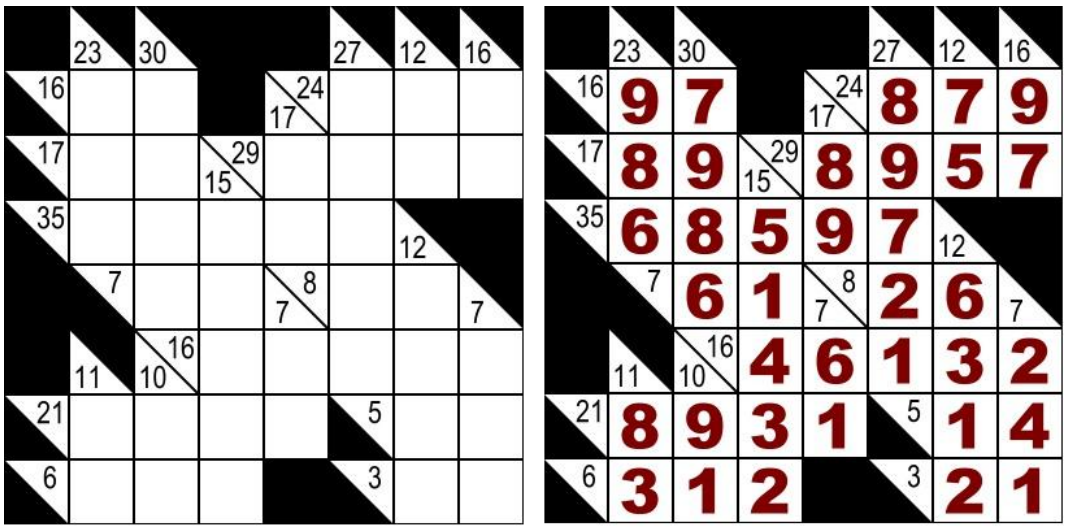
BAB III
ANALISIS DAN PERANCANGAN SISTEM

Pada bab ini akan dibahas mengenai analisis dan perancangan pada sistem yang dibangun yaitu implementasi algoritma *Harmony search* (HS) pada permainan *puzzle* angka.

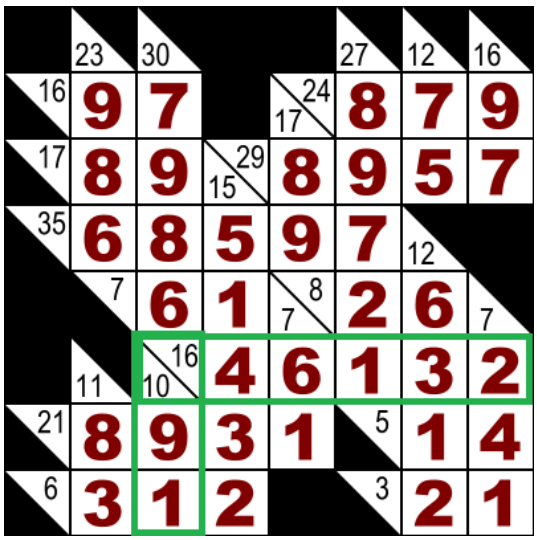
3.1 Analisis Masalah

Kakuro adalah permainan *puzzle* yang bersifat logika dan biasa disebut sebagai transliterasi matematis dari *puzzle* silang. Permainan ini adalah permainan penjumlahan angka dengan aturan mainnya yaitu dengan mengisi setiap kotak yang tersedia (kotak jawaban) dengan angka bulat dari 1 hingga 9 dengan beberapa syarat dalam pengisiannya. Soal-soal dalam permainan *puzzle* kakuro bersifat unik, karena soal-soal ini akan saling terkait baik dengan soal lainnya maupun jawaban dalam menyelesaikan permainan *puzzle* kakuro. Untuk menyelesaikan permainan *puzzle* kakuro ini, jumlah total pada nilai jawaban harus bernilai sama dengan nilai soal dan tidak boleh terdapat dua angka yang sama pada satu lajur (kolom maupun baris). Gambar 3.1 memperlihatkan penyelesaian *puzzle* kakuro.



Gambar 3.1 contoh penyelesaian *puzzle* kakuro

Oleh karena itu dibutuhkan teknik penyelesaian solusi dalam menyelesaikan jawaban pada permainan *puzzle* kakuro dengan beberapa syarat didalamnya. Kelayakan solusi pada permainan *puzzle* kakuro didefinisikan dengan suatu nilai akan dinyatakan layak apabila tidak terdapat nilai angka yang sama pada satu lajur (baik baris maupun kolom) serta nilai total pada kotak jawaban harus bernilai sama dengan nilai pada kotak soal. Fungsi solusi didefiniskan jika semua kotak kosong (kotak jawaban) telah terisi dengan nilai yang sesuai (sesuai dengan kelayakan solusi). Berikut gambaran kelayakan solusi pada permainan *puzzle* kakuro :



Gambar 3.2 contoh kelayakan solusi *puzzle* kakuro

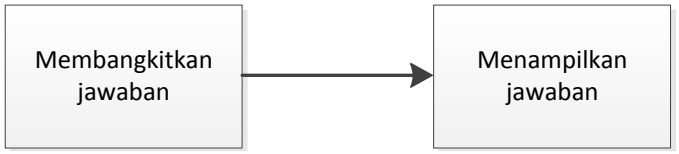
3.2 Analisis Game

Analisis *game* yang akan diteliti pada *game puzzle* kakuro adalah bagaimana cara menyelesaikan jawaban pada *puzzle* kakuro. Pada dasarnya permainan *puzzle* kakuro ini merupakan sebuah permainan yang menggunakan teknik komputasi dalam pencarian solusi salah satunya adalah dengan penerapan algoritma pencarian yaitu dengan memahami kemungkinan yang mungkin dari setiap solusi dari masalah yang disajikan. Aturan permainan dari *puzzle* kakuro sendiri mirip dengan sudoku. Ada beberapa algoritma pencarian yang sudah digunakan untuk menyelesaikan masalah yang disajikan pada sudoku.

Diantaranya adalah algoritma *harmony search*. Algoritma ini dalam beberapa penerapan pada berbagai problem menunjukan algoritma ini termasuk efektif dan mempunyai banyak kelebihan. Atas dasar itu dilakukan penelitian kemungkinan diterapkannya algoritma *harmony search* pada permainan *puzzle* kakuro. Pada *puzzle* kakuro ini algoritma *harmony search* akan digunakan sebagai penyelesai permainan. Berikut merupakan alur penyelesaian jawaban pada kotak jawaban:

1. Proses penyelesaian jawaban pada kotak jawaban kosong.

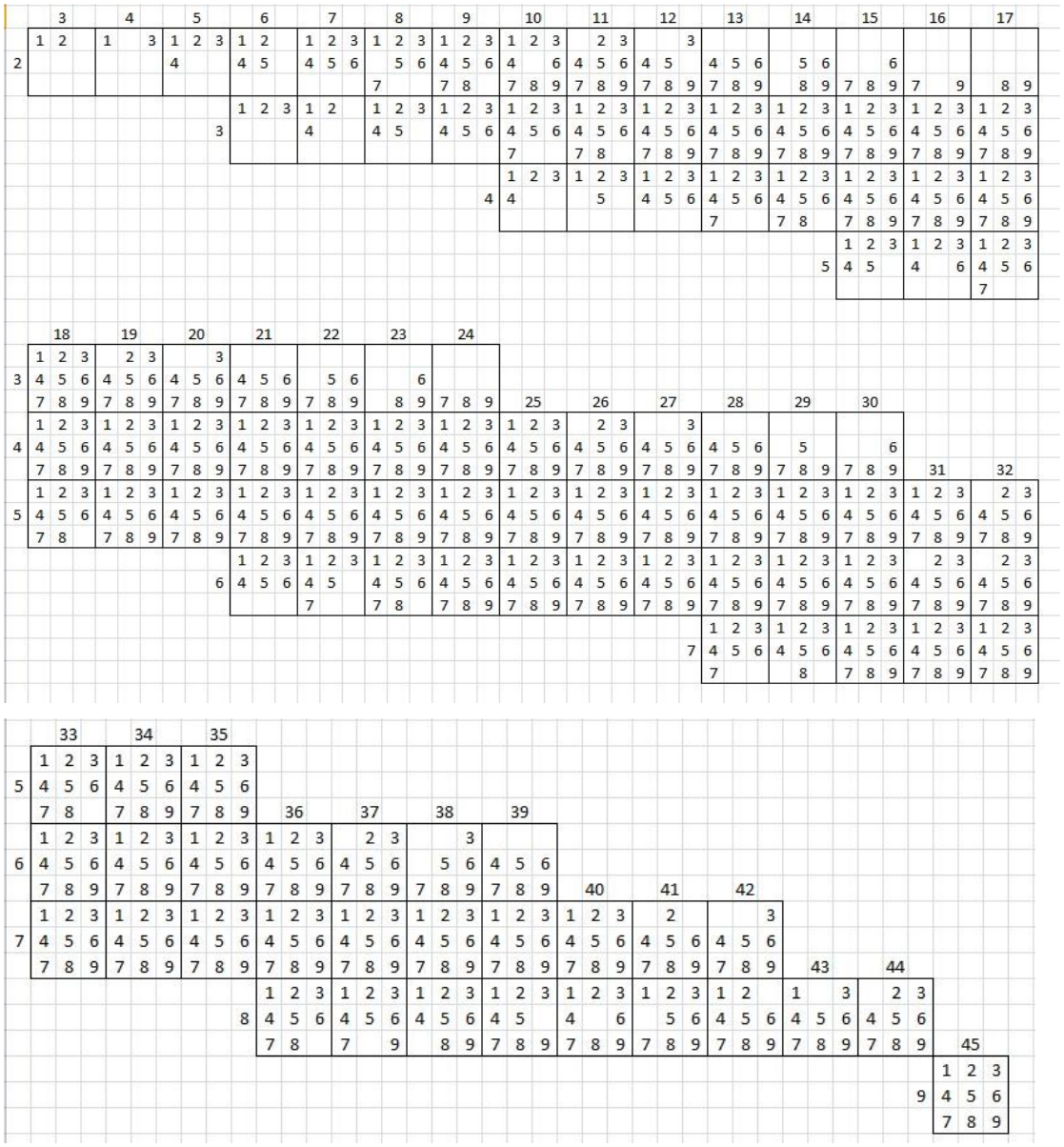
   Pada proses ini, penyelesaian jawaban dengan algoritma *harmony search* dengan sistem acak dalam pencarian nilai. Kemudian dilanjutkan dengan melakukan tahapan pengecekan nilai yaitu tidak boleh terdapat angka yang sama pada satu lajur serta total nilai jawaban pada satu lajur yang akan dibandingkan dengan nilai soal.



