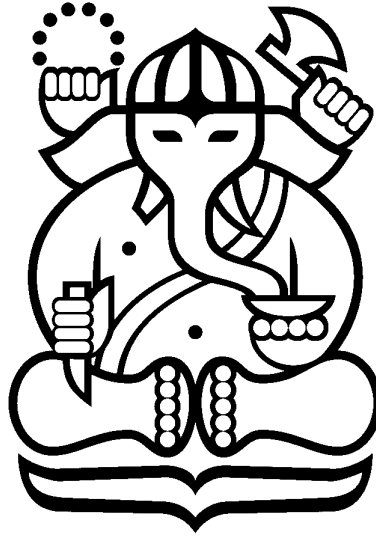


LAPORAN TUGAS KECIL 1
IF2211 STRATEGI ALGORITMA 2024/2025



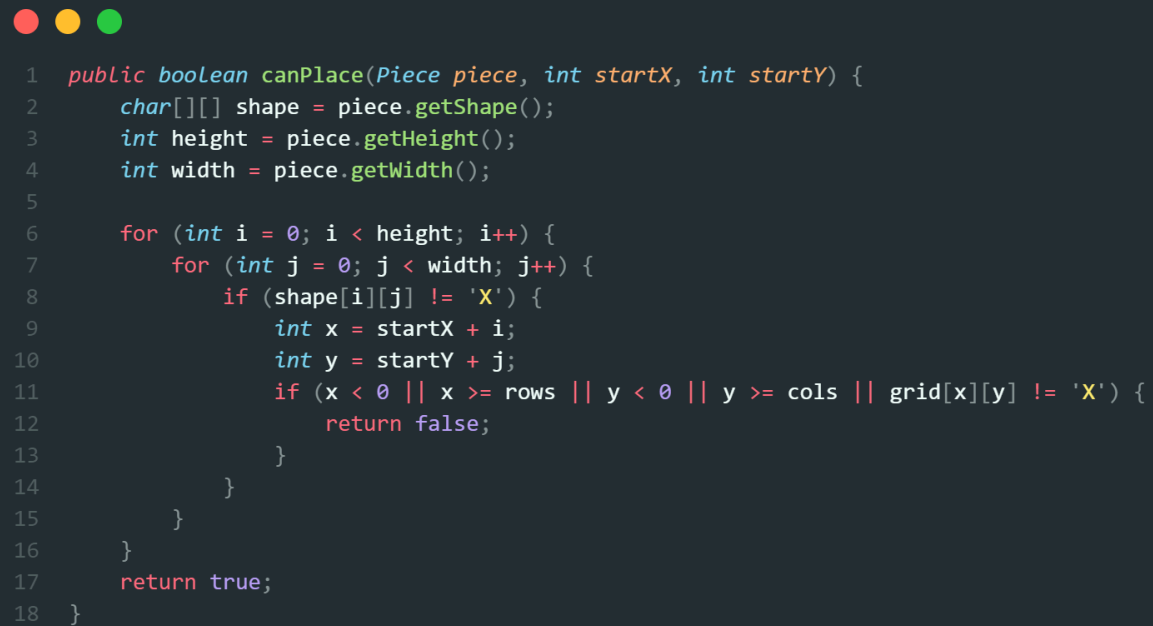
Oleh:
Muhammad Edo Raduputu Aprima
13523096

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132
2025

Implementasi Algoritma *Brute Force*

Program menggunakan Algoritma *Brute Force* untuk mencoba semua kemungkinan posisi setiap *piece* pada *board*. Dengan menggunakan algoritma *brute force*, dapat dijamin ditemukan solusi atau tidaknya karena semua kemungkinan akan ditinjau.


Cara kerja algoritma pada program ini dapat dibagi menjadi 3 bagian, yaitu inisialisasi, proses *brute force*, dan kondisi selesai.



```
1 public boolean canPlace(Piece piece, int startX, int startY) {
2     char[][] shape = piece.getShape();
3     int height = piece.getHeight();
4     int width = piece.getWidth();
5
6     for (int i = 0; i < height; i++) {
7         for (int j = 0; j < width; j++) {
8             if (shape[i][j] != 'X') {
9                 int x = startX + i;
10                int y = startY + j;
11                if (x < 0 || x >= rows || y < 0 || y >= cols || grid[x][y] != 'X') {
12                    return false;
13                }
14            }
15        }
16    }
17    return true;
18 }
```

Gambar 1.1 Source code canPlace()

