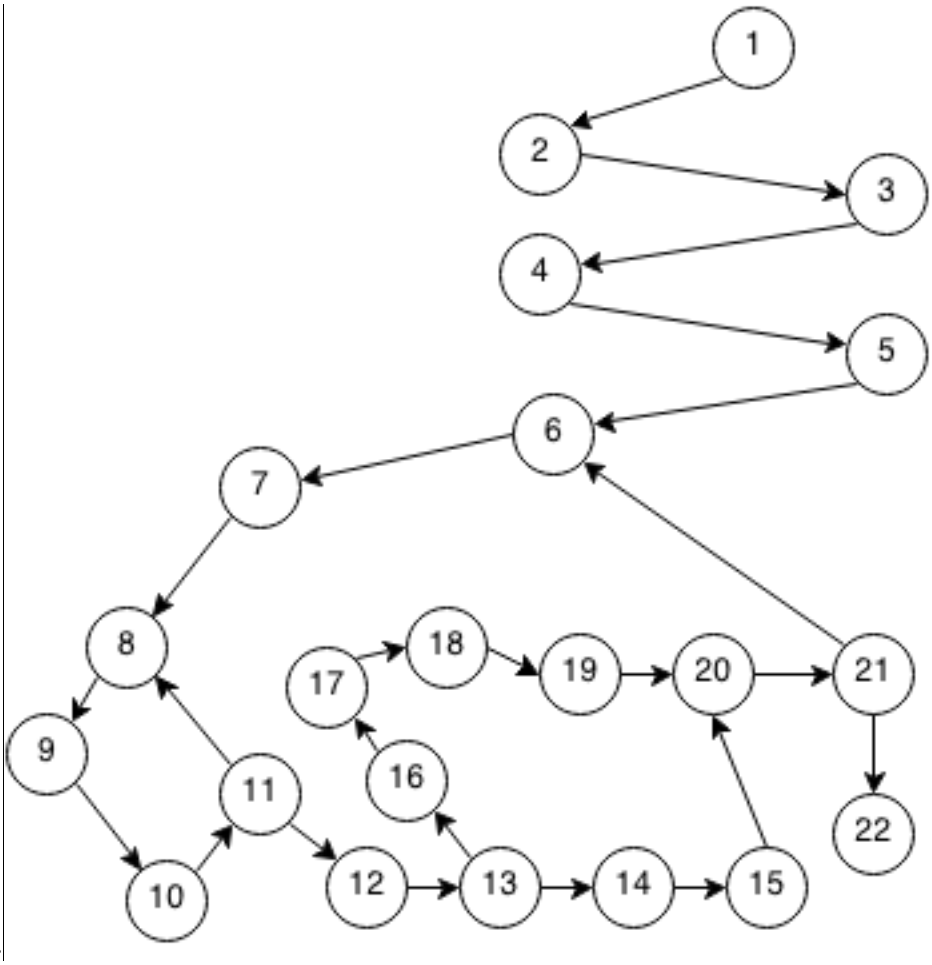
Path 5 : 11-2-3-4-5-6-7-8-9-10-11-14-15-6-7-8-9-10-11-12-13-16

Berikut adalah *graph* matriks pada diagram alir di atas untuk metode pencarian *Knuth-Morris-Pratt* fase pra-pencarian sebagai berikut:

Tabel 4.3 Tabel *graph* matriks diagram alir metode *Knuth-Morris-Pratt* fase pra-pencarian

nodes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	N(E)-1
1		1															0
2			1														0
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			1		1			2
12													1				0
13						1										1	1
14															1		0
15						1										1	1
16																	0
															SUM(E)+1		5

Berikut ini adalah diagram alir metode pencarian *Knuth-Morris-Pratt* pada tahap pencarian dengan *nodes* berdasarkan implementasi algoritma pada gambar 4.7



Gambar 4.10 Diagram alir metode pencarian *Knuth-Morris-Pratt* pada fase pencarian

Dari diagram alir diatas, dapat dihitung *cyclomatic complexity* dengan rumus :

V(G) = *Edge – Node* + (2P)

V(G) = 24 – 22 + 2 = 4

Dari hasil perhitungan *cyclomatic complexity* terdapat 4 *independent path* yaitu:

Path 1 : 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-20-21-22

Path 2 : 1-2-3-4-5-6-7-8-9-10-11-8-9-10-11-12-13-14-15-20-21-22

Path 3 : 1-2-3-4-5-6-7-8-9-10-11-12-13-16-17-18-19-20-21-22

Path 4 : 1-2-3-4-5-6-7-8-9-10-11-12-13-16-17-18-19-20-21-6-7-8-9-10-11-12-13-14-15-20-21-22

Berikut adalah graph matriks pada diagram alir di atas untuk metode pencarian *Knuth-Morris-Pratt* fase pencarian sebagai berikut:

Tabel 4. 4 Tabel *graph* matriks diagram alir metode *Knuth-Morris-Pratt* fase pencarian

node	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
1		1																					0
2			1																				0
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				1											1
12													1										0
13														1		1							1
14															1								0
15																				1			0
16																	1						0
17																		1					0
18																			1				0
19																				1			0
20																					1		0
21						1																1	1
22																							0
																					SUM(E)+1		4

4.2.2 Pengujian *Black Box*

Metode yang digunakan dalam pengujian *alpha* adalah metode *black box* yang fokus pada persyaratan fungsional dari perangkat lunak yang dibangun. Pengujian alpha dengan *black box* dibagi menjadi pengujian fungsionalitas pada *game* dan pengujian algoritma. Pengujian fungsionalitas aplikasi dijelaskan pada tabel 4.5.

4.2.2.1 Pengujian fungsionalitas pada *game*

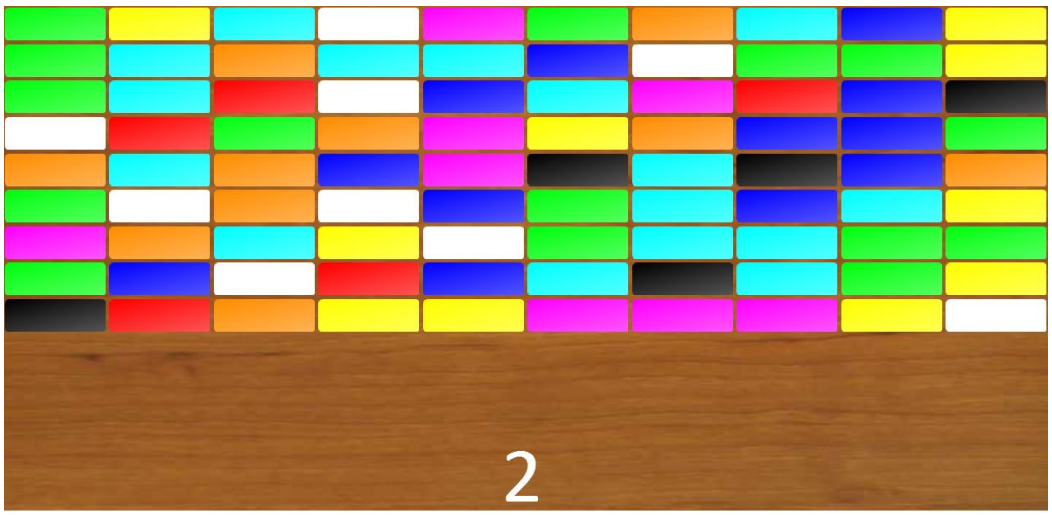
Tabel 4.5 Tabel pengujian aplikasi

Kasus dan Hasil uji Coba (data normal)			
Aktivitas Pengujian	**Hasil yang Diharapkan**	**Hasil Pengujian**	**Kesimpulan**
Klik Tombol START 	Men-*generate* dan menampilkan susunan warna utama	susunan warna utama berhasil di-*generate* dan ditampilkan	[✓] Berhasil [ ] Tidak Berhasil
Klik tombol masukan tebakan 	Menambah urutan tebakan pengguna sesuai dengan warna tombol	Urutan tebakan pengguna bertambah sesuai dengan warna tombol yang di-klik	[✓] Berhasil [ ] Tidak Berhasil
Klik tombol submit 	Melakukan proses pencarian kecocokan pola dan menampilkan hasil kecocokan	Hasil kecocokan tampil dari hasil pencarian	[✓] Berhasil [ ] Tidak Berhasil
Kasus dan Hasil uji Coba (data tidak normal)			
Textbox untuk masukan kolom dan baris tidak diisi atau bukan angka	Proses *generate* susunan warna utama gagal dan tidak dapat muncul	Susunan warna utama tidak muncul	[✓] Berhasil [ ] Tidak Berhasil
Urutan tebakan pengguna kosong	Tombol Submit tidak akan	Tombol Submit tidak muncul	[✓] Berhasil [ ] Tidak