Gambar 3.3 Proses penyelesaian jawaban pada kotak kosong

3.3 Analisis Masukan

Analisis masukan yang dilakukan dalam penelitian ini adalah menganalisis inputan *puzzle* dalam algoritma *Harmony Search* pada *game* kakuro. Masukan tersebut berupa nilai (angka), akan tetapi masukan diberi batasan nilai yaitu dari angka 1 hingga 9, berikut merupakan gambar tabel analisis masukan pada *game puzzle* kakuro. Pada gambar 3.4 dapat dilihat angka 2 – 9 di sebelah kiri itu menunjukan banyaknya kotak jawaban yang kosong, sedangkan angka dari 3 – 45 di bagian atas menunjukan nilai penjumlahan atau nilai jumlah kotak soal.



Gambar 3.4 analisis masukan pada *game puzzle* kakuro

3.4 Analisis Metode / Algoritma

Analisis algoritma dilakukan untuk mengetahui optimalitas suatu algoritma yang dapat diukur dari kompleksitas ruang dan kompleksitas waktunya. Algoritma *Harmony Search* akan diterapkan untuk mencari solusi optimal pada *puzzle* kakuro. Terdapat beberapa parameter yang harus ditentukan sebelum

algoritma *Harmony Search* digunakan secara efektif, algoritma *Harmony Search* mengadopsi sebuah parameter yang disebut *Harmony memory* Considering rate (HMCR) dan *Pitch Adjusting Rate* (PAR). Tahapan-tahapan yang dilakukan dalam pencarian solusi dengan algoritma *harmony search* adalah sebagai berikut :

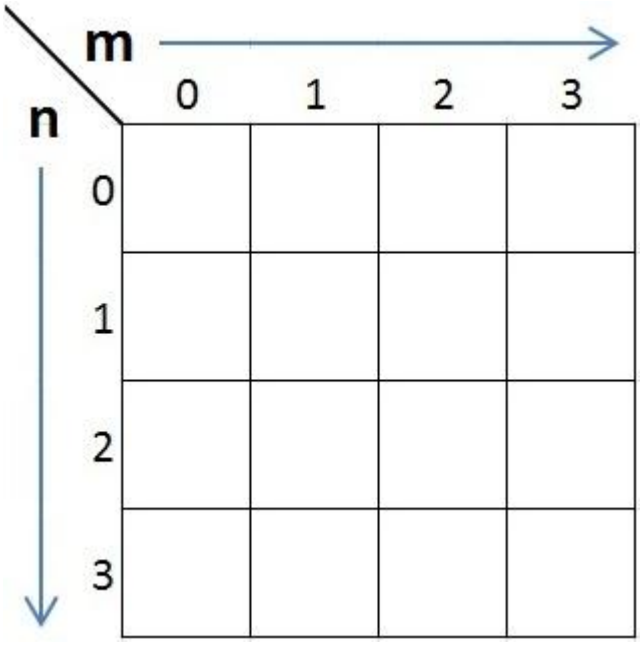
```
Input : fungsi tujuan, itmax
Output : x terbaik, nilai fungsi tujuan
%Tahap Inisialisasi
Tentukan HMCR (0.75 - 0.95)
Tentukan ukuran HMS %Ukuran sample misal 20
Tentukan pitch Adjusting rate (PAR = 0.3)
Tentukan pitch bandwitch b, x_L dan x_U
Bangkitkan  Harmony Memory secara random sebanyak HMS
while it < itmax do
    foreach i=1: jumlah variabel do
        Bangkitkan bilangan random r ≈ U (0,1)
        if r ≤ HMCR then
            Pilih satu nilai, xold_i , dari HM untuk variabel i
            Bangkitkan bilangan random r ≈ U (0,1)
            if r ≤ PAR then
                Lakukan pengaturan nilai xnew_i = xold_i ± b_i
                Jika xnew_i ≤ x_L maka xnew_i = x_L
                Jika xnew_i ≥ x_U maka xnew_i = x_U
            end
        else
            Bangkitkan vektor solusi baru xnew_i
        end
    end
    Bandingkan f (xnew) dengan f(xworst) di dalam HM
    Jika lebih baik, terima xnew dan simpan dalam HM
end
Temukan Solusi terbaik dan fungsi tujuan dari HM
```

Gambar 3.5 *pseudo code* algoritma *harmony search*

3.4.1 Analisis Algoritma *Harmony Search*

Sebelum meneliti cara kerja algoritma *Harmony Search* pada permainan *puzzle* kakuro ini, akan dibahas terlebih dahulu bagaimana cara menginisialisasikan kotak terlebih dahulu yang akan menghasilkan kotak kosong (kotak jawaban) dan kotak blok (kotak nilai soal / pembatas).

Dalam proses inisialisasi kotak pada *puzzle* kakuro, di setiap kotak *puzzle* kakuro akan direpresentasikan sebagai *array* dua dimensi, *P* dengan panjang kolom *n* dan panjang baris *m* yang akan di gambarkan sebagai berikut :

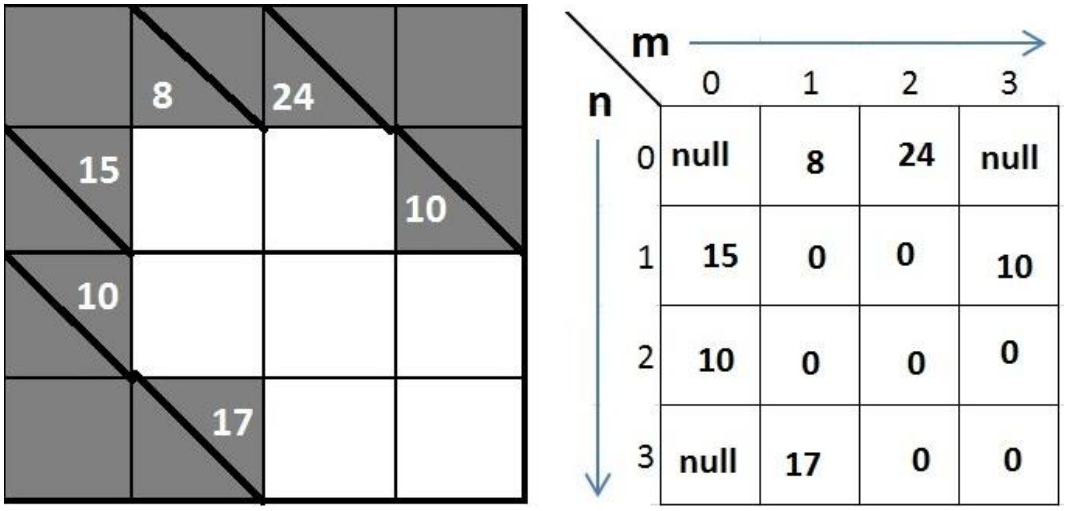


Gambar 3.6 kotak *puzzle* direpresentasikan sebagai *array P* dengan panjang m dan n

Proses inisialisasi 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]). Kemudian dilanjutkan dari atas ke bawah dengan posisi blok di pojok kiri atas sebagai awal pencarian. Dalam menentukan kotak blok dan kotak kosong dalam permainan *puzzle* kakuro ini, maka alur proses yang akan terjadi adalah sebagai berikut :

1. Apabila nilai indeks pada kotak yang dipilih terdapat nilai 0, maka kotak tersebut akan menjadi kotak kosong/kotak jawaban.
2. Apabila nilai indeks pada kotak yang dipilih terdapat nilai > 0, maka kotak tersebut akan menjadi kotak blok/kotak soal.
3. Sedangkan apabila nilai indeks pada kotak yang dipilih null, maka kotak tersebut akan menjadi kotak batasan.

Puzzle kakuro yang direpresentasikan sebagai array dapat dilihat pada gambar 3.7 berikut :

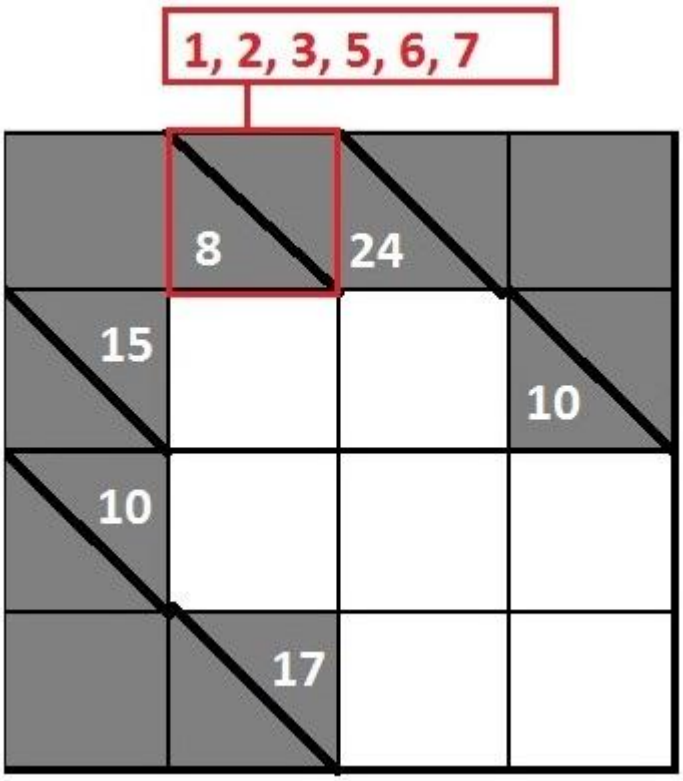


Gambar 3.7 Inisialisasi State *Puzzle* dan Struktur data

Analisis algoritma *harmony search* dalam menyelesaikan jawaban terjadi dalam kondisi disaat kotak jawaban kosong. Dalam penelitian ini, dilakukan pendekatan baru untuk menyelesaikan *puzzle* kakuro. Untuk keperluan penelitian ini, perlu dilakukan beberapa eliminasi dari semua kemungkinan kombinasi. Hal ini penting, sehingga pada saat pelaksanaan algoritma akan memberikan solusi terbaik dengan iterasi yang lebih sedikit yang diperlukan.

