
**Tugas Kecil 1 IF2211 Strategi Algoritma Semester II tahun
2024/2025 Penyelesaian IQ Puzzler Pro dengan Algoritma Brute
Force**

**by
William Andrian Dharma T
13523006**

Github Repository Link : https://github.com/kirisame-ame/Tucil1_13523006

I. IQ Puzzler Pro

IQ Puzzler Pro adalah permainan papan yang diproduksi oleh perusahaan Smart Games. Tujuan dari permainan ini adalah pemain harus dapat mengisi seluruh papan dengan piece yang telah tersedia. Komponen permainan ini terdiri dari:

1. Board (Papan) – Board komponen utama permainan yang menjadi bentuk yang harus diisi oleh *piece* yang disediakan
2. Piece – Piece adalah komponen-komponen yang harus semuanya digunakan untuk mengisi papan

Tujuan dari program ini adalah untuk mencari solusi pertama atau ketidakberadaannya dengan metode *brute force*.



Gambar 1. Permainan Fisik IQ Puzzler Pro

II. File Input

Program menerima input melalui file .txt yang mengikuti format berikut:

```
N M P
S
<Piece 1>
<Piece 2>
<Piece 3>
...
<Piece P>
```

Dimana:

1. N, M, P adalah jumlah baris, kolom, dan *piece*.
2. S adalah tipe bentuk papan (hanya DEFAULT yang didukung saat ini).
3. Piece adalah bentuk dari piece tersebut dalam huruf kapital A-Z unik dan berbentuk kontinu.

Validasi dari semua parameter dan syarat diatas dilakukan dan akan mengeluarkan *error message* jika tidak terpenuhi. Contoh *input* yang valid adalah sebagai berikut:

```
5 5 7
DEFAULT
A
AA
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
```

III. Algoritma Brute Force

Cara penyelesaian puzzle ini adalah dengan memanggil *recursive function* solve(int idx), idx adalah id *piece*, pada tiap *piece* untuk mencoba memuatkan ke dalam tiap titik pada *board*. Solve bekerja sebagai berikut secara pseudocode:

1. if idx = pieceNum then return boardFilled(), basecase rekursi yang mengecek apakah *board* terisi penuh saat semua *piece* sudah dicoba (tidak di-abstraksi dalam source code).
2. Jika masih ada *piece* untuk dicoba, ambil *piece* dengan id = idx, kita akan sebut *piece-1*, dan coba untuk fit sebuah posisi. Jika fit, panggil solve untuk

piece berikutnya, kita akan sebut *piece-2*, dan cek apakah berhasil. Jika solve *piece-2* berhasil, panggil solve untuk *piece* selanjutnya lagi, jika tidak, hapus *piece-1* dan coba kembali fit dengan semua transformasi dan posisi

3. yang berbeda. Jika *piece-1* dari awal tidak fit, return false.

```
function solve(int: idx) -> boolean
if idx = pieceNum then
    ➔ boardFilled() {cek papan terisi, tidak di-abstraksi dalam source code}

Piece: currPiece <- pieces.get(idx) {ambil object piece}
r traversal [0...rows-1]
  c traversal [0...cols-1]
    i traversal [0...1] {Mirror atau tidak}
      j traversal [0...3] {Orientasi piece}
        boolean[][] currShape = currPiece.transform(j,i)
        if Piece.insertPiece(r, c, currPiece.id, currShape) then {jika fit lanjut}
          if solve(idx+1) then {panggil rekursif piece berikutnya}
            -> true
          else
            Piece.removePiece(r,c,currShape) {hapus piece jika gagal solve}
        -> false {false jika tidak fit}
```

