

**Laporan Akhir**  
**Penerapan *Constraint Satisfaction Problem***  
**pada Games *Magic Square Puzzle***



**Disusun oleh:**

1. Lusiana Ros Romantika Siahaan – 12S18005
2. Yohana Polin Simatupang – 12S18018
3. Maria Puspita Sari PNababan – 12S18019
4. Roy Gunawan Napitupulu – 12S18043
5. Letare Aiglien Saragih – 12S18064
6. Grace Vidia Rosa Panjaitan – 12S18067

**10S3001 – KECERDASAN BUATAN**  
**FAKULTAS TEKNIK INFORMATIKA DAN ELEKTRO**  
**INSTITUT TEKNOLOGI DEL**  
**2020/2021**

# DAFTAR ISI

|   |    |
|---|----|
| DAFTAR ISI  | 1  |
| DAFTAR GAMBAR   | 2  |
| DAFTAR TABEL  | 2  |
| BAB 1   | 3  |
| 1.1. Latar Belakang   | 3  |
| 1.2. Tujuan   | 3  |
| 1.3. Manfaat  | 3  |
| 1.4. Ruang Lingkup  | 4  |
| BAB 2   | 5  |
| 2.1. Analisis   | 5  |
| 2.2. <i>Flowchart</i> Penyelesaian <i>Magic Square Puzzle</i> | 6  |
| 2.3. Implementasi Kode Program <i>Magic Square Puzzle</i>     | 9  |
| 2.3.1. Fungsi Main  | 9  |
| 2.3.2. Fungsi Grid 3x3  | 10 |
| 2.3.3. Fungsi Grid 4x4  | 14 |
| 2.3.4. Fungsi Grid 5x5  | 18 |
| 2.4. Evaluasi   | 23 |
| BAB 3   | 25 |
| 3.1. Pembagian Tugas  | 25 |
| 3.2. Kesimpulan   | 27 |
| <b>3.3 Saran</b>  | 27 |
| Daftar Pustaka  | 28 |

## **DAFTAR GAMBAR**

|                                     |   |
|-------------------------------------|---|
| Gambar 1. Flowchart MSP .....       | 6 |
| Gambar 2. Contoh puzzle 3 x 3 ..... | 8 |

## **DAFTAR TABEL**

|   |    |
|---|----|
| Tabel 1. Pembagian Tugas Anggota Tim Proyek ..... | 25 |
|---|----|

# BAB 1

## Pendahuluan

### 1.1. Latar Belakang

Seiring berkembangnya teknologi informasi diberbagai bidang, baik dari sisi perangkat lunak maupun dari sisi perangkat keras, dan semakin bertambahnya kebutuhan manusia, maka teknologi juga berkembang untuk tujuan pengetahuan. Salah satu bentuknya adalah kecerdasan buatan untuk menyelesaikan permasalahan dalam kehidupan sehari-hari.

Pak Natal adalah seorang yang suka menggunakan waktu luangnya untuk memainkan permainan sederhana. Salah satu permainan yang saat ini dimainkan oleh pak Natal adalah Magic Square Puzzle. Pak natal membutuhkan penyelesaian untuk permasalahan pada *magic square puzzle*. Sebagai solusi dari permasalahan tersebut maka kami menerapkan teknik kecerdasan buatan yaitu *Constraint Satisfaction Problem (CSP)*.

### 1.2. Tujuan

Berikut ini kami jabarkan tujuan dari pembuatan proposal pada *magic square puzzle*.

1. Pembaca dapat mengetahui cara penyelesain dari *magic square puzzle* dengan menerapkan kecerdasan buatan .
2. Pembaca dapat mempelajari serta mengenal lebih dalam tentang penggunaan dan pengembangan algoritma CSP.
3. Proposal digunakan sebagai media komunikasi antar anggota tim proyek untuk konstruksi algoritma *magic square puzzle*.

### 1.3. Manfaat

Manfaat dari proyek ini dijabarkan pada butir poin berikut.

1. Dokumentasi proyek dapat digunakan sebagai referensi untuk para pembaca mengenai pemanfaatan algoritma *Constraint Satisfaction Problem (CSP)*.
2. Mempermudah dalam mencari solusi untuk permainan *magic square puzzle*.

## 1.4. Ruang Lingkup

Ruang lingkup pada proyek ini dijelaskan sebagai berikut.

1. Aplikasi menggunakan CLI sebagai *interface*.
2. Program akan menyelesaikan permainan magic square puzzle dengan jumlah kotak mulai dari 3x3 sampai 5x5.
3. Program menggunakan konsep algoritma *Constraint Satisfaction Problem* (CSP).