1. Inisialisasi Kombinasi

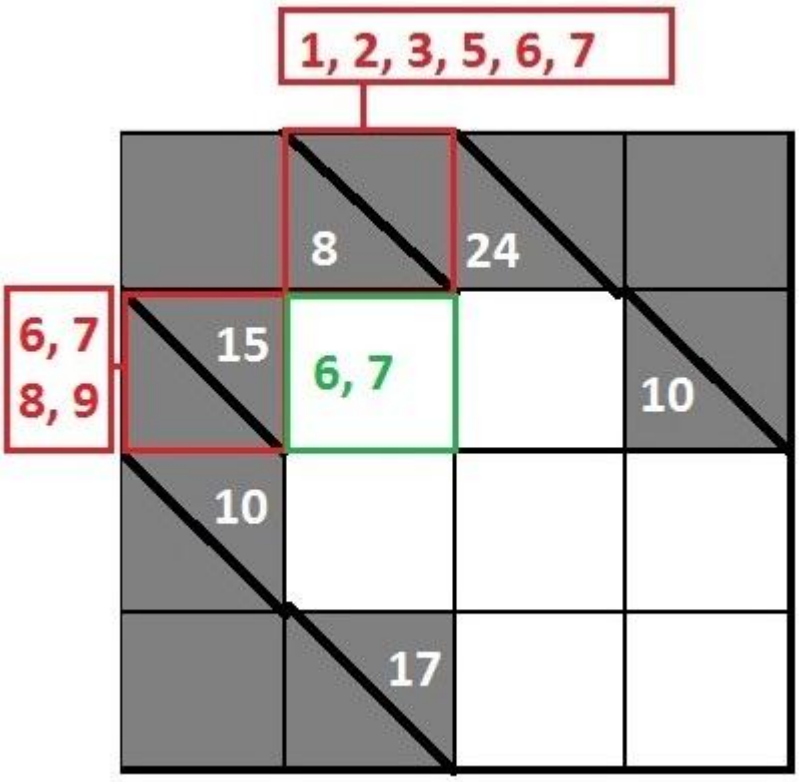
   Pertama-tama, semua kotak soal pada *puzzle* kakuro di cek, baik secara baris maupun kolom. Begitu juga dengan banyaknya kotak jawaban. Untuk setiap kemungkinan dari jumlah kombinasi yang dapat membentuk jumlah pada kotak soal akan dibangkitkan dan disimpan sebagai satu set angka untuk tahap perhitungan berikutnya. Misalnya pada contoh gambar 3.7, jika isi yang terdapat pada kotak soal adalah 8 (delapan), maka akan dicari kombinasi angka yang ketika dijumlahkan akan menghasilkan angka 8, diketahui bahwa hal itu dapat diperoleh dengan menggunakan kombinasi jumlah (2,6), (3,5), (1,7) sehingga kombinasi angka yang ditetapkan untuk nomor 8 adalah {1,2,3,5,6,7,}.



Gambar 3.8 kombinasi angka dari 8

2. Menghitung Kombinasi Silang

   Untuk semua kotak jawaban kosong dalam *puzzle* kakuro merupakan kotak persimpangan, setiap kombinasi set dari kotak persimpangan akan dihitung. Hal ini penting untuk mengurangi kombinasi yang dibutuhkan, sehingga mempercepat pencarian pada algoritma *harmony search*. Sebagai contoh, pada Gambar 3.8 dapat dilihat pada kotak hijau, itu merupakan kotak persimpangan yang dipengaruhi oleh kombinasi untuk kotak soal 8 (kolom) dan 15 (baris). Untuk kotak soal 8 kita memiliki kombinasi {1,2,3,5,6,7}, dan untuk nomor 15 kita memiliki kombinasi {6,9}{7,8}. Oleh karena itu, kotak persimpangan memiliki kombinasi {6,7}, kombinasi itu merupakan nilai yang mungkin pada kotak persimpangan.



Gambar 3.9 kotak persimpangan yang dipengaruhi oleh kombinasi dari angka 8 dan 15

3. Algoritma Harmony Search

Inisialisasi parameter

Pertama tetapkan beberapa nilai parameter berikut :

Harmony memory Consideration Rate (HMCR) = biasanya bernilai dari (0.7-0.9). Dalam contoh kasus ini dimisalkan HMCR=0.9

Pitch Adjusting Rate (PAR) = biasanya bernilai dari (0.1-0.3). Dalam contoh kasus ini dimisalkan PAR=0.1

Batas bawah x_l = 1

Batas atas x_u = 9

Fungsi Objektif untuk baris (Cazinave, 2010):

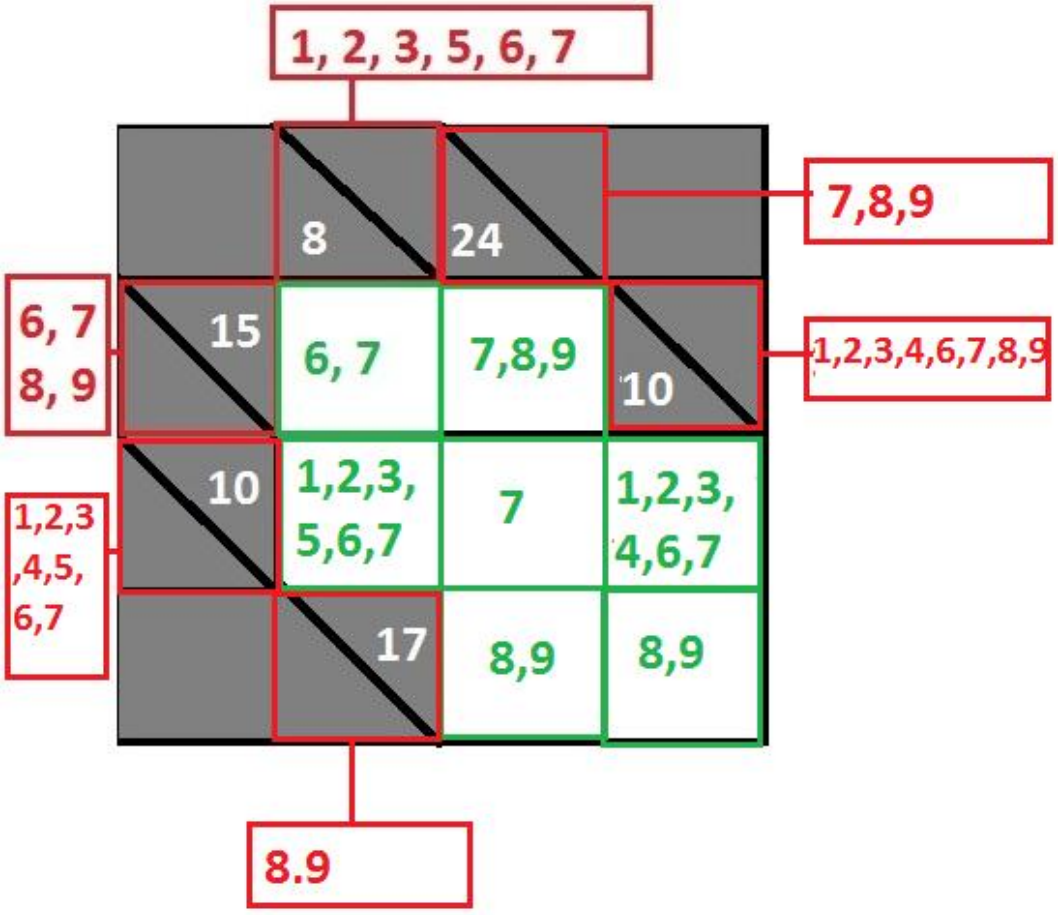
$$f = \sum_{h \in H} |TH(h) - CH(h)| \quad (2.4)$$

Fungsi Objektif untuk kolom :

$$f = \sum_{v \in V} |TV(v) - CV(v)| \quad (2.5)$$

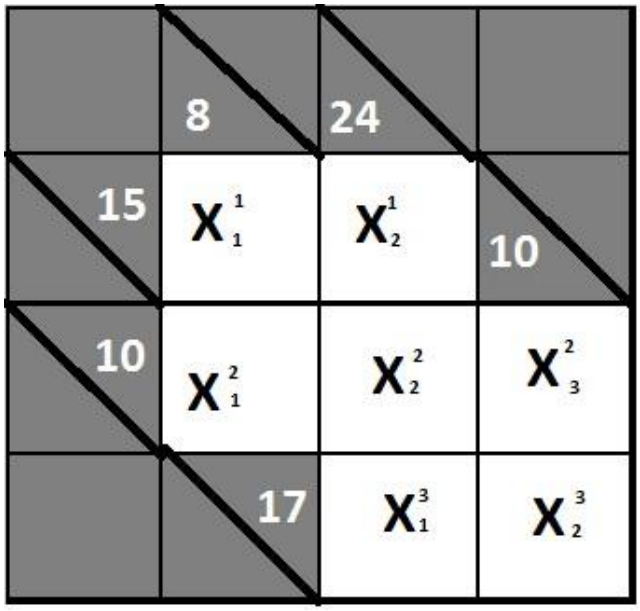
Inisialisasi *Harmony memory*

Masuk ke iterasi pertama, untuk mencari jawaban kotak soal, bangkitkan *harmony memory* dari setiap kombinasi yang didapat dari kotak persimpangan. Misalkan diambil contoh kasus seperti pada gambar 3.9 di bawah ini.



Gambar 3.10 kombinasi dari kotak persimpangan

Dari gambar diatas, dapat diketahui kandidat solusi yang mungkin untuk setiap kotak jawaban. Pada Gambar 3.10 dibawah ini dijelaskan pendeklarasian nama pada setiap kotak jawaban.



Gambar 3.11 pendeklarasian nama kotak jawaban

Data dari gambar 3.9 dan 3.10 dimasukan ke dalam tabel 3.1 dibawah ini

Tabel 3.1 Tabel *harmony memory*

Nama Kotak Jawaban	Kandidat Solusi / *Harmony memory*
x_1^1	6, 7
x_2^1	7, 8, 9
x_1^2	1, 2, 3, 5, 6, 7
x_2^2	7
x_3^2	1, 2, 3, 4, 6, 7
x_1^3	8,9
x_2^3	8,9

Iterasi pertama yaitu untuk mencari nilai dari kotak soal 15 menurut baris.

Improvisasi Harmoni Baru

Lakukan iterasi untuk x_1^1

Kemudian bangkitkan bilangan *random* r, bilangan *random* r diperoleh dari rand(0,1).

Misalkan r = 0.56, bandingkan r dengan HMCR

Jika r ≤ HMCR, ambil satu nilai secara acak dari HM

Jika r ≥ HMCR, ambil satu nilai dari rentang nilai x_l = 1 , x_u = 9.

Karena 0.56 ≤ 0.9, ambil satu nilai secara acak dari HM misal 7.

Kemudian bangkitkan kembali bilangan *random* r,

Misalkan r=0.3, bandingkan r dengan PAR

Jika r ≤ PAR, ambil satu nilai ketetanggaan dari x_1^1

Jika $x_1^1 \leq x_L$ maka $xnew_1^1 = x_L$

Jika $x_1^1 \geq x_U$ maka $xnew_1^1 = x_U$

Jika r ≥ PAR, maka nilai untuk x_1^1 dipertahankan.

Karena 0.3 ≥ 0.1, maka nilai x_1^1 tetap dipertahankan dengan nilai x_1^1 = 7.