Metode canPlace() digunakan untuk mengecek apakah sebuah *piece* dapat ditempatkan pada suatu posisi atau tidak, metode ini memastikan bahwa *piece* tidak keluar dari board. Selain itu, metode ini juga memastikan bahwa tidak ada *piece* yang saling tumpah tindih dengan *piece* lainnya.




```

1  public void placePiece(Piece piece, int startX, int startY) {
2      char[][] shape = piece.getShape();
3      int height = piece.getHeight();
4      int width = piece.getWidth();
5      char label = piece.getLabel();
6
7      for (int i = 0; i < height; i++) {
8          for (int j = 0; j < width; j++) {
9              if (shape[i][j] != 'X') {
10                 grid[startX + i][startY + j] = label;
11             }
12         }
13     }
14 }

```

Gambar 1.2 *Source code placePiece()*

Jika didapatkan sebuah posisi dimana sebuah *piece* dapat ditempatkan, maka metode ini akan menempatkan label *piece* pada sel *board* di posisi tersebut, hal ini akan diiterasi ada setiap sel *piece*.



```

1  public void removePiece(Piece piece, int startX, int startY) {
2      char[][] shape = piece.getShape();
3      int height = piece.getHeight();
4      int width = piece.getWidth();
5
6      for (int i = 0; i < height; i++) {
7          for (int j = 0; j < width; j++) {
8              if (shape[i][j] != 'X') {
9                  grid[startX + i][startY + j] = 'X';
10             }
11         }
12     }
13 }

```

Gambar 1.3 *Source code removePiece()*

Ketika solusi tidak ditemukan, maka algoritma akan mencoba posisi lain. Oleh karena itu dibutuhkan metode untuk menghapus *piece* dari *board* dengan mengembalikan sel ke bentuk awal atau inisialisasinya yaitu 'X'.



```
1  private boolean isBoardFullyCovered() {
2      for (int i = 0; i < board.getRows(); i++) {
3          for (int j = 0; j < board.getCols(); j++) {
4              if (board.getCell(i, j) == 'X') {
5                  return false;
6              }
7          }
8      }
9      return true;
10 }
```

Gambar 1.4 *Source code* isBoardFullyCovered()

Metode ini digunakan untuk mengecek apakah seluruh sel pada *board* sudah terisi atau belum, hal ini penting dilakukan dalam permainan puzzle ini, karena ketika semua *piece* sudah berhasil ditempatkan pada suatu posisi, kita perlu memeriksa apakah *pieces* tersebut sudah menutupi keseluruhan *board* atau belum.

```

1  public List<Piece> getAllTransformations() {
2      Set<String> seenShapes = new HashSet<>();
3      List<Piece> transformations = new ArrayList<>();
4
5      char[][] currentShape = shape;
6      for (int i = 0; i < 4; i++) {
7          currentShape = rotate90(currentShape);
8          addUniqueTransformation(transformations, seenShapes, new Piece(currentShape, label));
9      }
10
11     currentShape = mirror(shape);
12     for (int i = 0; i < 4; i++) {
13         currentShape = rotate90(currentShape);
14         addUniqueTransformation(transformations, seenShapes, new Piece(currentShape, label));
15     }
16
17     return transformations;
18 }

```

Gambar 1.5 Source code getAllTransformation()

```

1  private char[][] rotate90(char[][] shape) {
2      int h = shape.length;
3      int w = shape[0].length;
4      char[][] rotated = new char[w][h];
5
6      for (int i = 0; i < h; i++) {
7          for (int j = 0; j < w; j++) {
8              rotated[j][h - 1 - i] = shape[i][j];
9          }
10     }
11     return rotated;
12 }
13
14 private char[][] mirror(char[][] shape) {
15     int h = shape.length;
16     int w = shape[0].length;
17     char[][] mirrored = new char[h][w];
18
19     for (int i = 0; i < h; i++) {
20         for (int j = 0; j < w; j++) {
21             mirrored[i][w - 1 - j] = shape[i][j];
22         }
23     }
24     return mirrored;
25 }

```

Gambar 1.6 Source code rotate90() dan mirror()

Untuk mendapatkan semua kemungkinan posisi *piece* pada *board*, maka perlu metode untuk mendapatkan semua kemungkinan transformasi *piece*, pertama bentuk *piece* asli dapat di rotasi 90 derajat sebanyak empat kali, lalu bentuk *mirror*-nya juga di rotasi empat kali.

```
1 public boolean solve(int pieceIndex) {
2     iterationCount++;
3     if (pieceIndex == pieces.size()) {
4         return isBoardFullyCovered();
5     }
6
7     Piece piece = pieces.get(pieceIndex);
8     List<Piece> transformedPieces = piece.getAllTransformations();
9
10    for (Piece transformed : transformedPieces) {
11        for (int row = 0; row <= board.getRows() - transformed.getHeight(); row++) {
12            for (int col = 0; col <= board.getCols() - transformed.getWidth(); col++) {
13                if (board.canPlace(transformed, row, col)) {
14                    board.placePiece(transformed, row, col);
15                    if (solve(pieceIndex + 1)) {
16                        return true;
17                    }
18                    board.removePiece(transformed, row, col);
19                }
20            }
21        }
22    }
23    return false;
24 }
```

Gambar 1.7 Source code solve()

solve() adalah metode inti dalam implementasi *brute force* pada program ini, metode ini akan mencoba menempatkan setiap *piece* secara berurutan. Setiap *piece* yang sedang dicoba untuk ditempatkan akan diiterasi semua kemungkinan transformasinya, apabila suatu *piece* dapat ditempatkan, maka *piece* tersebut ditempatkan pada *board* lalu akan lanjut ke iterasi *piece* selanjutnya, begitu seterusnya hingga ditemukan solusi atau semua kemungkinan telah dicoba.