	dimunculkan		Berhasil

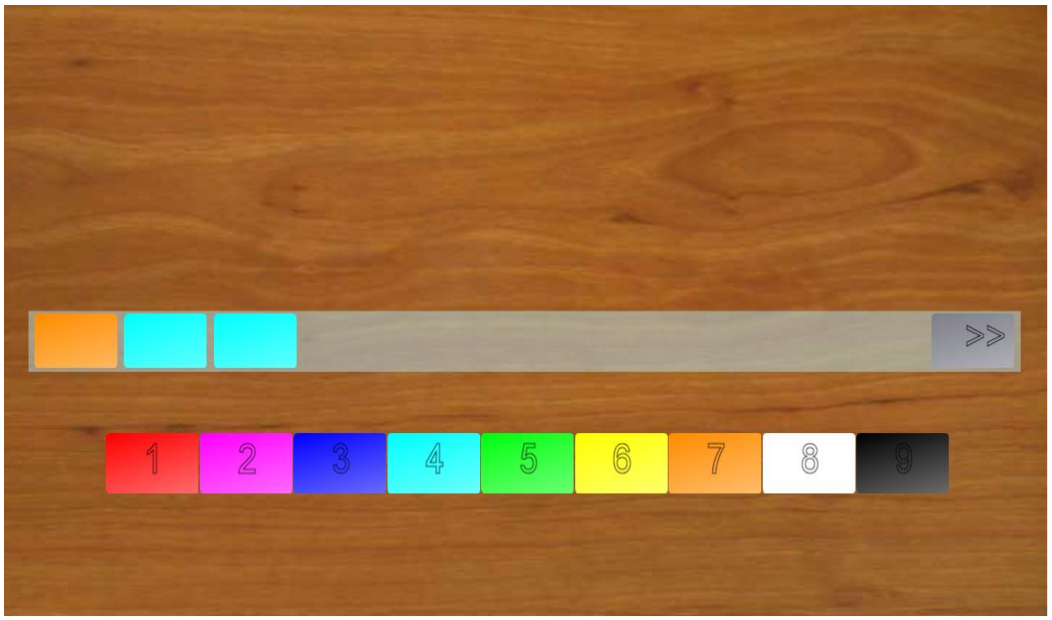
4.2.2.2 Pengujian algoritma pencarian pada *game*

Disini akan diuji apakah implementasi dari kedua algoritma pencarian (*Brute Force* dan KMP) dapat menunjukkan hasil pencariannya dengan benar. Sebagai contoh, berikut ini adalah susunan blok utama berdimensi 10x9.



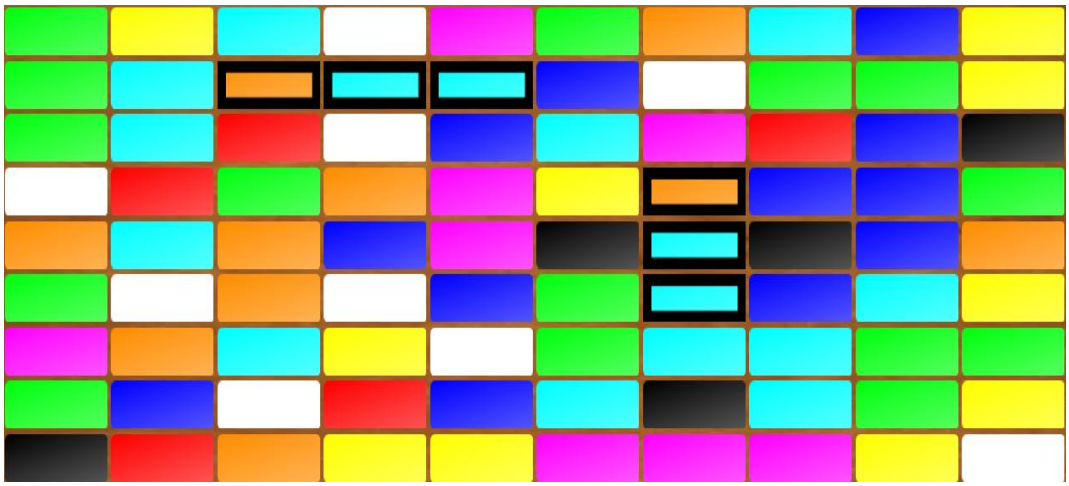
Gambar 4.11 Susunan blok utama 10x9

Selanjutnya pengguna memasukkan data tebakan sebagai berikut:



Gambar 4.12 Susunan blok masukan pengguna

Setelah pengguna selesai memasukkan data dan menekan tombol *submit*, maka tampilan susunan blok utama akan berubah menjadi seperti berikut ini:



Gambar 4.13 Susunan blok utama setelah pengguna memasukkan data tebakan

Dari gambar 4.10 terlihat bahwa *game* menandai tiga buah blok di dua segmen yang berbeda. Dua segmen blok tersebut adalah susunan yang cocok dengan susunan blok masukan pengguna, dimana pengguna memasukkan tiga buah susunan warna. Kesimpulannya, algoritma pencarian berhasil diimplementasikan di dalam *game* ini.

4.2.3 Pengujian Performansi

Pada pengujian performansi, beberapa percobaan akan dilakukan pada aplikasi dengan membagi beberapa jenis pengujian dengan parameter yang berbeda agar masing-masing algoritma dapat diketahui karakteristiknya.

Berdasarkan data pencariannya, percobaan dibagi menjadi tiga tipe, yaitu:

1) Pengujian Tipe 1

   Pada pengujian ini, data pola masukan pengguna diatur sedemikian rupa sehingga cocok dengan pola utama dari awal pencarian dilakukan.

2) Pengujian Tipe 2

Pada pengujian ini, data masukan pengguna diatur sedemikian rupa supaya pola yang cocok dengan pola utama berada di tengah-tengah pencarian.

3) Pengujian Tipe 3

Pada pengujian ini, data masukan pengguna diatur sedemikian rupa supaya tidak ada satupun kecocokan dengan pola utama.

4.2.3.1 Parameter Pengujian

Beberapa parameter yang dijadikan acuan pada penelitian ini diantaranya adalah:

1. Dimensi Blok Utama

   Parameter ini menunjukkan dimensi blok utama dalam format baris x kolom.

2. Panjang Masukan

   Panjang masukan adalah banyaknya data tebakan yang dimasukkan pengguna.

3. Pola Masukan

   Parameter ini menunjukkan pola warna yang dimasukkan oleh pengguna. Ini ditampilkan untuk kepentingan penggunaan memori pada tahap *precomputation*. Pola masukan direpresentasikan dengan inisial warna. Berikut daftar inisial warna dan artinya:

   1) mr = merah
   2) pt = putih
   3) kn = kuning
   4) br = biru
   5) ht = hitam
   6) hj = hijau
   7) bm = biru muda
   8) or = orange
   9) pn = pink

4. Jenis Percobaan

Menentukan percobaan seperti apa yang dilakukan sesuai dengan yang telah dijelaskan sebelumnya di bagian 4.2.3.

5. Banyak Perulangan (*precomputation*)

   Parameter ini menunjukkan banyaknya perulangan yang dilakukan sebuah algoritma pada tahap awal atau *precomputation* sebelum dilakukannya pencarian. Apabila sebuah algoritma tidak memiliki tahap awal sebelum pencarian, maka parameter ini akan diisi 0.
6. Banyak Perulangan

   Banyak perulangan yang ditampilkan disini adalah banyaknya perulangan yang dilakukan pada tahap awal + perulangan pada tahap pencarian.
7. Kompleksitas Waktu *precomputation* (Big-O)

   Parameter ini menunjukkan skalabilitas atau batas maksimal sebuah perulangan akan dilakukan oleh sebuah algoritma pada tahap awal atau *precomputation*. Apabila sebuah algoritma tidak memiliki tahap awal sebelum pencarian, maka parameter ini akan diisi 0.
8. Kompleksitas Waktu (Big-O)

   Parameter ini menunjukkan skalabilitas atau batas maksimal sebuah perulangan akan dilakukan oleh sebuah algoritma pada tahap pencarian.
9. Memori *precomputation* (Byte)