Lakukan iterasi untuk x_2^1 .

Bangkitkan bilangan *random* r, Misalkan r= 0.7

0.7 ≤ 0.9, ambil satu nilai dari HM, misalkan x_2^1 = 8

Bangkitkan bilangan *random* r, Misalkan r= 0.5

0.5 ≥ 0.1, maka nilai x_2^1 tetap dipertahankan dengan nilai x_2^1 = 8

Perbaharui *Harmony memory*

Cek nilai x_1^1 dan nilai x_2^1. Dengan cara memasukkan nilai tersebut ke fungsi objektif baris.

$$f = \sum_{h \in H} |TH(h) - CH(h)|$$

TH(h) = 15

CH(h) = $x_1^1 + x_2^1$

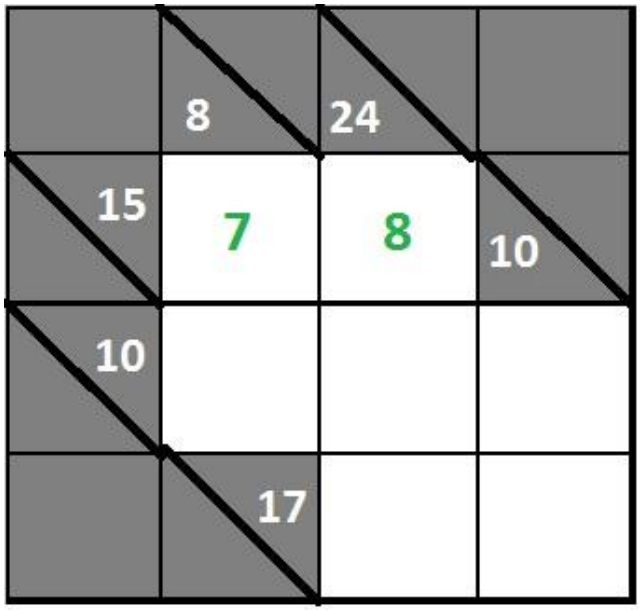
= 7 + 8

= 15

f = | 15 - 15 |

= 0

Karena Nilai yang didapatkan 0, Itu berarti nilai $(x_1^1 + x_2^1) =$ jumlah dalam kotak soal. Maka perbaharui *Harmony memory*. *Puzzle* kakuro yang sudah diperbaharui *harmony memory* nya dapat dilihat pada gambar 3.11 dibawah ini :



Gambar 3. 12 *Puzzle* kakuro yang sudah diperbaharui *harmony memory*

Masuk ke iterasi kedua, untuk mencari nilai 8 menurut kolom, bangkitkan *harmony memory*, *harmony memory* yang sudah diperbaharui dapat dilihat pada tabel 3.2 dibawah ini.

Tabel 3.2 Tabel *Harmony memory* yang sudah diperbaharui

Nama Kotak Jawaban	Kandidat Solusi / *Harmony memory*
x_1^1	7 (update)
x_2^1	8 (update)
x_1^2	1, 2, 3, 5, 6, 7
x_2^2	7
x_3^2	1, 2, 3, 4, 6, 7
x_1^3	8,9
x_2^3	8,9

Improvisasi Harmoni Baru

Lakukan iterasi untuk x_1^2, karena kotak jawaban x_1^1 sudah berisi nilai.

Bangkitkan bilangan *random* r,

Misalkan r = 0.86, bandingkan r dengan HMCR

$0.86 \leq 0.9$, ambil satu nilai secara acak dari HM misal nilai x_1^2 =1.

Bangkitkan bilangan *random* r, Misalkan r= 0.4

$0.4 \geq 0.1$, maka nilai x_1^2 tetap dipertahankan dengan nilai x_1^2 = 1

Perbaharui *Harmony memory*

Cek nilai x_1^1 dan nilai x_1^2. Dengan cara memasukkan nilai tersebut ke fungsi objektif kolom.

$$f = \sum_{v \in V} |TV(v) - CV(v)|$$

TV(v) = 8

CV(v) = $x_1^1 + x_1^2$

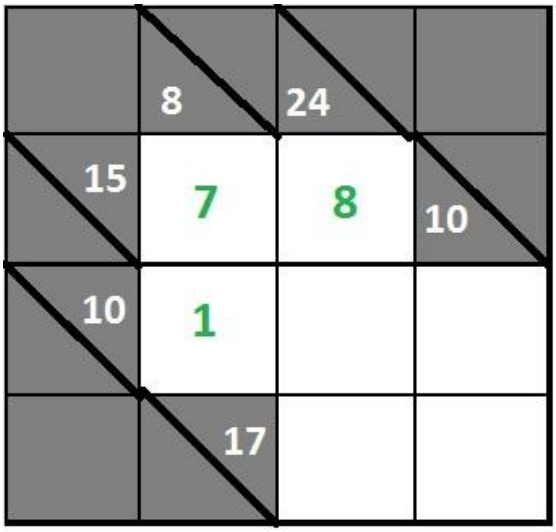
= 7 + 1

= 8

f = | 8 - 8 |

= 0

Karena Nilai yang didapatkan 0, Itu berarti nilai $(x_1^1 + x_1^2) =$ jumlah dalam kotak soal. Maka perbaharui *Harmony memory*. *Puzzle* kakuro yang sudah diperbaharui *harmony memory* nya dapat dilihat pada gambar 3.12 dibawah ini :



Gambar 3.13 *Puzzle* kakuro yang sudah diperbaharui

Masuk ke iterasi ketiga, untuk mencari nilai 10 menurut baris, bangkitkan *harmony memory*, *harmony memory* yang sudah diperbaharui dapat dilihat pada tabel 3.3 dibawah ini.

Tabel 3.3 Tabel *Harmony memory* yang sudah diperbaharui

Nama Kotak Jawaban	Kandidat Solusi / *Harmony memory*
x_1^1	7 (update)
x_2^1	8 (update)
x_1^2	1 (update)
x_2^2	7
x_3^2	1, 2, 3, 4, 6, 7
x_1^3	8,9
x_2^3	8,9

Improvisasi Harmoni Baru

Lakukan iterasi untuk x_3^2, karena kotak jawaban x_1^2 dan x_2^2 sudah berisi nilai.

Bangkitkan bilangan *random* r,

Misalkan r = 0.16, bandingkan r dengan HMCR

$0.16 \leq 0.9$, ambil satu nilai secara acak dari HM misal nilai x_3^2 =2.

Bangkitkan bilangan *random* r, Misalkan r= 0.7

$0.7 \geq 0.1$, maka nilai x_3^2 tetap dipertahankan dengan nilai x_3^2 = 2

Perbaharui *Harmony memory*

Cek nilai x_1^2 , x_2^2 dan nilai x_3^2. Dengan cara memasukkan nilai tersebut ke fungsi objektif baris.

$$f = \sum_{h \in H} |TH(h) - CH(h)|$$

TH(h) = 10

CH(h) = $x_1^2 + x_2^2 + x_3^2$

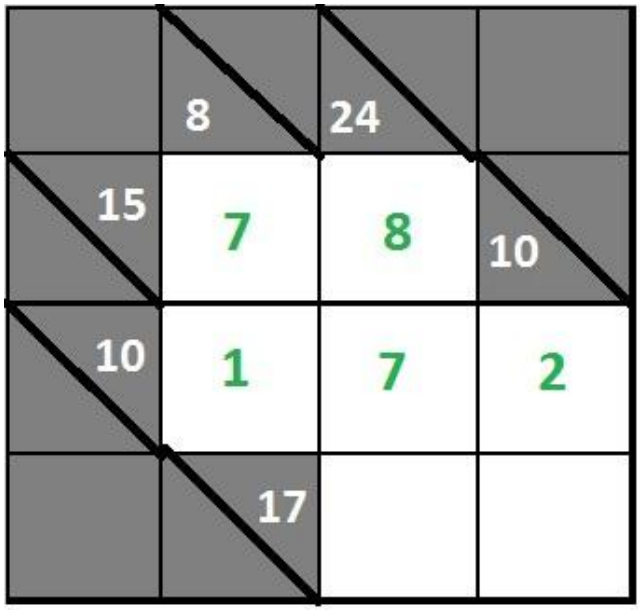
= 1 + 7 + 2

= 10

f = | 10 - 10 |

= 0

Karena Nilai yang didapatkan 0, Itu berarti nilai $(x_1^2 + x_2^2 + x_3^2) =$ jumlah dalam kotak soal. Maka perbaharui *Harmony memory*. *Puzzle* kakuro yang sudah diperbaharui *harmony memory* nya dapat dilihat pada gambar 3.13 dibawah ini :



Gambar 3.14 *Puzzle* kakuro yang sudah diperbaharui

Masuk ke iterasi keempat, untuk mencari nilai 24 menurut kolom, bangkitkan *harmony memory*, *harmony memory* yang sudah diperbaharui dapat dilihat pada tabel 3.4 dibawah ini.

Tabel 3.4 Tabel *Harmony memory* yang sudah diperbaharui

Nama Kotak Jawaban	Kandidat Solusi / *Harmony memory*
x_1^1	7 (update)
x_2^1	8 (update)
x_1^2	1 (update)
x_2^2	7 (update)
x_3^2	2 (update)
x_1^3	8,9
x_2^3	8,9

Improvisasi Harmoni Baru

Lakukan iterasi untuk x_1^3, karena kotak jawaban x_2^1 dan x_2^2 sudah berisi nilai.

Bangkitkan bilangan *random* r,

Misalkan r = 0.26, bandingkan r dengan HMCR

$0.26 \leq 0.9$, ambil satu nilai secara acak dari HM misal nilai x_1^3 =9.