## BAB 2

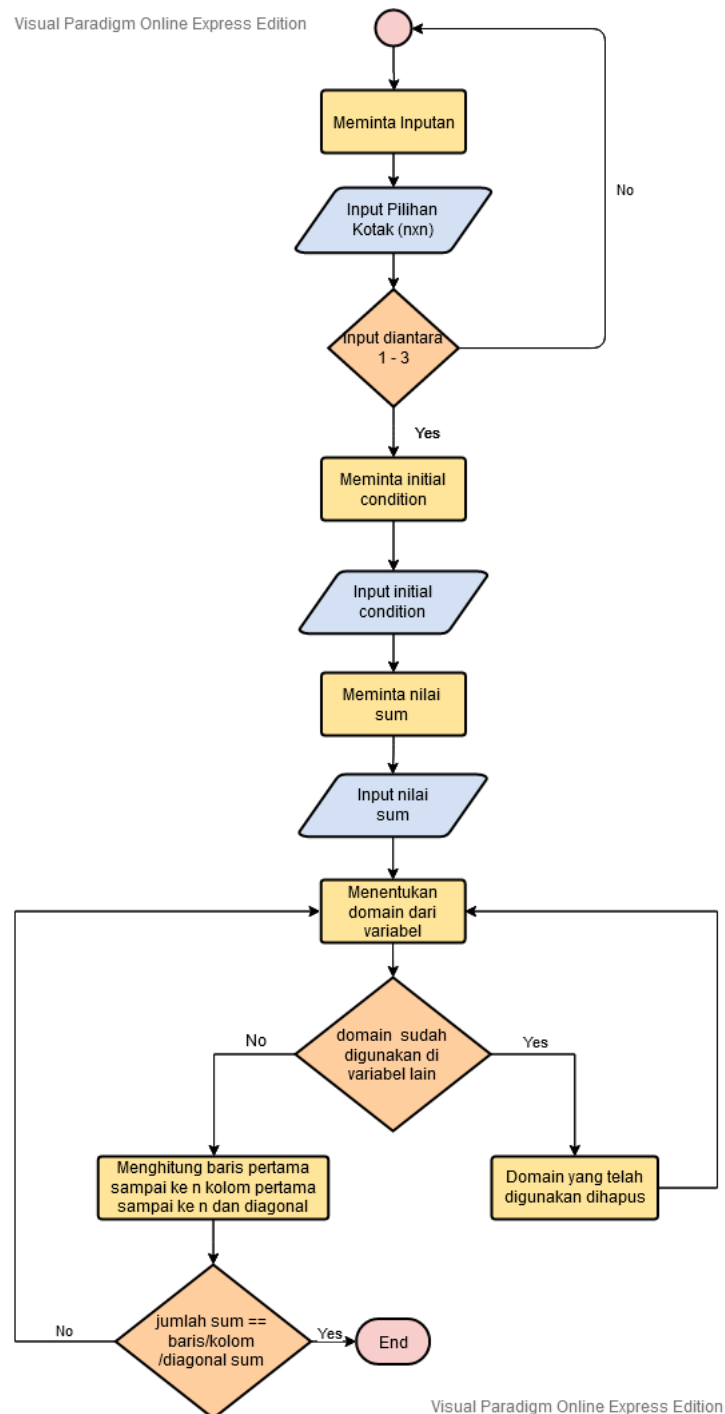
### ISI

#### 2.1. Analisis

Pada kondisi ini persoalan Magic Square Puzzle akan diselesaikan secara komputasi yaitu menggunakan algoritma CSP. Pada implementasi nya, nilai variabel adalah grid yang disediakan, yaitu grid 1,1, grid 1,2, grid 1,3, dan grid selanjutnya yang digunakan untuk menyimpan nilai dari angka yang akan dimasukkan. Sedangkan untuk nilai batasan atau *constraint* adalah jumlah bilangan yang ditambahkan untuk masing-masing grid. Sementara, untuk *domain*-nya adalah 1 sampai  $n \times n$  dengan  $n$  adalah jumlah baris atau kolom dari kotak. Spesifik metode yang digunakan adalah algoritma *Backtracking*. Hal ini dikarenakan Magic Square Puzzle dapat diselesaikan saat posisi variabel dalam kotak sudah memenuhi kondisi atau batasan-batasan tertentu yang nantinya batasan-batasan ini akan didefinisikan menggunakan algoritma CSP. Metode *backtracking* digunakan karena penyelesaian MSP nantinya akan memerlukan banyak aktivitas pengecekan apakah posisi variabel dalam kotak sudah memenuhi ketentuan yang diberikan. Maka untuk mempercepat penyelesaian masalah secara tepat digunakan backtracking karena backtracking merupakan algoritma berbasis DFS (*Depth First Search*) dimana metode ini tidak perlu memeriksa semua kemungkinan solusi yang ada, namun hanya melakukan pencarian yang mendekati solusi saja.

Pada persoalan Magic Square Puzzle yang diimplementasikan, algoritma Backtracking akan menguji jalur yang dilalui. Jika solusi tidak ditemukan, algoritma akan melakukan pengujian pada jalur lain yang mungkin hingga menemukan solusi yang tepat, yaitu untuk menemukan penjumlahan nilai yang tepat. Pada implementasi, kelompok akan menampilkan antarmuka pengguna dengan jenis *Command Line Interface* (CLI). Pemilihan jenis ini dikarenakan performa yang akan lebih baik dan penggunaan lebih efisien. Implementasi *magic square puzzle* akan menggunakan sebuah prosedur untuk menunjukkan *grid* di layar.

## 2.2. Flowchart Penyelesaian Magic Square Puzzle



Gambar 1. Flowchart MSP

- Proses dimulai dengan memasukkan input berupa jumlah kotak yang diinginkan, yang akan diidentifikasi melalui variabel  $n$ . Nilai  $n$  memiliki batasan dari nilai 1-3. Dari nilai  $n$  ini nanti akan menentukan *puzzle* ukuran berapa yang akan dihitung. Akan diberikan list:

1: Grid 3x3

2: Grid 4x4

3: Grid 5x5

Contoh jika memasukkan input angka 1, maka ukuran puzzle adalah 3 x 3

- Kemudian, akan dilanjutkan pada proses meminta *initial condition*. Pada proses ini, *user* akan diminta untuk memasukkan angka dan kotak yang akan diisi dengan angka tersebut. Jika angka yang dimasukkan pada sebuah kotak adalah 0, menunjukkan bahwa kotak tersebut akan diisi oleh sistem. Pengisian *initial condition* ini akan dilakukan secara berurut dari baris pertama.

Contoh inputan untuk puzzle 3 x 3 :

|       |
|-------|
| 4,0,1 |
| 0,8,0 |
| 2,0,2 |

- Kemudian dilakukan pengecekan kondisi dari nilai inputan yang diberikan. Apakah angka yang diinput oleh user termasuk dalam nilai variabel *n* yang dimiliki sistem, maka :
  - Jika Ya, akan dilanjutkan ke proses Meminta Nilai Sum . Dimana nilai Sum yang dimaksud pada proses ini adalah nilai penjumlahan untuk setiap baris, kolom maupun diagonal.
  - Jika Tidak , user akan kembali pada proses memasukkan input untuk jumlah matrix kotak
- Setelah dilakukan pemeriksaan terhadap algoritma yang akan digunakan kemudian dilakukan proses menentukan domain dari setiap variabel. Dimana:
  - Domain : {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
  - Varibel : { kotak-kotak yang terdapat pada puzzle tersebut seperti :  
 $grid[0][1], grid[0][2], ..., grid[n-row][n-column]$  } atau dapat kita lihat seperti gambar dibawah ini.



### example of puzzle 3x3

|             |   |             |   |   |
|-------------|---|-------------|---|---|
| R<br>O<br>W | 2 |             |   |   |
|             | 1 |             |   |   |
|             | 0 |             |   |   |
|             |   | 0           | 1 | 2 |
|             |   | C O L U M N |   |   |

Gambar 2. Contoh puzzle 3 x 3

- Setelah dilakukan penentuan domain pada setiap variabel maka langkah selanjutnya melakukan pemeriksaan apakah domain sudah digunakan pada variabel yang lain:
  - Jika ya, maka akan dilakukan penghapusan terhadap domain yang telah digunakan. Kemudian akan kembali ke dalam proses pemeriksaan apakah domain sudah digunakan pada variabel yang lain dan dapat kita lihat terdapat *backtracking* dalam hal pemeriksaan domain yang telah digunakan.
  - Jika tidak, maka akan dilakukan penghitungan terhadap tiap kotak-kotak tiap baris, tiap kolom, dan tiap diagonal. Setelah dilakukan penghitungan, maka hasil hitungan tersebut akan dibandingkan dengan nilai SUM yang sudah ditentukan sebelumnya. Jika masih belum sama dengan nilai SUM, maka akan dilakukan kembali penentuan domain dari variabel dan dilakukan penghitungan serta pengecekan ulang. Namun, jika seluruh baris, kolom, dan diagonal sudah sesuai dengan dengan jumlah sum, maka, program akan menampilkan hasil penyelesaian atau solusi dari magic square. Setelah itu, program selesai.


