BAB III
ANALISIS DAN PERANCANGAN SISTEM

III.1 Analisis Sistem

Analisis sistem dapat didefinisikan sebagai penguraian dari suatu sistem informasi yang utuh ke dalam bagian-bagian komponennya dengan maksud untuk mengidentifikasikan dan mengevaluasi permasalahan dan kebutuhan-kebutuhan yang diharapkan sehingga dapat diusulkan perbaikan-perbaikannya. Dalam membangun perangkat lunak ini dilakukan beberapa tahap analisis yaitu :

1. Analisis Masalah
2. Analisis *Game*
3. Analisis Masukan dan Keluaran
4. Analisis Algoritma
5. Analisis Kompleksitas
6. Analisis Kebutuhan Perangkat Lunak

III.1.1 Analisis Masalah

Berdasarkan latar belakang yang sudah diuraikan maka masalah yang didapat adalah sebagai berikut :

1. Belum diketahui perbandingan performansi antara Algoritma Reverse Colussi dengan Algoritma Karp-Rabin dalam mencari dan mencocokkan kata.
2. Belum diketahui mana algoritma yang lebih optimal antara Algoritma Reverse Colussi dengan Algoritma Karp-Rabin dalam mencari dan mencocokkan kata.

III.1.2 Analisis Game

Game ini merupakan sebuah permainan berjenis *Word Game* yang merupakan suatu permainan pencarian kata yang tersembunyi berdasarkan kata yang telah ditentukan didalam *wordlist.* Tujuan membangun *Game* ini yaitu untuk mengimplementasikan perbandingan antara kedua algoritma yaitu algoritma Karp-Rabin dan algoritma Reverse Colussi dalam mencari kata berdasarkan

performansi kecepatan waktu proses, dan pemakaian memori dalam 1 kali proses pencarian kata.

III.1.2.1. Gameplay

Secara umum terdapat 5 tahap dalam memainkan *game* ini, yaitu :

1. Memilih Metode algoritma

   Sebelum memulai permainan pemain diharuskan dulu memilih metode algoritma yang ingin di uji. Setelah memilih metode, maka sistem akan menampilkan layar permainan berdasarkan metode yang sudah dipilih sebelumnya. Kemudian pemain hanya harus menekan tombol *newgame* untuk memulai permainan.

2. Memulai permainan

   Setelah menekan tombol *newgame*, maka permainan dimulai dan sistem akan menampilkan papan *input*an kata (*virtual keyboard*) dan lahan hukuman pada layar.

3. Memasukkan kata (*input string*)

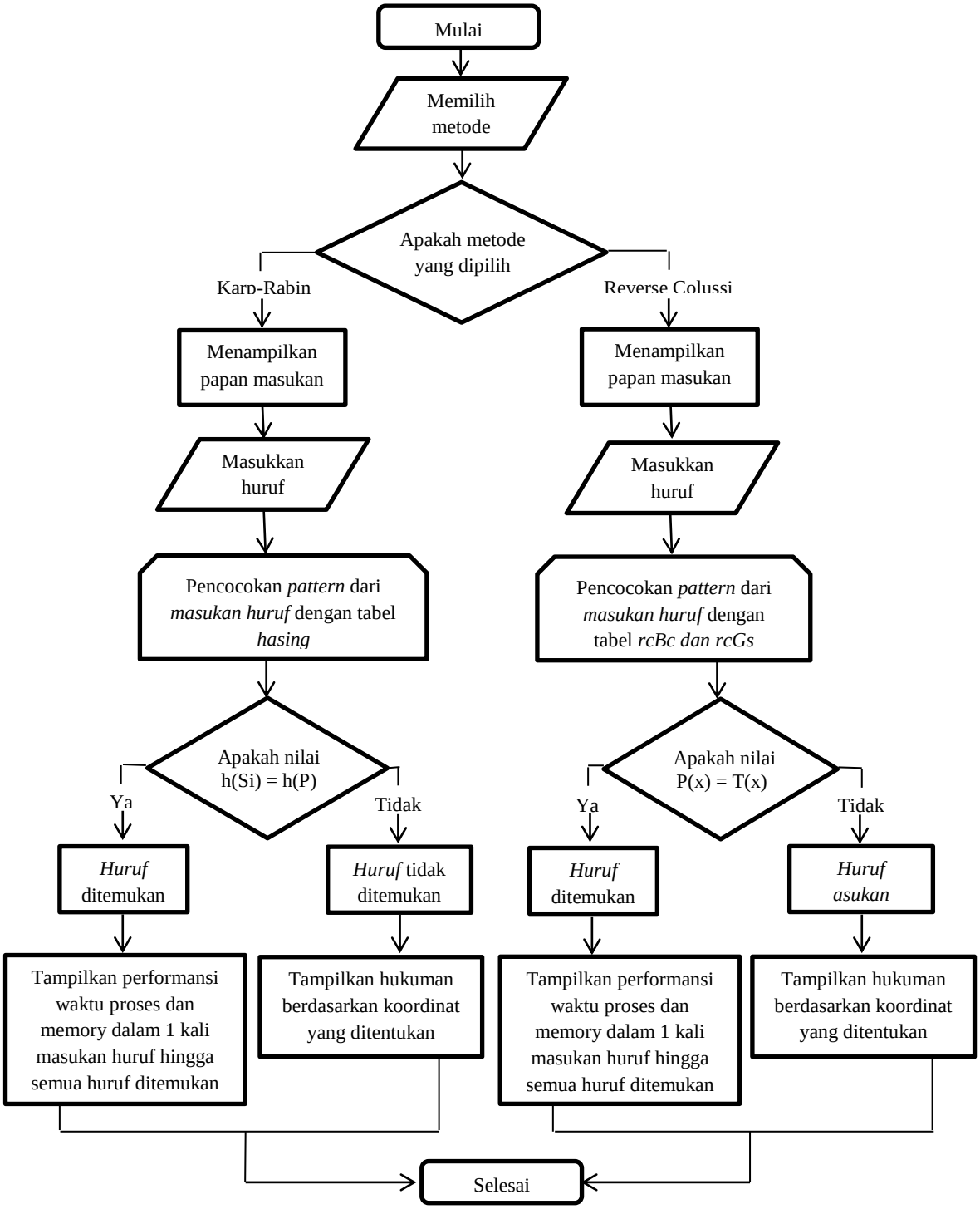
   Pada tahap ini pemain diharuskan memasukkan huruf pada papan *input*an (*virtual keyboard*) untuk menebak kata yang tersembunyi berdasarkan yang ditentukan pada *wordlist.* Apabila huruf yang dimasukkan cocok dengan kata yang tersembunyi, maka sistem akan menampilkan huruf tersebut pada layar dan pemain harus memilih huruf yang lain hingga semua kata yang tersembunyi ditemukan, maka pemain akan dinyatakan berhasil atau menang. Tetapi, jika huruf yang dimasukkan tidak cocok dengan kata yang tersembunyi, maka sistem akan mengaktifkan hukuman dengan menampilkan tiang gantung terlebih dahulu, tali gantungannya, munculkan kepala, badan, tangan kanan, tangan kiri, kaki kanan, dan terakhir kaki kiri. Jika ternyata semua sudah dimunculkan, maka pemain dinyatakan gagal atau kalah.

4. Keluar ke menu utama

   Pada tahap ini pemain diperbolehkan untuk kembali lagi ke menu pemilihan metode untuk memilih metode yang lain dengan cara menekan tombol keluar, maka sistem akan keluar dari permainan dan kembali ke halaman utama.

5. Keluar dari permainan

Pada tahap ini adalah sebuah tombol untuk keluar dari permainan dimana pemain hanya menekan tombol keluar maka sistem akan menutup permainan.



Gambar III.1. Diagram Blok Permainan

III.1.3 Analisis Masukan dan Keluaran

Analisis masukan yang dilakukan dalam penelitian ini adalah menganalisis parameter pada game Algojo menggunakan kedua algoritma yang dibandingkan, parameter tersebut adalah meng*input* huruf untuk menjawab kata yang tersembunyi dan menampilkan hukuman jika inputan tidak sesuai dengan kata yang tersembunyi serta menampilkan performansi waktu proses dan pemakaian *memory* disetiap 1 kali proses pencariannya. Berikut analisis masukan dan keluaran kedua algoritma dari *word game* Algojo :

1. Meng*input* 1 huruf yang dipilih oleh pemain

Pemain harus memilih huruf yang telah disediakan di papan huruf untuk menebak kata yang tersembunyi.

Data Masukan

Pemain memilih huruf yang telah disediakan di *array* papan huruf

Array :

abc = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W', 'X', 'Y','Z','_',];

Berikut adalah gambar papan *input*an huruf yang dapat dilihat pada gambar III.2.



Gambar III.2. Array Papan huruf

Data Keluaran

Setelah pemain memilih huruf di papan huruf, jika cocok dengan kata yang tersembunyi maka, sistem akan menampilkan huruf tersebut di layar jawaban dan tampilkan performansinya juga. Tapi, jika huruf yang dipilih tidak cocok maka, sistem akan menampilkan hukuman yang sudah dijelaskan di analisis *game* dengan koordinat-koordinat yang telah ditentukan.

1. Tampilkan jika huruf yang dipilih cocok

```
for (i = 0; i < 27; i++)
{
  var div = document.createElement('div');
  div.style.cursor = 'pointer';
  div.innerHTML = abc[i];
  div.onclick = getLetter;
  frag.appendChild(div);
}
letters.appendChild(frag);
drawCanvas();
window.scrollBy (0, 200);
```

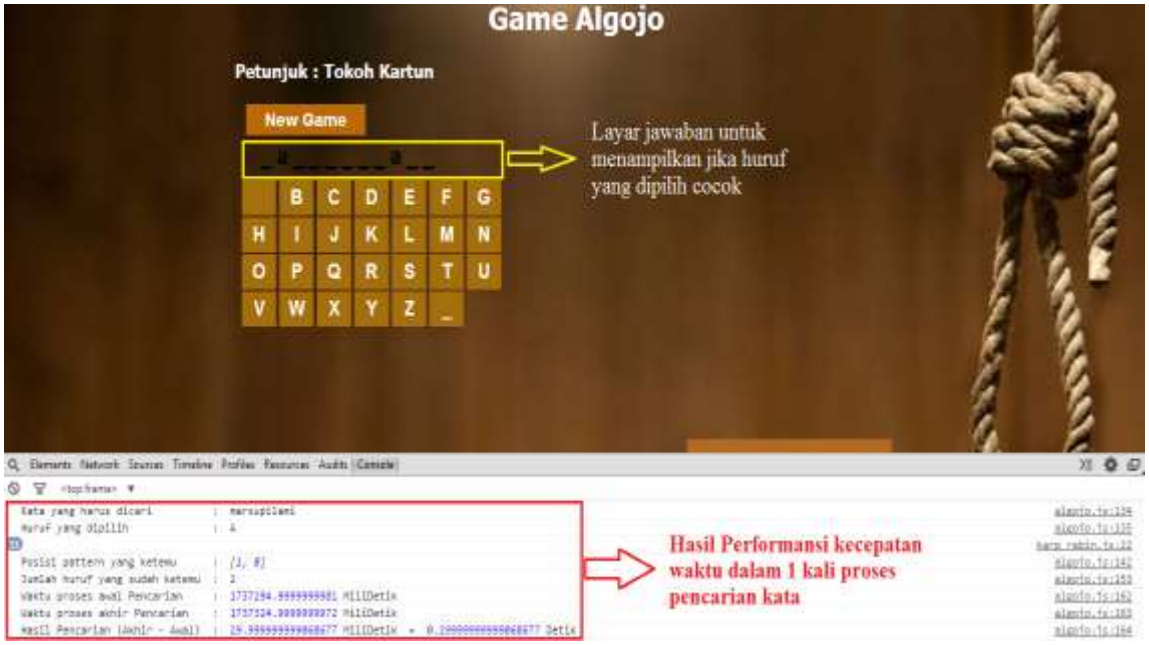
Huruf masukan yang dipilih

Jika huruf yang dipilih cocok dengan kata yang tersembunyi

Menampilkannya ke layar jawaban

Gambar III.3. *Source Code* huruf cocok

Berikut adalah gambar Tampilkan huruf jika cocok yang dapat dilihat pada gambar III.4.

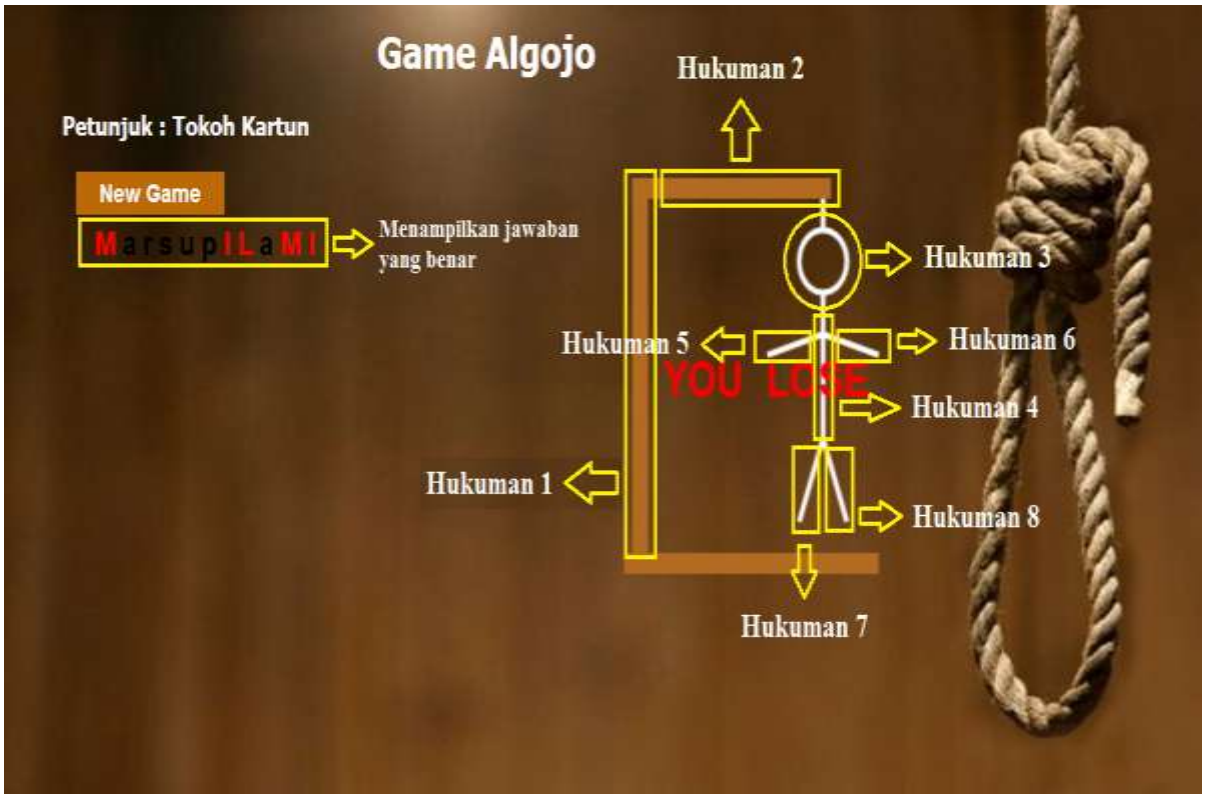


Gambar III.4. Tampilkan huruf jika cocok

2. Tampilkan hukuman jika salah input huruf

Hukuman jika salah masukkan huruf ada 8 tahap yaitu memunculkan tiang gantung, gantungannya, kepala, badan, tangan kiri, tangan kanan, kaki kiri, dan kaki kanan.