Bangkitkan bilangan *random* r, Misalkan r= 0.3

$0.3 \geq 0.1$, maka nilai x_1^3 tetap dipertahankan dengan nilai x_1^3 = 9

Perbaharui *Harmony memory*

Cek nilai x_2^1 , x_2^2 dan nilai x_1^3. Dengan cara memasukkan nilai tersebut ke fungsi objektif kolom.

$$f = \sum_{v \in V} |TV(v) - CV(v)|$$

TV(v) = 24

CV(v) = $x_2^1 + x_2^2 + x_1^3$

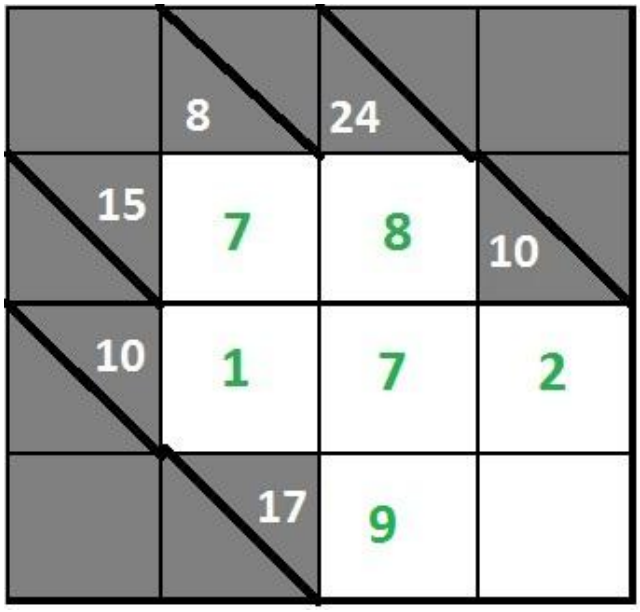
= 8 + 7 + 9

= 24

f = | 24 - 24 |

= 0

Karena Nilai yang didapatkan 0, Itu berarti nilai $(x_2^1 + x_2^2 + x_1^3) =$ jumlah dalam kotak soal. Maka perbaharui *Harmony memory*. *Puzzle* kakuro yang sudah diperbaharui *harmony memory* nya dapat dilihat pada gambar 3.14 dibawah ini :



Gambar 3.15 *Puzzle* kakuro yang sudah diperbaharui

Masuk ke iterasi kelima, untuk mencari nilai 17 menurut baris, bangkitkan *harmony memory*, *harmony memory* yang sudah diperbaharui dapat dilihat pada tabel 3.5 dibawah ini.

Tabel 3.5 Tabel *Harmony memory* yang sudah diperbaharui

Nama Kotak Jawaban	Kandidat Solusi / *Harmony memory*
x_1^1	7 (update)
x_2^1	8 (update)
x_1^2	1 (update)
x_2^2	7 (update)
x_3^2	2 (update)
x_1^3	9 (update)
x_2^3	8,9

Improvisasi Harmoni Baru

Lakukan iterasi untuk x_2^3, karena kotak jawaban x_1^3 sudah berisi nilai.

Bangkitkan bilangan *random* r,

Misalkan r = 0.96, bandingkan r dengan HMCR

0.9 ≥ 0.9, ambil satu nilai secara acak dari rentang nilai 1 – 9, misal nilai x_2^3 = 5.

Bangkitkan bilangan *random* r, Misalkan r= 0.2

0.2 ≥ 0.1, maka nilai x_2^3 tetap dipertahankan dengan nilai x_2^3 = 5

Perbaharui *Harmony memory*

Cek nilai x_1^3 dan nilai x_2^3. Dengan cara memasukkan nilai tersebut ke fungsi objektif baris.

$$f = \sum_{h \in H} |TH(h) - CH(h)|$$

TH(h) = 17

CH(h) = $x_1^3 + x_2^3$

= 9 + 5

= 14

f = | 17 - 14 |

= 3

Karena Nilai yang didapatkan 3, Itu berarti nilai

($x_1^3 + x_2^3$) ≠ jumlah dalam kotak soal. Maka *Harmony memory* tidak di perbaharui.

Masuk ke iterasi keenam, untuk mencari nilai 17 menurut baris, bangkitkan *harmony memory*.

Improvisasi Harmoni Baru

Lakukan iterasi untuk x_2^3, karena kotak jawaban x_1^3 sudah berisi nilai.

Bangkitkan bilangan *random* r,

Misalkan r = 0.29, bandingkan r dengan HMCR

0.29 ≥ 0.9, ambil satu nilai dari HM, misal nilai x_2^3 = 8.

Bangkitkan bilangan *random* r, Misalkan r= 0.5

0.5 ≥ 0.1, maka nilai x_2^3 tetap dipertahankan dengan nilai x_2^3 = 8

Perbaharui *Harmony memory*

Cek nilai x_1^3 dan nilai x_2^3. Dengan cara memasukkan nilai tersebut ke fungsi objektif baris.

$$f = \sum_{h \in H} |TH(h) - CH(h)|$$

TH(h) = 17

CH(h) = $x_1^3 + x_2^3$

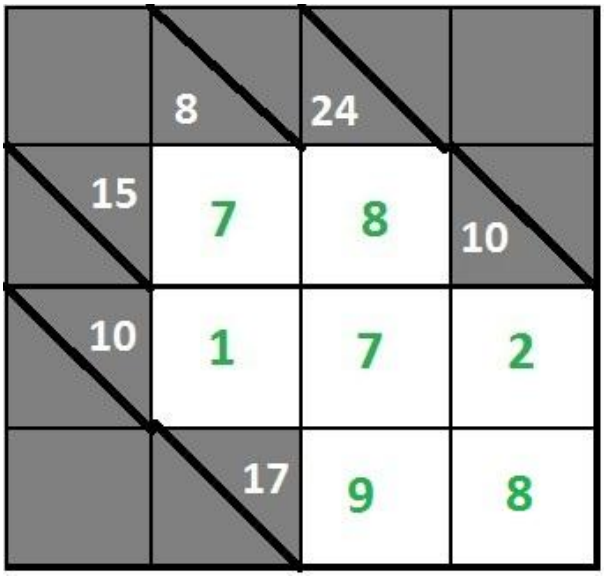
= 9 + 8

= 17

f = | 17 - 17 |

= 0

Karena Nilai yang didapatkan 0, Itu berarti nilai $(x_1^3 + x_2^3) =$ jumlah dalam kotak soal. Maka perbaharui *Harmony memory*. *Puzzle* kakuro yang sudah diperbaharui *harmony memory* nya dapat dilihat pada gambar 3.15 dibawah ini :



Gambar 3.16 *Puzzle* kakuro yang sudah diperbaharui

Masuk ke iterasi ketujuh, untuk mencari nilai 10 menurut kolom, bangkitkan *harmony memory*, *harmony memory* yang sudah diperbaharui dapat dilihat pada tabel 3.6 dibawah ini.

Tabel 3.6 Tabel *Harmony memory* yang sudah diperbaharui

Nama Kotak Jawaban	Kandidat Solusi / *Harmony memory*
x_1^1	7 (update)
x_2^1	8 (update)
x_1^2	1 (update)
x_2^2	7 (update)
x_3^2	2 (update)
x_1^3	9 (update)
x_2^3	8 (update)

Karena nilai x_3^2 dan x_2^3 sudah ada maka langsung ke langkah dibawah ini.

Perbaharui *Harmony memory*

Cek nilai x_3^2 dan nilai x_2^3. Dengan cara memasukkan nilai tersebut ke fungsi objektif kolom.

$$f = \sum_{v \in V} |TV(v) - CV(v)|$$

TV(v) = 10

CV(v) = $x_3^2 + x_2^3$

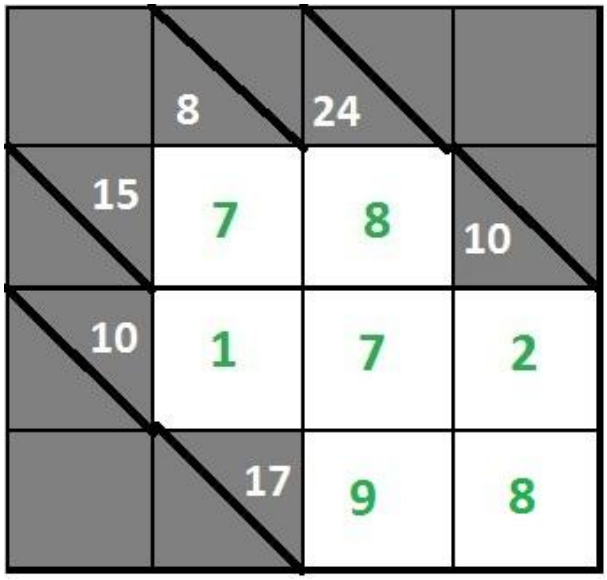
= 2 + 8

= 10

f = | 10 - 10 |

= 0

Karena Nilai yang didapatkan 0, Itu berarti nilai $(x_1^3 + x_2^3) =$ jumlah dalam kotak soal. Maka perbaharui *Harmony memory*. *Puzzle* kakuro yang sudah diperbaharui *harmony memory* nya dapat dilihat pada gambar 3.16 dibawah ini :

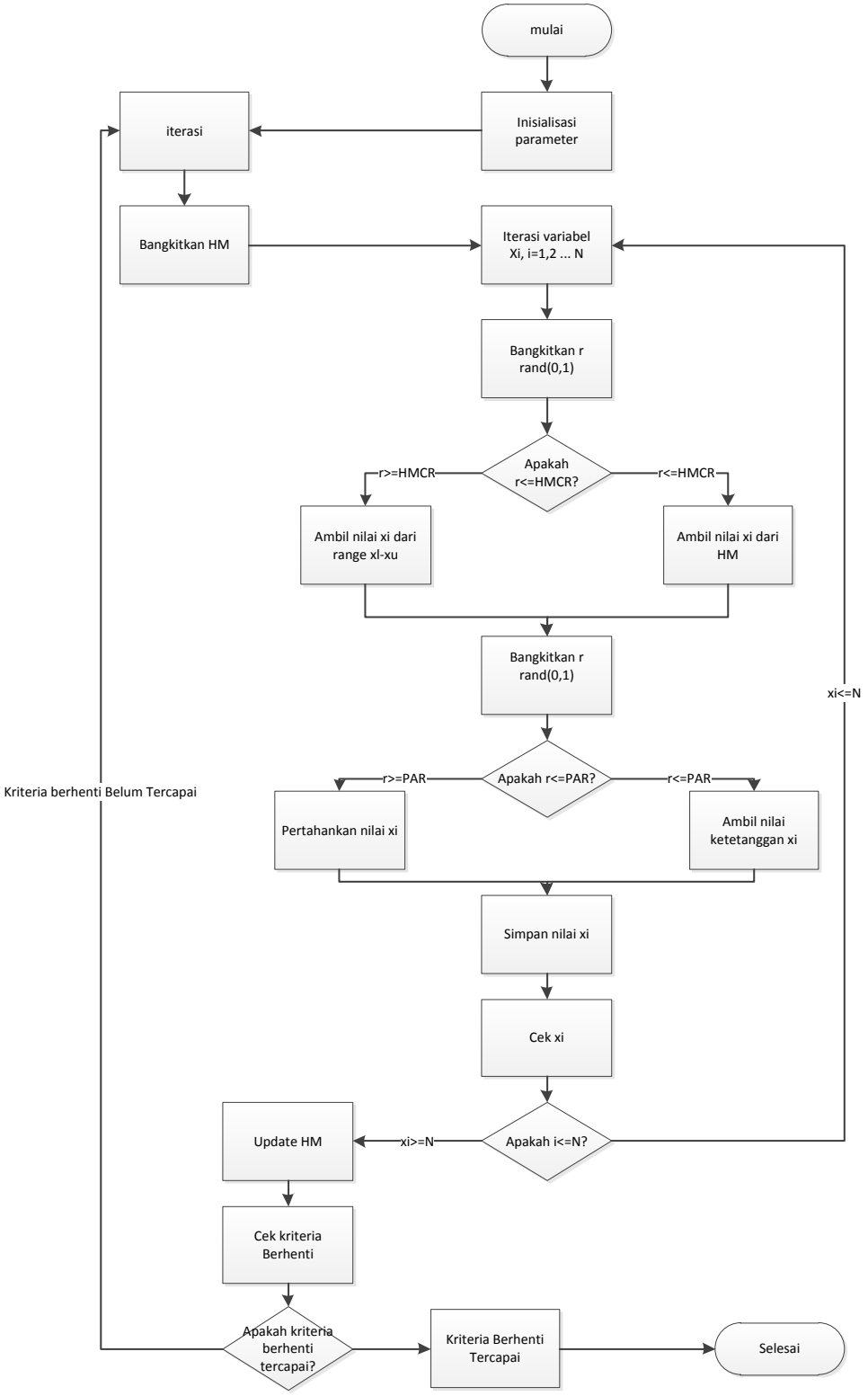


Gambar 3.17 Solusi Penyelesaian *Puzzle* Kakuro

Cek kriteria Penghentian

Karena solusi sudah ditemukan maka kriteria berhenti dicapai.

