BAB 3 ANALISIS DAN PERANCANGAN

Pada bab ini akan dibahas mengenai analisis permainan, yaitu konsep aturan dan cara bermain pada *game* yang berhubungan dengan program yang sedang dibangun yaitu *game color puzzle* dan metode yang digunakan untuk pemecahan solusinya yaitu algoritma pencarian pola dan string Knuth-Morris-Pratt.

3.1. Analisis Permainan

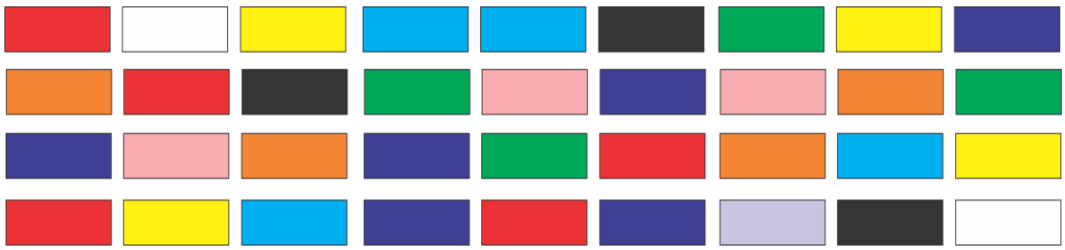
Game puzzle yang akan dibahas pada tugas akhir ini adalah sebuah *game* sejenis *word puzzle*. Pada permainan *word puzzle*, sejumlah kata dari sebuah matriks *cf char*. Kata adalah kumpulan dari karakter yang memiliki makna tertentu. Kata-kata yang dicari pada permainan seperti ini biasanya dapat dibaca dari kiri ke kanan, atas ke bawah, dan diagonal tergantung letak kata tersebut pada matirks *cf char*.

Permainan *word puzzle* seperti ini kemudian dimodifikasi sedemikian rupa dengan menghilangkan sebuah parameter masukan dengan tujuan meningkatkan akurasi perhitungan performansi pada saat aplikasi *game* diuji. Pada permainan *word puzzle* biasa, ada tiga parameter yang dibutuhkan dalam pencarian solusi: *Pattern/*kalimat, kata yang dicari, dan *database* kosakata. Namun secara konsep, algoritma KMP hanya membutuhkan dua masukan awal untuk dapat melakukan sebuah pencarian, yaitu *Pattern* utama dan kata atau *pattern* yang dicari. Penambahan parameter masukan lain yaitu *database* kosakata akan berpengaruh terhadap kompleksitas waktu algoritma yang digunakan karena pencarian pada basis data akan menambah beban pencarian. Karena hal tersebut, maka game *color puzzle* disini tidak menggunakan suatu basis data dan masukan untuk algoritma pencarian yang digunakan hanya ada dua. Alasan penggunaan warna dibandingkan dengan kata adalah karena tanpa menggunakan suatu basis data, maka kata yang tersusun pada *pattern* tidak akan memiliki makna khusus.

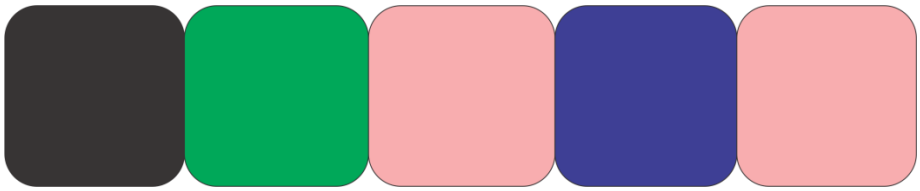
Game puzzle ini menggunakan algoritma Knuth-Morris-Pratt sebagai metode pencarian untuk proses pencocokan pola masukan dengan pola yang

dibandingkan. Pada proses pencocokan ini, akan didapat hasil berupa indeks atau posisi dimana kecocokan ditemukan, arah dari hasil pencarian, panjang iterasi dalam skenario pencarian terpanjang serta lamanya waktu total pencarian.

Misal sebuah susunan pola warna utama direpresentasikan sebagai *array* dua dimensi, *P* dengan panjang kolom *n* dan panjang baris *m*. Elemen-elemen pada *array P* kemudian dibandingkan dengan pola warna hasil inputan pengguna, direpresentasikan sebagai *array* satu dimensi, *p* dengan panjang *l*.



Gambar 3. 1 Susunan blok warna utama 9x4, direpresentasikan sebagai *array P* dengan panjang *n*=9 dan *m*=4.



Gambar 3. 2 Blok warna inputan pengguna *p* dengan panjang *l*=5

Susunan pola warna utama diilustrasikan pada gambar 3.1 sebagai array *P* dengan panjang 9 kolom dan 4 baris. Array *P* ini kemudian dijadikan landasan pencarian untuk susunan array *p* yang dimasukkan pengguna, digambarkan pada gambar 3.2 sebagai *array* dengan panjang 5. Pencarian dilakukan secara berurut mulai dari horizontal, vertikal, diagonal naik, dan diagonal turun. Apabila terjadi kecocokan elemen array *p* di dalam array *P*, maka indeks *P* dimana dimulainya kecocokan terjadi disimpan ke dalam sebuah array penampung hasil, *H*. Artinya, ada kemungkinan kecocokan terjadi lebih dari sekali untuk array *p* di dalam array *P*.

3.1.1. Analisis Pencarian Solusi Pada Permainan

Pada analisis permainan telah dijelaskan bahwa di dalam *game* ini, program mencari kecocokan warna yang diinput pengguna pada susunan warna utama. Sebelum menjelaskan cara-cara yang digunakan sebagai metode penyelesaian kasus pada game ini, ada beberapa hal dan istilah yang perlu didefinisikan di dalam game ini.

1. Warna pada setiap blok akan diwakilkan dengan inisial huruf untuk kemudahan penjelasan dalam analisis ini.
2. $P[n][m]$ merepresentasikan susunan pola utama dengan panjang kolom n dan baris m.
3. $p[l]$ merepresentasikan susunan pola masukan pengguna dengan panjang l.
4. Proses pencarian awal dilakukan pada susunan pola utama dengan mengiterasi dari kiri ke kanan, posisi blok di pojok kiri atas sebagai awal pencarian (elemen array $P[0][0]$). Ketika pencarian selesai, maka pencarian berikutnya akan dilakukan secara vertikal dari atas ke bawah dengan posisi blok di pojok kiri atas sebagai awal pencarian. Ketika pencarian kedua selesai, maka dilakukan pencarian berikutnya secara diagonal dari kiri atas ke kanan bawah dengan posisi blok di kanan atas sebagai awal pencarian (panjang indeks [n] pada *array P*). Ketika pencarian ketiga selesai, maka dilakukan pencarian berikutnya secara diagonal dari kiri bawah ke kanan atas dengan posisi blok di kanan bawah sebagai awal pencarian (panjang indeks [m] dan panjang indeks [n] pada *array P*).
5. *Array* yang digunakan untuk menyimpan informasi hasil pencarian blok di sini, *H*, merupakan *array* asosiatif satu dimensi yang menyimpan informasi posisi indeks *array P* dimana ditemukan kecocokan, panjang urutan pola yang cocok, dan arah pola kecocokan.

3.1.1.1. Analisis Pencarian Solusi dengan Algoritma *Naïve Search/brute force*

Metode pencarian naif (*naïve search*) merupakan metode yang paling mudah dimengerti dan diimplementasikan sebagai algoritma pencarian umum.