Berikut adalah gambar tampilkan hukuman jika huruf yang dipilih tidak cocok yang dapat dilihat pada gambar III.5



Gambar III.5. Tampilkan hukuman jika huruf tidak cocok

Keterangan :

a. Hukuman 1 : Tiang Gantungan
b. Hukuman 2 : Gantungannya
c. Hukuman 3 : Kepala
d. Hukuman 4 : Badan
e. Hukuman 5 : Tangan Kiri
f. Hukuman 6 : Tangan Kanan
g. Hukuman 7 : Kaki Kiri
h. Hukuman 8 : Kaki Kanan

Berikut adalah rincian kemunculan hukuman berdasarkan koordinat yang ditentukan :

a. Tiang Gantungan

   Tiang gantungan memiliki titik kordinat x = [30 , 185] dan y = [30 , 10].

b. Gantungannya

   Gantungannya memiliki titik kordinat x = [150 ,10].

c. Kepala

   Kepala memiliki titik kordinat x = [160 , 45] dan Lingkaran [145, 45, 15, 0, (math.PI/180)*360].

d. Badan

   Badan memiliki titik kordinat x = [145 , 60] dan y = [145 , 130].

e. Tangan Kiri

   Tangan kiri memiliki titik kordinat x = [145 , 80] dan y = [110 , 90].

f. Tangan Kanan

   Tangan kanan memiliki titik kordinat x = [145 , 80] dan y = [180 , 90].

g. Kaki Kiri

   Kaki kiri memiliki titik kordinat x = [145 , 130] dan y = [130 , 170].

h. Kaki Kanan

   Kaki kanan memiliki titik kordinat x = [145 , 130] dan y = [160 , 170].

Memunculkan juga pernyataan "*YOU LOSE*" pada titik koordinat x = [45,110] dan menampilkan juga jawaban dari kata yang tersembunyi pada layar jawaban.

III.1.4 Analisis Algoritma

Analisis algoritma adalah bahasan utama dalam ilmu komputer. Dalam menguji suatu algoritma, dibutuhkan beberapa kriteria untuk mengukur efisiensi algoritma. Terdapat dua tipe analisis algoritma yaitu :

1. Memeriksa kebenaran algoritma dapat dilakukan dengan cara perurutan, memeriksa bentuk logika, implementasi algoritma, pengujian dengan data dan menggunakan cara matematika untuk membuktikan kebenaran.
2. Penyederhanaan algoritma membagi algoritma menjadi bentuk yang sederhana

III.1.4.1. Analisis Kasus Algoritma

Analisis kasus algoritma adalah konstruktor terpenting dalam algoritma, yaitu memungkinkan kita untuk membuat teks yang sama namun menghasilkan eksekusi yang berbeda-beda. Konstruksi dari suatu analisis kasus dapat dimulai dari menentukan semua kondisi yang mungkin atau dimulai dari menentukan variasi aksi. Berikut analisis kasus algoritma *Reverse Colussi* dan algoritma *Karp-Rabin*.

III.1.4.1.1. Algoritma Karp-Rarbin

Algoritma *Karp-Rabin* ini tidak melakukan pergeseran yang rumit untuk menyelesaikan masalah, algoritma ini mempercepat pengecekan kata pada suatu teks dengan menggunakan fungsi *hash* dalam mencari suatu *string*. Fungsi *hash* adalah fungsi yang menerima masukan *string* yang panjangnya sembarang dan mengkonversinya menjadi *string* keluaran yang panjangnya tetap (*fixed*) umumnya berukuran jauh lebih kecil daripada ukuran *string* semula. Pada algoritma ini untaian *string* akan diubah menjadi *integer* berdasarkan bilangan ASCII-nya. Pendekatan utamanya adalah *string* yang sama akan memiliki nilai *hash* yang sama. Hal yang penting yang harus dilakukan sebelum melakukan pencocokan dengan algoritma *Karp-Rabin* adalah mengubah sumber *string* dan *pattern* yang dicari menjadi untaian *integer*. Karakteristik algoritma *Karp-Rabin* :

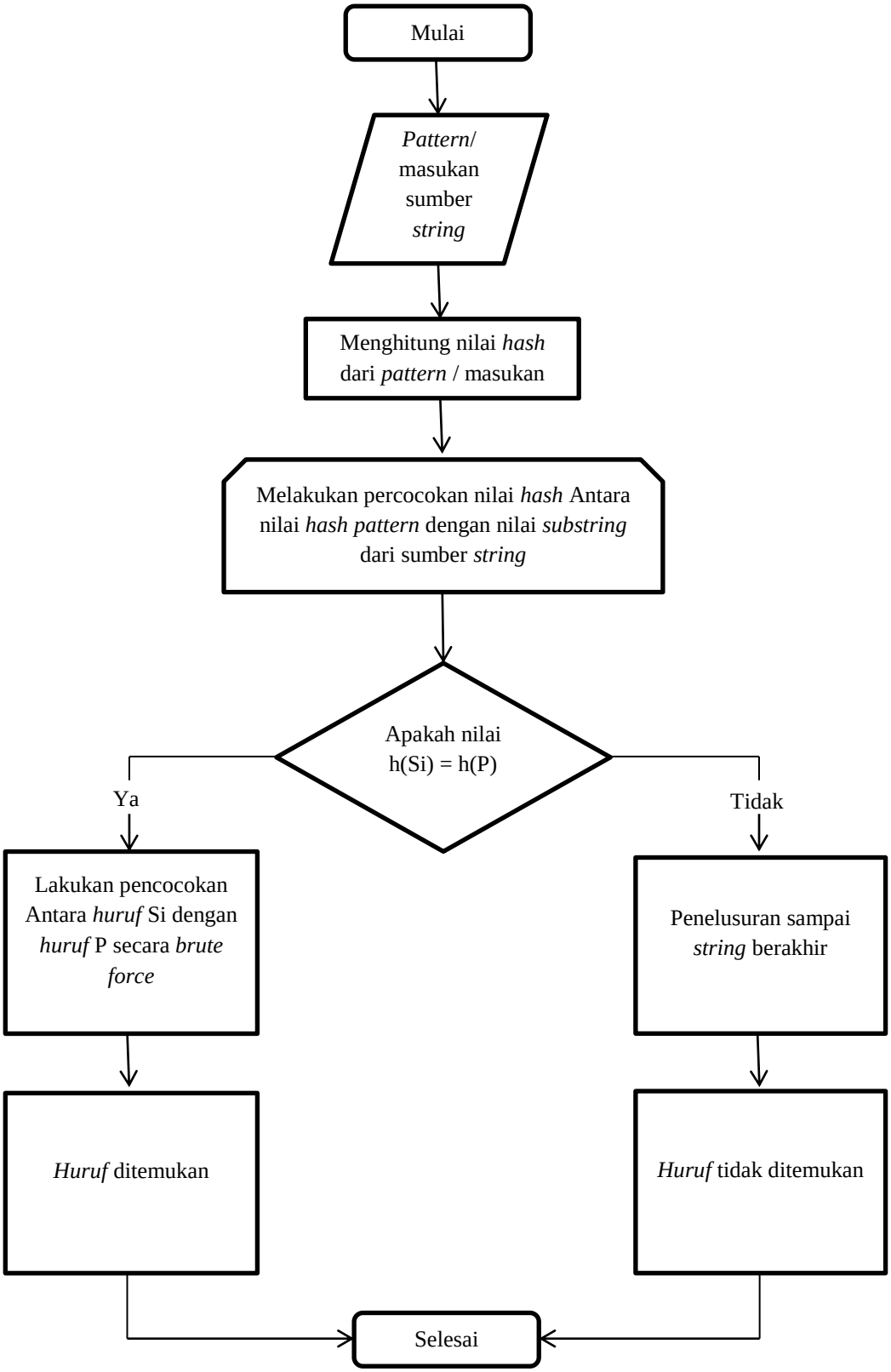
1. Menggunakan Fungsi Rolling Hashing.
2. Melakukan pencocokan dari kiri ke kanan
3. Pergeseran dilakukan secara brute-force

Berikut adalah *pseudocode* algoritma *Karp-Rabin* dengan menggunakan notasi *Big-O* :

```
function RabinKarp (input A: string[1..m], teks: string[1..n])
{IS : Melakukan pencarian string A pada string teks dengan
algoritma Rabin Karp
FS : keluaran yang diharapkan pencarian string A berhasil
dilakukan ada atau tidak pada string teks}
Deklarasi
      i : integer
      ketemu = Boolean
Algoritma:
   ketemu <- false                                  O(1)
      hs <- hash(A[1..m])                           O(1)
         for i <- 1 to n-m+1 do                     O(n)
            hsub <- hash(teks[1..i+m-1])            O(1)
               if hsub = hs then                    O(1)
                  if teks[i..i+m-1] = A then        O(1)
                     ketemu <- true                 O(1)
                  else
                     hsub <- hash(teks[i+1..i+m])   O(1)
                  endif
               endif
         endfor
   return ketemu                                    O(1)
endfunction
```

Gambar III.6. *Pseudocode* Algoritma *Karp-Rabin* dengan notasi *Big-O*

Berikut adalah Flowchart Algortima Karp-Rabin dalam pencarian *string* menggunakan fungsi *hasing* :



Gambar III.7 *Flowchart* Alur Kerja Algoritma *Karp-Rabin*

Berikut ini adalah penggunaan fungsi *hash* dengan basis. Misalnya menggunakan contoh *string* ”DORAEMON” sebagai sumber *string* dan ”O” sebagai *string* yang dicari

1. Dengan menggunakan a=2 sebagai basis lalu tentukan panjang sumber *string* disini ”DORAEMON” n=8. Sedangkan untuk *pattern* yang dicari ”O” panjangnya m=1.
2. Selanjutnya ubah *pattern* yang dicari dengan menggunakan fungsi *Rolling hash.* Dengan persamaan :

$$\mathbf{H = C0*am\text{-}1 + C1*am\text{-}2 + ... + C[m\text{-}1]*a0}$$

Persamaan III.1. ubah *pattern* menggunakan fungsi *Rolling Hash* [5].

H = nilai *hash*

C = nilai ASCII karakter

a = nilai basis

m = banyaknya karakter

Jumlah ASCII dari kata DORAEMON :

D	O	R	A	E	M	O	N
68	79	82	65	69	77	79	78

Jadi nilai *hash* dari ”O” adalah 79 didapat dari :

H = 79*20

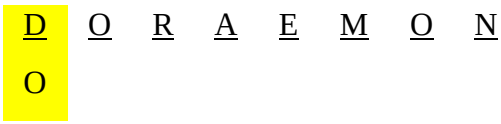
= 158/2

= 79

Percobaan ke 1 :

Hash(O) = 79

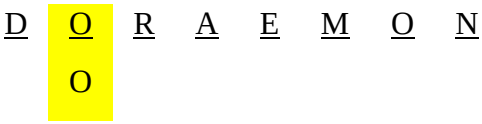
Hash (y[0]) = 68



Gambar III.8 Percobaan ke 1 fungsi *Rolling hash*

Nilai *hash* dari indeks ke 0 tidak cocok dengan nilai *hash* target, maka dilakukan pergeseran.

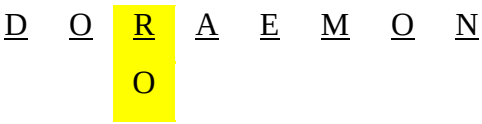
Percobaan ke 2 :
Hash(O) = 79
Hash (y[1]) = 79



Gambar III.9 Percobaan ke 2 fungsi *Rolling hash*

Terjadi kecocokan nilai *hash* dari indeks ke 1, maka algoritma menandai lokasi penemuan dan melanjutkan pencarian sampai karakter pada sumber *string* habis.

Percobaan ke 3 :
Hash(O) = 79
Hash (y[2]) = 82



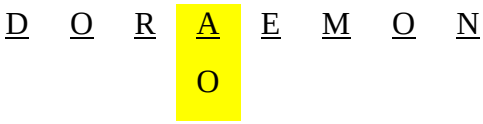
Gambar III.10 Percobaan ke 3 fungsi *Rolling hash*

Nilai *hash* dari indeks ke 2 tidak cocok dengan nilai *hash* target, maka dilakukan pergeseran.

Percobaan ke 4 :

Hash(O) = 79

Hash (y[3]) = 65



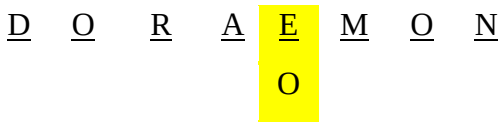
Gambar III.11 Percobaan ke 4 fungsi *Rolling hash*

Nilai *hash* dari indeks ke 3 tidak cocok dengan nilai *hash* target, maka dilakukan pergeseran.

Percobaan ke 5 :

Hash(O) = 79

Hash (y[4]) = 69



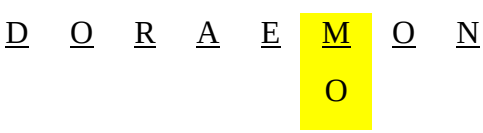
Gambar III.12 Percobaan ke 5 fungsi *Rolling hash*

Nilai *hash* dari indeks ke 4 tidak cocok dengan nilai *hash* target, maka dilakukan pergeseran.

Percobaan ke 6 :

Hash(O) = 79

Hash (y[5]) = 77



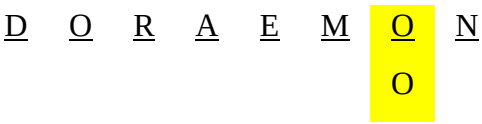
Gambar III.13 Percobaan ke 6 fungsi *Rolling hash*

Nilai *hash* dari indeks ke 5 tidak cocok dengan nilai *hash* target, maka dilakukan pergeseran.

Percobaan ke 7 :

Hash(O) = 79

Hash (y[6]) = 79



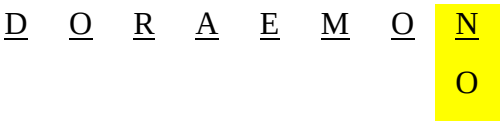
Gambar III.14 Percobaan ke 7 fungsi *Rolling hash*

Terjadi kecocokan nilai *hash* pada indeks ke 6, maka algoritma menandai lokasi penemuan dan melanjutkan pencarian sampai karakter pada sumber *string* habis.

Percobaan ke 8 :

Hash(O) = 79

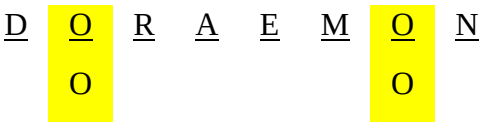
Hash (y[7]) = 78



Gambar III.15 Percobaan ke 8 fungsi *Rolling hash*

Nilai *hash* dari indeks ke 7 tidak cocok dengan nilai *hash* target. Karena *string* sudah habis, maka pencarian selesai.

Maka hasil dari pencarian berdasarkan pergeseran diatas menggunakan fungsi *hash* adalah huruf " O " ditemukan di indeks ke " 1 " dan " 6 ".