Alur kerja Algoritma akan digambarkan pada alur *flowchart* berikut :



Gambar 3.18 Flowchart Penyelesaian Permainan

3.5 Analisis Kebutuhan Perangkat Lunak

3.5.1 Analisis Kebutuhan Non Fungsional

Analisis non fungsional merupakan analisis yang dibutuhkan untuk menentukan spesifikasi kebutuhan sistem. Spesifikasi ini juga meliputi elemen atau komponen-komponen apa saja yang akan dibutuhkan sistem yang akan dibangun, sampai sistem tersebut diimplementasikan. Adapun kebutuhan non fungsionalnya yaitu meliputi analisis kebutuhan perangkat keras, analisis kebutuhan perangkat lunak, dan analisis pengguna.

1. Analisis Kebutuhan Perangkat Keras

Perangkat keras diperlukan untuk menjalankan suatu aplikasi guna mendukung proses kerja dari suatu sistem. Analisis perangkat keras digunakan untuk mengetahui mengenai spesifikasi perangkat keras dalam membangun dan menerapkan aplikasi, terdapat dua spesifikasi yang akan dijelaskan, yaitu spesifikasi perangkat keras pengembang dan spesifikasi perangkat keras pengguna.

Berikut ini merupakan spesifikasi perangkat keras yang digunakan dalam membangun aplikasi ini, yaitu:

Tabel 3.7 Spesifikasi Perangkat Keras

Perangkat Keras	Spesifikasi
Prosesor	Prosesor 3.6GHz QuadCore
VGA	2 GB DDR5
Resolusi Layar	1366 x 768 *Pixel*
Memory	*Memory* 4GB DDR3
Hard Disk	*Hard Disk* 250GB
Keyboard dan *Mouse*	-

Berikut ini merupakan spesifikasi perangkat keras pengguna atau spesifikasi minimum untuk menjalankan aplikasi ini, yaitu:

Tabel 3.8 Spesifikasi Perangkat Keras Pengguna

Perangkat Keras	Spesifikasi
Prosesor	Prosesor 2GHz
VGA	256 MB
Resolusi Layar	1366 x 768 *Pixel*
Memory	*Memory* 1GB
Hard Disk	*Hard Disk* 80GB
Keyboard dan *Mouse*	-

2. Analisis kebutuhan perangkat lunak (*software*)

Analisis perangkat lunak digunakan untuk mengetahui mengenai spesifikasi perangkat lunak untuk membangun dan menerapkan aplikasi, terdapat dua spesifikasi yang akan dijelaskan, yaitu spesifikasi perangkat lunak pengembang dan spesifikasi perangkat lunak pengguna.Berikut ini merupakan spesifikasi perangkat lunak yang digunakan dalam membangun aplikasi ini, yaitu: Kebutuhan perangkat lunak (*software*) untuk penerapan algoritma *Harmony Search* pada permainan *puzzle* kakuro dapat dilihat pada Tabel 3.9.

Tabel 3.9 Spesifikasi Perangkat Lunak

Perangkat Lunak	Spesifikasi
Sistem Operasi	Windows 7,
Bahasa pemrograman	Visual Studio 2012

Berikut ini merupakan spesifikasi perangkat lunak yang digunakan dalam menjalankan aplikasi ini, yaitu:

Tabel 3.10 Spesifikasi Perangkat Lunak Pengguna

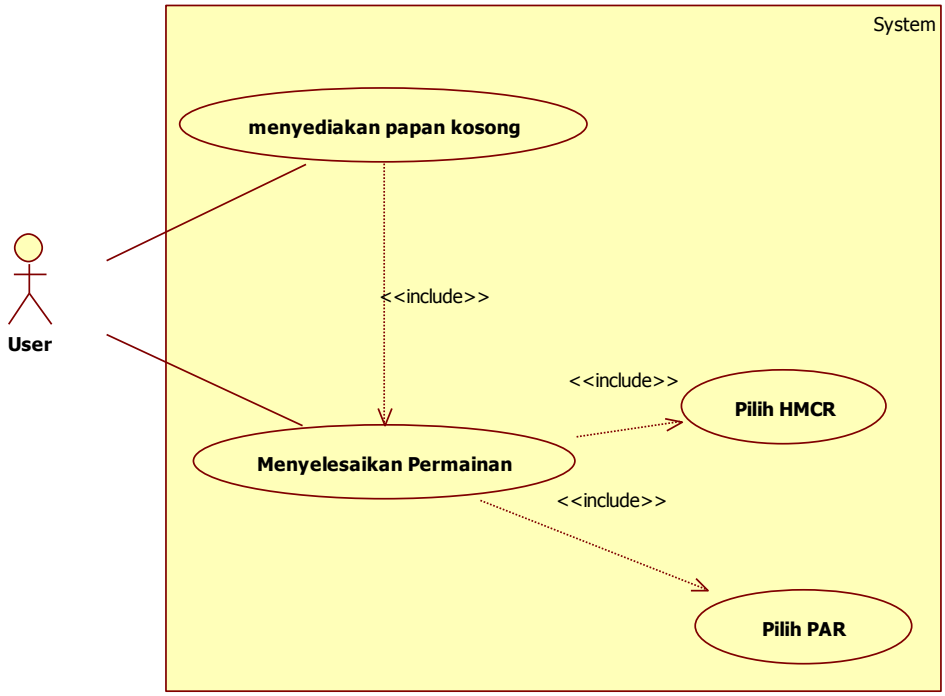
Perangkat Lunak	Spesifikasi
Sistem Operasi	Windows 7,

3.5.2 Analisis Kebutuhan Fungsional

Pemodelan aplikasi yang akan dibangun dilakukan dengan metode analisis perancangan dan pengembangan perangkat lunak berorientasi objek dan menggunakan pemodelan *Unified Modeling Language* (*UML*). Pemodelan mencakup *use case diagram, activity diagram, class diagram* dan *sequence diagram*. Proses yang dirancang diuraikan menjadi beberapa bagian yang dapat membentuk sistem tersebut menjadi satu kesatuan komponen.

3.5.2.1 Use Case Diagram aplikasi *puzzle* angka

Use case diagram untuk memodelkan aplikasi untuk mencari langkah terpendek pada *puzzle* kakuro dengan algoritma Harmony Search dapat dilihat pada Gambar 3.19.



Gambar 3.19 *Use Case Diagram* Aplikasi *Puzzle* Kakuro

1. Definisi *Use Case*

Definisi *use case* berfungsi untuk menjelaskan proses yang terdapat pada setiap *use case.* Tabel 3.11 menjelaskan definisi *Use Case* aplikasi *Puzzle* Kakuro.

Tabel 3.11 Definisi Use Case

Use Case	Definisi
menyediakan Papan kosong	Proses untuk menampilkan papan *puzzle* kakuro
Menyelesaikan Permainan	Proses untuk menyelesaikan permainan *puzzle* kakuro

2. Skenario Use Case

Skenario merupakan penjelasan lebih detail dari *use case* dari awal sampai akhirnya diperoleh output.

1) Skenario Use Case menyediakan papan kosong, diperlihatkan pada Tabel 3.12.

Tabel 3.12 Skenario Use Case Menyediakan Papan *Puzzle* Kakuro

Identifikasi			
Nama Use Case		Menyediakan Papan kosong	
Tujuan		Membuat Papan *Puzzle* Kakuro	
Aktor		user	
Skenario Utama			
Kondisi Awal		Halaman Utama Form *Puzzle* kakuro.	
No	**Aksi Aktor**	**No**	**Respons Sistem**
1	Memilih Ukuran *Puzzle*	2	Menampilkan Papan kosong dengan ukuran sesuai pilihan user
Kondisi Akhir	Sistem menampilkan papan *puzzle* kakuro		

2) Skenario Use Case memainkan permainan, diperlihatkan pada Tabel 3.13.