IV. Source Code

```
# Board.java
1. public static boolean solve(int idx){
2.     if(idx==pieceNum){
3.         for (int i = 0; i < rows; i++) {
4.             for(int j=0;j<cols;j++){
5.                 if(board[i][j]==0){
6.                     return false;
7.                 }
8.             }
9.         }
10.        return true;
11.    }
12.    Piece currPiece = pieces.get(idx);
13.    for(int r=0;r<rows;r++){
14.        for(int c=0;c<cols;c++){
15.            // All transformations
16.            for(int i=0;i<2;i++){
17.                for(int j=0;j<4;j++){
18.                    iter++;
19.                    boolean[][] currShape = currPiece.transform(j, i);
20.                    if(Piece.insertPiece(r, c, currPiece.id, currShape)){
21.                        if(solve(idx+1)){
22.                            return true;
23.                        }
24.                    } else{
25.                        Piece.removePiece(r,c,currShape);
26.                    }
27.                }
28.            }
29.        }
30.    }
31.    return false;
32. }
```

```

33.     }
34.
# Piece.java
1.     /**
2.      * transform a piece
3.      * @param rot : rotation number
4.      * @param mirror : mirror number
5.      * @return transformed boolean matrix
6.      */
7.     public boolean[][] transform(int rot,int mirror){
8.         if(mirror==0){
9.             return rotate(this.shape, rot);
10.        }
11.        return rotate(reverse(this.shape, mirror), rot);
12.    }
13.    /**
14.     * @param r : row index
15.     * @param c : col index
16.     * @param id : piece id
17.     * @param p : piece shape
18.     * @return
19.     */
20.    public static boolean insertPiece(int r,int c,int id,boolean[][] p){
21.        int[][] n_board = Board.getBoard();
22.        for(int i=0;i<p.length;i++){
23.            for(int j=0;j<p[0].length;j++){
24.                if(r+i>=n_board.length || c+j>=n_board[0].length){
25.                    return false;
26.                }
27.                else if(p[i][j]){
28.                    if(n_board[r+i][c+j]==0){
29.                        n_board[r+i][c+j] = id;
30.                    }
31.                    else{
32.                        return false;
33.                    }
34.                }
35.            }
36.        }
37.        Board.setBoard(n_board);
38.        return true;
39.    }
40.    public static void removePiece(int r,int c,boolean[][] p){
41.        int[][] n_board = Board.getBoard();
42.        for(int i=0;i<p.length;i++){
43.            for(int j=0;j<p[0].length;j++){
44.                if(r+i<n_board.length && c+j<n_board[0].length && p[i][j]){
45.                    n_board[r+i][c+j] = 0;
46.                }
47.            }
48.        }
49.        Board.setBoard(n_board);
50.    }
51.    /**
52.     * @param quad rotation index
53.     * <li> 1 : 90 degrees CCW
54.     * <li> 2 : 180 degrees
55.     * <li> 3 : 270 degrees CCW
56.     * <li> returns original shape otherwise
57.     */
58.    public boolean[][] rotate(boolean[][] m,int quad){