## 2.3. Implementasi Kode Program *Magic Square Puzzle*

### 2.3.1. Fungsi Main

Merupakan fungsi utama yang digunakan untuk menyelesaikan kasus MSP (*Magic Square Puzzle*). Dimana ketika menjalankan aplikasi akan meminta inputan dari user grid yang diinginkan. Jika yang diisi oleh user `choice == 3` maka selanjutnya akan masuk kedalam fungsi grid 3x3, jikalau `choice == 4` maka selanjutnya akan masuk kedalam fungsi grid 4x4, jika `choice == 5` maka selanjutnya akan masuk kedalam fungsi grid 5x5, dan yang terakhir jika inputan yang diberikan bukan grid 3,4, dan 5 maka akan menampilkan “Grid tidak ditemukan”.

```
# Function utama aplikasi
import subprocess
while True:
    try:
        choice = int(input("Please enter your choice: "))
    except ValueError:
        print("Insert another value!")
        continue
    else:
        break
if choice == 3:
    subprocess.call(['python', 'grid_3x3.py'])
elif choice == 4:
    subprocess.call(['python', 'grid_4x4.py'])
elif choice == 5:
    subprocess.call(['python', 'grid_5x5.py'])
else:
    print("Grid tidak ditemukan!")
```

Berikut merupakan tampilan ketika grid yang diinput oleh user bukan grid 3,4,dan 5



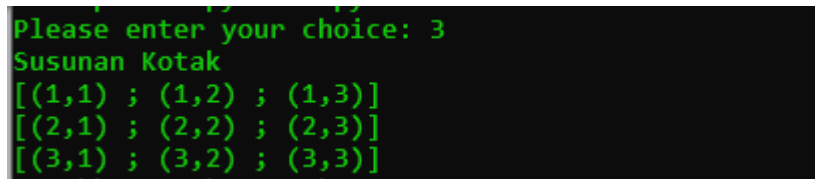
```
Please enter your choice: 2
Grid tidak ditemukan!
```

### 2.3.2. Fungsi Grid 3x3

Ketika user mengisi jumlah grid yang akan digunakan yaitu 3 maka akan tampil grid yang diinginkan.

```
#3 X 3
print("Susunan Kotak ")
print("[(1,1) ; (1,2) ; (1,3)]")
print("[(2,1) ; (2,2) ; (2,3)]")
print("[(3,1) ; (3,2) ; (3,3)]")
```

Tampilan gridnya seperti gambar dibawah ini



```
Please enter your choice: 3
Susunan Kotak
[(1,1) ; (1,2) ; (1,3)]
[(2,1) ; (2,2) ; (2,3)]
[(3,1) ; (3,2) ; (3,3)]
```

#### 2.3.2.1. Definisikan nilai dari setiap (row,col)

Pada bagian ini user akan mengisi nilai dari tiap kolom dan barisnya dimana yang diisi adalah nilai yang dianggap sudah benar dimana tidak boleh ada nilai yang berulang dan nilai yang diinput adalah bebas, selain itu akan diisi dengan 0 dan mengisi total nilai setiap baris,kolom, dan diagonalnya.

```
satusatu = input("Masukkan Angka Kotak 1,1 = ")
print(type(satusatu))
a = int(satusatu)
satudua = input("Masukkan Angka Kotak 1,2 = ")
print(type(satudua))
b = int(satudua)
satutiga = input("Masukkan Angka Kotak 1,3 = ")
print(type(satutiga))
c = int(satutiga)

duasatu = input("Masukkan Angka Kotak 2,1 = ")
print(type(duasatu))
d = int(duasatu)
duadua = input("Masukkan Angka Kotak 2,2 = ")
print(type(duadua))
e = int(duadua)
duatiga = input("Masukkan Angka Kotak 2,3 = ")
print(type(duatiga))
f = int(duatiga)

tigasatu = input("Masukkan Angka Kotak 3,1 = ")
print(type(tigasatu))
g = int(tigasatu)
tigadua = input("Masukkan Angka Kotak 3,2 = ")
print(type(tigadua))
h = int(tigadua)
tigatiga = input("Masukkan Angka Kotak 3,3 = ")
```

Tampilannya adalah sebagai berikut:

```
Masukkan Angka Kotak 1,1 = 0
<class 'str'>
Masukkan Angka Kotak 1,2 = 7
<class 'str'>
Masukkan Angka Kotak 1,3 = 6
<class 'str'>
Masukkan Angka Kotak 2,1 = 0
<class 'str'>
Masukkan Angka Kotak 2,2 = 5
<class 'str'>
Masukkan Angka Kotak 2,3 = 0
<class 'str'>
Masukkan Angka Kotak 3,1 = 4
<class 'str'>
Masukkan Angka Kotak 3,2 = 0
<class 'str'>
Masukkan Angka Kotak 3,3 = 8
<class 'str'>
Masukkan Total yang diinginkan : 15
<class 'str'>
```

### 2.3.2.2. Fungsi Pengecek total nilai setiap baris, kolom, dan diagonal

Pada bagian tersebut setiap baris, kolom dan diagonal akan diperiksa apakah setiap baris, kolom, dan diagonal sudah sesuai dengan total nilai yang telah diinput oleh user sebelumnya. Dimana pengecekan setiap kotak tersebut sebanyak 3 kali.

```
kotakmsp_tkt = []
kotakmsp_tkt.append([a,b,c])
kotakmsp_tkt.append([d,e,f])
kotakmsp_tkt.append([g,h,i])

def checkingMSP(kotakmsp_tkt):
    global totalnilai
    for row in range(0,3):
        for col in range(0,3):
            if kotakmsp_tkt[row][col]==0:
                return False
    for row in range(0,3):
        if(kotakmsp_tkt[row][0]+kotakmsp_tkt[row][1]+kotakmsp_tkt[row][2])!=total
nilai:
            return False
    for col in range(0,3):
        if (kotakmsp_tkt[0][col]+kotakmsp_tkt[1][col]+kotakmsp_tkt[2][col])!=tota
lnilai:
            return False
    if (kotakmsp_tkt[0][0]+kotakmsp_tkt[1][1]+kotakmsp_tkt[2][2])!=totalnilai:
        return False
    if (kotakmsp_tkt[0][2]+kotakmsp_tkt[1][1]+kotakmsp_tkt[2][0])!=totalnilai:
        return False
    return True
```

### 2.3.2.3. Fungsi untuk melakukan penyelesaian dengan mencari kotak yang kosong

Pada bagian tersebut akan dilakukan penyelesaian dengan cara memeriksa setiap baris dan kolom apakah nilai yang akan diisi sudah memenuhi atau tidak. Jika sedang dilakukan pemeriksaan maka akan menampilkan “Nilai pada kotak-kotak sedang diperiksa” dimana pemeriksaan dilakukan sebanyak 9 kali pada setiap kotaknya. Jikalau baris ataupun kolom sudah selesai diperiksa akan menampilkan “Nilai pada kotak-kotak sudah selesai diperiksa. Kemudian akan menampilkan baris dan kolom yang sedang diperiksa.

```
def penyelesaiantkt(kotakmsp_tkt):
    for i in range(0,9):
        row=i//3
        col=i%3
        if kotakmsp_tkt[row][col]==0:
            print("Nilai pada kotak-kotak sedang diperiksa")
            for value in range (1,10):
                if not(value in kotakmsp_tkt[0] or value in kotakmsp_tkt[1] or value in kotakmsp_tkt[2]):
                    kotakmsp_tkt[row][col]= value
                    if checkingMSP(kotakmsp_tkt):
                        print("Nilai pada kotak-kotak sudah selesai diperiksa.")
                        return True
                    else:
                        if penyelesaiantkt(kotakmsp_tkt):
                            return True
            break
    print(kotakmsp_tkt)
    kotakmsp_tkt[row][col] = 0
```