Tabel 3.13 Skenario Use Case memainkan permainan

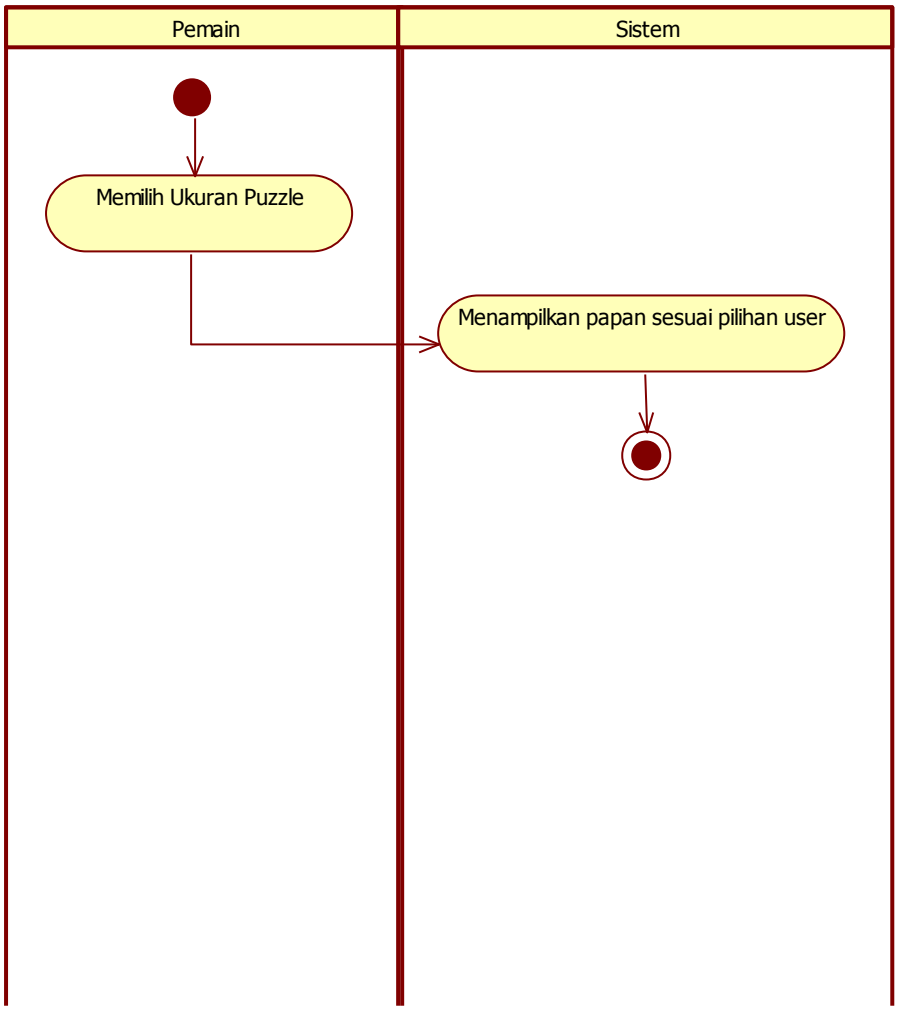
Identifikasi			
Nama Use Case		Menyelesaikan permainan	
Tujuan		Permainan *puzzle* kakuro Terselesaikan	
Aktor		user	
Skenario Utama			
Kondisi Awal		Halaman Utama Form *Puzzle* kakuro.	
No	**Aksi Aktor**	**No**	**Respons Sistem**
1	Memilih Papan *Puzzle*		
		2	Menampilkan papan *puzzle* kakuro
3	Pilih HMCR		
4	Pilih PAR		
5	Tampilkan *Harmony memory*		
		6	Menampilkan *Puzzle* yang terisi *Harmony memory*
7	Klik Solve		
		8	Penyelesaian Permainan oleh Algoritma harmony Search
		9	Solusi Ditemukan
Kondisi Akhir		Sistem menampilkan solusi dari *puzzle* kakuro	

3.5.2.2 Activity Diagram

Activity Diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, serta proses paralel yang mungkin terjadi pada beberapa eksekusi.

1. Activity Diagram membuat Papan

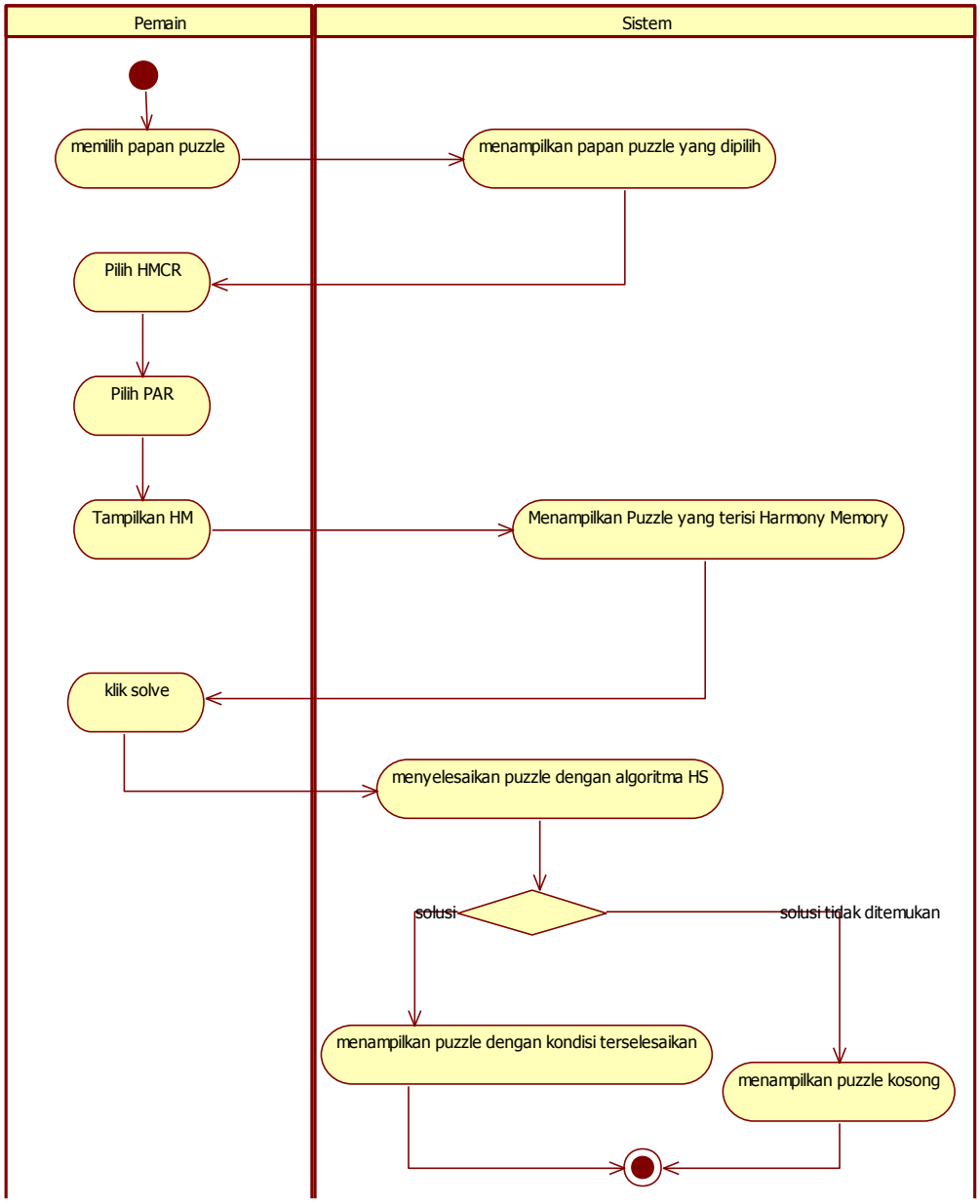
Berikut ini merupakan diagram yang menunjukan alur aksi pada aktivitas Membuat papan yang dapat dilihat pada gambar 3.20.



Gambar 3.20 Activity Diagram menyediakan Papan

2. Activity Diagram Menyelesaikan Permainan

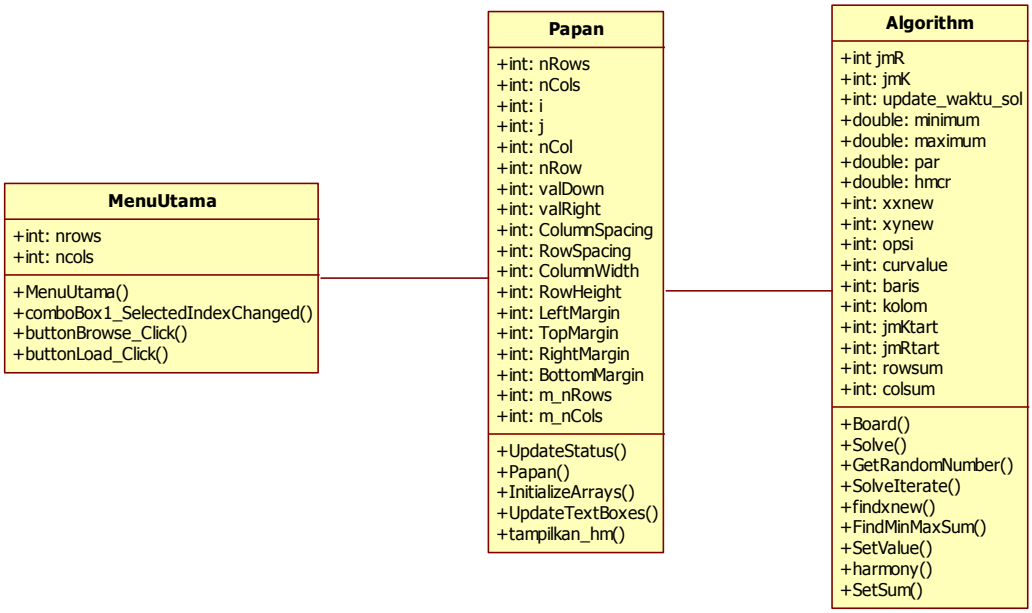
Berikut ini merupakan diagram yang menunjukan alur aksi pada aktivitas Menyelesaikan permainan yang dapat dilihat pada gambar 3.21.



Gambar 3.21 Activity Diagram Menyelesaikan Permainan

3.5.2.3 Class Diagram

Class diagram merupakan gambaran sistem berupa relasi antar kelas-kelas ataupun paket-paketnya. *Class diagram* dijelaskan melalui gambar berikut.



Gambar 3.22 *Class diagram* permainan *puzzle* kakuro

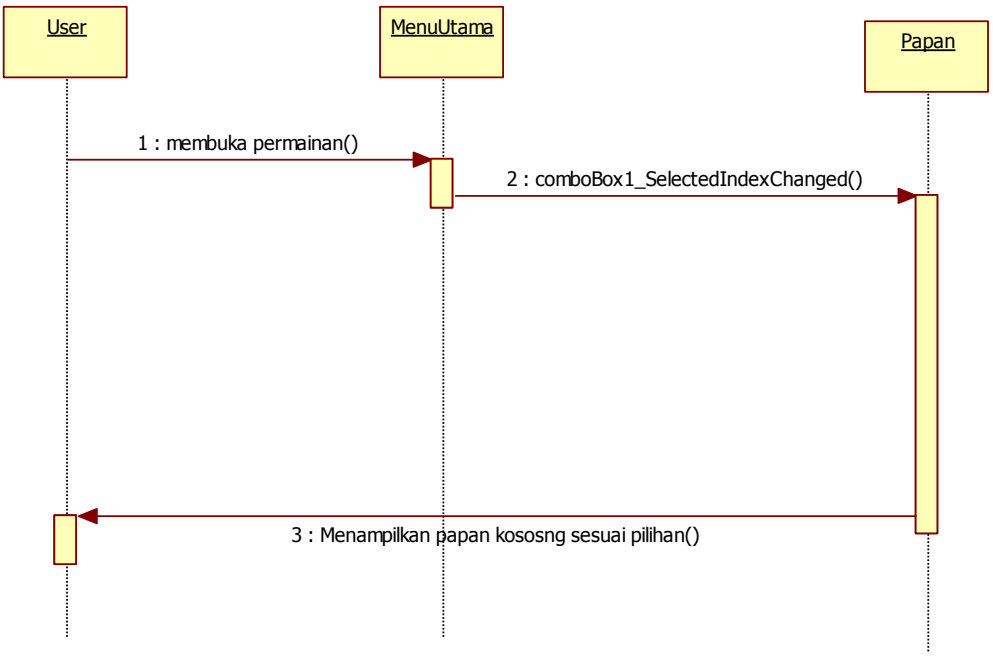
Penjelasan dari masing-masing class diagram dapat dilihat pada tabel 3.14