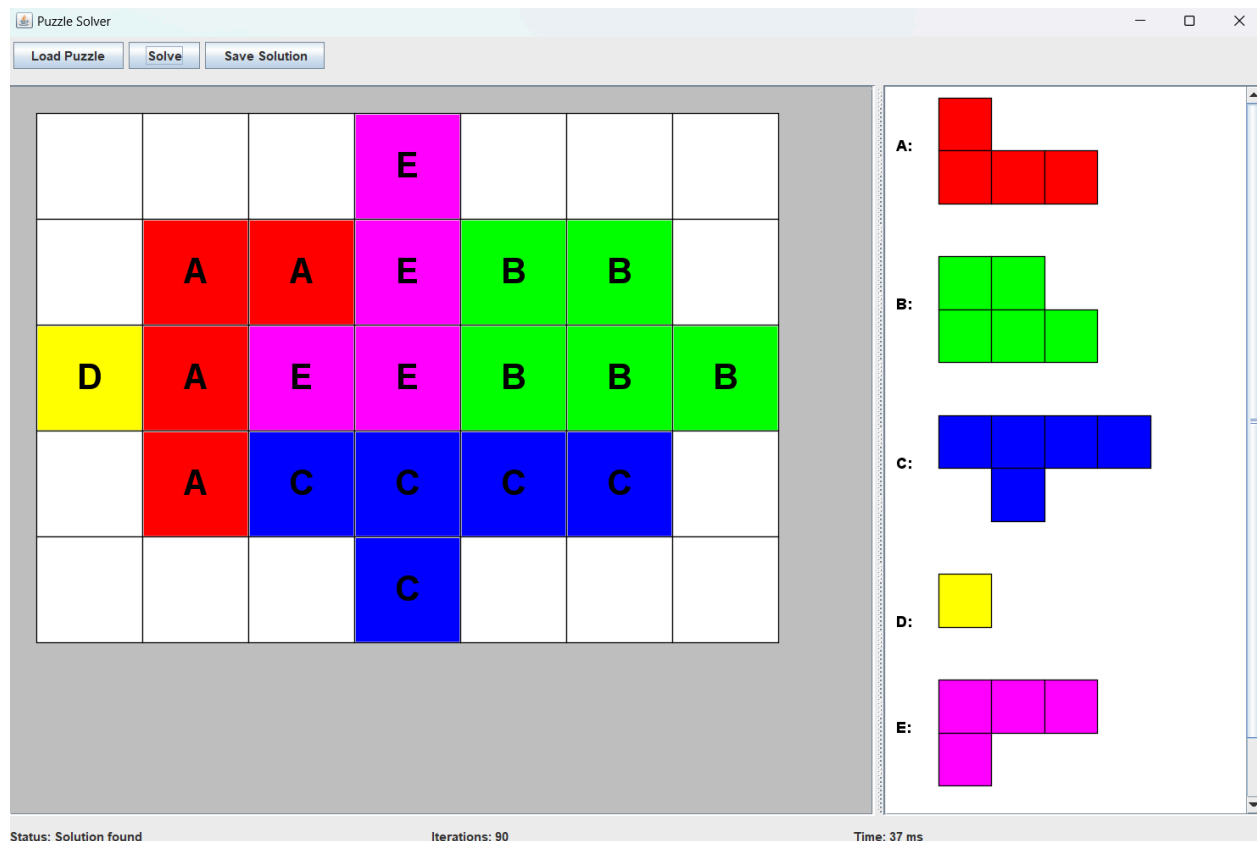
Sehingga secara singkat, alur algoritma program adalah sebagai berikut; Pertama, akan diambil semua transformasi dari sebuah *piece* dengan metode `getAllTransformation()`, untuk setiap transformasi akan dicek apakah *piece* dapat ditempatkan pada *board* dengan metode `canPlace()`, jika valid maka akan dipanggil metode `placePiece()` untuk melabeli sel tersebut pada *board*. Selanjutnya jika solusi tidak ditemukan maka *piece* akan dihapus dari *board* dengan

removePiece() dan algoritma akan lanjut mengecek transformasi atau posisi berikutnya. Algoritma ini akan dilakukan terus menerus hingga ditemukan solusi yang valid atau ketika semua kemungkinan posisi sudah ditinjau.