   Parameter ini menunjukkan berapa banyak memori yang digunakan oleh sebuah algoritma pada tahap awal atau *precomputation*. Apabila sebuah algoritma tidak memiliki tahap awal sebelum pencarian, maka parameter ini akan diisi 0.
10. Memori Pencarian (Byte)

    Parameter ini menunjukkan berapa banyak memori yang digunakan oleh sebuah algoritma pada tahap pencarian.
11. Total Memori Digunakan (Byte)

    Parameter ini menunjukkan berapa banyak memori yang digunakan oleh sebuah algoritma pada tahap *precomputation* + pada tahap pencarian.

Pada halaman berikut adalah hasil percobaan yang dilakukan dengan parameter masukan yang berbeda-beda.

Tabel 4.6 Tabel hasil percobaan

no	dimensi blok utama	panjang masukan	Pola Masukan	jenis percobaan	Banyak Perulangan (*precomputation*)		banyak perulangan		kompleksitas waktu *precomputation* (big-O)		kompleksitas waktu (big-O)		memori *precomputation* (byte)		memori pencarian (byte)		total memori digunakan (byte)	
					Brute Force	Knuth-Morris-Pratt	Brute Force	Knuth-Morris-Pratt	Brute Force	Knuth-Morris-Pratt	Brute Force	Knuth-Morris-Pratt	Brute Force	Knuth-Morris-Pratt	Brute Force	Knuth-Morris-Pratt	Brute Force	Knuth-Morris-Pratt
1	3x3	3	or, pn, pn	Tipe 1	0	2	12	6	0	0	18	18	0	0	32	40	32	40
2	3x3	3	br, or, pt	Tipe 2	0	2	12	6	0	0	18	18	0	0	32	40	32	40
3	3x3	3	mr, mr, mr	Tipe 3	0	2	9	8	0	0	18	18	0	0	32	40	32	56
4	4x4	3	hj, or, ht	Tipe 1	0	2	18	13	0	0	36	28	0	0	32	40	32	40
5	4x4	4	pt, ht, ht, hj	Tipe 2	0	3	19	12	0	0	40	32	0	0	32	40	32	40
6	4x4	4	pt, ht, pt, ht	Tipe 3	0	3	18	14	0	0	40	32	0	16	32	40	32	56
7	5x5	3	pn, kn, hj	Tipe 1	0	2	27	22	0	0	60	40	0	0	32	40	32	40
8	5x5	4	hj, hj, or, br	Tipe 2	0	4	31	22	0	0	70	45	0	8	32	40	32	48
9	5x5	5	mr, bm, mr, bm, kn	Tipe 3	0	5	27	25	0	0	75	50	0	16	32	40	32	56
10	6x6	4	pn, bm, hj, mr	Tipe 1	0	3	44	31	0	0	108	60	0	0	32	40	32	40
11	6x6	5	pn, pn, pt, ht, pt	Tipe 2	0	5	41	36	0	0	120	66	0	8	32	40	32	48
12	6x6	6	hj, hj, hj, hj, hj, hj	Tipe 3	0	5	40	35	0	0	126	72	0	40	32	40	32	80
13	7x7	5	or, hj, or, ht, or	Tipe 1	0	5	62	46	0	0	175	84	0	8	32	40	32	48
14	7x7	6	or, or, kn, bm, pn, pn	Tipe 2	0	6	63	44	0	0	189	91	0	8	32	40	32	48
15	7x7	7	or, or, or, or, hj, bm, br	Tipe 3	0	9	60	51	0	0	196	98	0	24	32	40	32	64
16	8x8	6	kn, kn, kn, hj, pt, pt	Tipe 1	0	7	80	60	0	0	264	112	0	16	32	40	32	56
17	8x8	7	kn, hj, hj, bm, kn, ht, or	Tipe 2	0	7	82	56	0	0	280	120	0	8	32	40	32	48
18	8x8	8	kn, hj, kn, hj, kn, hj, kn, hj	Tipe 3	0	7	78	60	0	0	288	128	0	48	32	40	32	88
19	9x9	7	bm, mr, hj, br, bm, br, kn	Tipe 1	0	7	96	75	0	0	378	144	0	8	32	40	32	48
20	9x9	8	kn, pn, pt, or, kn, pn, hj, ht	Tipe 2	0	8	99	72	0	0	396	153	0	16	32	40	32	56
21	9x9	9	pt, ht, pt, ht, pt, ht, or, kn, hj	Tipe 3	0	10	87	82	0	0	405	162	0	32	32	40	32	72
22	10x10	8	kn, ht, kn, kn, kn, hj, hj, br	Tipe 1	0	10	118	93	0	0	520	180	0	8	32	40	32	48
23	10x10	9	bm, pt, pt, pt, or, pn, pt, br, bm	Tipe 2	0	8	113	91	0	0	540	190	0	8	32	40	32	48
24	10x10	10	hj, br, mr, mr, mr, hj, br, or, hj, br	Tipe 3	0	10	117	99	0	0	550	200	0	16	32	40	32	56
25	20x20	8	pn, kn, pt, pn, bm, hj, hj, pn	Tipe 1	0	8	465	386	0	0	2640	560	0	8	32	40	32	48
26	20x20	9	pn, pn, pn, pt, or, or, bm, pt, pn	Tipe 2	0	10	467	394	0	0	2880	580	0	16	32	40	32	56
27	20x20	10	or, or, hj, hj, br, mr, or, or, hj, hj	Tipe 3	0	10	442	390	0	0	3100	600	0	32	32	40	32	72
28	30x30	8	pt, mr, or, bm, bm, hj, br, or	Tipe 1	0	7	1034	884	0	0	6360	1140	0	0	32	40	32	40
29	30x30	9	br, bm, br, bm, br, hj, pn, pn, or	Tipe 2	0	10	1040	857	0	0	7020	1170	0	24	32	40	32	64
30	30x30	10	mr, kn, hj, mr, kn, kn, mr, br, pn, mr	Tipe 3	0	11	954	889	0	0	7650	1200	0	16	32	40	32	56