Pada kasus pencarian di game ini, langkah-langkah yang dilakukan untuk mencari kecocokan pola dengan menggunakan metode *naïve search* atau *brute force* adalah sebagai berikut:

a) Arah Horizontal:
   1. Buat peubah *i, j, k,* dan *m*. Inisiasikan semuanya dengan nilai 0.
   2. Jika belum ada, buat sebuah *array* asosiatif untuk menampung indeks hasil, *H*. Dimana masing-masing elemen *H* adalah peubah *array* yang merepresentasikan arah hasil pencarian, yaitu: “horizontal”, “vertikal”, “diagonal_turun”, dan “diagonal_naik”.
   3. Bandingkan kecocokan antara array *P* ke-[*m*][*k*+*i*] dengan array *p* ke-*i* (*P*[*m*][*k*+*i*] = *p*[*i*] ?).
   4. Apabila terjadi kecocokan:
      a. Tambahkan nilai *i* dengan 1.
      b. Apabila nilai *i* belum sama dengan jumlah array *p* dan nilai *k*+*i* belum melebihi jumlah kolom pada array *P* - 1, ulangi langkah 3.
      c. Apabila nilai *i* sama dengan jumlah array *p-1* dan nilai *k*+*i* belum melebihi jumlah kolom pada array *P* - 1, maka pencarian dikatakan berhasil. Simpan nilai variabel *k* dan *m* ke dalam array *H* indeks ke-*j*. Kemudian tambahkan variabel *j* dengan 1. Tambahkan nilai *k* dengan 1. Nilai *i* menjadi 0. Kemudian ulangi langkah 3.
      d. Apabila nilai *k*+*i* melebihi jumlah kolom pada array *P* - 1, maka pencarian di baris *m* dikatakan gagal. Hentikan pencarian. Tambahkan nilai *m* dengan 1. Nilai *k* dan *i* masing-masing menjadi 0.
      e. Apabila nilai *m* belum melebihi jumlah baris pada *array P* – 1, ulangi langkah 3.
      f. Apabila nilai *m* melebihi jumlah baris pada *array P* – 1, maka pencarian secara keseluruhan dikatakan gagal.
   5. Apabila terjadi ketidakcocokan:
      a. Tambahkan nilai *k* dengan 1. Nilai *i* menjadi 0.
      b. Apabila nilai *k* belum melebihi jumlah kolom pada array *P*-1, ulangi langkah 3.

c. Apabila nilai *k* melebihi jumlah kolom pada array *P*-1, maka pencarian di baris *m* dikatakan gagal. Hentikan pencarian. Tambahkan nilai *m* dengan 1. Nilai *k* dan *i* masing-masing menjadi 0.
d. Apabila nilai *m* belum melebihi jumlah baris pada *array P* – 1, ulangi langkah 3.
e. Apabila nilai *m* melebihi jumlah baris pada *array P* – 1, maka pencarian secara keseluruhan dikatakan gagal.

Berikut ini merupakan *pseudocode* dari metode *brute force*

```
Deklarasi
i, j, k, m : integer
H : Array

Algoritma:
I, j, k, m ← 0
while (m ≤ P.length-1) do
     if (P[m][k+i] = p[i]) then
          if (i = p.length-1) and (k+i ≤ P[m].length-1) then
               H[j][0] ← k
               H[j][1] ← m
               j ← j+1
               k ← k+1
               i ← 0
          else if (k+i > P[m].length-1) then
              m ← m+1
              k ← 0
              i ← 0
              if (m > P.length-1) then
                    break
              endif
         else
               i ← i+1
          endif
     else
          k ← k+1
          i ← 0
          if (k > P[m].length-1) then
               m ← m+1
               k ← 0
               i ← 0
              if (m > P.length-1) then
                    break
              endif

          endif
      endif
endwhile
```

Gambar 3.3 Pseudocode metode *Brute Force* untuk pencarian pada pola horizontal

b) Arah Vertikal:

1. Buat peubah *i, j, k,* dan *m*. Inisiasikan semuanya dengan nilai 0.
2. Jika belum ada, buat sebuah *array* asosiatif untuk menampung indeks hasil, *H*. Dimana masing-masing elemen *H* adalah peubah *array* yang merepresentasikan arah hasil pencarian, yaitu: "horizontal", "vertikal", "diagonal_turun", dan "diagonal_naik".
3. Bandingkan kecocokan antara array *P* ke-[*k*+*i*][*m*] dengan array *p* ke-*i* (*P*[*k*+*i*][*m*] = *p*[*i*] ?).
4. Apabila terjadi kecocokan:
   a. Tambahkan nilai *i* dengan 1.
   b. Apabila nilai *i* belum sama dengan jumlah array *p-1* dan nilai *k*+*i* belum melebihi jumlah baris pada array *P* - 1, ulangi langkah 3.
   c. Apabila nilai *i* sama dengan jumlah array *p* dan nilai *k*+*i* belum melebihi jumlah baris pada array *P* - 1, maka pencarian dikatakan berhasil. Simpan nilai variabel *k* dan *m* ke dalam array *H* indeks ke-*j*. Kemudian tambahkan variabel *j* dengan 1. Tambahkan nilai *k* dengan 1. Nilai *i* menjadi 0. Kemudian ulangi langkah 3.
   d. Apabila nilai *k*+*i* melebihi jumlah baris pada array *P* - 1, maka pencarian di kolom *m* dikatakan gagal. Hentikan pencarian. Tambahkan nilai *m* dengan 1. Nilai *k* dan *i* masing-masing menjadi 0.
   e. Apabila nilai *m* belum melebihi jumlah baris pada *array P* – 1, ulangi langkah 3.
   f. Apabila nilai *m* melebihi jumlah baris pada *array P* – 1, maka pencarian secara keseluruhan dikatakan gagal.
5. Apabila terjadi ketidakcocokan:
   a. Tambahkan nilai *k* dengan 1. Nilai *i* menjadi 0.
   b. Apabila nilai *k* belum melebihi jumlah baris pada array *P*-1, ulangi langkah 3.
   c. Apabila nilai *k* melebihi jumlah baris pada array *P*-1, maka pencarian di kolom *m* dikatakan gagal. Hentikan pencarian. Tambahkan nilai *m* dengan 1. Nilai *k* dan *i* masing-masing menjadi 0.

d. Apabila nilai *m* belum melebihi jumlah kolom pada *array P* – 1, ulangi langkah 3.
e. Apabila nilai *m* melebihi jumlah kolom pada *array P* – 1, maka pencarian secara keseluruhan dikatakan gagal.