Gambar III.16 Hasil pergeseran dengan fungsi *Rolling hash*

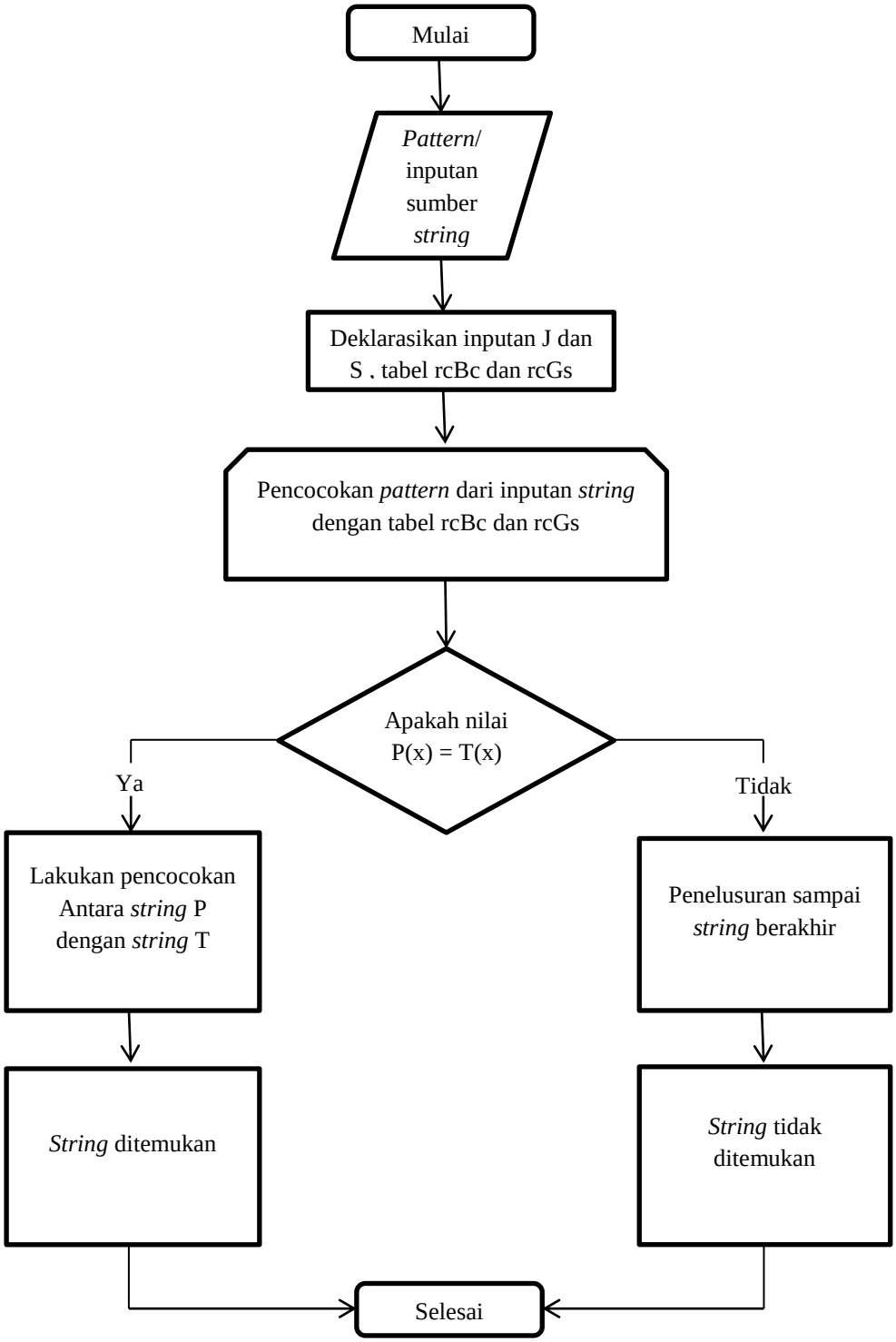
III.1.4.1.2. Algoritma Reverse Colussi

Pada algoritma *reverse colussi*, harus ditentukan terlebih dahulu pemrosesan awalnya kemudian melakukan pencocokan *string* dengan menggunakan tabel rcBc dan rcGs. Berikut adalah *pseudocode* algoritma *Reverse Colussi* dalam Bahasa Javascript :

```
function searchReverseColussi(pattern, text){                    O(1)
  var result = new Array();                                      O(1)
  var _text = text.toString();                                   O(1)
  var _pattern = pattern.toString();                             O(1)
  var initial_index = 0;                                         O(1)
  var charleft = 0;                                              O(1)
  while (initial_index != -1){                                   O(n)
      console.log("");                                           O(1)
      initial_index = _text.indexOf(_pattern[0]);                O(1)
      if (initial_index != -1){                                  O(1)
        if(_text.substr(initial_index).length >=                 O(1)
        _pattern.length){
          if(_text[initial_index+(_pattern.length-1)] ==         O(1)
        _pattern[_pattern.length-1]){
            result[result.length] = initial_index + charleft; O(1)
          }
        }
            _text = _text.substr(initial_index+1);               O(1)
            charleft = charleft + initial_index+1;               O(1)
      }
  }
  rc_memory = sizeof(result) + sizeof(_text) +                   O(1)
  sizeof(_pattern) + sizeof(initial_index) + sizeof(charleft);
      return result;                                             O(1)
}
```

Gambar III.17. *Pseudocode* Algoritma *Reverse Colussi* dalam Javascript

Berikut ini adalah *Flowcart* alur kerja algoritma *Reverse Colussi* menggunakan tabel rcBc dan rcGs :



Gambar III.18. *Flowcart* Alur Kerja Algoritma *Reverse Colussi*

Berikut ini adalah pencarian algoritma *Reverse Colussi* dengan menggunakan 2 fase pencarian yaitu fase pemrosesan awal dan fase pencocokan *string* berdasarkan tabel rcBc dan rcGs.

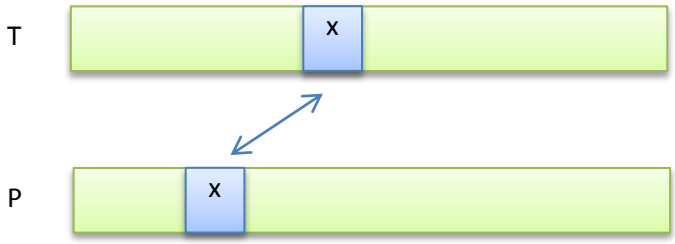
Fase pemrosesan awal

Pertama algoritma harus menentukan *special position* terlebih dahulu. Misal Pada algoritma *reverse colussi*, harus ditentukan beberapa *point* yang dianggap spesial dan beberapa *point* yang dianggap tidak spesial. Spesial *point* memungkinkan angka pergeseran yang lebih kecil dari pada bukan spesial *point.* Oleh karena itu, dalam algoritma *reverse colussi*, Ti merupakan karakter ke-i pada T ($1 \leq i \leq n$). Dan Pj adalah karakter ke-j pada P ($1 \leq j \leq m$). Berikut gambaran mengenai apa yang disebut sepasang karakter.

Aturan tabel rcBc (Reverse Colussi Bad Character)

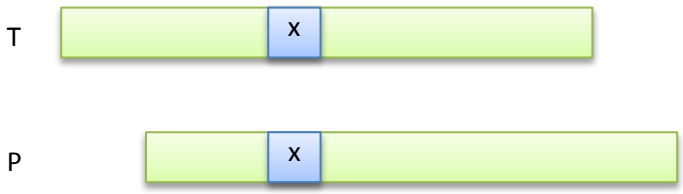
Kondisi 1 :

Untuk setiap karakter x pada T, menemukan karakter x yang terdekat pada P yang terletak pada sisi kiri dari karakter x pada T.



Gambar III.19. Fase pemrosesan awal

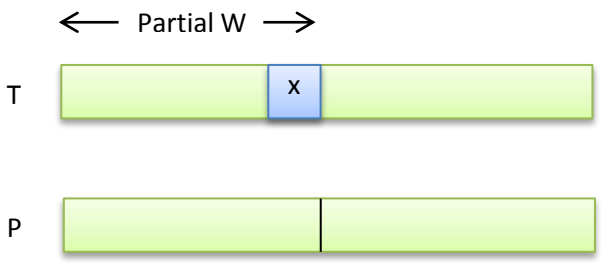
Jika terdapat karakter x pada P yang terletak disebelah kiri dari karakter x pada T, maka geser P sampai kedua karakter x tersebut sejajar.



Gambar III.20. Fase pemrosesan awal

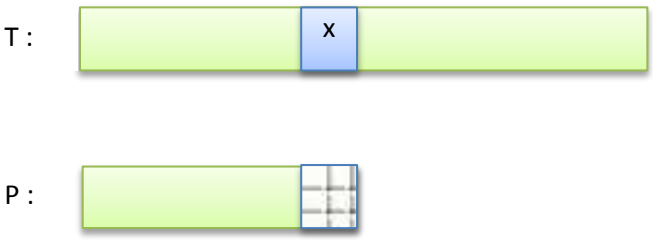
Kondisi 2 :

Jika tidak ada, maka buat *partial window* berdasarkan posisi karakter x pada T dan string yang ada di sebelah kirinya. Setelah ini, proses pencarian hanya sebatas ukuran *window* saja dan hitung sisa panjang pattern di *partial W*.



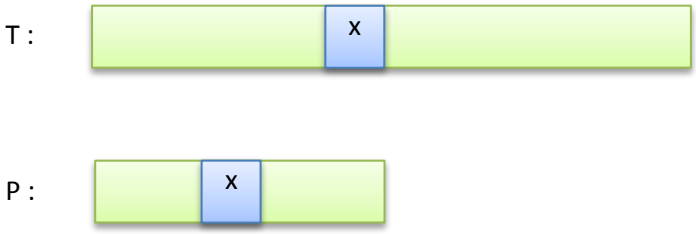
Gambar III.21. Fase pemrosesan awal

Misalkan karakter x yang terakhir pada *window* T tidak cocok dengan karakter terakhir pada P



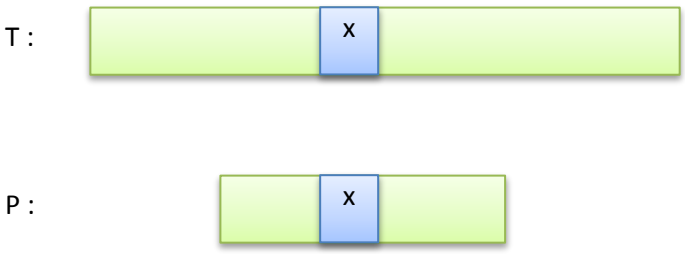
Gambar III.22. Fase pemrosesan awal

Misalkan terdapat karakter x pada P:



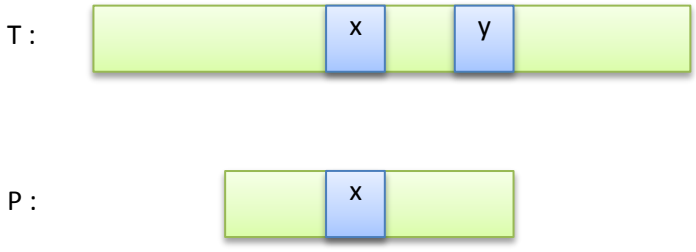
Gambar III.23. Fase pemrosesan awal

Maka, sejajarkan posisi x pada P dengan posisi x pada T



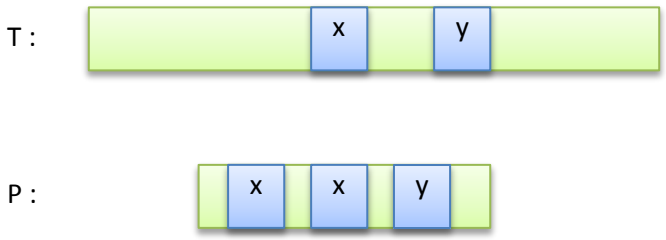
Gambar III.24. Fase pemrosesan awal

Misalkan karakter terakhir y dari *window* pada T tidak cocok dengan karakter terakhir dari P:



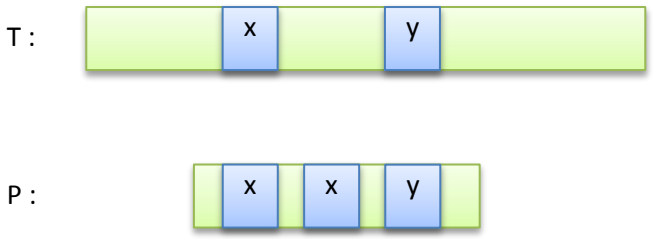
Gambar III.25. Fase pemrosesan awal

Jika tidak cocok, kemudian cari pasangan x dan y pada P. Misal posisi x dan y sebagai berikut:



Gambar III.26. Fase pemrosesan awal

Maka geser sampai posisi pasangan x dan y pada P sejajar dengan posisi x dan y pada T, sepasang karakter sudah diperoleh.



Gambar III.27. Fase pemrosesan awal

Pencarian pasangan karakter

Dalam algoritma reverse colussi, pencarian pasangan karakter dapat kita lakukan dengan menggunakan bantuan tabel rcBc (Reverse Colussi Bad Character). Misalkan:

Y : karakter terakhir dari window pada T

s : panjang pergeseran

k : bilangan integer

m : panjang pattern

T : inputan huruf

P : pattern atau kata yang dicari

Kondisi 1:

Jika ditemukan Pm-k-1=Y and Pm-k-s-1=Pm-s-1,

masukkan k terkecil ke tabel rcBc[Y,s]

Kondisi 2:

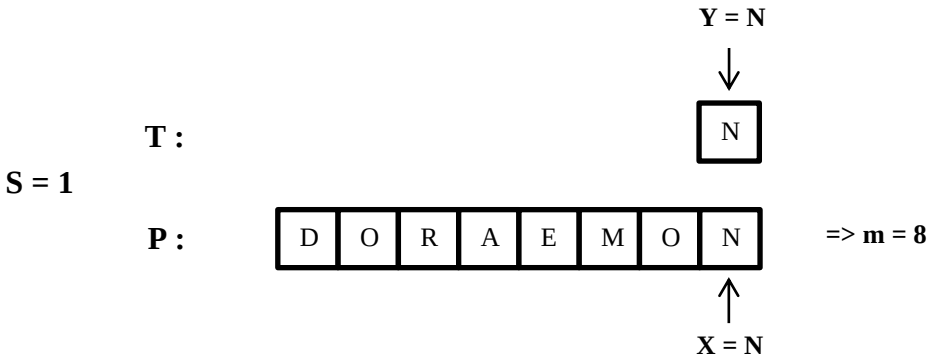
Jika ditemukan Pm-k-1=Y and k>m-s-1,

masukkan k terkecil ke tabel rcBc[Y,s]

Kondisi 3:

Jika tidak ditemukan,

masukkan m (panjang pattern) ke tabel rcBc[Y,s]

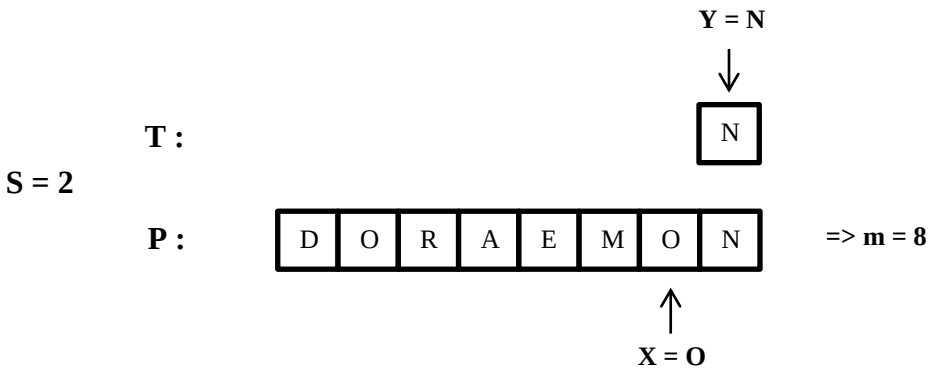


Gambar III.28. mencari pasangan karakter N ke N

XY = N N Tidak terdapat pada P, maka : rcBc [Y,1] = m => rcBc [Y,1] = 8

Tabel III.1. rcBc (Reverse Colussi Bad Character)

x \ s	1	2	3	4	5	6	7	8
N	8							
O								
M								
E								
A								
R								
D								



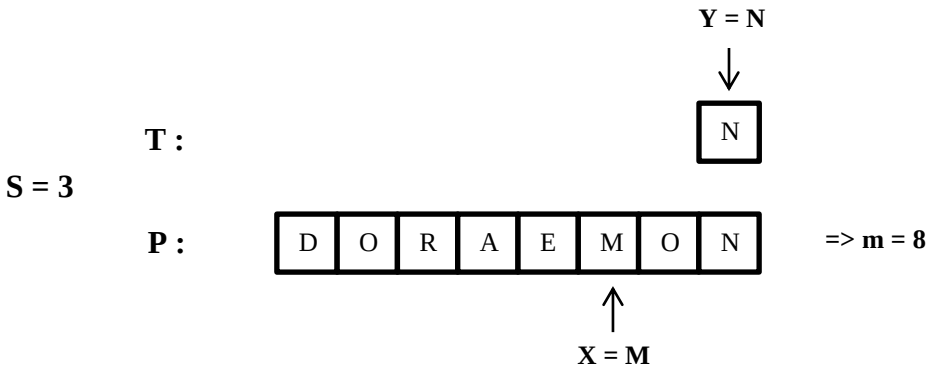
Gambar III.29. mencari pasangan karakter O ke N