Test Case Input/Output

```
1 5 7 5
2 CUSTOM
3 ...X...
4 .XXXXX.
5 XXXXXXX
6 .XXXXX.
7 ...X...
8 A
9 AAA
0 BB
1 BBB
2 CCCC
3 C
4 D
5 EEE
6 E
```

Gambar 2.1 Input testcase 1 (Custom)



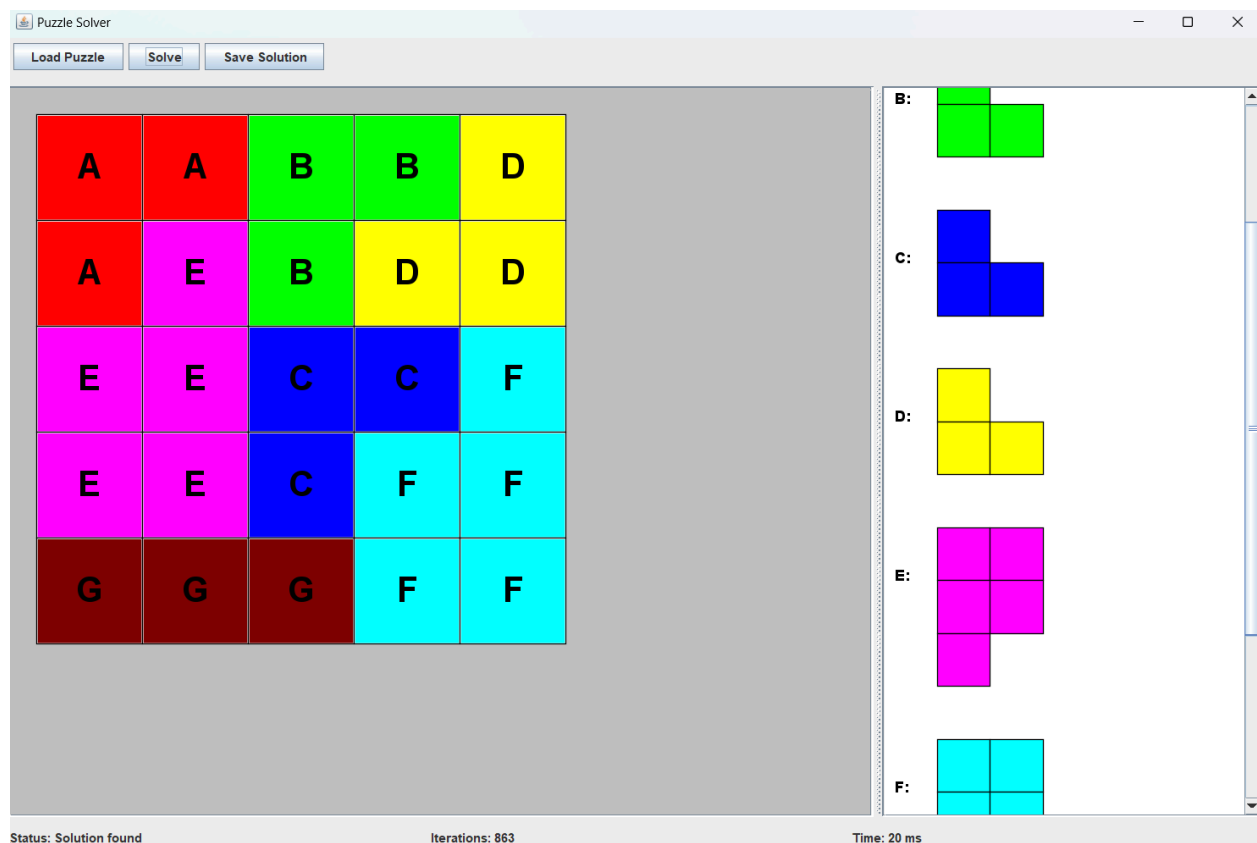
Gambar 2.2 Output testcase 1 (Custom)

```

1  5 5 7
2  DEFAULT
3  A
4  AA
5  B
6  BB
7  C
8  CC
9  D
10 DD
11 EE
12 EE
13 E
14 FF
15 FF
16 F
17 GGG

```

Gambar 2.3 Input testcase 2 (Default)