Berikut ini merupakan *pseudocode* dari pencarian vertikal dengan metode *brute force* dengan langkah-langkah yang telah dijabarkan sebelumnya.

```
Deklarasi
i, j, k, m : integer
H : Array

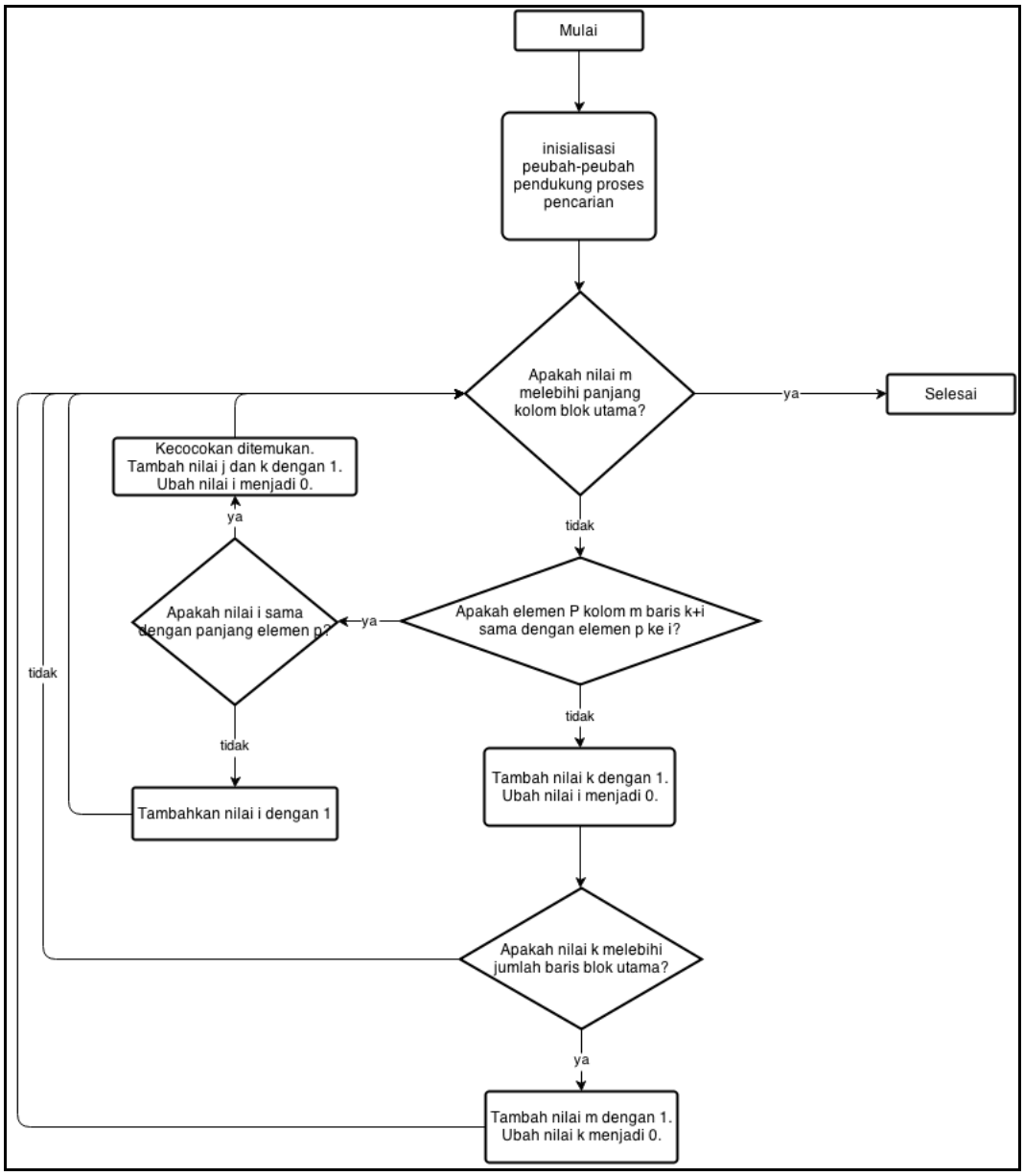
Algoritma:
I, j, k, m ← 0
while (m ≤ P[m].length-1) do
     if (P[k+i][m] = p[i]) then
          if (i = p.length-1) and (k+i ≤ P.length-1) then
               H[j][0] ← m
               H[j][1] ← k
               j ← j+1
               k ← k+1
               i ← 0
          else if (k+i > P.length-1) then
              m ← m+1
              k ← 0
              i ← 0
              if (m > P[m].length-1) then
                    break
              endif
         else
              i ← i+1
         endif
     else
          k ← k+1
          i ← 0
          if (k > P.length-1) then
               m ← m+1
               k ← 0
               i ← 0
              if (m > P[m].length-1) then
                    break
              endif

          endif
      endif
endwhile
```

Gambar 3.4 Pseudocode metode *Brute Force* untuk pencarian pada pola vertikal

Kompleksitas waktu dari algoritma *brute force* di atas adalah: **O((nm-T_{m-1})r)** Dimana *n* adalah jumlah masukan dari pola utama untuk satu baris, *m* adalah jumlah masukan pengguna, T_{m-1} adalah fungsi triangular untuk setiap *m-1*, dan *r* adalah jumlah baris dari susunan pola utama.

Berikut adalah *flowchart* dari algoritma *Brute Force* pada *game*:



Gambar 3.5 Flowchart dari algoritma *Brute Force* pada *game*

Menggunakan contoh susunan blok warna sebelumnya, algoritma *brute force* akan mencoba mencari solusi pencarian. Disini, warna dari masing-masing blok warna yang telah dicontohkan akan diwakilkan dengan inisial warna. Berikut adalah inisial warna dan artinya:

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

Tabel 3.1 Array P (Susunan Blok Utama)

Index	0	1	2	3	4	5	6	7	8
P	mr	pt	kn	br	br	ht	hj	kn	un
Index	9	10	11	12	13	14	15	16	17
P	or	mr	ht	hj	pn	un	pn	or	hj
Index	18	19	20	21	22	23	24	25	26
P	un	pn	or	un	hj	mr	or	br	kn
Index	27	28	29	30	31	32	33	34	35
P	mr	kn	br	un	mr	un	pt	ht	pt

Tabel 3.2 Array p (Susunan masukan pengguna)

Index	0	1	2	3	4
p	ht	hj	pn	un	pn

Hasil akhir pencarian didapat satu buah kecocokan (j = 1) dengan index P ke-11 sebagai index awal dimana kecocokan terjadi (H[0] = 11). Total pencarian

membutuhkan 46 kali iterasi dari total kompleksitas waktu O(36x5) atau O(180) atau dari iterasi maksimal 180 kali dalam kasus terburuk.

3.1.1.2. Analisis Pemecahan Masalah dengan Algoritma Knuth-Morris-Pratt

Pada algoritma KMP, kita menyimpan informasi yang digunakan untuk melakukan pergeseran lebih jauh, tidak hanya satu karakter seperti algoritma *Brute Force*. Algoritma ini melakukan pencocokan dari kiri ke kanan.

Sebelum melakukan proses pencarian dengan menggunakan metode KMP, kita akan melakukan perhitungan pra-pencarian untuk mencari *overlap* dalam pola yang dicari. *Overlap* dalam hal ini adalah urutan kata/pola yang mengulang di dalam satu pola yang dicari sehingga kita dapat melakukan pergeseran lebih jauh pada tahap pencarian.

Berikut algoritma yang dilakukan dalam mencari *overlap* dari pola m:

```
Deklarasi
K, v : integer
c : String
overlap : Array
input: Pola P sepanjang m
Algoritma:
overlap[0] = 0
for k:=0 to m-1 do
      c:=P[k+1]
      v:=overlap[k]
      while P[v] != c AND  v != 0 do
            v:=overlap[v]
      end while
      if P[v] = c then
            overlap[k+1]:=v+1
      else
            overlap[k+1]:=0
      end if
end for
```

Gambar 3.6 Pseudocode penacrian *overlap*

Berikut adalah *flowchat* dari proses pencarian *overlap* berdasarkan algoritma diatas:



Gambar 3.7 Flowchart tahap pencarian *overlap* dari algoritma KMP pada *game*

Dengan menggunakan contoh pola yang berbeda dengan penyelesaian kasus secara *Brute Force*, berikut adalah hasil *overlap* dari sebuah array p dengan menggunakan algoritma diatas.