XY = | O | N | Tidak terdapat pada P, maka :

rcBc [Y,2] = m => rcBc [Y,2] = 8

Tabel III.2. rcBc (Reverse Colussi Bad Character)

X \ S	1	2	3	4	5	6	7	8
N	8	8						
O								
M								
E								
A								
R								
D								



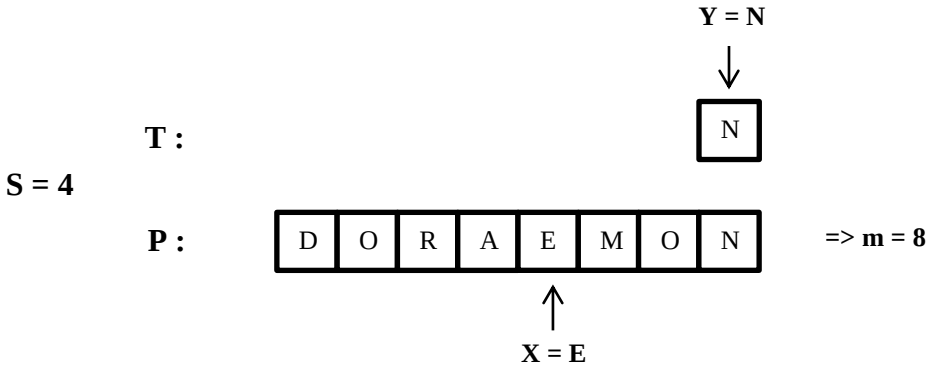
Gambar III.30. mencari pasangan karakter M ke N

XY = | M | | N | Tidak terdapat pada P, maka :

rcBc [Y,3] = m => rcBc [Y,3] = 8

Tabel III.3. rcBc (Reverse Colussi Bad Character)

X \ S	1	2	3	4	5	6	7	8
N	8	8	8					
O								
M								
E								
A								
R								
D								



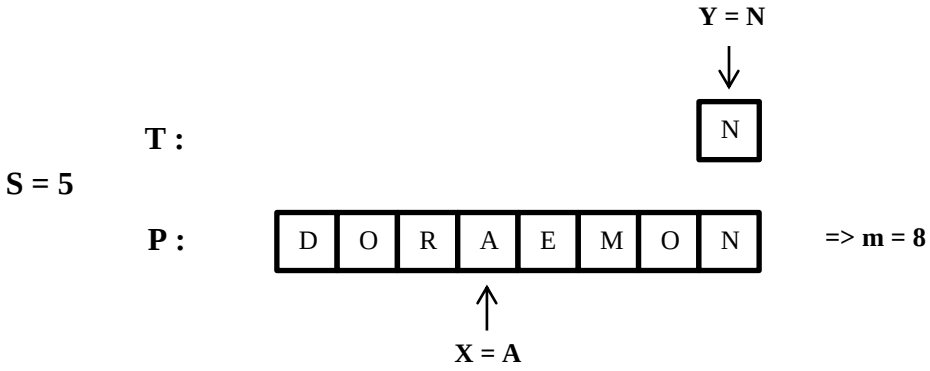
Gambar III.31. mencari pasangan karakter E ke N

XY = | E | | | N | Tidak terdapat pada P, maka :

rcBc [Y,4] = m => rcBc [Y,4] = 8

Tabel III.4. rcBc (Reverse Colussi Bad Character)

X \ S	1	2	3	4	5	6	7	8
N	8	8	8	8				
O								
M								
E								
A								
R								
D								



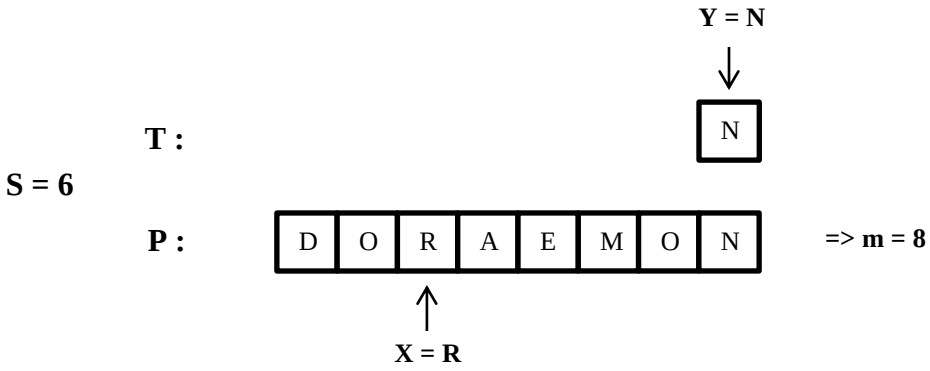
Gambar III.32. mencari pasangan karakter A ke N

XY = A | | | | N Tidak terdapat pada P, maka :

rcBc [Y,5] = m => rcBc [Y,5] = 8

Tabel III.5. rcBc (Reverse Colussi Bad Character)

X \ S	1	2	3	4	5	6	7	8
N	8	8	8	8	8			
O								
M								
E								
A								
R								
D								



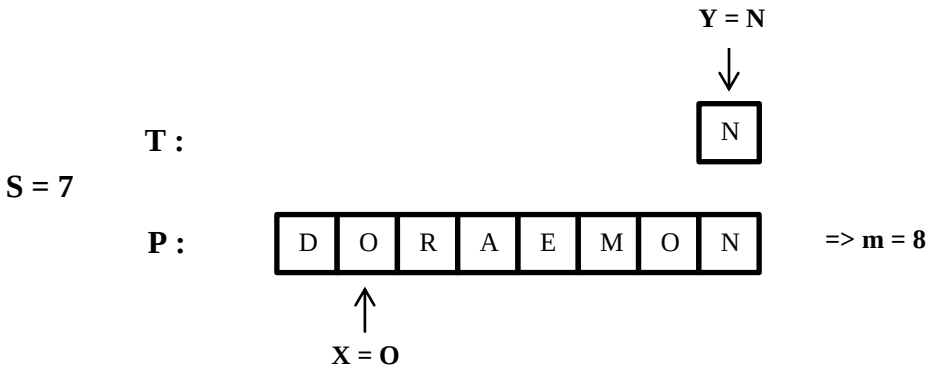
Gambar III.33. mencari pasangan karakter R ke N

XY = | R | | | | | N | Tidak terdapat pada P, maka :

rcBc [Y,6] = m => rcBc [Y,6] = 8

Tabel III.6. rcBc (Reverse Colussi Bad Character)

x \ S	1	2	3	4	5	6	7	8
N	8	8	8	8	8	8		
O								
M								
E								
A								
R								
D								



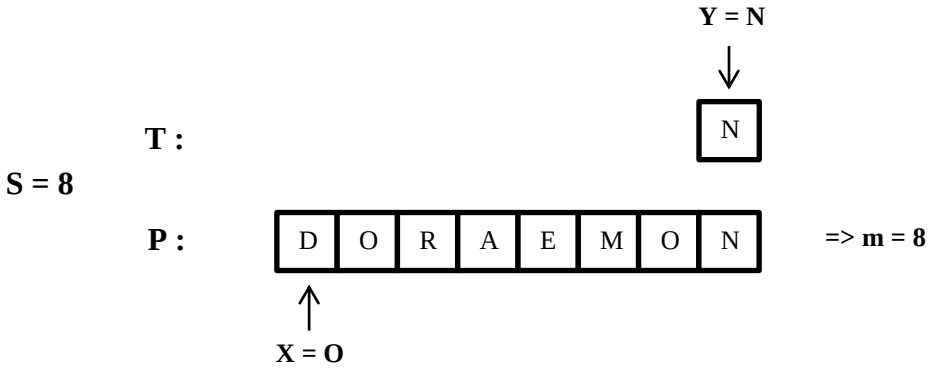
Gambar III.34. mencari pasangan karakter O ke N

XY = | O | | | | | | N | Tidak terdapat pada P, maka :

rcBc [Y,7] = m => rcBc [Y,7] = 8

Tabel III.7. rcBc (Reverse Colussi Bad Character)

x \ S	1	2	3	4	5	6	7	8
N	8	8	8	8	8	8	8	
O								
M								
E								
A								
R								
D								



Gambar III.35. mencari pasangan karakter D ke N

XY = | D | | | | | | | N | Tidak terdapat pada P, maka :

rcBc [Y,8] = m => rcBc [Y,8] = 8

Tabel III.8. rcBc (Reverse Colussi Bad Character)

X \ S	1	2	3	4	5	6	7	8
N	8	8	8	8	8	8	8	8
O								
M								
E								
A								
R								
D								

Selanjutnya lakukan seperti pencarian diatas untuk mencari tabel rcBc pada karakter O,M,E,A,R, dan D. Sehingga hasil tabel rcBc nya sebagai berikut :

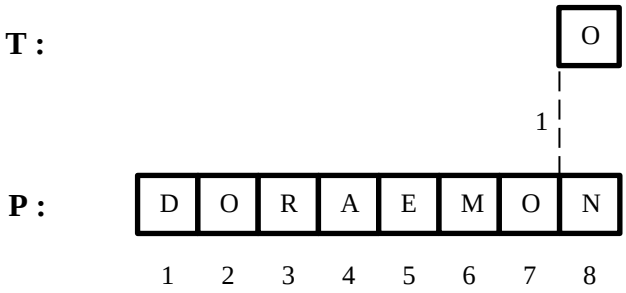
Tabel III.9. rcBc (Reverse Colussi Bad Character)

X \ S	1	2	3	4	5	6	7	8
N	8	8	8	8	8	8	8	8
O	8	8	6	6	6	6	6	6
M	8	8	8	8	8	8	2	2
E	8	8	8	8	8	3	3	3
A	8	8	8	8	4	4	4	4
R	8	5	8	5	5	5	5	5
D	8	8	8	8	8	8	8	8

Fase Pencocokan *String*

Pada tahap ini algoritma akan mencocokan *string* yang di*input* berdasarkan *pattern* yang telah ditentukan terlebih dahulu oleh algoritma pada fase pemrosesan awal berdasarkan tabel rcBc dan rcGs.

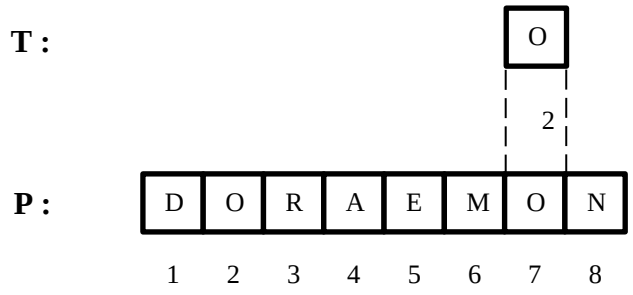
Percobaan ke-1



Gambar III.36. Fase pencocokan *pattern* percobaan ke - 1

Sesuai kondisi 1 (*rcBc*[O][*s*], *s* = 8), maka geser s = 1

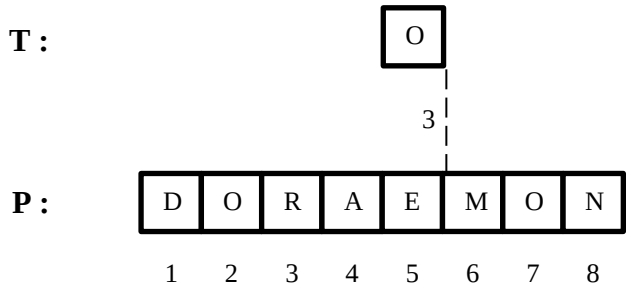
Percobaan ke-2



Gambar III.37. Fase pencocokan *pattern* percobaan ke - 2

Sesuai kondisi 1 (*rcBc*[O][*s*], *s* = 8), maka geser s = 2

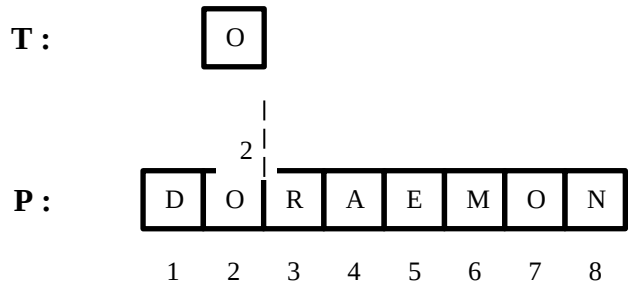
Percobaan ke-3



Gambar III.38. Fase pencocokan *pattern* percobaan ke - 3

Sesuai kondisi 2 (*rcBc*[O][*s*], *s* = 6), maka geser s = 3

Percobaan ke-4

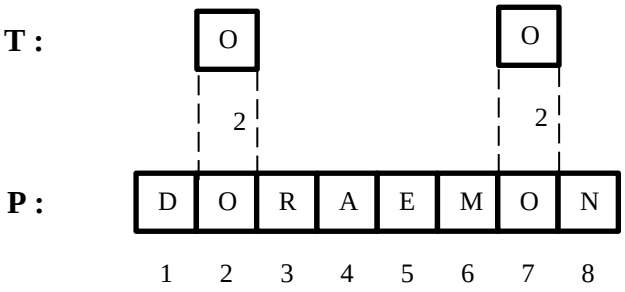


Gambar III.39. Fase pencocokan *pattern* percobaan ke - 4

Sesuai kondisi 1 (*rcBc*[O][*s*], *s* = 8), maka geser s = 2

Untuk mencocokkan huruf O pada kata DORAEMON memerlukan percobaan sebanyak 4 kali percobaan dengan hasil pencocokkan sebanyak 2 kali, yaitu :

Hasil pencocokkan



Gambar III.40. Hasil pencocokan *pattern*

III.1.5 Analisis Kompleksitas

Pada analisis kompleksitas ini akan dijelaskan parameter-parameter apa saja yang ingin diketahui dalam membandingkan performansi antara kedua algoritma untuk mendapatkan hasil performansi mana yang lebih optimal.

III.1.5.1 Analisis Kompleksitas Algoritma *Karp-Rabin*

Pada algoritma *Karp-Rabin* ini parameter yang ingin diketahui yaitu kompleksitas waktu yang dibutuhkan dalam 1 kali proses dan melihat berapa persentase *memory* dan *processor* yang dibutuhkan dalam 1 kali proses tersebut.

Untuk mendapatkan hasil dari kompleksitas waktu proses yang dibutuhkan dalam 1 kali proses / percobaan yaitu menggunakan perhitungan *Big-O*. berdasarkan *pseudocode* dari analisis kasus *Karp-Rabin* di atas, maka berikut tabel perhitungan *Big-O* nya :

Tabel III.10. Perhitungan *Big-O* Karp-Rabin

Pseudocode	***Big-O***
`ketemu ← false`	O(1)
`hs ← hash(s[1..m])`	O(1)
`for i ← 1 to n-m+1 do`	O(n)
`hsub ← hash(teks[1..i+m-1])`	O(1)
`if hsub = hs then`	O(1)
`if teks[i..i+m-1] = s then`	O(1)
`ketemu ← true`	O(1)
`hsub ← hash(teks[i+1..i+m])`	O(1)
`return ketemu`	O(1)
`Jumlah`	**O(n)**

Berdasarkan hasil perhitungan performansi algoritma *Karp-Rabin* menggunakan notasi *Big-O* didapat kompleksitas waktu dengan O(n) . dengan n adalah ukuran inputan.

III.1.5.2. Analisis Kompleksitas Algoritma *Reverse Colussi*

Pada algoritma *Reverse Colussi* ini parameter yang ingin diketahui yaitu kompleksitas pencocokan kata yang diinput dengan kata yang ada di *word list*.