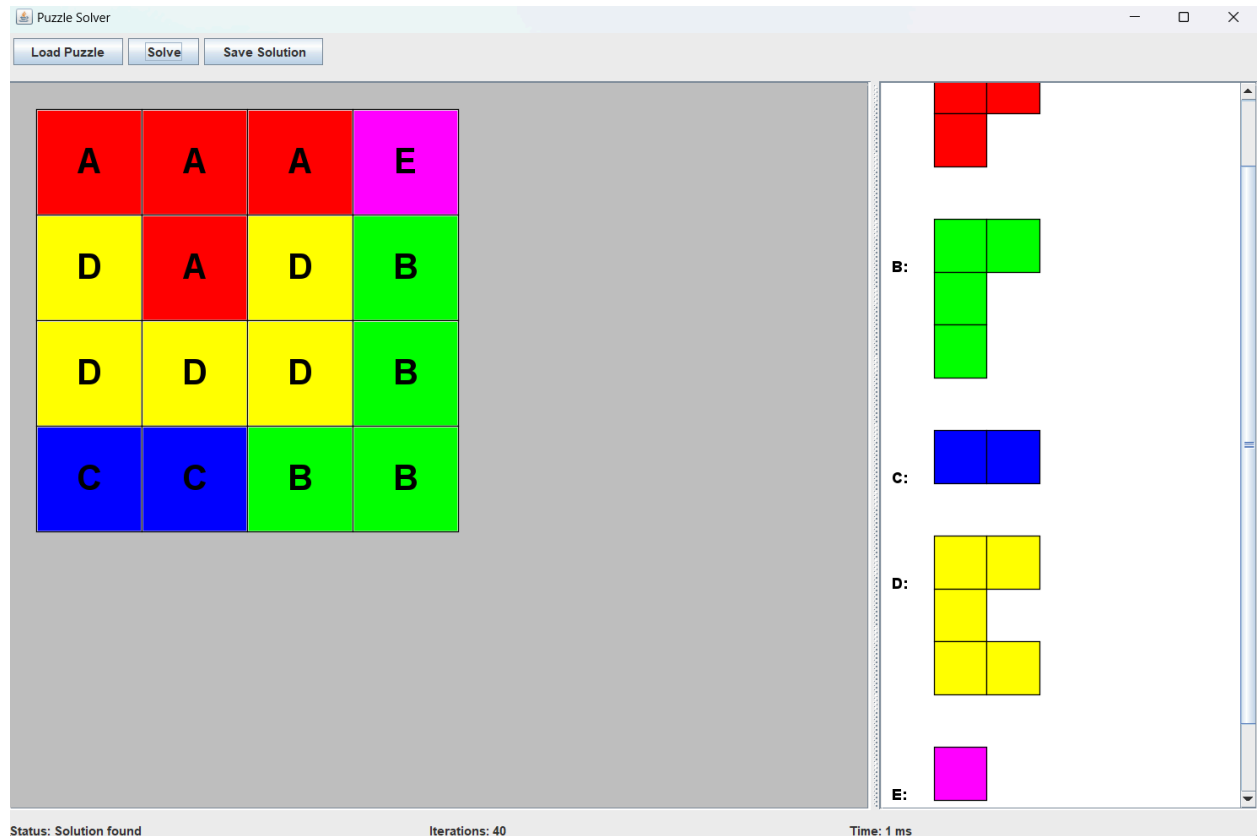
Gambar 2.4 Output testcase 2 (Default)

```

1  4 4 5
2  DEFAULT
3  A
4  AA
5  A
6  BB
7  B
8  B
9  CC
10 DD
11 D
12 DD
13 E

```

Gambar 2.5 Input testcase 3 (Default)