Tabel 3.14 Deskripsi Class Diagram

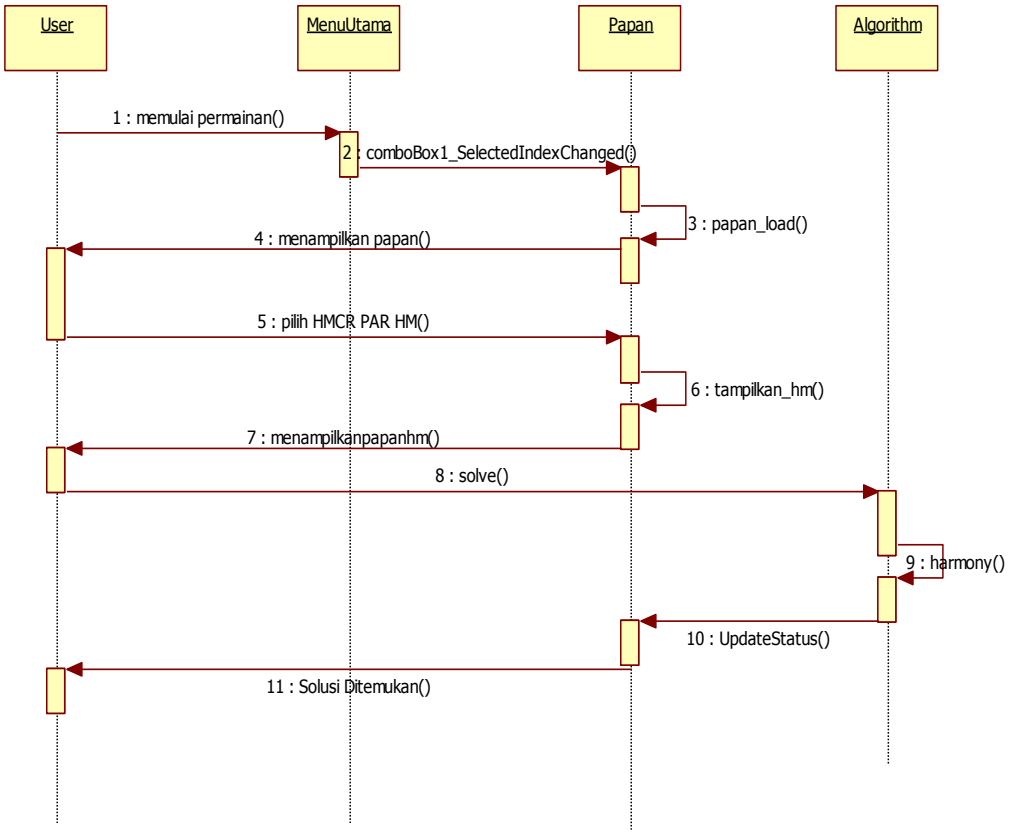
User Application		
Class	**Jenis Class**	**Deskripsi**
MenuUtama	View	Class yang menampilkan halaman awal aplikasi
Papan	Control	Class yang berisikan ukuran papan *puzzle* kakuro, inisialisasi array
Algorithm	Control	Class yang berisikan algoritma *harmony search*

3.5.2.4 Sequence Diagram

Sequence diagram menggambarkan tindakan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. *Sequence diagram* dapat dilihat pada gambar berikut.



Gambar 3.23 *Sequence diagram* menyediakan papan



Gambar 3.24 *Sequence diagram* menyelesaikan permainan

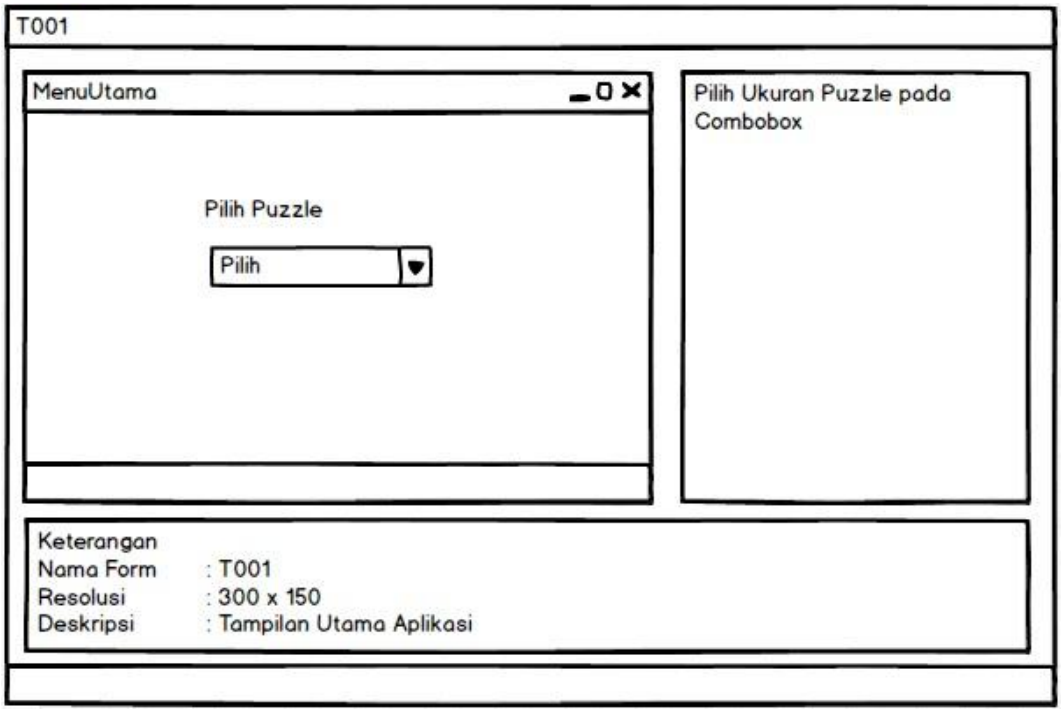
3.6 Perancangan Sistem

Perancangan sistem adalah suatu bagian dari metodologi pembangunan suatu perangkat lunak yang dilakukan setelah tahapan analisis sistem untuk memberikan gambaran secara terperinci sistem yang akan dibangun. Dalam perancangan sistem pada aplikasi ini, maka dilakukan beberapa langkah, yaitu:

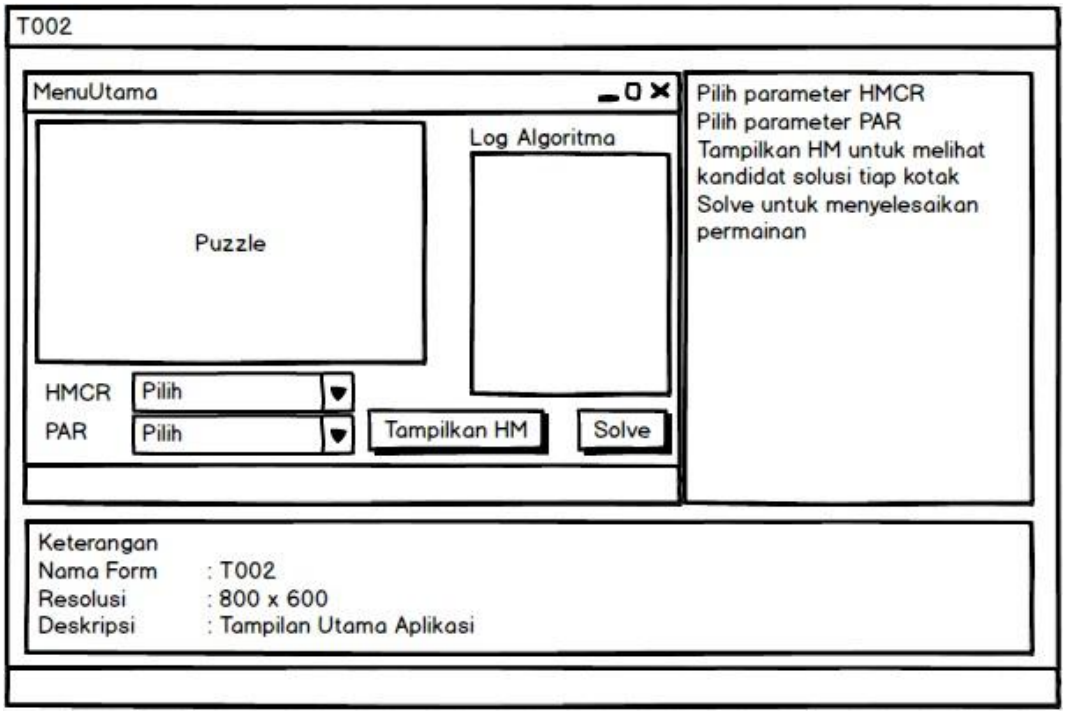
1. Perancangan Antarmuka
2. Perancangan pesan
3. Jaringan Semantik

3.2.1 Perancangan Antarmuka

Perancangan antarmuka dibutuhkan untuk mewakili keadaan sebenarnya dari aplikasi yang akan dibangun. Perancangan antarmuka pada aplikasi ini adalah sebagai berikut:



Gambar 3.25 Antarmuka permainan Halaman Utama



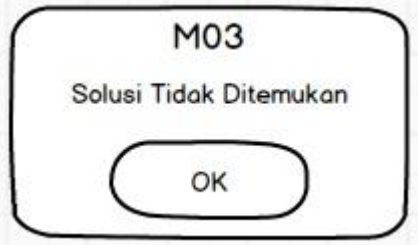
Gambar 3.26 Antarmuka permainan Penyelesai Permainan

3.2.2 Perancangan Pesan

Perancangan pesan merupakan pesan yang muncul guna memberi peringatan atau pemberitahuan kepada *user*. Perancangan pesan pada aplikasi ini adalah sebagai berikut:



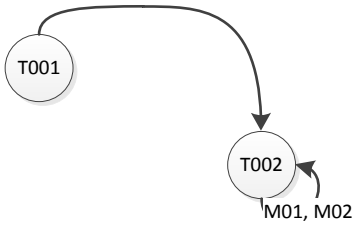
Gambar 3.27 Pesan Solusi Ditemukan



Gambar 3.28 Pesan Solusi Tidak Ditemukan

3.2.3 Jaringan Semantik

Jaringan semantik merupakan gambaran tentang relasi dari masing-masing halaman yang bisa diakses oleh pengguna. Jaringan semantik pada aplikasi ini adalah sebagai berikut:



Gambar 3.29 Jaringan Semantik