Untuk mendapatkan hasil dari kompleksitas pencocokan kata yang diinput dengan kata yang ada di *word list* yaitu menggunakan pencarian dari tabel rcBc. berdasarkan perhitungan dari analisis kasus *Reverse Colussi* di atas.

Tabel III.11. Pseudocode Algoritma Reverse Colussi

Pseudocode	***Big-O***
`var result = new Array();`	O(1)
`var _text = text.toString();`	O(1)
`var _pattern = pattern.toString();`	O(1)
`var initial_index = 0;`	O(1)
`var charleft = 0;`	O(1)
`while (initial_index != -1){`	O(n)
`initial_index = _text.indexOf(_pattern[0]);`	O(1)
`if (initial_index != -1){`	O(1)
`if(_text.substr(initial_index).length >= O(1)` `_pattern.length){`	O(1)
`if(_text[initial_index+(_pattern.length-1)]==` `O(1)_pattern[_pattern.length-1]){`	O(1)
`result[result.length] = initial_index + charleft; O(1)}}`	O(1)
`_text = _text.substr(initial_index+1);`	O(1)
`charleft = charleft + initial_index+1;`	O(1)
`Jumlah`	**O(n)**

Berdasarkan hasil perhitungan performansi algoritma Reverse Colussi menggunakan notasi Big-O didapat kompleksitas waktu dengan $O(m^2)$ sebagai tahap 1 jika pencarian *multi pattern* dan kompleksitas waktu O(n) sebagai tahap 2 jika pencarian single pattern dengan n dan m^2 adalah ukuran inputan. pada fungsi pencarian (while (initial_index != -1)).

III.1.6. Analisis Kebutuhan Perangkat Lunak

Analisis kebutuhan perangkat lunak dibagi menjadi 2, yaitu analisis kebutuhan *non-fungsional* dan analisis kebutuhan *fungsional*. Tanpa perangkat

lunak maka perangkat keras tidak dapat berjalan, begitu juga sebaliknya. Maka perangkat lunak dan perangkat keras harus saling mendukung satu sama lainnya.

III.1.6.1. Analisis Kebutuhan Non – Fungsional

Analisis kebutuhan *non-fungsional* merupakan analisis yang dibutuhkan untuk menentukan spesifikasi kebutuhan sistem. Spesifikasi ini juga meliputi *elemen* atau komponen-komponen apa saja yang dibutuhkan untuk sistem yang akan dibangun sampai dengan sistem tersebut diimplementasikan. Analisis kebutuhan ini juga menentukan spesifikasi masukan yang diperlukan sistem keluaran yang akan dihasilkan sistem dan proses yang dibutuhkan untuk mengolah masukan sehingga menghasilkan suatu keluaran yang diinginkan.

Pada analisis kebutuhan sistem *non-fungsional* ini dijelaskan analisis kebutuhan perangkat lunak dan analisis kebutuhan perangkat keras.

III.1.6.1.1. Analisis Perangkat Keras

Agar aplikasi dapat berjalan dengan baik, maka dibutuhkan perangkat keras yang sesuai dengan kebutuhan aplikasi. Dibagian ini akan dijelaskan analisis kebutuhan perangkat keras dari sisi pengembang dan dari sisi pemain.

1. Dari sisi pengembang

Perangkat keras yang digunakan untuk membangun *game* ini adalah sebagai berikut :

a. *Processor* 2,4 GHz
b. *Memory* 1 GB
c. *Space* kosong di *harddisk* 5 GB
d. VGA *card* 256 MB
e. *Monitor* 1280 x 720
f. *Mouse* dan *keyboard*
g. *Speaker*

2. Dari sisi *user*

Spesifikasi minimum perangkat keras yang dibutuhkan agar dapat menjalankan *game* simulasi ini adalah sebagai berikut :

a. *Processor* dengan kecepatan 1.8 GHz.
b. *Memory* 512 MB
c. *Space* kosong di *harddisk* 100 MB
d. VGA *card (onboard)*
e. Monitor
f. *Mouse* dan *keyboard*
g. Speaker

III.1.6.1.2. Analisis Perangkat Lunak

Tanpa perangkat lunak maka perangkat keras tidak dapat berjalan, begitu juga sebaliknya. Maka perangkat lunak dan perangkat keras harus saling mendukung satu sama lainnya. Perangkat keras hanya berfungsi jika diberikan instruksi-instruksi kepadanya. Instruksi-instruksi inilah yang disebut dengan perangkat lunak.

1. Dari sisi pengembang

Perangkat lunak yang digunakan untuk membangun dan mengembangkan aplikasi *game* simulasi ini adalah sebagai berikut :

a. Sistem operasi *Windows* XP, Vista, *Windows* 7.
b. Aplikasi *Code Editor* seperti : *Ultra Edit, Notepad ++, Sublime Text* dan lain–lain.
c. *Adobe Photoshop* sebagai pengolah grafis.
d. Aplikasi *Browser* untuk menguji dan menjalankan aplikasi

2. Dari sisi pengguna

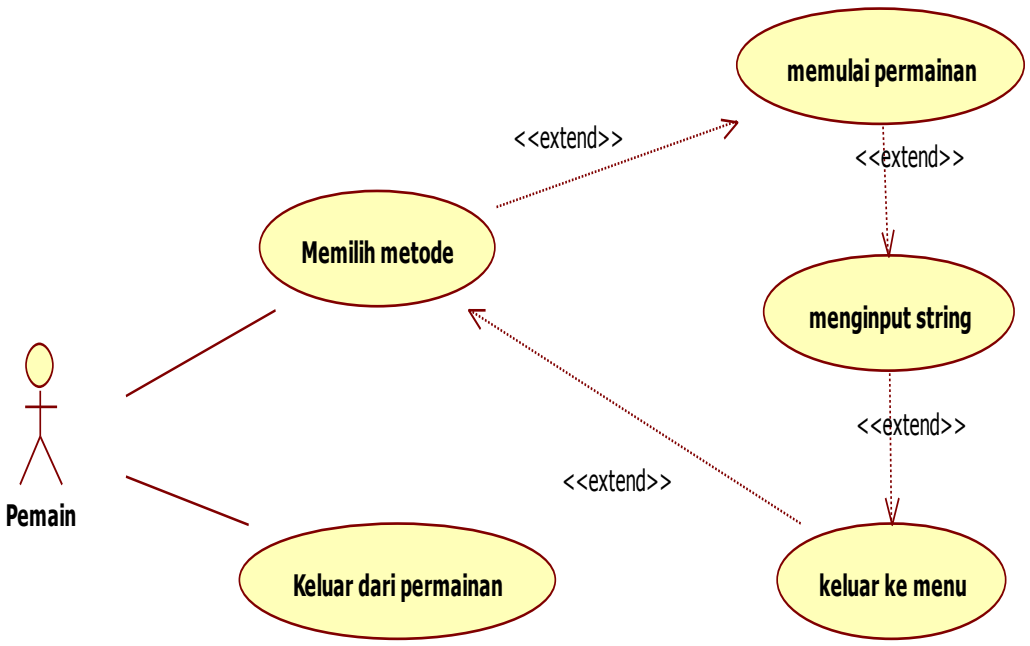
Untuk menjalankan aplikasi *game* simulasi ini dibutuhkan perangkat lunak, antara lain :

a. Sistem operasi *windows*.
b. Aplikasi *Browser* seperti : *Mozilla Firefox, Chrome, Opera,* dan lain-lain.

III.1.6.2. Analisis Kebutuhan *Fungsional*

Analisis kebutuhan *fungsional* menggambarkan proses kegiatan yang akan diterapkan dalam sebuah sistem dan menjelaskan kebutuhan yang diperlukan sistem agar sistem dapat berjalan dengan baik sesuai kebutuhan. Pada bagian ini akan dibahas tentang *Use Case Diagram, Activity Diagram, Class Diagram Dan Sequence Diagram.*

III.1.6.2.1 *Use case Diagram*



Gambar III.41. ***Use Case Diagram***

1. *Actor Definition*

Actor definition berfungsi untuk menjelaskan *Actor* yang terdapat pada *Use Case Diagram*. *Actor Definition* diterangkan pada *Table* III.12 *Actor Definition.*

***Table* III.12 Actor Definition**

No	*Actor*	Deskripsi
1	Pemain	Orang yang memainkan game

2. *Use Case Definition*

Use Case Definition berfungsi untuk menjelaskan fungsi *Use Case* yang terdapat pada *Use Case Diagram*.*Use Case Definition* diterangkan pada Tabel *Use Case Definition* dibawah ini.

Table* III.13 *Use Case Definition

No	*Use Case*	Deskripsi
1	Memilih metode	Proses memilih metode permainan
2	Memulai permainan	Proses memulai permainan berdasarkan metode yang dipilih
3	Menginput *string*	Proses memilih *string* yang cocok dengan kata yang dicari
4	Keluar Ke Menu Utama	Proses untuk keluar ke menu utama
5	Keluar dari permainan	Proses keluar dari permainan

Skenario *Use Case*

Use Case Skenario berisi mengenai *scenario* (*flow of event*) untuk beberapa *Use Case* utama, yang menggambarkan urutan interaksi *actor dengan Use Case* tersebut dari awal sampai akhir. *Use Case scenario* yang terdapat pada pembangunan aplikasi ini antara lain sebagai berikut :

***III.1.6.2.1.1. Use Case* Skenario Memilih Metode**

Use Case Scenario memilih metode berfungsi untuk menampilkan menu untuk memilih metode permainan.

***Table* III.14 Skenario *Use Case* Memilih Metode**

identifikasi	
Nomor	1
Nama	Memilih Metode
Tujuan	Memulai permainan dengan metode yang berbeda dari 2 algoritma

Deskripsi	Proses untuk memilih metode permainan
Aktor	Pemain
Skenario Utama	
Kondisi awal	Pemain berada di menu utama
Aksi Aktor	**Reaksi Sistem**
1. Memilih Metode	
	2. Menampilkan layar permainan dan papan input *string*
Kondisi Akhir	Pemain memainkan permainan berdasarkan metode yang dipilih

III.1.6.2.1.2. Use Case Skenario Memulai permainan

Use Case Skenario memulai permainan berfungsi untuk memainkan permainan mencari kata.

***Table* III.15 Skenario *Use Case* Memulai Permainan**

Identifikasi	
Nomor	2
Nama	Memulai permainan
Tujuan	Memulai permainan mencari kata
Deskripsi	Proses memulai permainan mencari kata yang telah ditentukan didalam permainan
Aktor	Pemain
Skenario Utama	
Kondisi awal	Pemain berada metode permainan yang dipilih
Aksi Aktor	**Reaksi Sistem**
1. Klik tombol *new game*	
	2. Menampilkan papan permainan
3. Menginput *string*	
	4. Cek *string* cocok atau tidak dengan kata yang ditentukan
Kondisi Akhir	Jika cocok maka pemain terus menginput *string* sampai kotak jawaban terisi semua dan permainan berakhir, anda menang. jika tidak cocok maka hukuman akan di aktifkan apabila melakukan 8 kali kesalahan maka permainan berakhir dan anda kalah.

III.1.6.2.1.3. Use Case Skenario Meng*input String*

Use Case Skenario meng*input string* berfungsi untuk memasukkan kata untuk mengisi kotak jawaban sesuai kata yang di tentukan.

Table III.16 Skenario *Use Case* Memasukkan Kata

Identifikasi	
Nomor	3
Nama	Meng*input string*
Tujuan	Untuk memasukkan kata jawaban dari kotak kata yang ditentukan
Deskripsi	Proses untuk memasukkan kata jawaban kata yang ditentukan
Aktor	Pemain
Skenario Utama	
Kondisi awal	Pemain berada di layar permainan
Aksi Aktor	**Reaksi Sistem**
1. Meng*input string*	
	5. Menampilkan ke kotak jawaban jika *string* yang dipilih benar, jika salah, aktifkan hukuman
Kondisi Akhir	Jika berhasil menebak, maka *You Win* Jika gagal menebak, maka *You Lose*

III.1.6.2.1.4. Use Case Skenario Keluar Ke Menu Utama

Use Case Skenario keluar ke menu utama yang berfungsi untuk keluar dari permainan dan kembali ke menu utama untuk memilih metode lainnya.

Table III.17 Skenario *Use Case* Keluar Ke Menu Utama

Identifikasi	
Nomor	5
Nama	Keluar ke menu utama
Tujuan	Untuk dari permainan dan kembali ke menu utama
Deskripsi	Proses untuk keluar dari permainan dan kembali ke menu utama untuk memulai dengan metode lain atau keluar dari permainan
Aktor	Pemain
Skenario Utama	
Kondisi awal	Pemain berada di metode permainan yang dipilih

Aksi Aktor	**Reaksi Sistem**
1. Memilih tombol keluar	
	2. Keluar dari permainan dan kembali ke menu utama
Kondisi Akhir	Pemain keluar dari permainan dan kembali ke menu utama untuk memilih bermain dengan metode lainnya atau keluar dari permainan untuk berhenti bermain

III.1.6.2.1.5. *Use Case* Skenario Keluar Dari Permainan

Use Case Skenario keluar dari permainan berfungsi untuk keluar dan berhenti bermain.

Table III.18 Skenario *Use Case* Keluar Dari Permainan

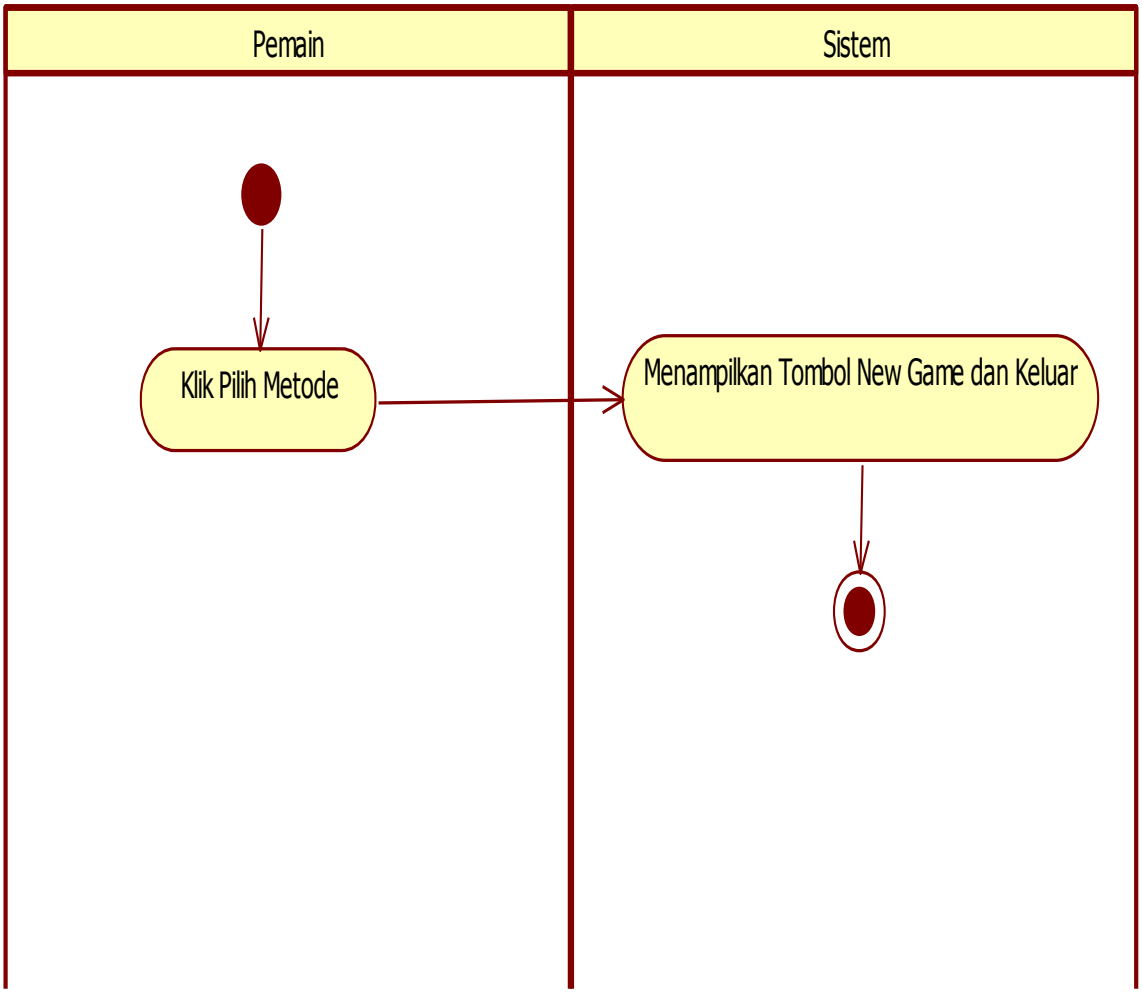
Identifikasi	
Nomor	6
Nama	Keluar Dari Permainan
Tujuan	Keluar untuk berhenti bermain
Deskripsi	Proses untuk keluar dari permainan dan berhenti bermain
Aktor	Pemain
Skenario Utama	
Kondisi awal	Pemain berada di menu utama
Aksi Aktor	**Reaksi Sistem**
III.1.6 Memilih tombol keluar dari permainan	
	IV.1.6 Keluar dari permainan
Kondisi Akhir	Pemain keluar dari permainan dan berhenti bermain

III.1.6.2.2. *Activity Diagram*

Activity diagram menggambarkan berbagai alur aktivitas dalam *system* yang sedang dirancang, bagaimana masing-masing alur berawal, decision yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses parallel yang mungkin terjadi pada beberapa eksekusi. Berikut ini *activity diagram* pada aplikasi *game* Algojo.