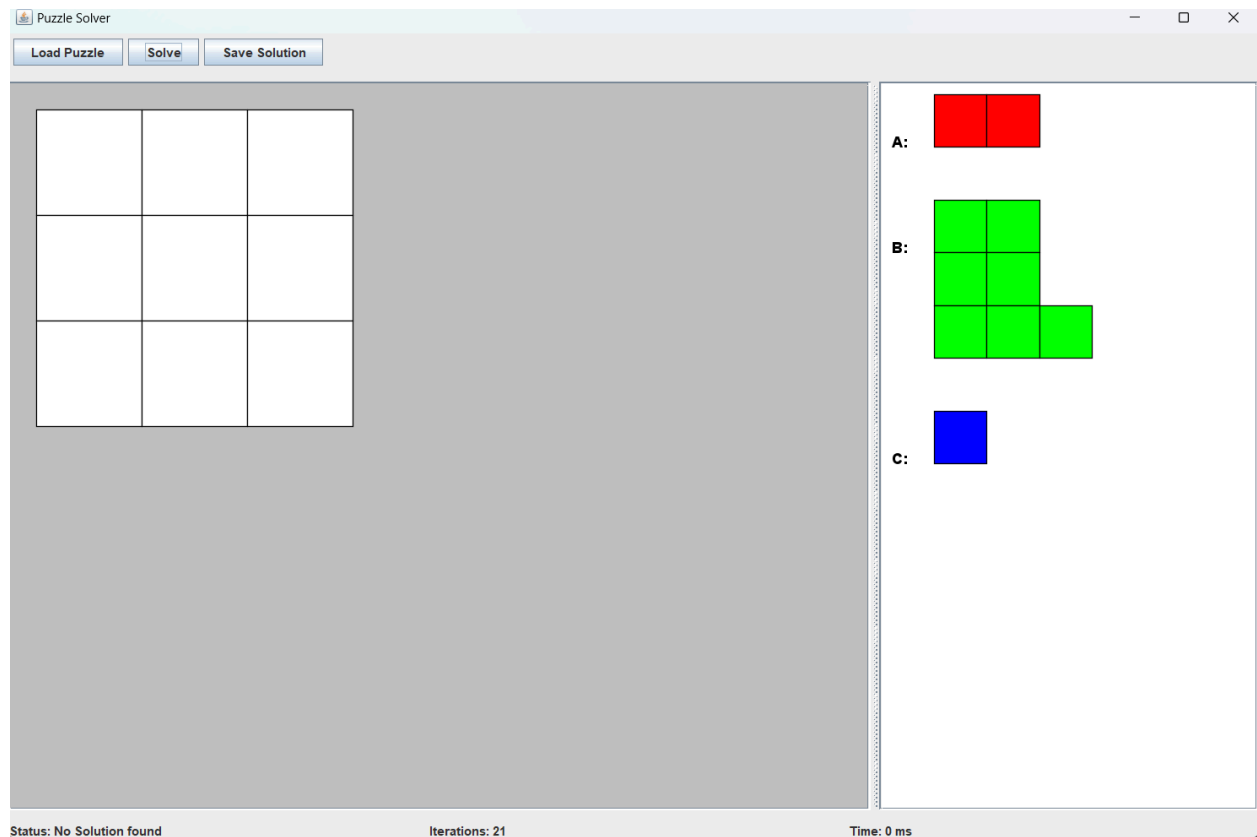
Gambar 2.6 Output testcase 3 (Default)

```

1  3 3 3
2  DEFAULT
3  AA
4  BB
5  BB
6  BBB
7  C

```

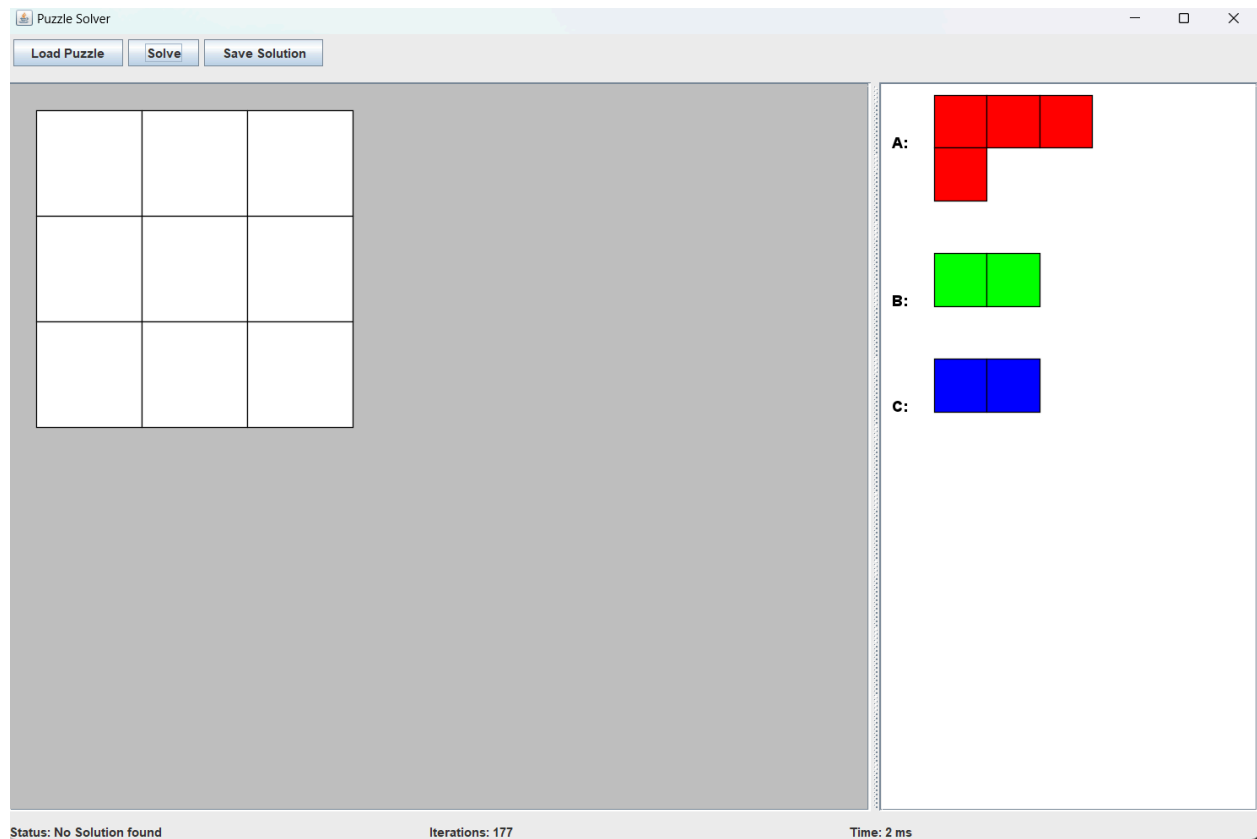
Gambar 2.7 Input testcase 4 (Default)