Tampilannya adalah sebagai berikut:

```
Nilai pada kotak-kotak sedang diperiksa
Nilai pada kotak-kotak sedang diperiksa
Nilai pada kotak-kotak sedang diperiksa
Nilai pada kotak-kotak sedang diperiksa
[[1, 7, 6], [2, 5, 3], [4, 9, 8]]
[[1, 7, 6], [2, 5, 3], [4, 9, 0]]
Nilai pada kotak-kotak sedang diperiksa
Nilai pada kotak-kotak sedang diperiksa
[[1, 7, 6], [2, 5, 8], [4, 3, 9]]
[[1, 7, 6], [2, 5, 8], [4, 3, 0]]
Nilai pada kotak-kotak sedang diperiksa
[[1, 7, 6], [2, 5, 8], [4, 9, 3]]
[[1, 7, 6], [2, 5, 8], [4, 9, 0]]
[[1, 7, 6], [2, 5, 8], [4, 9, 0]]
Nilai pada kotak-kotak sedang diperiksa
Nilai pada kotak-kotak sedang diperiksa
[[1, 7, 6], [2, 5, 9], [4, 3, 8]]
[[1, 7, 6], [2, 5, 9], [4, 3, 0]]
Nilai pada kotak-kotak sedang diperiksa
[[1, 7, 6], [2, 5, 9], [4, 8, 3]]
[[1, 7, 6], [2, 5, 9], [4, 8, 0]]
[[1, 7, 6], [2, 5, 9], [4, 8, 0]]
[[1, 7, 6], [2, 5, 9], [4, 0, 0]]
Nilai pada kotak-kotak sedang diperiksa
```



#### 2.3.2.4. Fungsi Penyelesaian

Pada bagian ini akan dilakukan tahap penyelesaian dari MSP tersebut. Jika MSP berhasil diselesaikan maka akan ditampilkan grid penyelesaiannya, jika tidak dapat diselesaikan maka akan menampilkan "Magic Square Puzzle tidak dapat diselesaikan. Silahkan run kembali."

```
solved = penyelesaiantkt(kotakmsp_tkt)
if solved:
    print("Hasil Penyelesaian:")
    print (" [ {} ] [ {} ] [ {} ]".format(kotakmsp_tkt[0][0], kotakmsp_tkt[0][1], kotakmsp_tkt[0][2]))
    print (" [ {} ] [ {} ] [ {} ]".format(kotakmsp_tkt[1][0], kotakmsp_tkt[1][1], kotakmsp_tkt[1][2]))
    print (" [ {} ] [ {} ] [ {} ]".format(kotakmsp_tkt[2][0], kotakmsp_tkt[2][1], kotakmsp_tkt[2][2]))
else:
    print("Magic Square Puzzle tidak dapat diselesaikan. Silahkan run kembali.")
```

Tampilannya adalah sebagai berikut

```
Hasil Penyelesaian:
[ 2 ] [ 7 ] [ 6 ]
[ 9 ] [ 5 ] [ 1 ]
[ 4 ] [ 3 ] [ 8 ]
```

#### 2.3.3. Fungsi Grid 4x4

Ketika user mengisi jumlah grid yang akan digunakan yaitu 4 maka akan tampil grid yang diinginkan.

```
print("Susunan Kotak ")
print("[(1,1) ; (1,2) ; (1,3) ; (1,4)]")
print("[(2,1) ; (2,2) ; (2,3) ; (2,4)]")
print("[(3,1) ; (3,2) ; (3,3) ; (3,4)]")
print("[(4,1) ; (4,2) ; (4,3) ; (4,4)]")
```

Tampilannya adalah sebagai berikut:

```
Please enter your choice: 4
Susunan Kotak
[(1,1) ; (1,2) ; (1,3) ; (1,4)]
[(2,1) ; (2,2) ; (2,3) ; (2,4)]
[(3,1) ; (3,2) ; (3,3) ; (3,4)]
[(4,1) ; (4,2) ; (4,3) ; (4,4)]
```

### 2.3.3.1. Definisikan nilai dari setiap (row,col)

Pada bagian ini user akan mengisi nilai dari tiap kolom dan barisnya dimana yang diisi adalah nilai yang dianggap sudah benar dimana tidak boleh ada nilai yang berulang dan nilai yang diinput adalah bebas, selain itu akan diisi dengan 0 dan mengisi total nilai setiap baris,kolom, dan diagonalnya.

```
satuA = input("Masukkan Angka Kotak 1,1 = ")
print(type(satuA))
a = int(satuA)
satuB = input("Masukkan Angka Kotak 1,2 = ")
print(type(satuB))
b = int(satuB)
satuC = input("Masukkan Angka Kotak 1,3 = ")
print(type(satuC))
c = int(satuC)
satuD = input("Masukkan Angka Kotak 1,4 = ")
print(type(satuD))
d = int(satuD)

duaA = input("Masukkan Angka Kotak 2,1 = ")
print(type(duaA))
e = int(duaA)
duaB = input("Masukkan Angka Kotak 2,2 = ")
print(type(duaB))
f = int(duaB)
duaC = input("Masukkan Angka Kotak 2,3 = ")
print(type(duaC))
g = int(duaC)
duaD = input("Masukkan Angka Kotak 2,4 = ")
print(type(duaD))
h = int(duaD)

tigaA = input("Masukkan Angka Kotak 3,1 = ")
print(type(tigaA))
i = int(tigaA)
tigaB = input("Masukkan Angka Kotak 3,2 = ")
print(type(tigaB))
j = int(tigaB)
tigaC = input("Masukkan Angka Kotak 3,3 = ")
print(type(tigaC))
k = int(tigaC)
tigaD = input("Masukkan Angka Kotak 3,4 = ")
print(type(tigaD))
l = int(tigaD)

empatA = input("Masukkan Angka Kotak 4,1 = ")
print(type(empatA))
m = int(empatA)
empatB = input("Masukkan Angka Kotak 4,2 = ")
print(type(empatB))
n = int(empatB)
empatC = input("Masukkan Angka Kotak 4,3 = ")
print(type(empatC))
o = int(empatC)
empatD = input("Masukkan Angka Kotak 4,4 = ")
print(type(empatD))
p = int(empatD)
total = input("Masukkan Total yang diinginkan : ")
print(type(total))
totalnilai = int(total)
```



Tampilannya adalah sebagai berikut:

```
Masukkan Angka Kotak 1,1 = 0
<class 'str'>
Masukkan Angka Kotak 1,2 = 0
<class 'str'>
Masukkan Angka Kotak 1,3 = 14
<class 'str'>
Masukkan Angka Kotak 1,4 = 4
<class 'str'>
Masukkan Angka Kotak 2,1 = 12
<class 'str'>
Masukkan Angka Kotak 2,2 = 0
<class 'str'>
Masukkan Angka Kotak 2,3 = 0
<class 'str'>
Masukkan Angka Kotak 2,4 = 9
<class 'str'>
Masukkan Angka Kotak 3,1 = 0
<class 'str'>
Masukkan Angka Kotak 3,2 = 10
<class 'str'>
Masukkan Angka Kotak 3,3 = 11
<class 'str'>
Masukkan Angka Kotak 3,4 = 0
<class 'str'>
Masukkan Angka Kotak 4,1 = 13
<class 'str'>
Masukkan Angka Kotak 4,2 = 3
<class 'str'>
Masukkan Angka Kotak 4,3 = 2
<class 'str'>
```