```

```

59.         switch (quad) {
60.             case 1->{
61.                 boolean[][] t = transpose(m);
62.                 return reverse(t, 0);
63.             }
64.             case 2->{
65.                 return reverse(reverse(m, 0),1);
66.             }
67.             case 3->{
68.                 boolean[][] t = transpose(m);
69.                 return reverse(t, 1);
70.             }
71.             default ->{
72.                 return m;
73.             }
74.         }
75.     }
76.     public boolean[][] transpose(boolean[][] m){
77.         boolean[][] t = new boolean[m[0].length][m.length];
78.         for (int i = 0; i < m.length; i++) {
79.             for(int j=0;j<m[0].length;j++){
80.                 t[j][i] = m[i][j];
81.             }
82.         }
83.         return t;
84.     }
85.     /**
86.      *
87.      * @param axis
88.      * <li> 0: reverse by rows
89.      * <li> 1: reverse by columns
90.      */
91.     @SuppressWarnings("ManualArrayToCollectionCopy")
92.     public boolean[][] reverse(boolean[][] m,int axis){
93.         if(axis==0){
94.             boolean[][] res = new boolean[m.length][m[0].length];
95.             for(int i=m.length-1;i>=0;i--){
96.                 for(int j=0;j<m[0].length;j++){
97.                     res[m.length-1-i][j] = m[i][j];
98.                 }
99.             }
100.            return res;
101.        }
102.        if(axis==1){
103.            boolean[][] res = new boolean[m.length][m[0].length];
104.            // Access entire rows
105.            for (int i=0;i<m.length;i++) {
106.                for (int j = m[0].length-1; j>=0; j--) {
107.                    res[i][m[0].length-1-j] = m[i][j];
108.                }
109.            }
110.            return res;
111.        }
112.        return m;
113.    }
114.

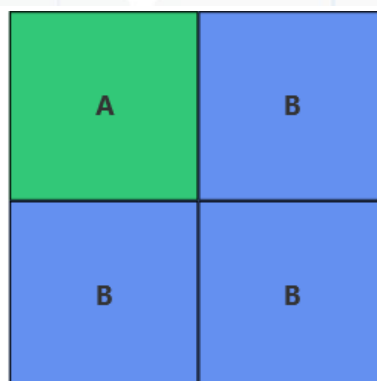
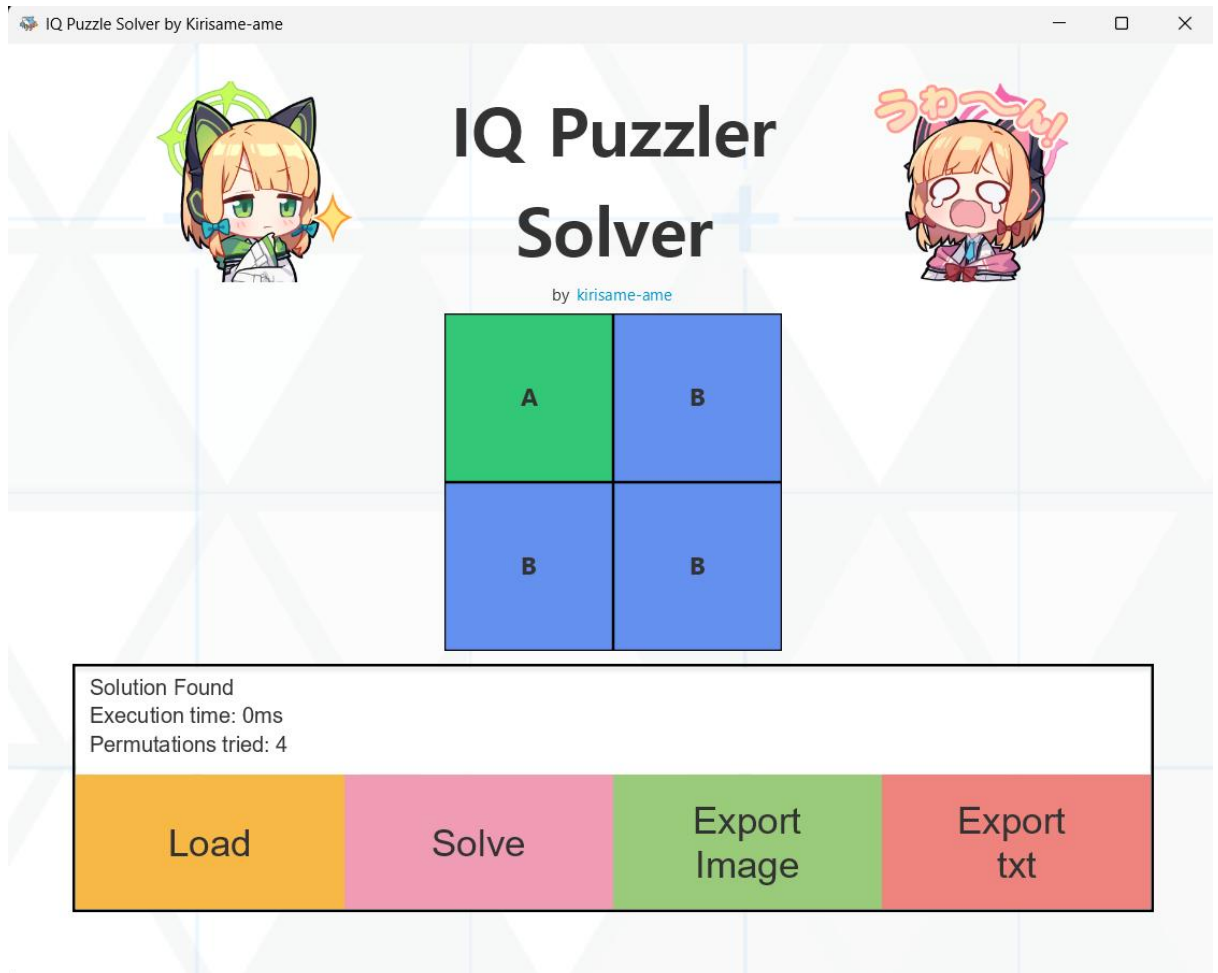
```