Gambar 2.8 Output testcase 4 (Default)

1	3 3 3
2	DEFAULT
3	AAA
4	A
5	BB
6	CC

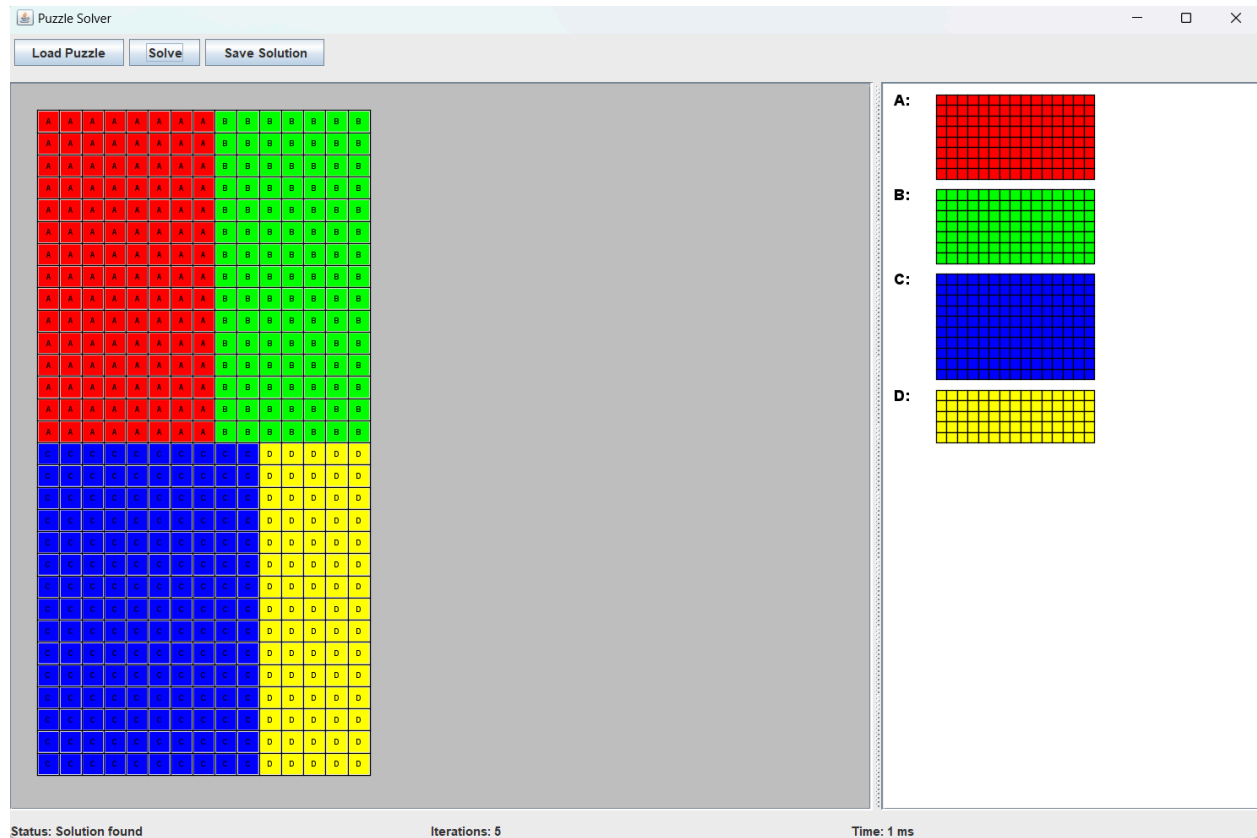
Gambar 2.9 Input testcase 5 (Default)