### 2.3.3.2. Fungsi Pengecek total nilai setiap baris, kolom, dan diagonal

Pada bagian tersebut setiap baris, kolom dan diagonal akan diperiksa apakah setiap baris, kolom, dan diagonal sudah sesuai dengan total nilai yang telah diinput oleh user sebelumnya. Dimana pemeriksaan dilakukan sebanyak 4 kali.

```
kotakmsp = []
kotakmsp.append([a,b,c,d])
kotakmsp.append([e,f,g,h])
kotakmsp.append([i,j,k,l])
kotakmsp.append([m,n,o,p])

def checkingMSP(kotakmsp):
    global totalnilai
    for row in range(0,4):
        for col in range(0,4):
            if kotakmsp[row][col]==0:
                return False
    for row in range(0,4):
        if (kotakmsp[row][0]+kotakmsp[row][1]+kotakmsp[row][2]+kotakmsp[row][3])!=totalnilai:
            return False
    for col in range(0,4):
        if (kotakmsp[0][col]+kotakmsp[1][col]+kotakmsp[2][col]+kotakmsp[3][col])!=totalnilai:
            return False
    if (kotakmsp[0][0]+kotakmsp[1][1]+kotakmsp[2][2]+kotakmsp[3][3])!=totalnilai:
        return False
    if (kotakmsp[0][3]+kotakmsp[1][2]+kotakmsp[2][1]+kotakmsp[3][0])!=totalnilai:
        return False
    return True
```

### 3.2.3.2. Fungsi untuk melakukan penyelesaian dengan mencari kotak yang kosong

Pada bagian tersebut akan dilakukan penyelesaian dengan cara memeriksa setiap baris dan kolom apakah nilai yang akan diisi sudah memenuhi atau tidak. Jika sedang dilakukan pemeriksaan maka akan menampilkan “Nilai pada kotak-kotak sedang diperiksa” dimana batas pemeriksaanyang dilakukan sebanyak 16 kali pada setiap kotaknya. Jikalau baris ataupun kolom sudah selesai diperiksa akan menampilkan “Nilai pada kotak-kotak sudah selesai diperiksa”. Kemudian akan menampilkan baris dan kolom yang sedang diperiksa.”

```
def penyelesaiantkt(kotakmsp):
    for i in range(0,16):
        row=i//4
        col=i%4
        if kotakmsp[row][col]==0:
            print("Nilai pada kotak-kotak sedang diperiksa")
            for value in range (1,17):
                if not(value in kotakmsp[0] or value in kotakmsp[1] or value in kotakmsp[2] or value in kotakmsp[3]):
                    kotakmsp[row][col]= value
                    if checkingMSP(kotakmsp):
                        print("Nilai pada kotak-kotak sudah selesai diperiksa.")
                        return True
            else:
                if penyelesaiantkt(kotakmsp):
                    return True
        break
    print(kotakmsp)
    kotakmsp[row][col] = 0
```

Tampilannya adalah sebagai berikut:

```
Nilai pada kotak-kotak sedang diperiksa
Nilai pada kotak-kotak sedang diperiksa
Nilai pada kotak-kotak sedang diperiksa
Nilai pada kotak-kotak sedang diperiksa
Nilai pada kotak-kotak sedang diperiksa
[[1, 5, 14, 4], [12, 6, 7, 9], [8, 10, 11, 15], [13, 3, 2, 16]]
[[1, 5, 14, 4], [12, 6, 7, 9], [8, 10, 11, 15], [13, 3, 2, 0]]
Nilai pada kotak-kotak sedang diperiksa
[[1, 5, 14, 4], [12, 6, 7, 9], [8, 10, 11, 16], [13, 3, 2, 15]]
[[1, 5, 14, 4], [12, 6, 7, 9], [8, 10, 11, 16], [13, 3, 2, 0]]
[[1, 5, 14, 4], [12, 6, 7, 9], [8, 10, 11, 16], [13, 3, 2, 0]]
Nilai pada kotak-kotak sedang diperiksa
Nilai pada kotak-kotak sedang diperiksa
[[1, 5, 14, 4], [12, 6, 7, 9], [15, 10, 11, 8], [13, 3, 2, 16]]
[[1, 5, 14, 4], [12, 6, 7, 9], [15, 10, 11, 8], [13, 3, 2, 0]]
Nilai pada kotak-kotak sedang diperiksa
[[1, 5, 14, 4], [12, 6, 7, 9], [15, 10, 11, 16], [13, 3, 2, 8]]
[[1, 5, 14, 4], [12, 6, 7, 9], [15, 10, 11, 16], [13, 3, 2, 0]]
[[1, 5, 14, 4], [12, 6, 7, 9], [15, 10, 11, 16], [13, 3, 2, 0]]
Nilai pada kotak-kotak sedang diperiksa
Nilai pada kotak-kotak sedang diperiksa
[[1, 5, 14, 4], [12, 6, 7, 9], [16, 10, 11, 8], [13, 3, 2, 15]]
[[1, 5, 14, 4], [12, 6, 7, 9], [16, 10, 11, 8], [13, 3, 2, 0]]
Nilai pada kotak-kotak sedang diperiksa
[[1, 5, 14, 4], [12, 6, 7, 9], [16, 10, 11, 15], [13, 3, 2, 8]]
[[1, 5, 14, 4], [12, 6, 7, 9], [16, 10, 11, 15], [13, 3, 2, 0]]
[[1, 5, 14, 4], [12, 6, 7, 9], [16, 10, 11, 15], [13, 3, 2, 0]]
[[1, 5, 14, 4], [12, 6, 7, 9], [16, 10, 11, 0], [13, 3, 2, 0]]
Nilai pada kotak-kotak sedang diperiksa
```

### 2.3.3.3. Fungsi Penyelesaian

Pada bagian ini akan dilakukan tahap penyelesaian dari MSP tersebut. Jika MSP tersebut makan akan ditampilkan grid penyelesaiannya, jikalau tidak dapat diselesaikan maka akan menampilkan “Magic Square Puzzle tidak dapat diselesaikan. Silahkan run kembali.”.

```
solved = penyelesaian_tkt(kotakmsp_tkt)
if solved:
    print("Hasil Penyelesaian:")
    print (" [ {} ] [ {} ] [ {} ]".format(kotakmsp_tkt[0][0], kotakmsp_tkt[0][1], kotakmsp_tkt[0][2]))
    print (" [ {} ] [ {} ] [ {} ]".format(kotakmsp_tkt[1][0], kotakmsp_tkt[1][1], kotakmsp_tkt[1][2]))
    print (" [ {} ] [ {} ] [ {} ]".format(kotakmsp_tkt[2][0], kotakmsp_tkt[2][1], kotakmsp_tkt[2][2]))
else:
    print("Magic Square Puzzle tidak dapat diselesaikan. Silahkan run kembali.")
```