Tabel 3.3 Hasil pencarian *overlap* pada tabel p

Index	0	1	2	3	4
p	ht	hj	Pn	Un	Pn
Overlap	0	0	0	0	0

Setelah selesai mencari milai-nilai *overlap*, sekarang kita akan mencari kecocokan pola p pada pola utama P berikut:

Tabel 3.4. Tabel P

Index	0	1	2	3	4	5	6	7	8
P	mr	pt	kn	br	br	ht	hj	kn	un
Index	9	10	11	12	13	14	15	16	17
P	or	mr	ht	hj	pn	un	pn	or	hj
Index	18	19	20	21	22	23	24	25	26
P	un	pn	or	un	hj	mr	or	br	kn
Index	27	28	29	30	31	32	33	34	35
P	mr	kn	br	un	mr	un	pt	ht	pt

Berdasarkan informasi nilai overlap yang sebelumnya dicari, maka proses pencarian dilakukan. Berikut adalah pseudocode untuk pencarian pola dengan menggunakan algoritma Knuth-Morris-Pratt untuk urutan pola horizontal (dari kiri ke kanan):

```
Deklarasi
i, j, k, n, m: integer

Algoritma:
i:=0, j:=0, k:=0, n:=36, m:=8
while n-k ≥ m do
      while j ≤ m AND P[i] = p[j] do
            i++, j++
      end while
      if j > m then output k
      if overlap[j-1] > 0 then
            k:=i-overlap[j-1]
      else
            if i = k then
                  i++
            end if
            k:=i
      end if
      if j>1 then
            j:=overlap[j-1]+1
      end if
      end while
```

Gambar 3.8 *Pseudocode string matching* secara horizontal dengan metode *Knuth-Morris-Pratt*

Kompleksitas waktu dari algoritma *Knuth-Morris-Pratt* di atas adalah **O((n+m)r).** Dimana *n* adalah jumlah masukan dari pola utama untuk satu baris, *m* adalah jumlah masukan dan *r* adalah jumlah baris dari susunan pola utama.

Hasil iterasi dari pencarian pola dengan menggunakan metode KMP Pada iterasi ke-18 dengan menggunakan KMP, kecocokan ditemukan.

3.2. 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 mengenai perangkat keras (*hardware),* perangkat lunak (*software),* dan pengguna (*user*) sebagai bahan analisis kekurangan dan kebutuhan yang harus dipenuhi dalam perancangan.

3.2.1. Analisis Kebutuhan Perangkat Keras

Perangkat keras yang dapat digunakan untuk menjalankan program puzzle game ini adalah dengan spesifikasi minimal sebagai berikut:

Tabel 3.5. Tabel Spesifikasi Kebutuhan Perangkat Keras

Nama Perangkat	**Spesifikasi**
Processor	1.0Ghz single core
Harddisk Drive	4Gb free
RAM	512 Mb

3.2.2. Analisis Kebutuhan Perangkat Lunak

Beberapa perangkat lunak yang dibutuhkan dalam menjalankan aplikasi puzzle game ini adalah sebagai berikut:

Tabel 3.6. Tabel Spesifikasi Kebutuhan Perangkat Lunak

Nama Perangkat Lunak	Spesifikasi
Web Browser	Chrome, Firefox, Safari, Internet Explorer versi terbaru
Framework	Enchant.js
Sistem Operasi	Windows, Mac, Linux

3.2.3. Analisis Pengguna (*user)*

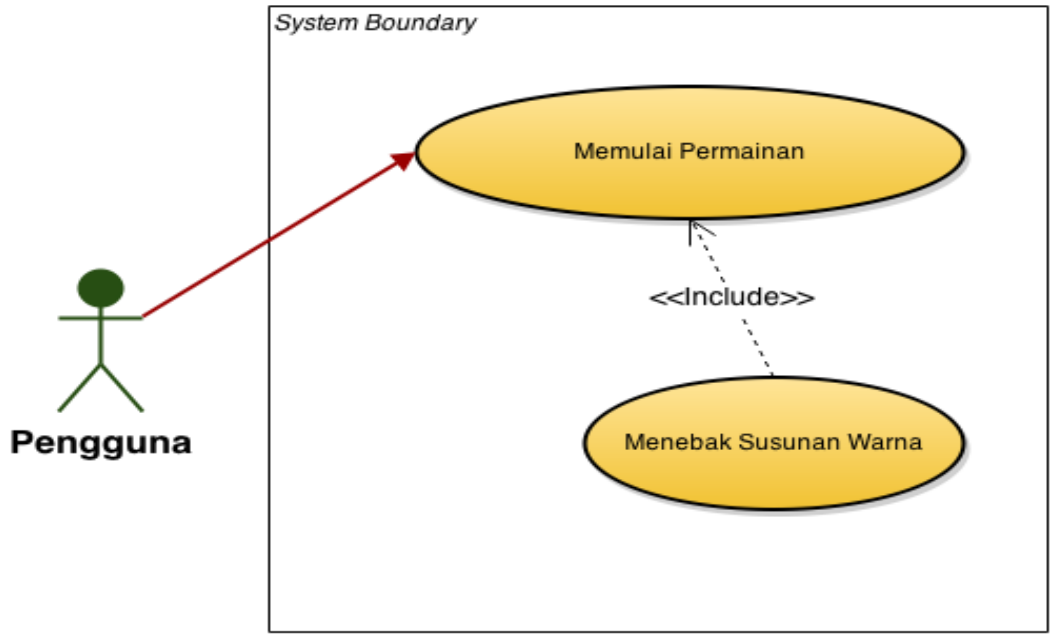
Aplikasi ini tidak membedakan level pengguna, sehingga semua pengguna dapat menggunakan fasilitas yang sama, dengan asumsi telah memiliki kemampuan dalam menggunakan dan berinteraksi dengan komputer personal atau laptop.

3.3. Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional akan membahas perancangan sistem di dalam aplikasi. Perancangan sistem adalah penggambaran, perancangan dan pembuatan sketsa. Pada tahap ini, tools / alat bantu yang digunakan untuk menganalisis dan merancang sistem adalah UML (Unified Modeling Language).

3.3.1. *Use Case Diagram*

Use case diagram digunakan untuk memodelkan proses bisnis dan merepresentasikan sebuah interaksi antara aktor dengan sistem. Aktor disini adalah pengguna, yang mana pengguna dapat memulai permainan dan menebak susunan warna.



Gambar 3.9 *Use Case* pada program *game*

3.3.1.1. Skenario *Use Case*

Skenario setiap bagian pada *use case* menunjukkan proses apa yang terjadi pada setiap bagian didalam *use case* tersebut, dimana pengguna memberikan perintah pada setiap bagian dan respon apa yang diberikan oleh sistem kepada pengguna setelah pengguna memberikan perintah pada setiap bagian-bagian *use case*.