V. Testing

File .txt input testing terdapat pada directory test/tc, hasil testing terdapat pada test/screenshots, test/img_out, test/text_out

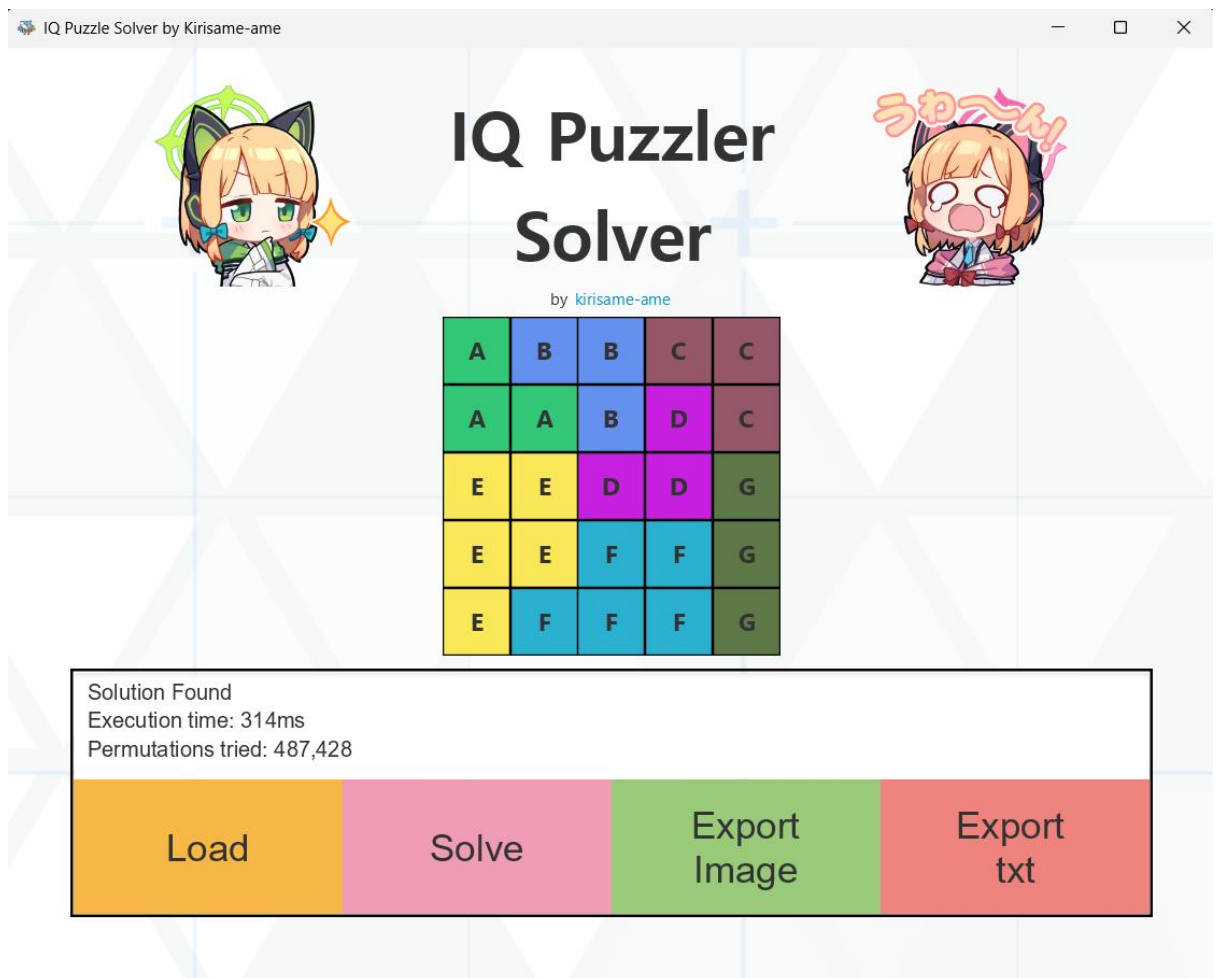
A. Test case 1

```
AB
BB
Solution Found
Execution time: 0ms
Permutations tried: 4
```