III.1.6.2.2.1. *Activity Diagram* Memilih Metode

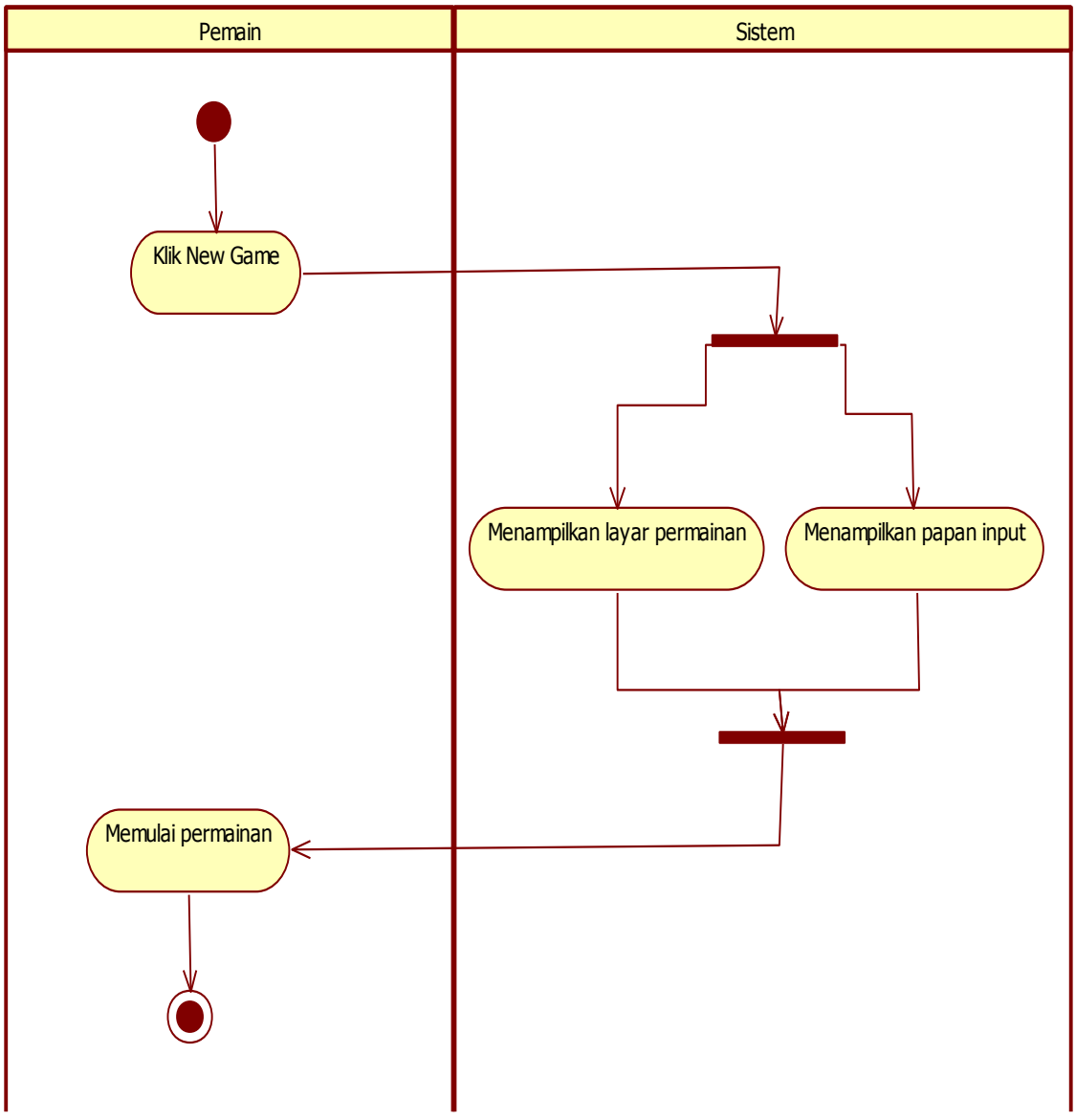
Activity diagram ini menjelaskan alur dari aktivitas memilih metode antar aktor dan sistem yang dijelaskan pada Gambar III.42.



Gambar III.42. *Activity Diagram* Memilih Metode

III.1.6.2.2.2. *Activity Diagram* Memulai Permainan

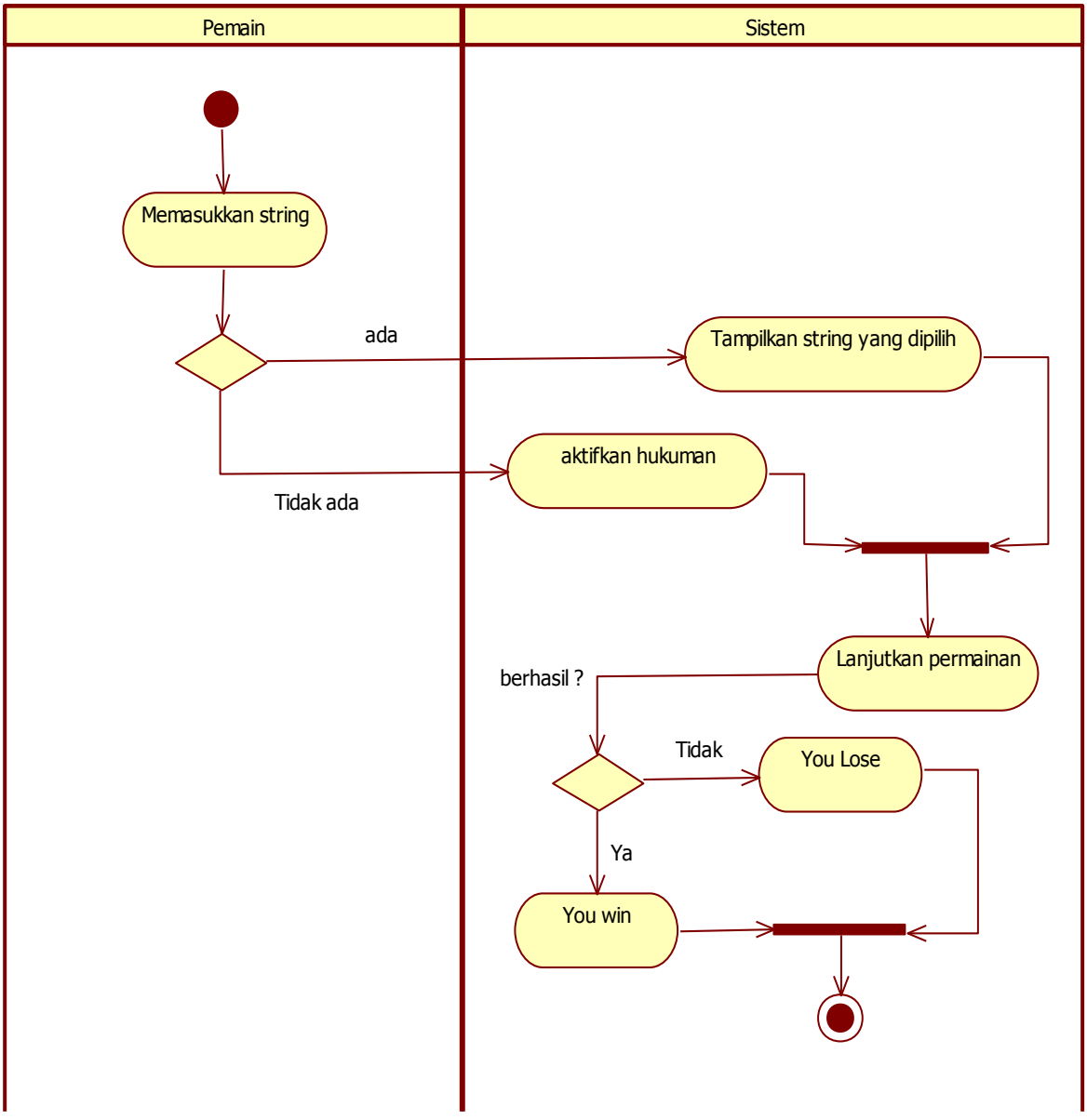
Activity diagram ini menjelaskan alur dari aktivitas memulai permainan antar aktor dan sistem yang dijelaskan pada Gambar III.43.



Gambar III.43. *Activity Diagram* Memulai Permainan

III.1.6.2.2.3. *Activity Diagram* Meng*input String*

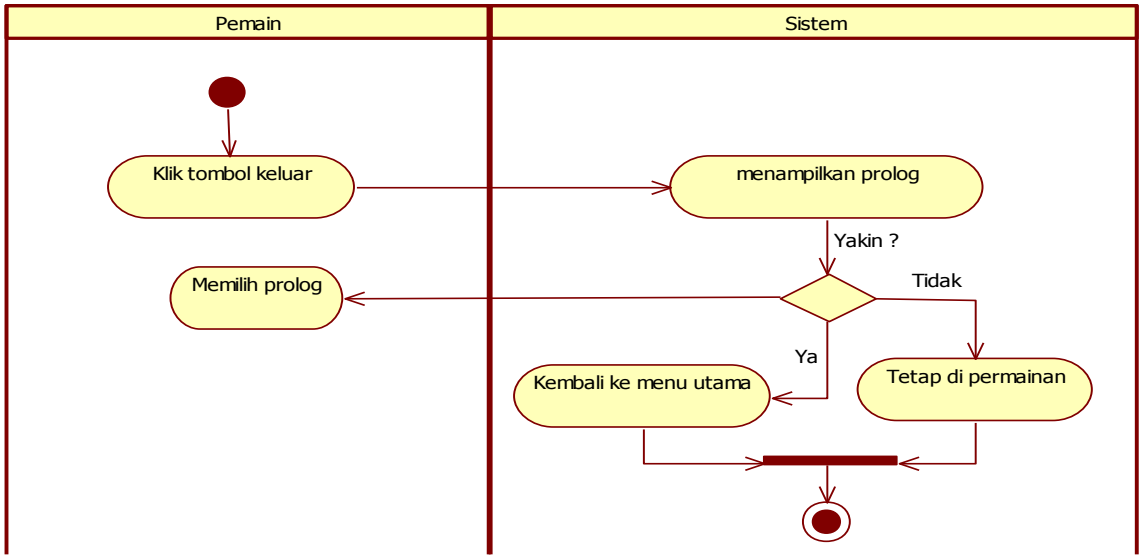
Activity diagram ini menjelaskan alur dari aktivitas memasukkan *string* antar aktor dan sistem yang dijelaskan pada Gambar III.44.



Gambar III.44. *Activity Diagram* Memasukkan *String*

III.1.6.2.2.4. *Activity Diagram* Keluar Ke Menu Utama

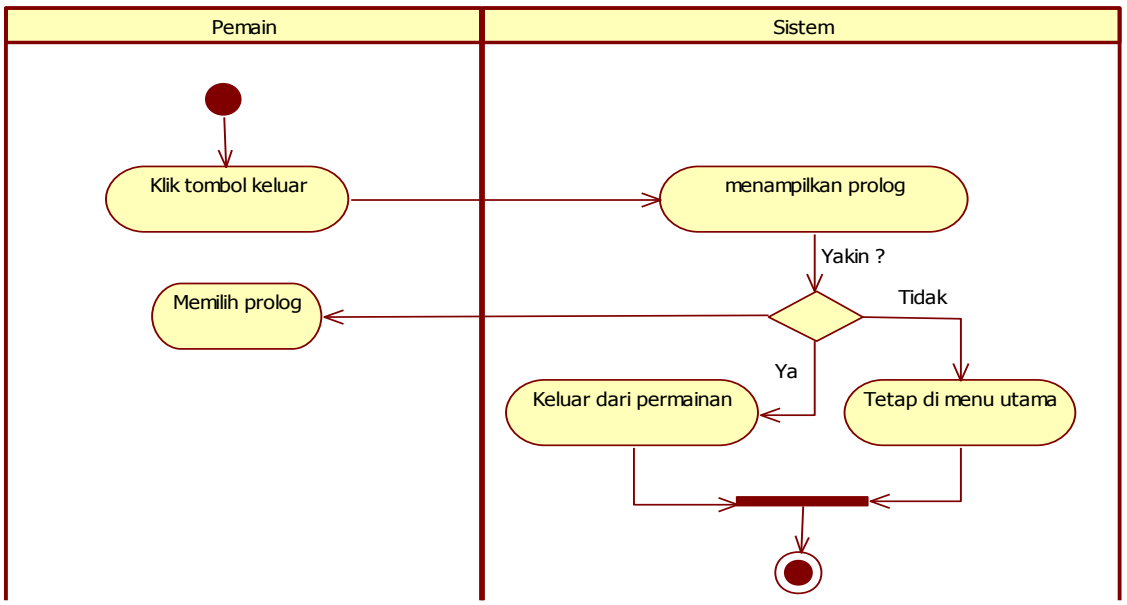
Activity diagram ini menjelaskan alur dari aktivitas keluar ke menu utama antar aktor dan sistem yang dijelaskan pada Gambar III.45.



Gambar III.45. *Activity Diagram* Keluar Ke Menu Utama

III.1.6.2.2.5. *Activity Diagram* Keluar Dari Permainan

Activity diagram ini menjelaskan alur dari aktivitas keluar dari permainan antar aktor dan *system* yang dijelaskan pada Gambar III.46.