Tabel 3.7. Skenario *Use Case* Memulai Permainan

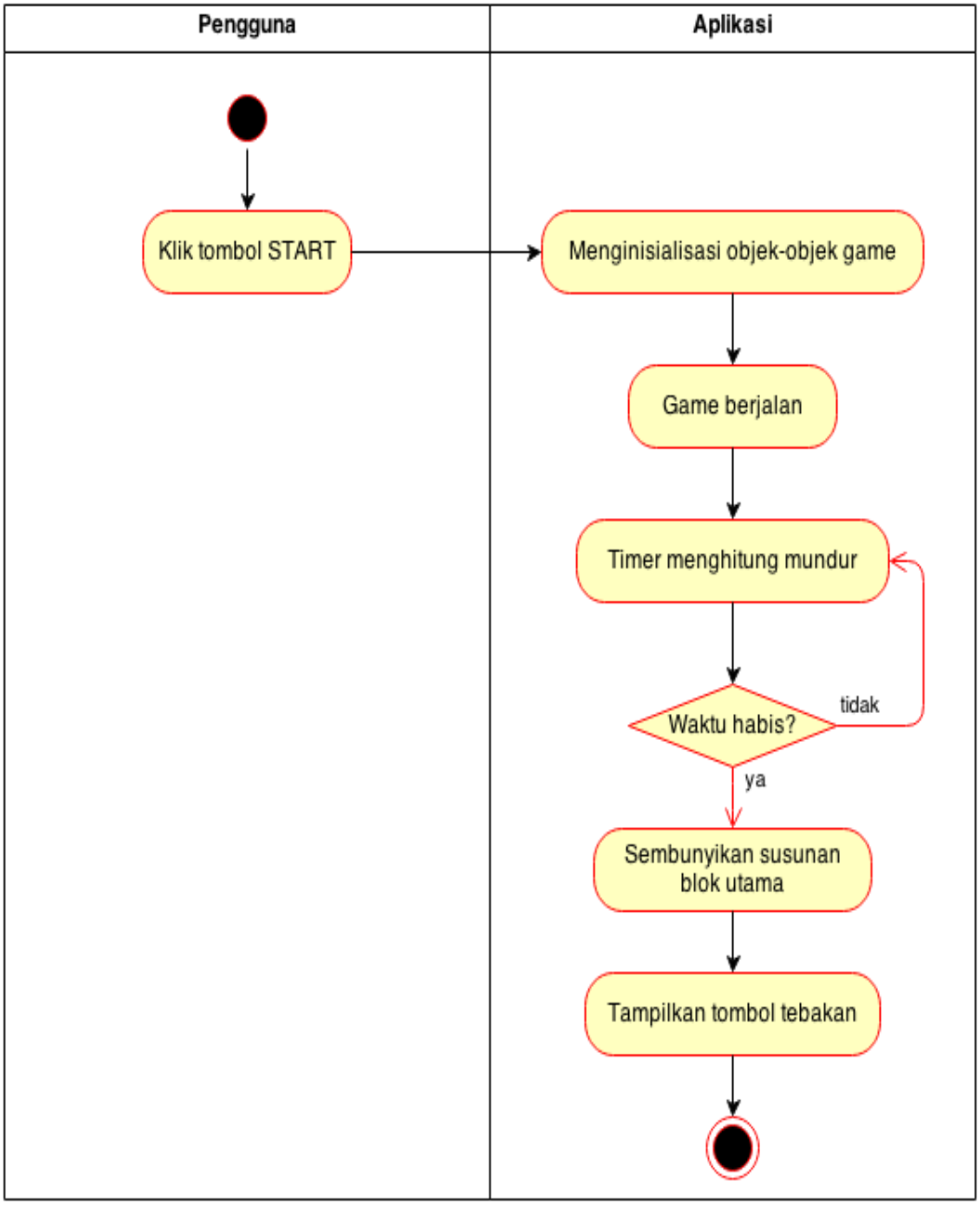
Identifikasi		
Nama	Memulai Permainan	
Tujuan	Menginstruksikan program untuk menjalankan permainan	
Deskripsi	Proses inisiasi dimulainya permainan	
Aktor	Pengguna	
Skenario Utama		
Kondisi Awal	Pengguna berada di menu utama pada program	
Aksi actor		Reaksi Sistem
1. Menekan tombol “START”		2. Menginstansiasi objek-objek yang diperlukan dalam permainan
Kondisi Akhir	Objek-objek permainan terinstansiasi dan permainan dimulai	

Tabel 3.8. Skenario *Use Case* Menebak Susunan Warna

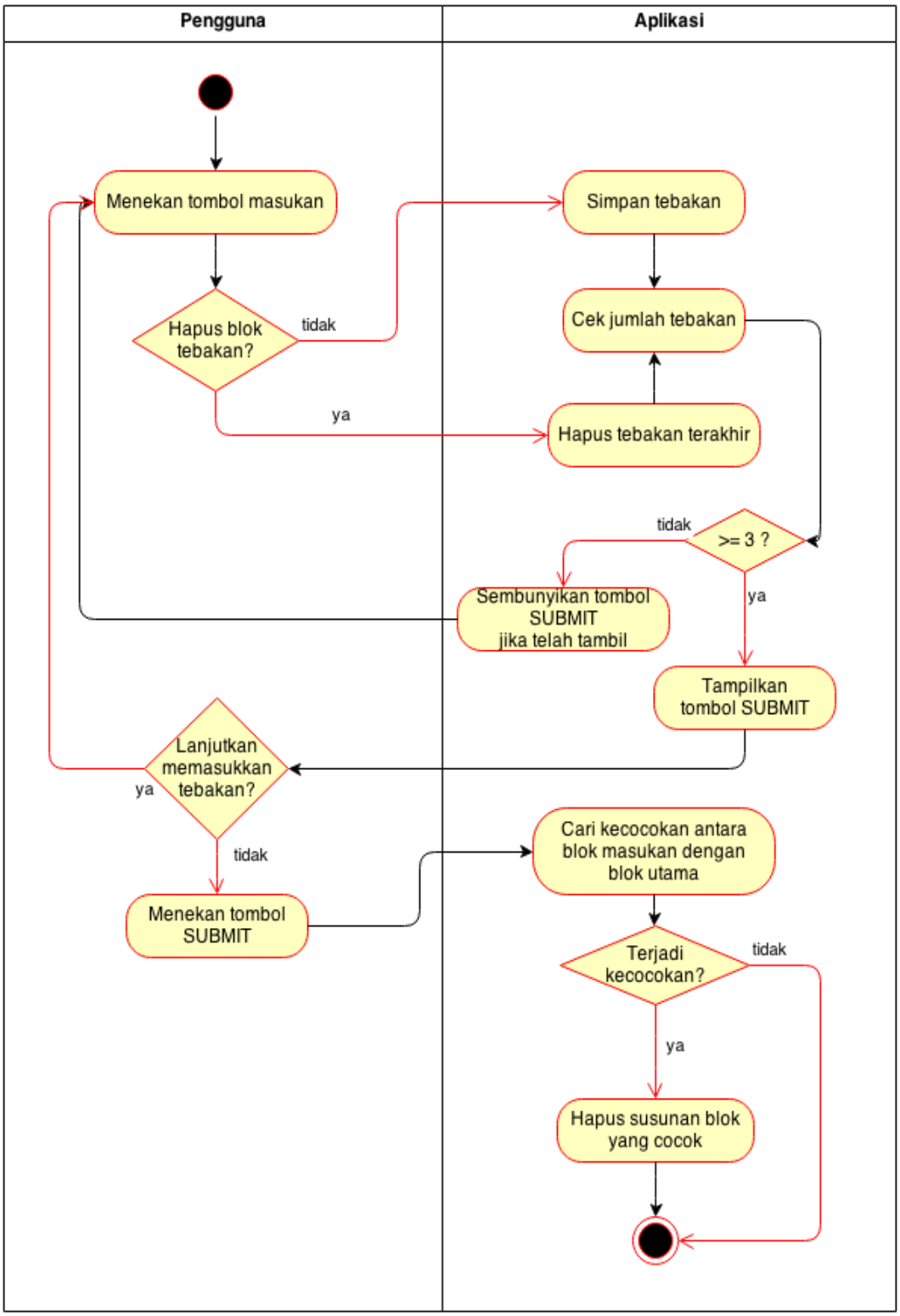
Identifikasi		
Nama	Menebak Susunan Warna	
Tujuan	Mencari kecocokan susunan warna	
Deskripsi	Proses pengguna memasukkan tebakan susunan warna	
Aktor	Pengguna	
Skenario Utama		
Kondisi Awal	Aplikasi menampilkan tombol masukan kepada pengguna	
Aksi actor		Reaksi Sistem
1. Memasukkan masukan tebakan		2. Menyimpan masukan pengguna
3. Menekan tombol SUBMIT		4. Mencari kecocokan masukan pengguna dengan susunan utama
Kondisi Akhir	Tampilan kembali menampilkan susunan blok utama dan menghapus susuan blok yang cocok dengan masukan pengguna	

3.3.2. Activity Diagram

Activity Diagram mendeskripsikan alur aktifitas dari setiap use case. Activity diagram pada program ini ditunjukkan pada gambar-gambar berikut.