B. Test case 2

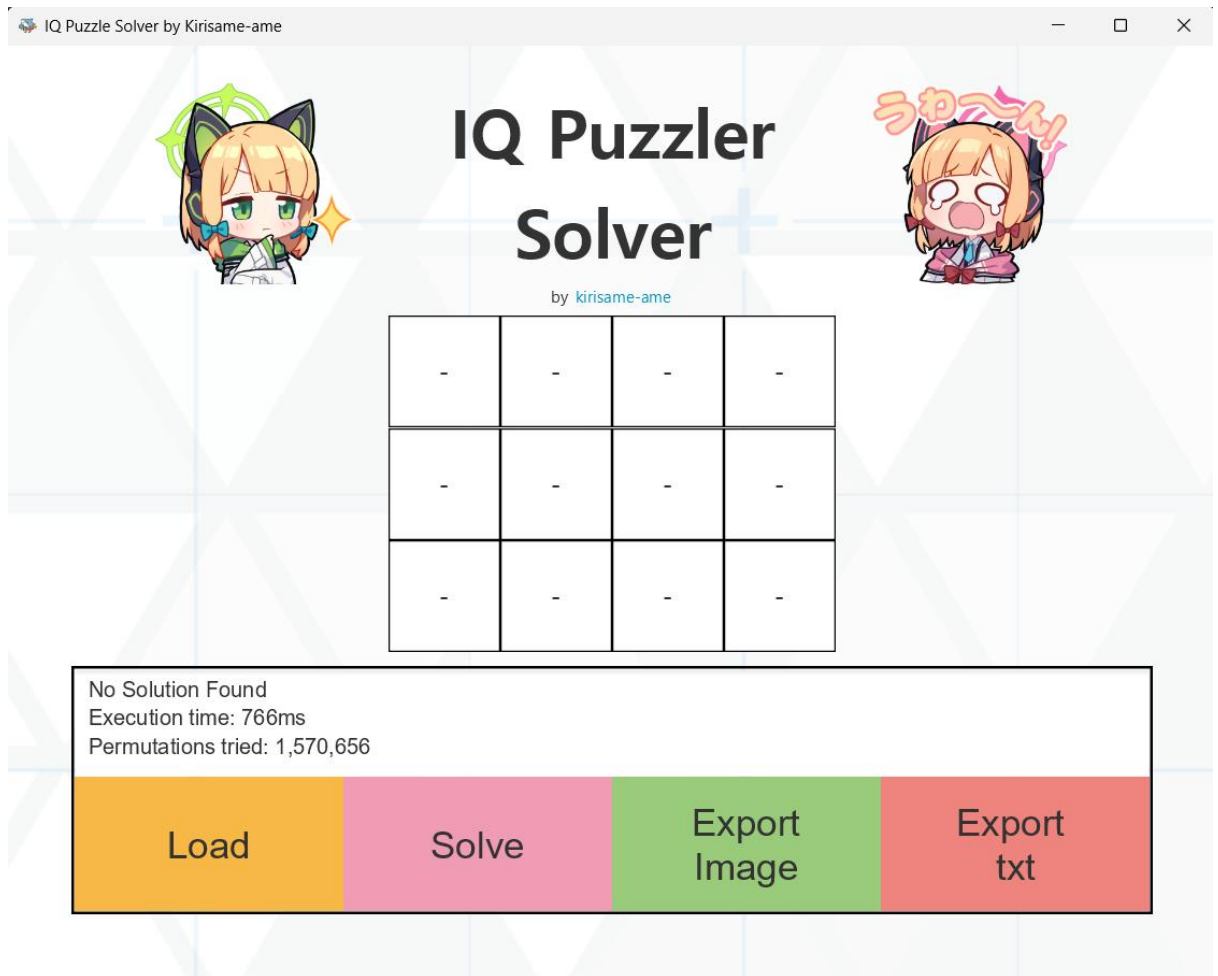
```
ABBCC
AABDC
EEDDG
EEFFG
EFFF6
Solution Found
Execution time: 314ms
Permutations tried:
487,428
```