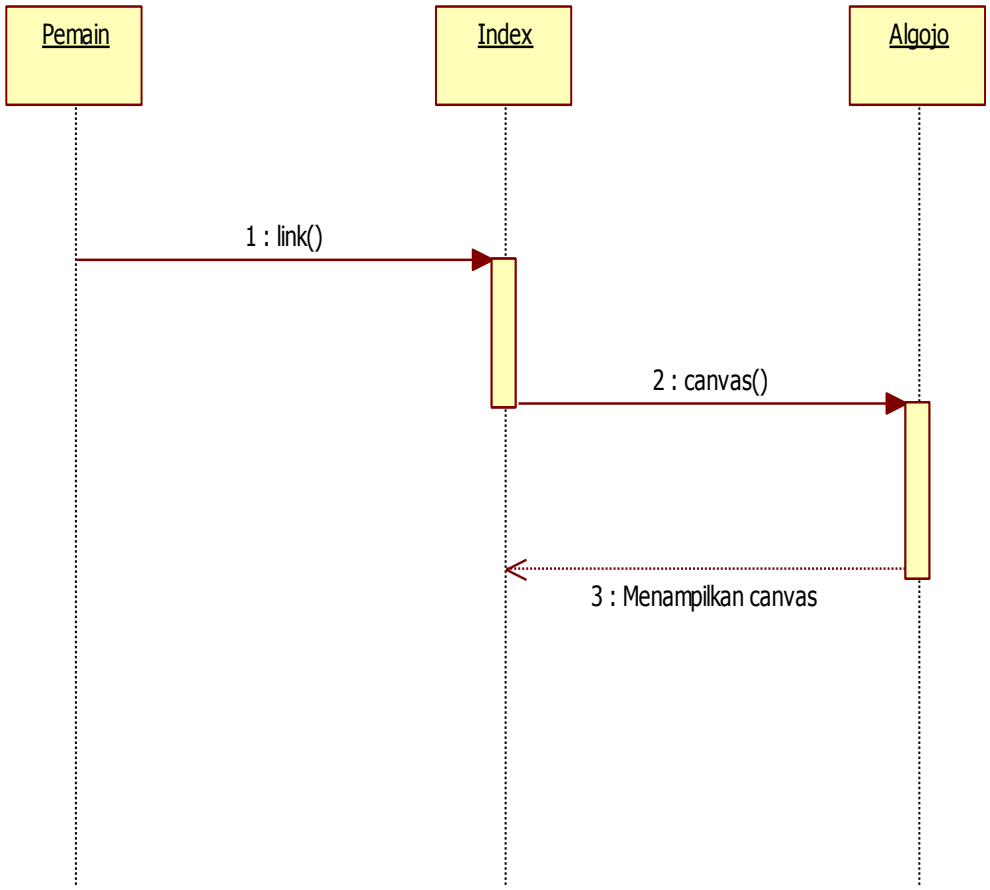
Gambar III.46. *Activity Diagram* Keluar Dari Permainan

III.1.6.2.3. *Sequence Diagram*

Sequence Diagram menggambarkan interaksi antar masing-masing objek pada setiap *use case* dalam urutan waktu. Interaksi ini berupa pengiriman serangkaian data antar objek-objek yang saling berinteraksi. *Sequence Diagram* pada "Perbandingan Algoritma Reverse Colussi dengan Algoritma Karp-Rabin dalam mencari dan mencocokkan *string* pada *word game"* terdiri dari *:*

III.1.6.2.3.1. *Sequence diagram* pilih metode

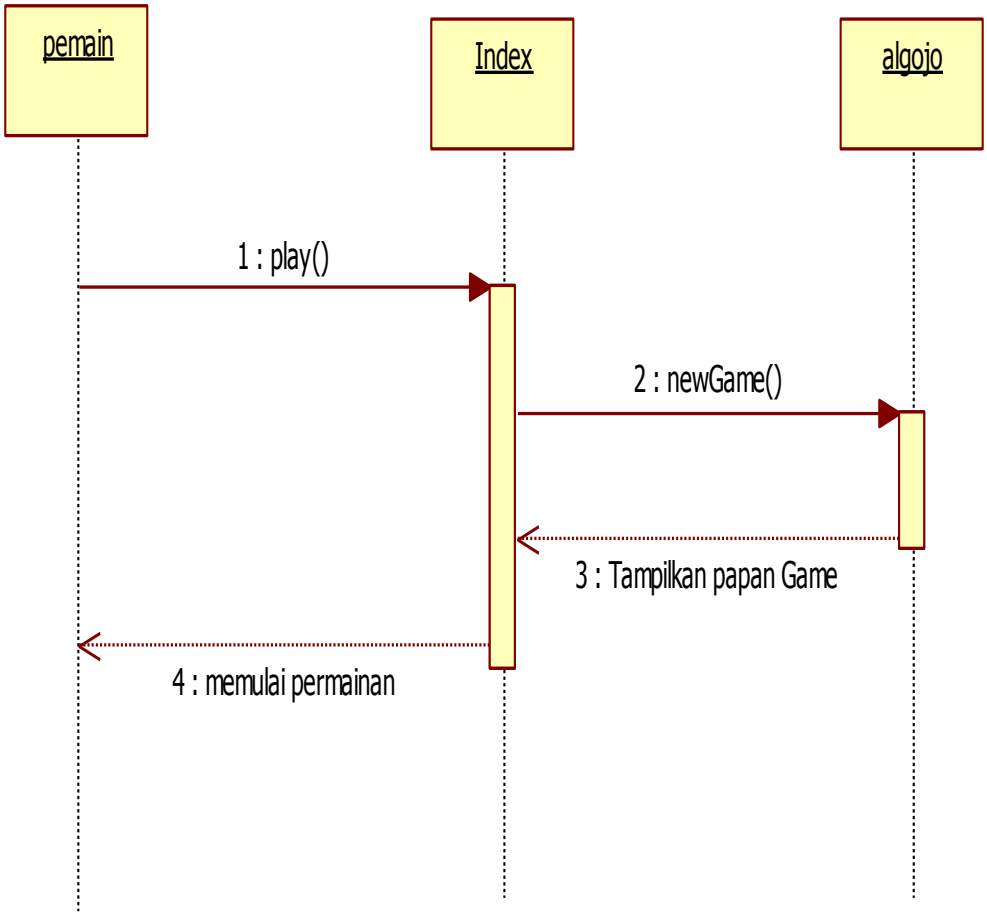
Diagram ini menjelaskan interaksi antar partisipan di dalam aktivitas memilih metode permainan sebelum bermain.



Gambar III.47. *Sequence diagram* pilih metode

III.1.6.2.3.2. *Sequence diagram* memulai permainan

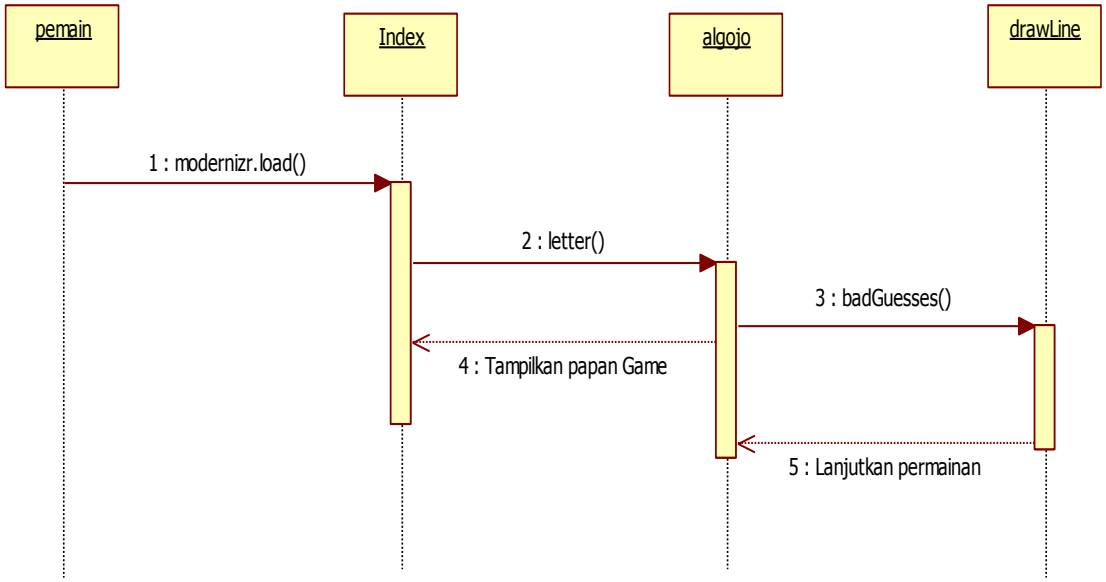
Diagram ini menjelaskan interaksi antar partisipan di dalam aktivitas memulai permainan saat metode telah dipilih.



Gambar III.48. *Sequence diagram* memulai permainan

III.1.6.2.3.3. *Sequence diagram* meng*input string*

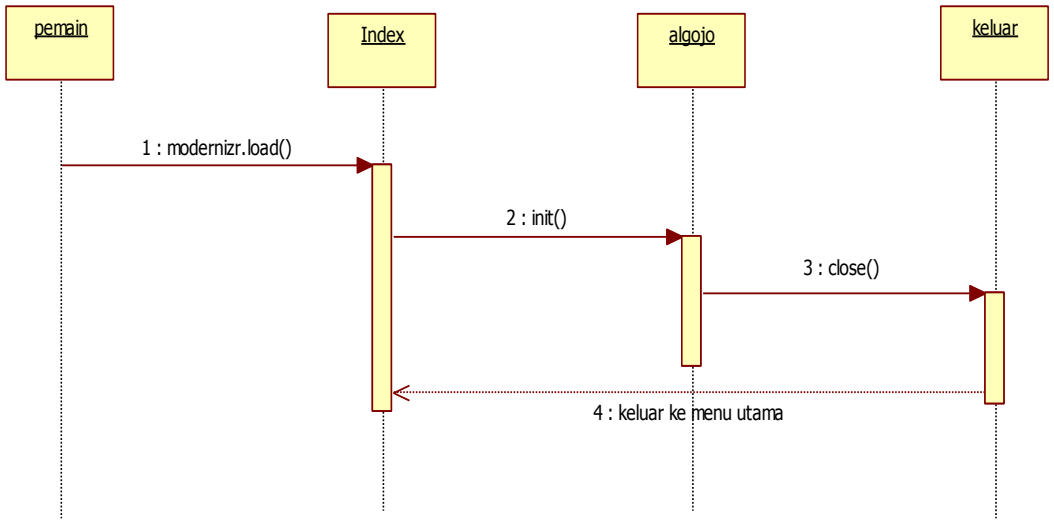
Diagram ini menjelaskan interaksi antar partisipan didalam aktivitas meng*input* huruf berdasarkan kata yang telah ditetapkan.



Gambar III.49. *Sequence diagram* meng*input string*

III.1.6.2.3.4. *Sequence diagram* keluar ke menu utama

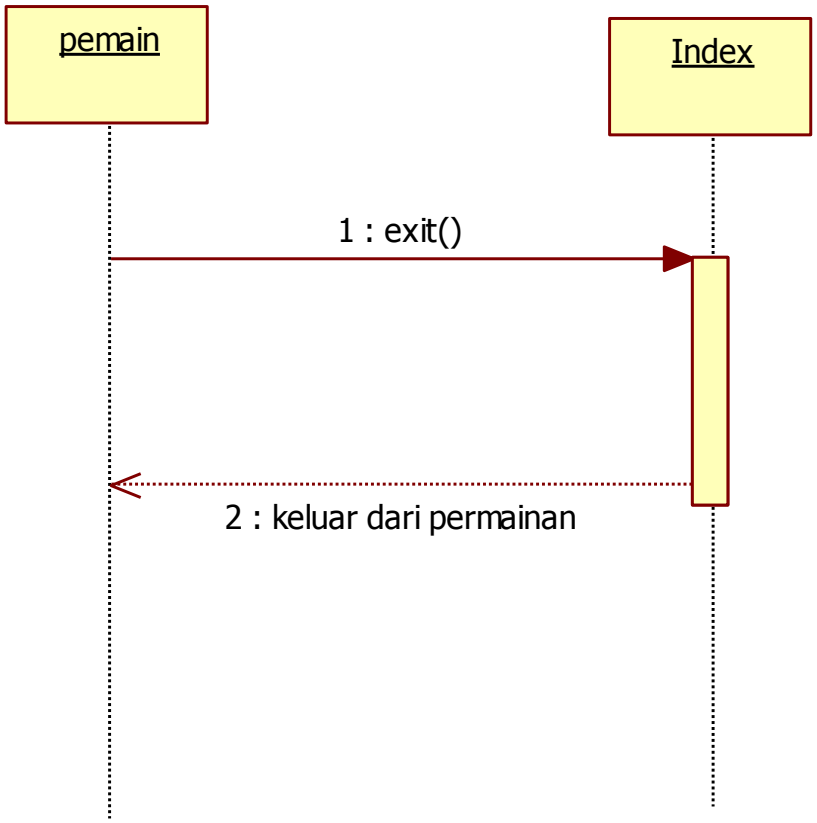
Diagram ini menjelaskan interaksi antar partisipan di dalam aktivitas keluar ke menu utama.permainan.



Gambar III.50. *Sequence diagram* keluar ke menu utama

III.1.6.2.3.5. *Sequence diagram* keluar dari permainan

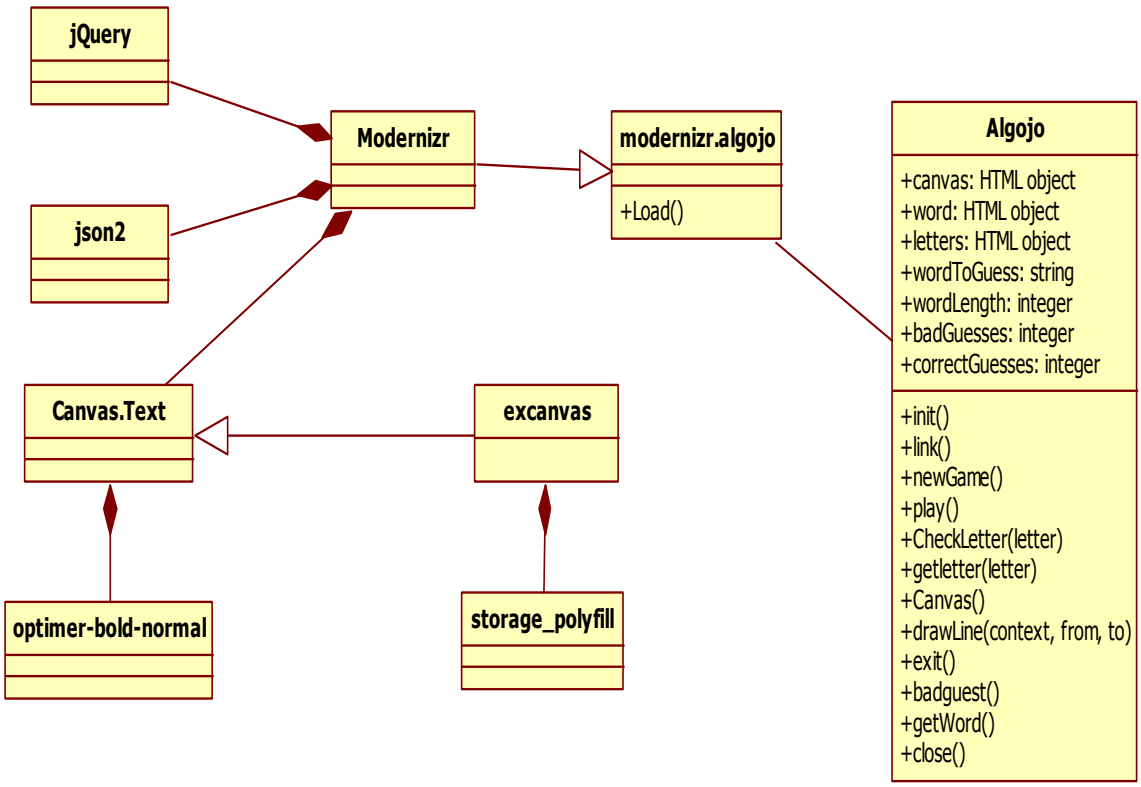
Diagram ini menjelaskan interaksi antar partisipan di dalam aktivitas keluar dari permainan.



Gambar III.51. *Sequence diagram* keluar dari permainan

III.1.6.2.4. Class Diagram

Class Diagram menggambarkan struktur dan hubungan antar objek-objek yang ada pada sistem. Struktur itu meliputi atribut dan *method* yang ada pada masing-masing *class* seperti terlihat pada gambar berikut ini.