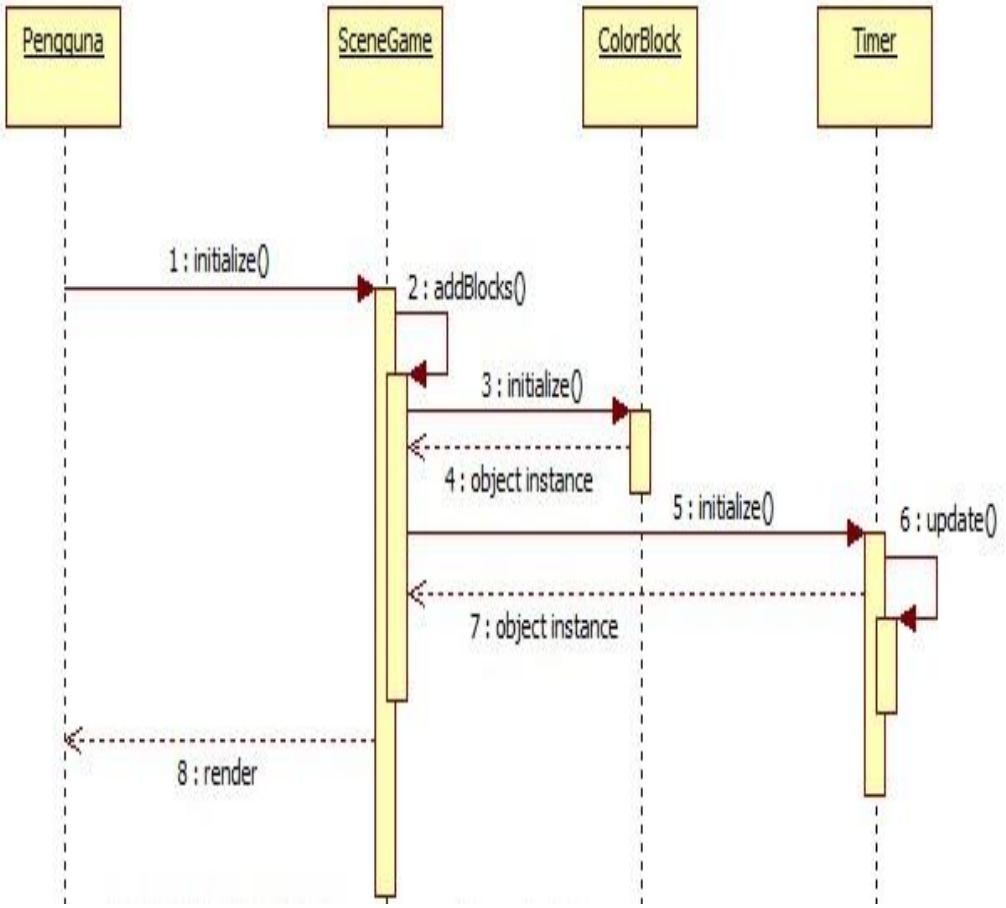
Gambar 3.10 *Activity Diagram* Memulai Permainan



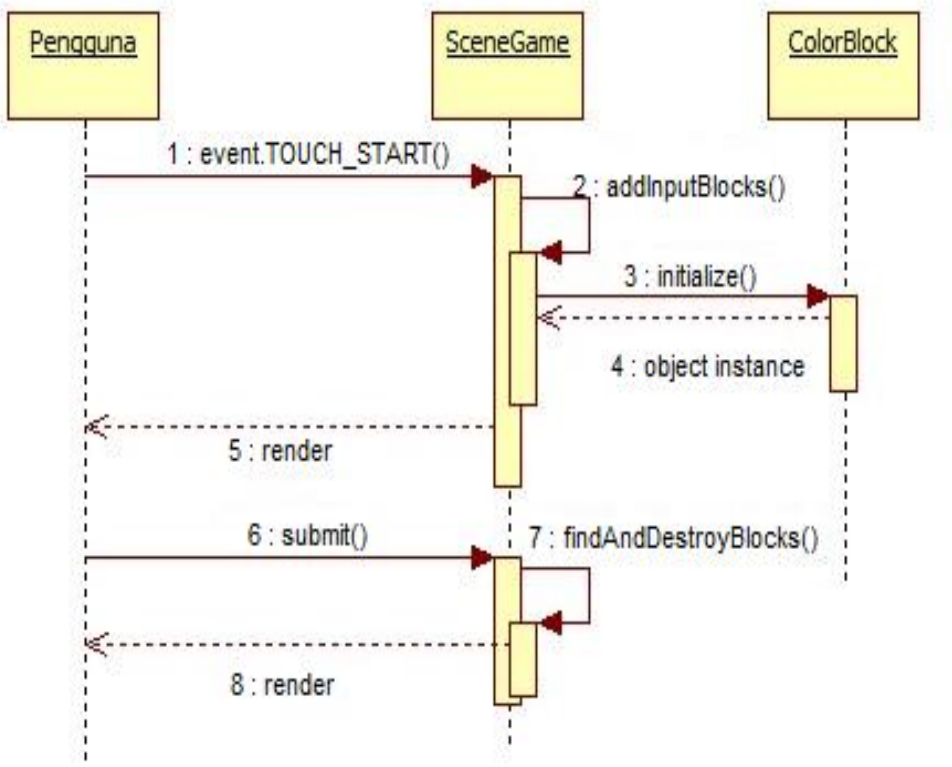
Gambar 3.11 *Activity Diagram* Menebak Susunan Warna

3.3.3. 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.