A	B	B	C	C
A	A	B	D	C
E	E	D	D	G
E	E	F	F	G
E	F	F	F	G

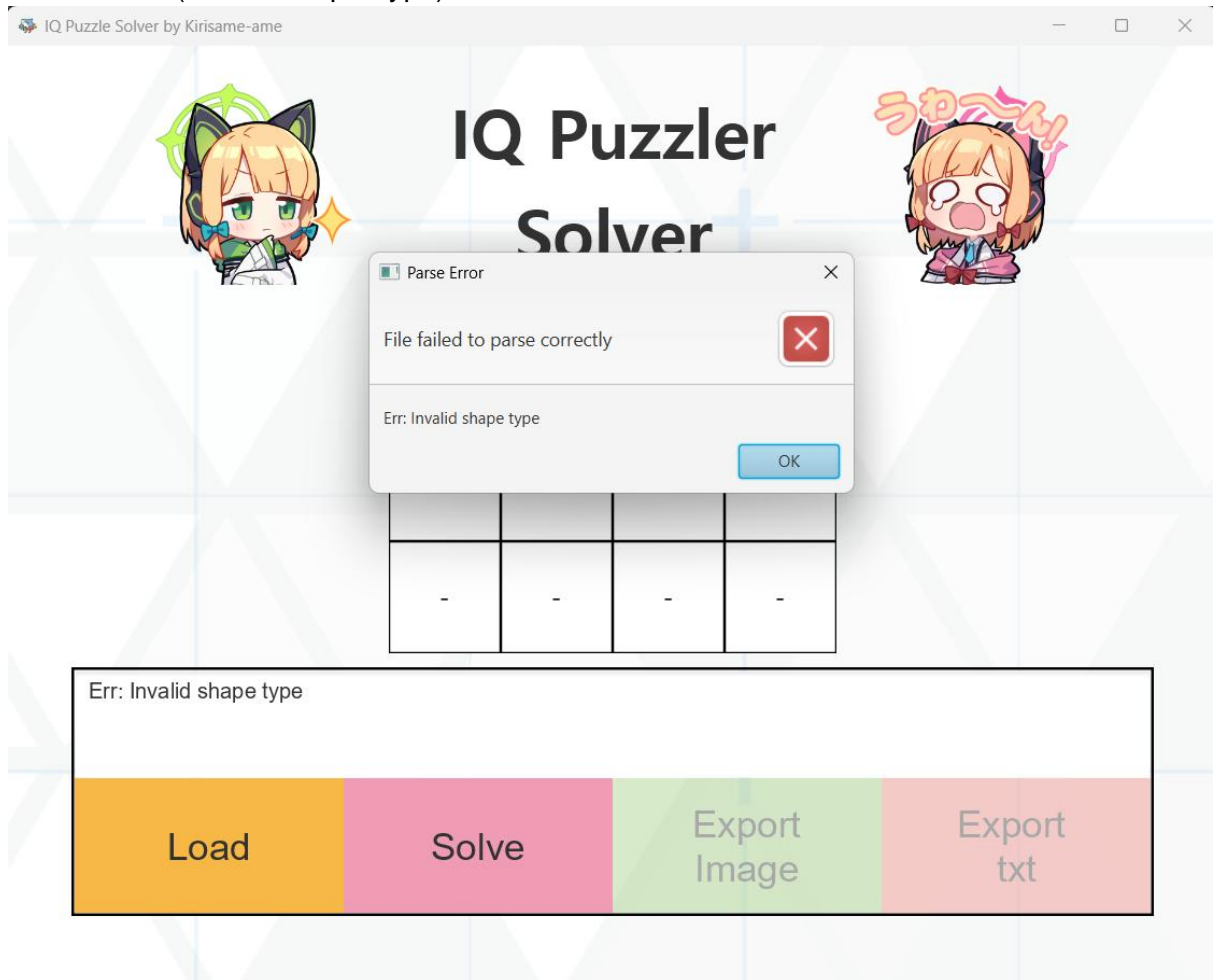
C. Test Case 3 (No solution)