Gambar 2.10 Output testcase 5 (Default)

```
1 30 15 4
2 DEFAULT
3 AAAAAAAAAAAAAA
4 AAAAAAAAAAAAAA
5 AAAAAAAAAAAAAA
6 AAAAAAAAAAAAAA
7 AAAAAAAAAAAAAA
8 AAAAAAAAAAAAAA
9 AAAAAAAAAAAAAA
10 AAAAAAAAAAAAAA
11 BBBBBBBBBBBBBB
12 BBBBBBBBBBBBBB
13 BBBBBBBBBBBBBB
14 BBBBBBBBBBBBBB
15 BBBBBBBBBBBBBB
16 BBBBBBBBBBBBBB
17 BBBBBBBBBBBBBB
18 CCCCCCCCCCCCCC
19 CCCCCCCCCCCCCC
20 CCCCCCCCCCCCCC
21 CCCCCCCCCCCCCC
22 CCCCCCCCCCCCCC
23 CCCCCCCCCCCCCC
24 CCCCCCCCCCCCCC
25 CCCCCCCCCCCCCC
26 CCCCCCCCCCCCCC
27 CCCCCCCCCCCCCC
28 DDDDDDDDDDDDD
29 DDDDDDDDDDDDD
30 DDDDDDDDDDDDD
31 DDDDDDDDDDDDD
32 DDDDDDDDDDDDD
```

Gambar 2.11 Input testcase 6 (Default)



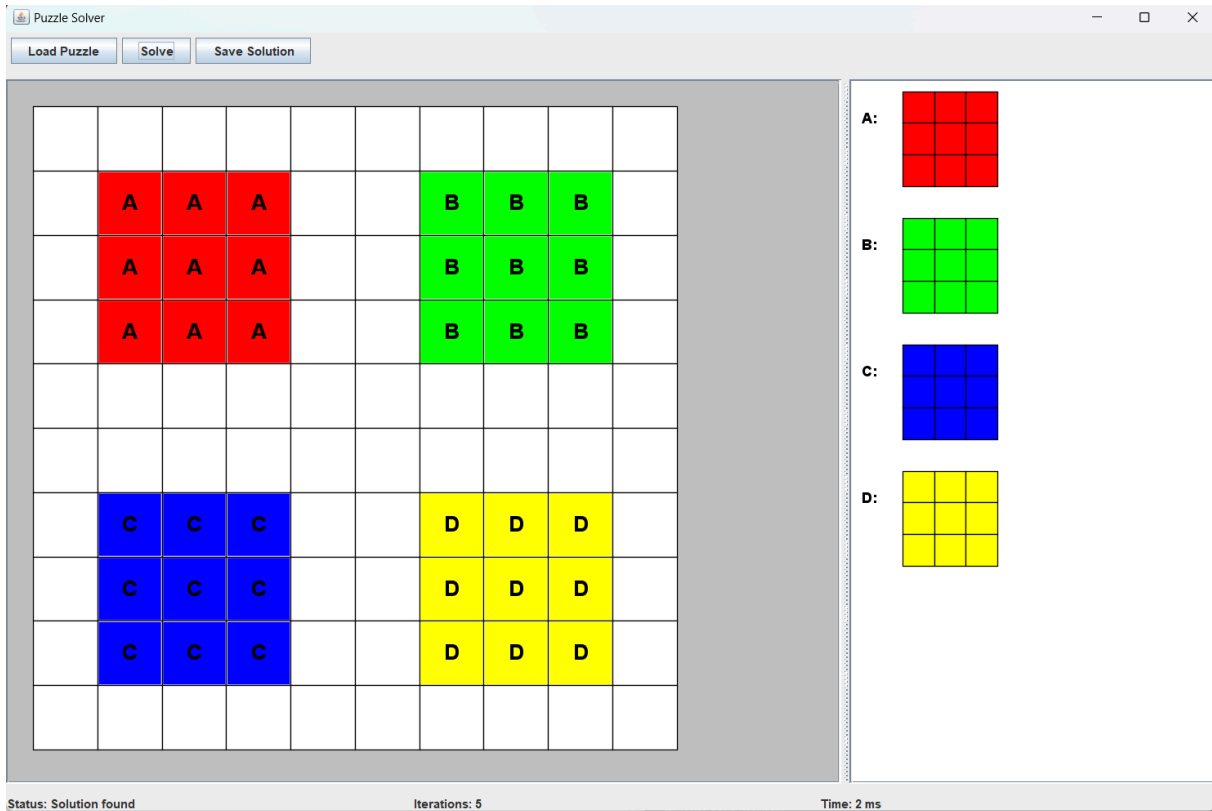
Gambar 2.12 Output testcase 6 (Default)

```

1  10 10 4
2  CUSTOM
3  .....
4  .XXX. .XXX.
5  .XXX. .XXX.
6  .XXX. .XXX.
7  .....
8  .....
9  .XXX. .XXX.
10 .XXX. .XXX.
11 .XXX. .XXX.
12 .....
13 AAA
14 AAA
15 AAA
16 BBB
17 BBB
18 BBB
19 CCC
20 CCC
21 CCC
22 DDD
23 DDD
24 DDD

```

Gambar 2.13 Input testcase 7 (Custom)



Gambar 2.14 Output testcase 7 (Custom)

Link Github Repository

https://github.com/poetoeee/Tucil1_13523096