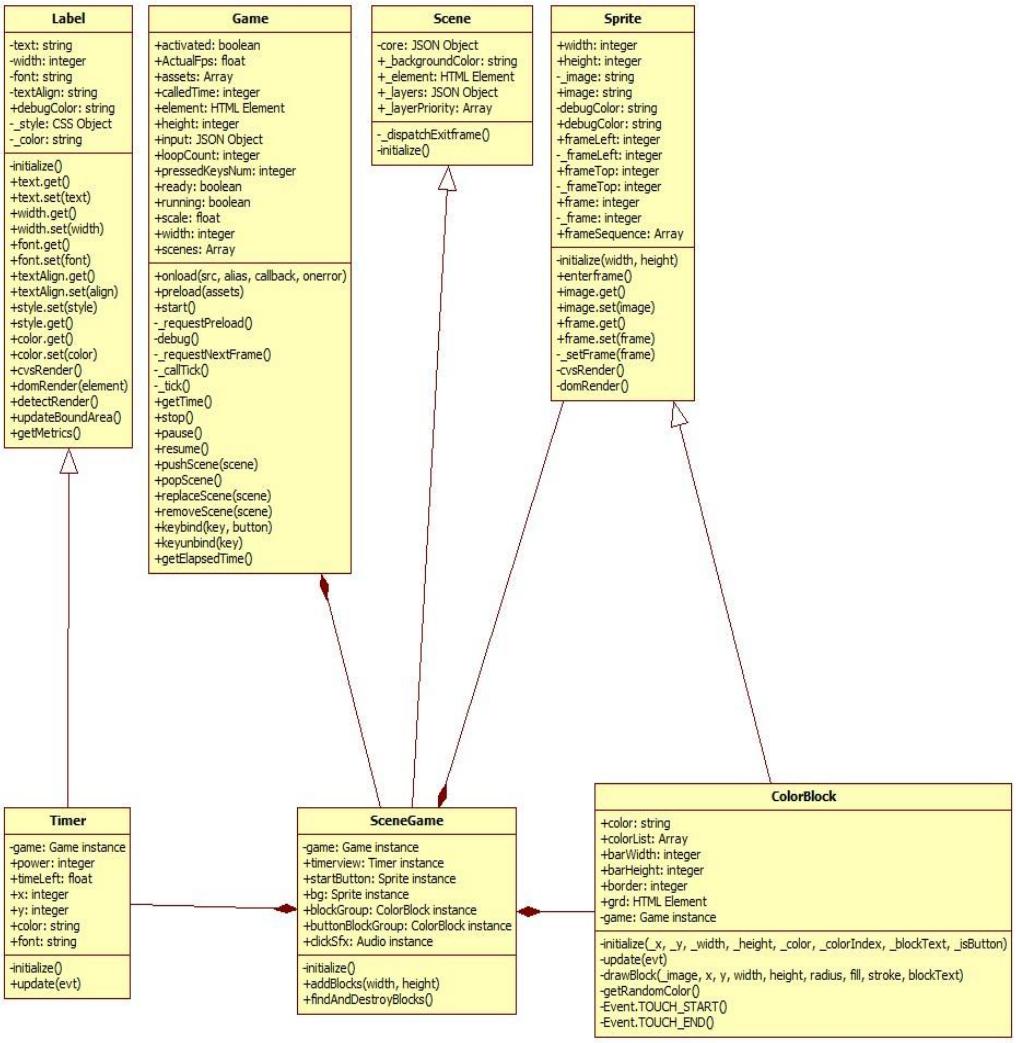
Gambar 3.12 *Sequence Diagram* Memulai Permainan



Gambar 3.13 *Sequence Diagram* Menebak Susunan Warna

3.3.2. 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 3.14 *Class Diagram*

3.4. Perancangan Sistem

Perancangan sistem merupakan kegiatan untuk merancang aplikasi yang akan di bangun. tahapannya dimulai dari perancangan komponen Permainan, perancangan antarmuka, dan jaringan semantik.

3.4.1. Perancangan komponen Permainan

Perancangan komponen pada permainan puzzle ini terdiri dari pengguna, scene, block, dan timer yang akan dijelaskan sebagai berikut :

1. Pengguna

   Pada permainan ini penguna berinteraksi dengan blok-blok khusus untuk menginput tebakan warna dari susunan blok utama yang dilihatnya.

2. Scene

   Scene merupakan sebidang area di dalam jendela browser di mana program game ini akan berjalan.

3. Block

   Block merupakan komponen kunci dari program game puzzle ini. Block memiliki properti warna yang disesuaikan dengan fungsinya. Seperti block dalam susunan blok warna utama, dimana properti warnanya acak. Ada juga block khusus dimana pengguna dapat menggunakan block tersebut sebagai tombol inputan, tombol submit, pause, dan quit.

4. Timer

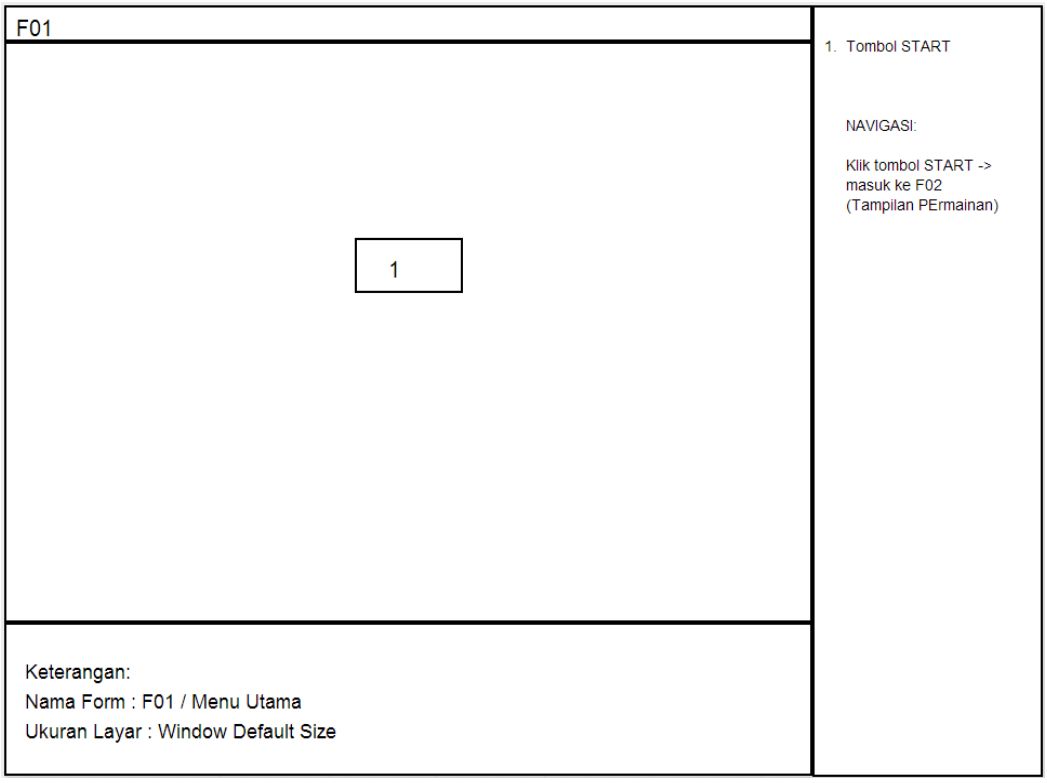
   Timer merupakan objek di dalam game yang berfungsi mengontrol berapa lama susunan blok warna utama akan tampil. Objek ini juga digunakan untuk menghitung skor akhir.

3.4.2. Perancangan Antarmuka

Perancangan antarmuka bertujuan untuk memberikan gambaran visual tentang aplikasi yang akan dibangun, sehingga akan mempermudah dalam mengimplementasikan aplikasi serta akan memudahkan pembuatan aplikasi.