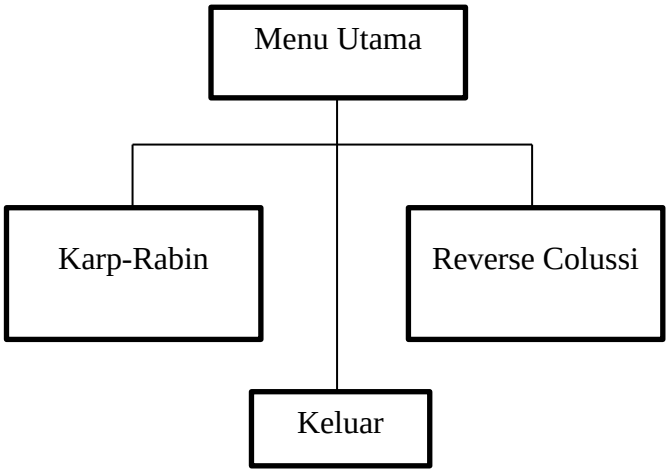
Gambar III.52 *Class Diagram*

III.2 Perancangan Sistem

Perancangan sistem merupakan kegiatan untuk merancang aplikasi yang akan di bangun, tahapannya dimulai dari perancangan struktur *menu*, perancangan antarmuka, perancangan pesan dan jaringan semantik.

III.2.1 Perancangan struktur menu

Perancangan struktur *menu* pada *game* pencarian dan pencocokan *string* pada gambar III.53 berikut :



Gambar III.53. Struktur *Menu*

Keterangan :

1. Menu *Reverse Colussi*

   Merupakan *submenu* untuk memulai permainan menggunakan metode Algoritma *Reverse Colussi* dalam mencari dan mencocokkan kata.

2. Menu *Karp-Rabin*

   Merupakan *submenu* untuk memulai permainan menggunakan metode Algoritma *Karp-Rabin* dalam mencari dan mencocokkan kata.

3. Menu Keluar

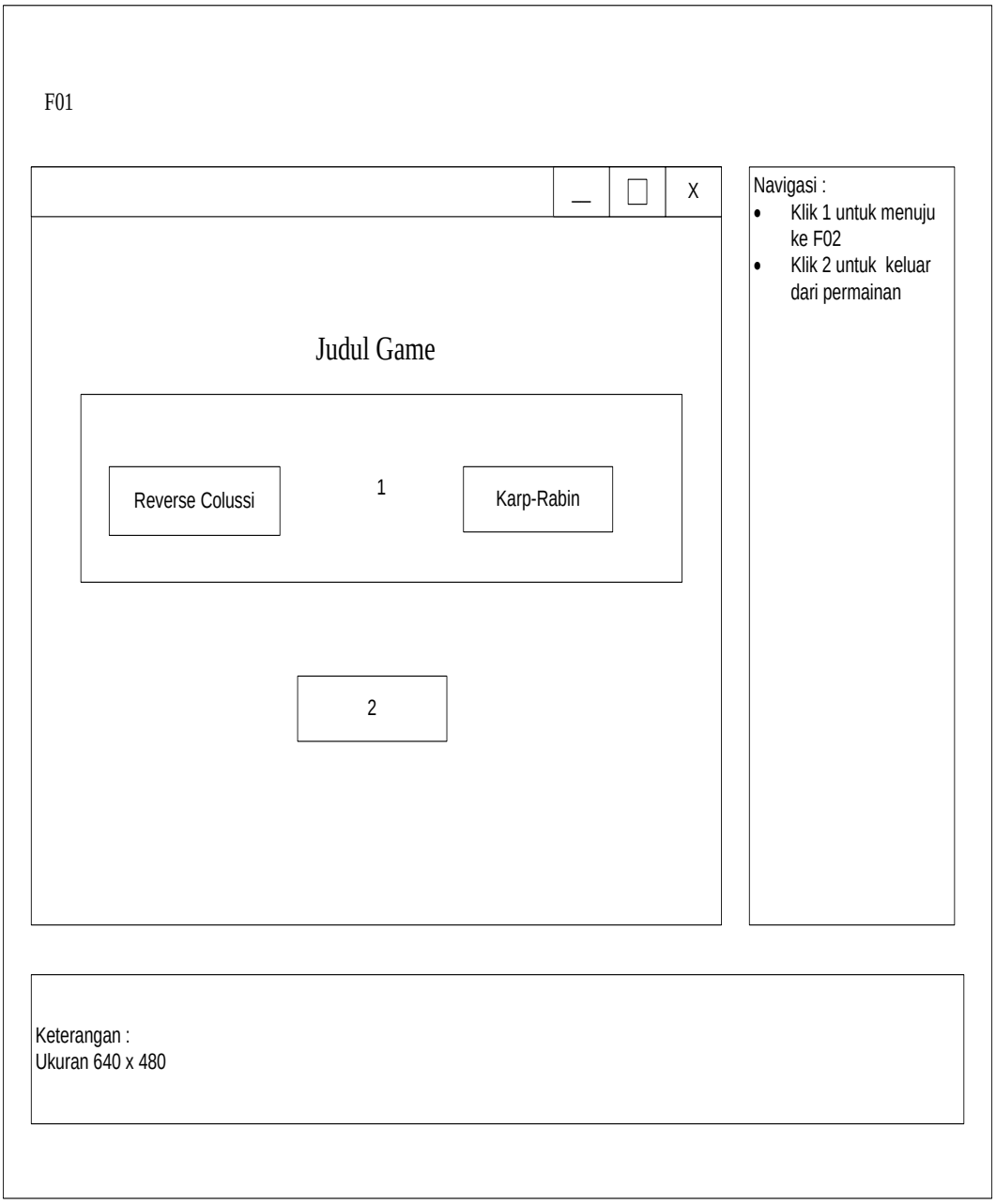
   Merupakan *submenu* yang berfungsi untuk keluar dari permainan.

III.2.2 Perancangan Antarmuka

Perancangan antarmuka bertujuan untuk memberikan gambaran tentang aplikasi yang akan dibangun, sehingga akan mempermudah dalam mengimplementasikan aplikasi serta akan memudahkan pembuatan aplikasi. Pada bagian ini akan dijelaskan perancangan antarmuka *menu* utama memilih metode, memulai permainan, dan meng*input string* sedangkan perancangan antarmuka *menu* keluar akan ditampilkan diperancangan pesan.

III.2.2.1 Perancangan Antarmuka *Menu* Utama

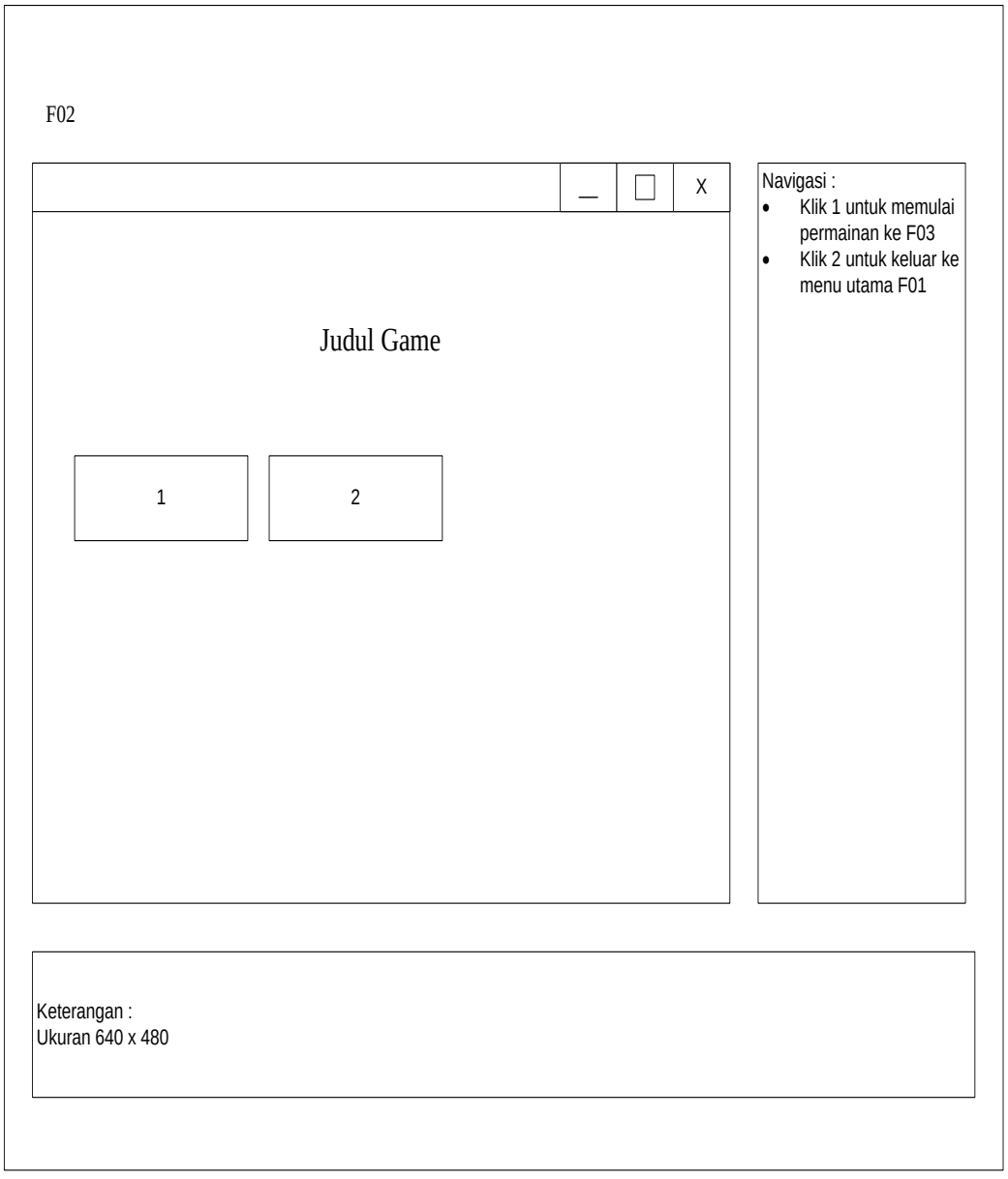
Menu utama merupakan *menu* yang akan muncul pertama kali pada saat pengguna membuka *game*.



Gambar III.54. Perancangan Antarmuka *Menu* Utama

III.2.2.2 Perancangan Antarmuka Memulai Permainan

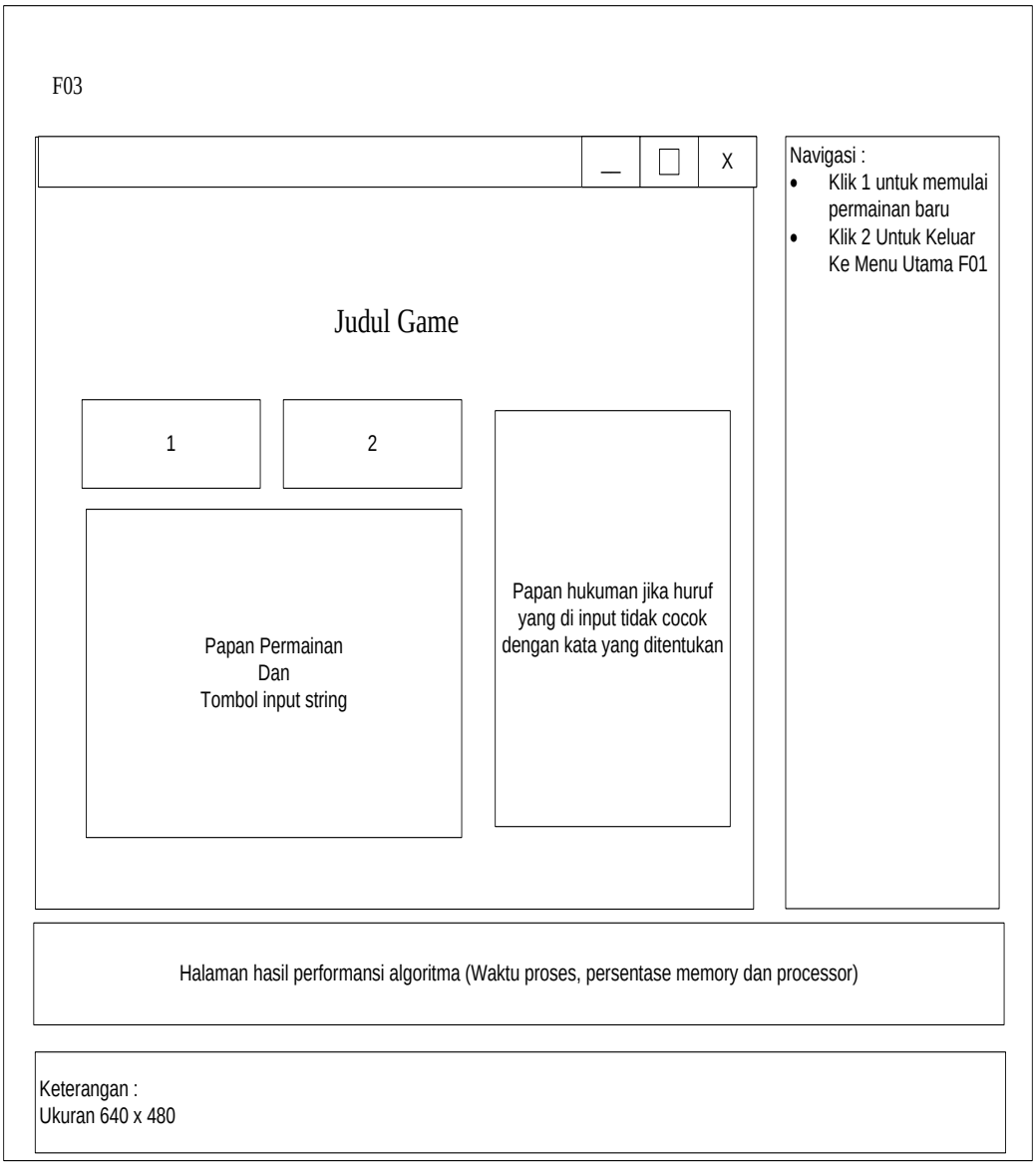
Memulai permainan adalah *menu* memulai permainan pencarian dan pencocokkan kata dengan metode algoritma yang dipilih di *menu* utama.



Gambar III.55. Perancangan Antarmuka Memulai Permainan

III.2.2.3 Perancangan Antarmuka Memasukkan Huruf

Menu memasukkan huruf adalah tahapan dalam bermain pencarian dan memilih huruf dengan metode algoritma yang dipilih di *menu* utama.



Gambar III.56. Perancangan Antarmuka Memasukkan Huruf

III.2.3 Perancangan Pesan

Menu keluar adalah *menu* keluar dari permainan dimana sebelum keluar dari permainan pemain akan ditanya dahulu apakah yakin mau keluar atau tidak.

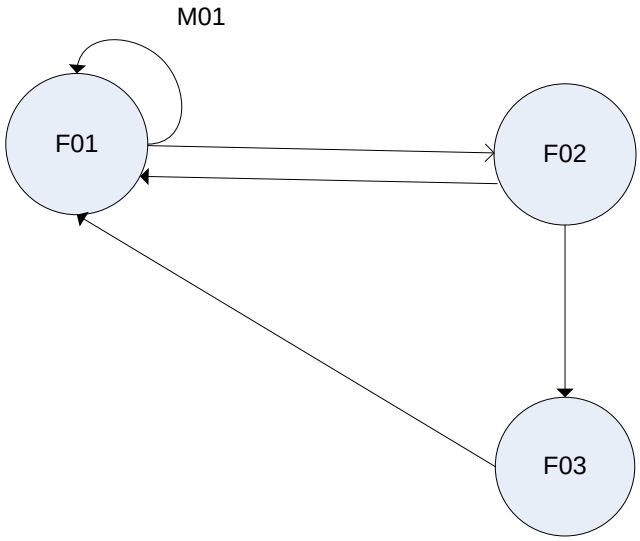


Gambar III.57. Perancangan Antarmuka Pesan Keluar.

III.2.4 Jaringan Semantik

Untuk mengetahui keterhubungan dari setiap yang dibuat dapat menggunakan jaringan semantik.

Gambar III.58 dibawah ini adalah jaringan semantik pada *word game* Algojo.



Gambar III.58 Gambar Jaringan Semantik