Berdasarkan hasil pengujian, terlihat bahwa kedua algoritma memiliki keunggulannya tersendiri. *Brute Force* dapat melakukan pencarian dengan beban memori yang relatif lebih ringan dibandingkan dengan *Knuth-Morris-Pratt*, namun banyakya perulangan dan kompleksitas waktu (big-O) - yang menunjukkan batas luar algoritma - *Knuth-Morris-Pratt* lebih unggul disbanding *Brute Force* dikarenakan nilai yang lebih kecil. Artinya, metode pencarian *Knuth-Morris-Pratt* relatif lebih terskala dalam hal lamanya waktu pencarian dibandingkan dengan metode pencarian *Brute Force*.

Ada beberapa karakteristik khusus ketika percobaan dilakukan dengan tipe percobaan yang berbeda. Misalnya, algoritma *Brute Force* menghasilkan jumlah perulangan paling kecil ketika melakukan jenis percobaan 3 dibandingkan dengan tipe 1 dan 2. Sedangkan metode KMP justru sebaliknya, dimana pengujian dengan tipe 3 justru menghasilkan jumlah perulangan yang lebih banyak dibandingkan dengan tipe percobaan 1 dan 2. Apabila pencarian merupakan *one stop search* (pencarian berhenti apabila terjadi kecocokan) dan kecocokan pola terjadi di awal pencarian, maka metode *Brute Force* akan lebih dulu menyelesaikan pencarian dibandingkan metode *Knuth-Morris-Pratt*. Metode *Brute Force* paling cepat berjalan apabila tidak ada satupun kecocokan yang terjadi. Sebaliknya metode KMP paling lambat bekerja apabila tidak ada kecocokan yang terjadi.

Banyaknya memori yang digunakan didapat dari perhitungan manual setiap peubah pembantu yang digunakan pada saat pencarian dengan mereferensikan standar penggunaan memori di *Javascript*. Nilai ini tidak berubah di setiap pengujian dikarenakan memori yang dihitung tidak termasuk data yang dicari ataupun data masukan pengguna. Di tabel 4.6 terlihat bahwa konsumsi memori metode KMP lebih besar disbanding konsumsi memori *Brute Force.* Ini dikarenakan metode *Knuth-Morris-Pratt* membutuhkan ruang memori tambahan untuk menyimpan data berupa *array* yang berisi informasi *overlap*.