3.4.2.1. Perancangan Tampilan Awal

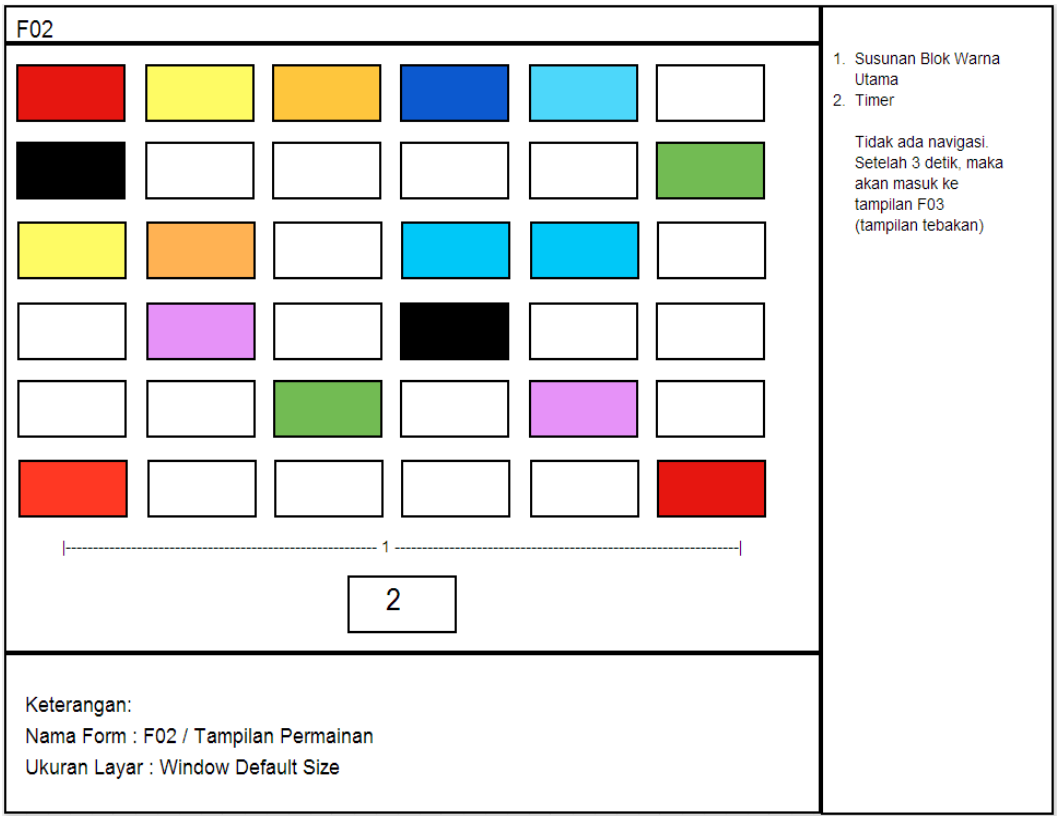
Tampilan awal adalah tampilan yang pertama kali muncul ketika pengguna menjalankan program. Terdapat dua buah kotak masukan dan satu buah buah tombol START.



Gambar 3.15 Perancangan Tampilan Awal

3.4.2.2. Perancangan Tampilan Permainan

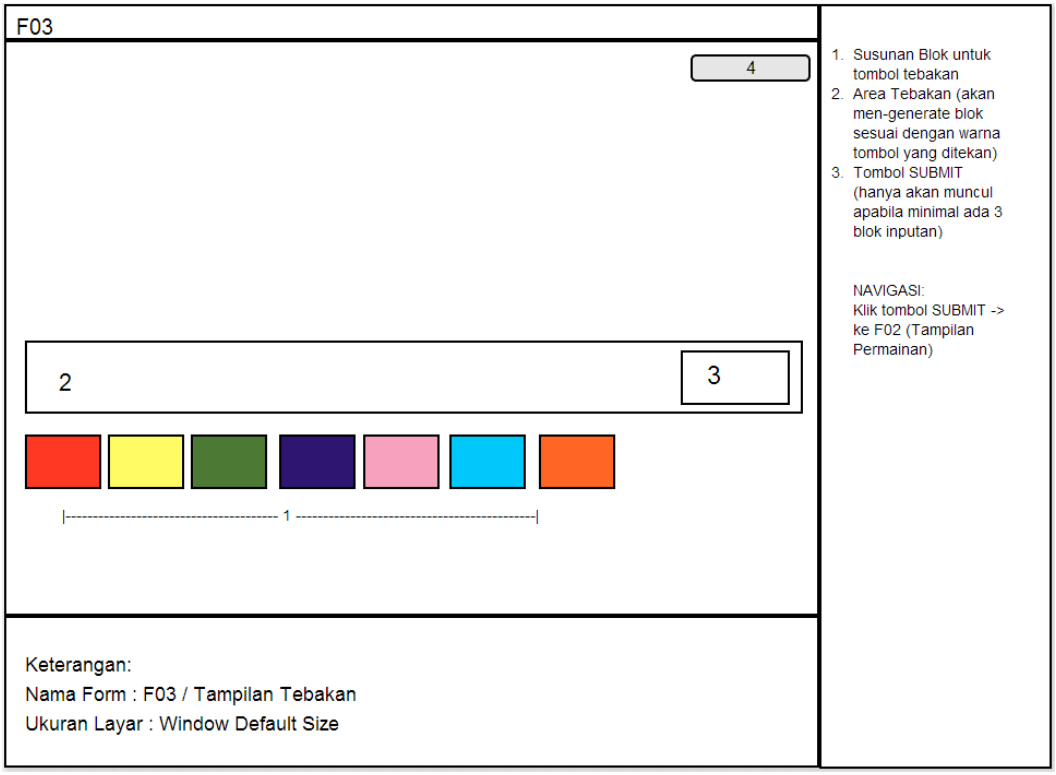
Tampilan permainan ini muncul ketika pengguna menekan tombol START. Susunan blok akan muncul secara otomatis dan timer mulai berjalan.



Gambar 3.16 Perancangan Tampilan Permainan

3.4.2.3. Perancangan Tampilan Tebakan

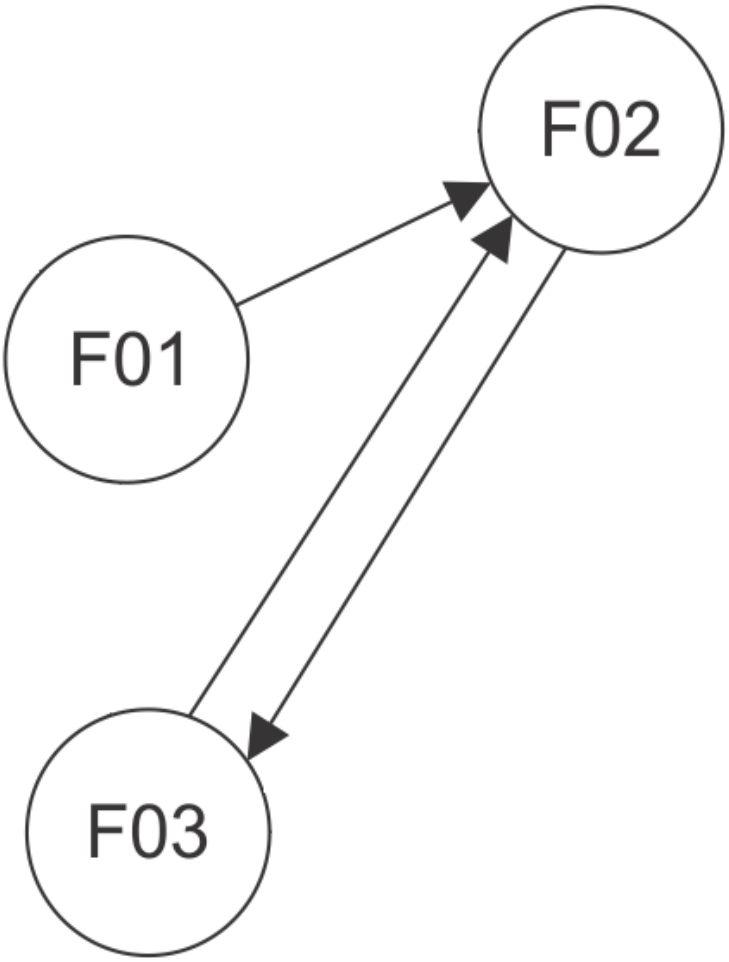
Ketika selang beberapa waktu setelah tampilan permainan muncul, tampilan tebakan ini akan ditampilkan.



Gambar 3.17 Perancangan Tampilan Tebakan

3.4.3. Jaringan Semantik

Jaringan simantik menggambarkan keterhubungan navigasi menu dari satu halaman ke halaman lainnya.



Gambar 3.18 Jaringan Semantik