Tampilannya adalah sebagai berikut:

```
Hasil Penyelesaian:
[ 1 ] [ 15 ] [ 14 ] [ 4 ]
[ 12 ] [ 6 ] [ 7 ] [ 9 ]
[ 8 ] [ 10 ] [ 11 ] [ 5 ]
[ 13 ] [ 3 ] [ 2 ] [ 16 ]
```

### 2.3.4. Fungsi Grid 5x5

Ketika user mengisi jumlah grid yang akan digunakan yaitu 5 maka akan tampil grid yang diinginkan.

```
print("Susunan Kotak ")
print("[ (1,1) ; (1,2) ; (1,3) ; (1,4) ; (1,5) ]")
print("[ (2,1) ; (2,2) ; (2,3) ; (2,4) ; (2,5) ]")
print("[ (3,1) ; (3,2) ; (3,3) ; (3,4) ; (3,5) ]")
print("[ (4,1) ; (4,2) ; (4,3) ; (4,4) ; (4,5) ]")
print("[ (5,1) ; (5,2) ; (5,3) ; (5,4) ; (5,5) ]")
```

Tampilan gridnya seperti gambar dibawah ini

```
Susunan Kotak
[ (1,1) ; (1,2) ; (1,3) ; (1,4) ; (1,5) ]
[ (2,1) ; (2,2) ; (2,3) ; (2,4) ; (2,5) ]
[ (3,1) ; (3,2) ; (3,3) ; (3,4) ; (3,5) ]
[ (4,1) ; (4,2) ; (4,3) ; (4,4) ; (4,5) ]
[ (5,1) ; (5,2) ; (5,3) ; (5,4) ; (5,5) ]
```

#### 2.3.4.1. Definisikan nilai dari setiap (row,col)

Pada bagian ini user akan mengisi nilai dari tiap kolom dan barisnya dimana yang diisi adalah nilai yang dianggap sudah benar dimana tidak boleh ada nilai yang berulang dan nilai yang

diinput adalah bebas. Selain itu akan diisi dengan 0 dan mengisi total nilai setiap baris,kolom, dan diagonalnya.

```
satusatu = input("Masukkan Angka Kotak 1,1 = ")
print(type(satusatu))
a = int(satusatu)
satudua = input("Masukkan Angka Kotak 1,2 = ")
print(type(satudua))
b = int(satudua)
satutiga = input("Masukkan Angka Kotak 1,3 = ")
print(type(satutiga))
c = int(satutiga)
satuempat = input("Masukkan Angka Kotak 1,4 = ")
print(type(satuempat))
d = int(satuempat)
satulima = input("Masukkan Angka Kotak 1,5 = ")
print(type(satulima))
e = int(satulima)

duasatu = input("Masukkan Angka Kotak 2,1 = ")
print(type(duasatu))
f = int(duasatu)
duadua = input("Masukkan Angka Kotak 2,2 = ")
print(type(duadua))
g = int(duadua)
duatiga = input("Masukkan Angka Kotak 2,3 = ")
print(type(duatiga))
h = int(duatiga)
duaempat = input("Masukkan Angka Kotak 2,4 = ")
print(type(duaempat))
i = int(duaempat)
dualima = input("Masukkan Angka Kotak 2,5 = ")
print(type(dualima))
j = int(dualima)

tigasatu = input("Masukkan Angka Kotak 3,1 = ")
print(type(tigasatu))
k = int(tigasatu)
tigadua = input("Masukkan Angka Kotak 3,2 = ")
print(type(tigadua))
l = int(tigadua)
tigatiga = input("Masukkan Angka Kotak 3,3 = ")
print(type(tigatiga))
m = int(tigatiga)
tigaempat = input("Masukkan Angka Kotak 3,4 = ")
print(type(tigaempat))
n = int(tigaempat)
tigalima = input("Masukkan Angka Kotak 3,5 = ")
print(type(tigalima))
o = int(tigalima)
```

```

empatsatu = input("Masukkan Angka Kotak 4,1 = ")
print(type(empatsatu))
p = int(empatsatu)
empatdua = input("Masukkan Angka Kotak 4,2 = ")
print(type(empatdua))
q = int(empatdua)
empattiga = input("Masukkan Angka Kotak 4,3 = ")
print(type(empattiga))
r = int(empattiga)
empatempat = input("Masukkan Angka Kotak 4,4 = ")
print(type(empatempat))
s = int(empatempat)
empatlima = input("Masukkan Angka Kotak 4,5 = ")
print(type(empatlima))
t = int(empatlima)

limasatu = input("Masukkan Angka Kotak 5,1 = ")
print(type(limasatu))
u = int(limasatu)
limadua = input("Masukkan Angka Kotak 5,2 = ")
print(type(limadua))
v = int(limadua)
limatiga = input("Masukkan Angka Kotak 5,3 = ")
print(type(limatiga))
w = int(limatiga)
limaempat = input("Masukkan Angka Kotak 5,4 = ")
print(type(limaempat))
x = int(limaempat)
limalima = input("Masukkan Angka Kotak 5,5 = ")
print(type(limalima))
y = int(limalima)

total = input("Masukkan Total yang diinginkan : ")
print(type(total))
totalnilai = int(total)

```

Tampilannya adalah sebagai berikut:

```

Masukkan Angka Kotak 1,1 = 17
<class 'str'>
Masukkan Angka Kotak 1,2 = 24
<class 'str'>
Masukkan Angka Kotak 1,3 = 1
<class 'str'>
Masukkan Angka Kotak 1,4 = 8
<class 'str'>
Masukkan Angka Kotak 1,5 = 15
<class 'str'>
Masukkan Angka Kotak 2,1 = 23
<class 'str'>
Masukkan Angka Kotak 2,2 = 5
<class 'str'>
Masukkan Angka Kotak 2,3 = 7
<class 'str'>
Masukkan Angka Kotak 2,4 = 14
<class 'str'>
Masukkan Angka Kotak 2,5 = 16
<class 'str'>
Masukkan Angka Kotak 3,1 = 4
<class 'str'>
Masukkan Angka Kotak 3,2 = 6
<class 'str'>
Masukkan Angka Kotak 3,3 = 13
<class 'str'>
Masukkan Angka Kotak 3,4 = 20
<class 'str'>
Masukkan Angka Kotak 3,5 = 22
<class 'str'>
Masukkan Angka Kotak 4,1 = 10
<class 'str'>
Masukkan Angka Kotak 4,2 = 12
<class 'str'>

```

### 2.3.4.2. Fungsi Pengecek total nilai setiap baris, kolom, dan diagonal

Pada bagian tersebut setiap baris, kolom dan diagonal akan diperiksa apakah setiap baris, kolom, dan diagonal sudah sesuai dengan total nilai yang telah diinput oleh user sebelumnya. Dimana pengecekan setiap kotak tersebut sebanyak 5 kali.

```
def checkingMSP(kotakmsp):
    global totalnilai
    for row in range(0,5):
        for col in range(0,5):
            if kotakmsp[row][col]==0:
                return False
    for row in range(0,5):
        if (kotakmsp[row][0]+kotakmsp[row][1]+kotakmsp[row][2]+kotakmsp[row][3]+kotakmsp[row][4])!=totalnilai:
            return False
    for col in range(0,4):
        if (kotakmsp[0][col]+kotakmsp[1][col]+kotakmsp[2][col]+kotakmsp[3][col]+kotakmsp[4][col])!=totalnilai:
            return False
    if (kotakmsp[0][0]+kotakmsp[1][1]+kotakmsp[2][2]+kotakmsp[3][3]+kotakmsp[4][4])!=totalnilai:
        return False
    if (kotakmsp[0][4]+kotakmsp[1][3]+kotakmsp[2][2]+kotakmsp[3][1]+kotakmsp[4][0])!=totalnilai:
        return False
    return True
```

### 2.3.4.3. Fungsi untuk melakukan penyelesaian dengan mencari kotak yang kosong

Pada bagian tersebut akan dilakukan penyelesaian dengan cara memeriksa setiap baris dan kolom apakah nilai yang akan diisi sudah memenuhi atau tidak. Jika sedang dilakukan pemeriksaan maka akan menampilkan “Nilai pada kotak-kotak sedang diperiksa” dimana pemeriksaan dilakukan sebanyak 25 kali pada setiap kotaknya. Jikalau baris ataupun kolom sudah selesai diperiksa akan menampilkan “Nilai pada kotak-kotak sudah selesai diperiksa. Kemudian akan menampilkan baris dan kolom yang sedang diperiksa.

```
def penyelesaiantkt(kotakmsp):
    for i in range(0,25):
        row=i//5
        col=i%5
        if kotakmsp[row][col]==0:
            print("Nilai pada kotak-kotak sedang diperiksa")
            for value in range (1,26):
                if not(value in kotakmsp[0] or value in kotakmsp[1] or value in kotakmsp[2] or value in kotakmsp[3] or value in kotakmsp[4]):
                    kotakmsp[row][col]= value
                    if checkingMSP(kotakmsp):
                        print("Nilai pada kotak-kotak sudah selesai diperiksa.")
                        return True
            else:
                if penyelesaiantkt(kotakmsp):
                    return True
        break
    print(kotakmsp)
```

Tampilannya adalah sebagai berikut:

```
class Solusi:
    Nilai pada kotak-kotak sedang diperiksa
    Nilai pada kotak-kotak sedang diperiksa
    Nilai pada kotak-kotak sedang diperiksa
    [[17, 24, 1, 8, 15], [23, 5, 7, 14, 16], [4, 6, 13, 20, 22], [10, 12, 19, 21, 3], [9, 18, 25, 2, 11]]
    [[17, 24, 1, 8, 15], [23, 5, 7, 14, 16], [4, 6, 13, 20, 22], [10, 12, 19, 21, 3], [9, 18, 25, 2, 0]]
    Nilai pada kotak-kotak sedang diperiksa
    Nilai pada kotak-kotak sudah selesai diperiksa.
```

#### 2.3.4.4. Fungsi Penyelesaian

Pada bagian ini akan dilakukan tahap penyelesaian dari MSP tersebut. Jika MSP berhasil diselesaikan maka akan ditampilkan grid penyelesaiannya, jika tidak dapat diselesaikan maka akan menampilkan “Magic Square Puzzle tidak dapat diselesaikan. Silahkan run kembali.”.

```
solved = penyelesaian_tkt(kotakmsp)
if solved:
    print("Hasil Penyelesaian:")
    print ("[ {} ] [ {} ] [ {} ] [ {} ] [ {} ]".format(kotakmsp[0][0],
    kotakmsp[0][1], kotakmsp[0][2], kotakmsp[0][3], kotakmsp[0][4]))
    print ("[ {} ] [ {} ] [ {} ] [ {} ] [ {} ]".format(kotakmsp[1][0],
    kotakmsp[1][1], kotakmsp[1][2], kotakmsp[1][3], kotakmsp[1][4]))
    print ("[ {} ] [ {} ] [ {} ] [ {} ] [ {} ]".format(kotakmsp[2][0],
    kotakmsp[2][1], kotakmsp[2][2], kotakmsp[2][3], kotakmsp[2][4]))
    print ("[ {} ] [ {} ] [ {} ] [ {} ] [ {} ]".format(kotakmsp[3][0],
    kotakmsp[3][1], kotakmsp[3][2], kotakmsp[3][3], kotakmsp[3][4]))
    print ("[ {} ] [ {} ] [ {} ] [ {} ] [ {} ]".format(kotakmsp[4][0],
    kotakmsp[4][1], kotakmsp[4][2], kotakmsp[4][3], kotakmsp[4][4]))
else:
    print("Tidak dapat diselesaikan!")
```

Tampilannya adalah sebagai berikut:

```
Hasil Penyelesaian:
[ 17 ] [ 24 ] [ 1 ] [ 8 ] [ 15 ]
[ 23 ] [ 5 ] [ 7 ] [ 14 ] [ 16 ]
[ 4 ] [ 6 ] [ 13 ] [ 20 ] [ 22 ]
[ 10 ] [ 12 ] [ 19 ] [ 21 ] [ 3 ]
[ 11 ] [ 18 ] [ 25 ] [ 2 ] [ 9 ]
```

## 2.4. Evaluasi

Setelah melakukan pengujian pada aplikasi *Magic Square Puzzle*, kami menemukan beberapa evaluasi untuk aplikasi ini. Berikut ini adalah evaluasi yang telah kami rancang.

1. Kompleksitas pada masing-masing jumlah kotak berbeda. Semakin tinggi jumlah kotak, semakin lama program akan berjalan.

| Grid  | n | CheckingMSP<br>( $2n^2 + 4n + 5$ ) | PenyelesaianTKD<br>( $50n^2(n^2+1)$ ) | Total (Waktu) |
|-------|---|------------------------------------|---------------------------------------|---------------|
| 3 X 3 | 3 | 52                                 | 4500                                  | 4552          |
| 4 X 4 | 4 | 101                                | 13600                                 | 13701         |
| 5 X 5 | 5 | 155                                | 32500                                 | 33655         |

```
def checkingMSP(kotakmsp):  
    global totalnilai  
    for row in range(0,4):  
        for col in range(0,4):  
            if kotakmsp[row][col]==0:  
                return False  
        for row in range(0,4):  
            if (kotakmsp[row][0]+kotakmsp[row][1]+kotakmsp[row][2]+kotakmsp[row][3])!=totalnilai:  
                return False  
        for col in range(0,4):  
            if (kotakmsp[0][col]+kotakmsp[1][col]+kotakmsp[2][col]+kotakmsp[3][col])!=totalnilai:  
                return False  
        if (kotakmsp[0][0]+kotakmsp[1][1]+kotakmsp[2][2]+kotakmsp[3][3])!=totalnilai:  
            return False  
        if (kotakmsp[0][3]+kotakmsp[1][2]+kotakmsp[2][1]+kotakmsp[3][0])!=totalnilai:  
            return False  
    return True
```

From 0 to n

From 0 to m

From 0 to n

From 0 to m



```

def penyelesaian_tkt(kotakmsp):
    for i in range(0,16):
        row=i//4
        col=i%4
        if kotakmsp[row][col]==0:
            print("Nilai pada kotak-kotak sedang diperiksa")
            for value in range (1,17):
                if not(value in kotakmsp[0] or value in kotakmsp[1]
or value in kotakmsp[2] or value in kotakmsp[3]):
                    kotakmsp[row][col]= value
                    if checkingMSP(kotakmsp):
                        print("Nilai pada kotak-
kotak sudah selesai diperiksa.")
                        return True
                    else:
                        if penyelesaian_tkt(kotakmsp):
                            return True
            break
    print(kotakmsp)
    kotakmsp[row][col] = 0

```

From 1 to n\*n

From value to  
n\*n+

2. Ketika angka yang dimasukkan ke dalam grid sudah lengkap dan sesuai dengan *constraint*, pesan yang ditampilkan sama dengan pesan error untuk permasalahan *puzzle* yang tidak bisa diselesaikan oleh program.

## **BAB 3**

### **Penutup**

#### **3.1. Pembagian Tugas**

Berikut ini adalah penjabaran mengenai tugas masing-masing anggota kelompok yang dilaksanakan oleh tim selama proyek berlangsung.

Tabel 1. Pembagian Tugas Anggota Tim Proyek

| No | Tugas   | Nama   |
|----|---|--|
| 1. | Pertemuan: Pemilihan Algoritma yang menjadi solusi      | Lusiana Siahaan, Yohana Simatupang, Maria Nababan, Roy Napitupulu, Letare Saragih, Grace Panjaitan |
| 2. | Proposal: Pendokumentasian Penggunaan Konsep Kecerdasan | Roy Napitupulu, Maria Nababan  |
| 3. | Proposal: Pendokumentasian Flowchart dan Penjelasan     | Lusiana Siahaan, Grace Panjaitan, Letare Saragih   |
| 4. | Proposal: Penulisan Pendahuluan                         | Lusiana Siahaan, Yohana Simatupang, Maria Nababan, Roy Napitupulu, Letare Saragih, Grace Panjaitan |
| 5. | Proposal: Daftar Isi                                    | Yohana Polin   |
| 6. | Proposal: Daftar Pustaka                                | Letare Saragih   |
| 7. | Proposal: Gantt Chart dan Jadwal                        | Yohana Polin   |
| 8. | Proposal: Pembagian Tugas                               | Lusiana Siahaan  |
| 9. | Proposal: Finalisasi                                    | Lusiana Siahaan, Yohana Simatupang, Maria Nababan, Roy Napitupulu, Letare Saragih,                 |

|     |   |  |
|-----|---|--|
|     |   | Grace Panjaitan  |
| 10. | Program: Pembuatan GitHub Repository dan <i>issue</i> masing-masing anggota tim | Lusiana Siahaan  |
| 11. | Program: Pembuatan Desain   | Roy Napitupulu   |
| 12. | Program: Implementasi Program   | Lusiana Siahaan, Yohana Simatupang, Maria Nababan, Roy Napitupulu, Letare Saragih, Grace Panjaitan |
| 13. | Program: Testing and Repair   | Lusiana Siahaan, Yohana Simatupang, Maria Nababan, Roy Napitupulu, Letare Saragih, Grace Panjaitan |
| 14. | Presentase Akhir  | Lusiana Siahaan, Yohana Simatupang, Maria Nababan, Roy Napitupulu, Letare Saragih, Grace Panjaitan |
| 15. | Pertemuan: Pembahasan terkait konten laporan akhir                              | Lusiana Siahaan, Yohana Simatupang, Maria Nababan, Roy Napitupulu, Letare Saragih, Grace Panjaitan |
| 16. | Laporan Akhir: Pembuatan Sampul   |  |
| 16. | Laporan Akhir: Bab Pendahuluan  | Letare Saragih, Grace Panjaitan  |
| 17. | Laporan Akhir: Bab Isi  | Lusiana Siahaan, Yohana Simatupang, Maria Nababan, Roy Napitupulu, Letare Saragih, Grace Panjaitan |
| 18. | Laporan Akhir: Bab Penutup  | Lusiana Siahaan, Yohana Polin  |

### 3.2. Kesimpulan

Pada kondisi ini persoalan Magic Square Puzzle akan diselesaikan secara komputasi yaitu menggunakan algoritma Constraint Satisfaction Problem (CSP). Spesifik metode yang digunakan adalah algoritma Backtracking. Hal ini dikarenakan Magic Square Puzzle dapat diselesaikan saat posisi variabel dalam kotak sudah memenuhi kondisi atau batasan-batasan tertentu yang nantinya batasan-batasan ini akan didefinisikan menggunakan algoritma CSP. Metode Backtracking dipilih karena Backtracking merupakan algoritma rekursif yang mencoba memecahkan masalah tertentu dengan menguji semua kemungkinan jalur menuju solusi hingga solusi ditemukan. Setiap kali jalur diuji, jika solusi tidak ditemukan, algoritma melakukan backtrack untuk menguji jalur lain yang mungkin dan seterusnya sampai solusi ditemukan atau semua jalur telah diuji. Berdasarkan perhitungan kompleksitas yang dilakukan untuk ketiga kode program (implementasikan grid 3 x 3, grid 4 x 4, dan grid 5 x 5), dapat dilihat bahwa magic square puzzle dengan grid 3 x 3 lebih baik karena memiliki kompleksitas paling tinggi yaitu  $n^4$  sehingga waktu eksekusi yang lebih cepat.

### 3.3 Saran

Sistem ini memiliki kekurangan, sehingga masih diperlukan saran perbaikan sistem yang membantu sistem dalam pengembangannya. Adapun saran-saran tersebut yaitu:

1. Sebaiknya sistem penyelesaian persoalan Magic Square Puzzle dapat dikembangkan lebih lanjut dengan menambah jumlah grid yang diimplementasikan dan penggunaan metode penyelesaian lainnya.
2. Sebaiknya tampilan antarmuka sistem penyelesaian persoalan Magic Square Puzzle dapat lebih dikembangkan agar tampilan antarmuka sistem tersebut terlihat lebih menarik.

## Daftar Pustaka

Bhavya Gupta. (2016). Generalized Form of a 4x4 Magic Square[online]. Available: [https://www.academia.edu/21053073/Generalized Form of a 4x4 Magic Square](https://www.academia.edu/21053073/Generalized_Form_of_a_4x4_Magic_Square)

Bryan Yu. (2020). CS50's Introduction To Artificial Intelligence with Python[online]. Available: <https://cs50.harvard.edu/ai/2020/notes/3/#:~:text=Brian%20Yu>

Farhan Makarim. (2017). Analisis Penggunaan Algoritma Backtracking dalam Penjadwalan Kuliah[online]. Available: <http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2016-2017/Makalah2017/Makalah-IF2211-2017-003.pdf>

Hendry.(2010).Mengenal Magic Square [online]. Available: <http://hendrydext.blogspot.com/2010/01/mengenal-magic-square.html>

Maisaroh.(2016).CSP(Constraint Satisfaction Problems)[online]. Available: <http://maisarohmae23.blogspot.com/2016/10/csp-constraint-satisfaction-problem.html>

Samuel I.G Situmeang S.TI.,M.Sc. (2020). Constraint Satisfaction Problems[online]. Available: <https://cis.del.ac.id>

Techfor ID.(2020). CLI VS GUI : Mana yang Lebih Baik Menurutmu ?[online]. Available: <https://www.techfor.id/cli-vs-gui-mana-yang-lebih-baik-menurutmu/>