No Solution Found
Execution time: 766ms
Permutations tried:
1,570,656

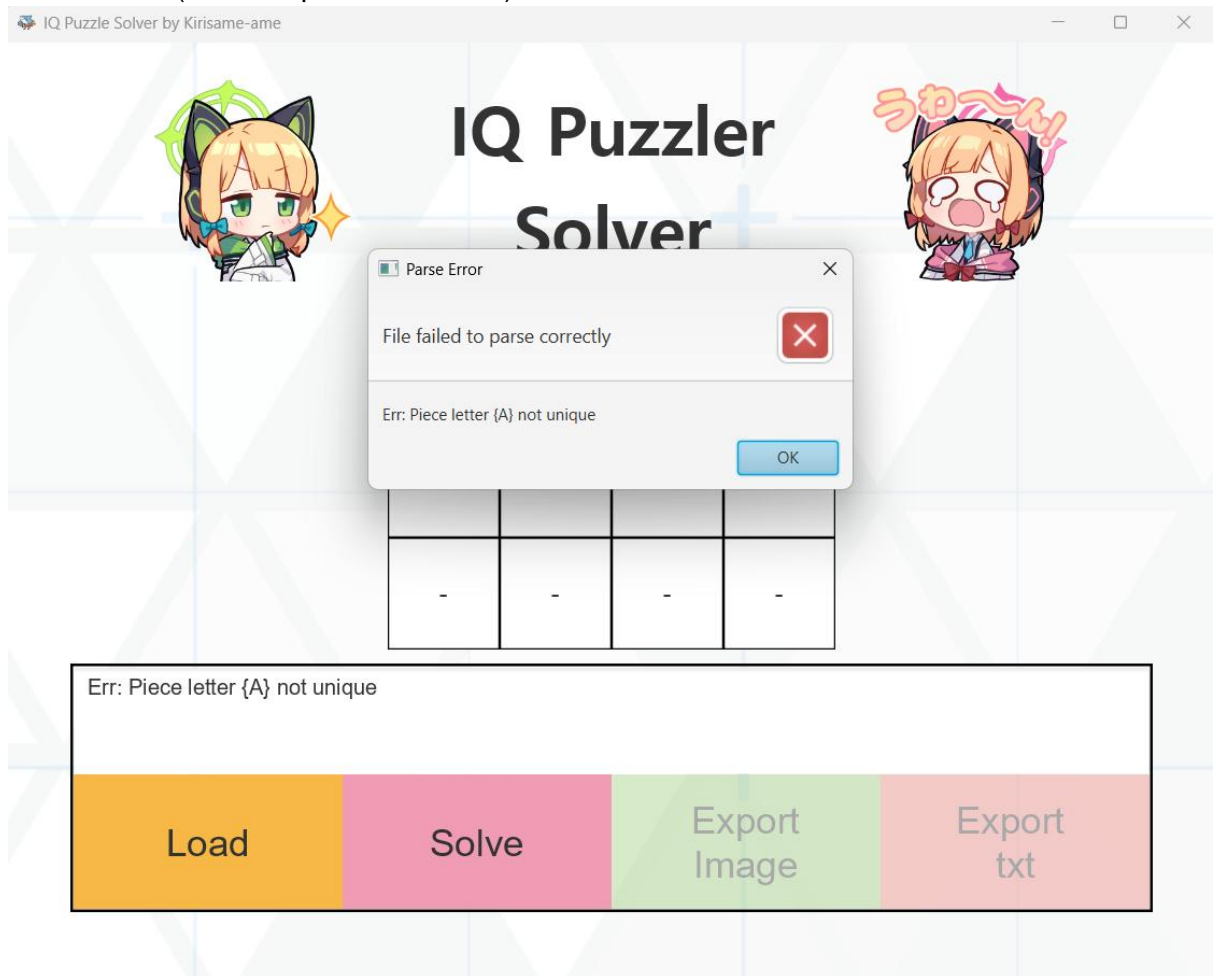


-	-	-	-
-	-	-	-
-	-	-	-

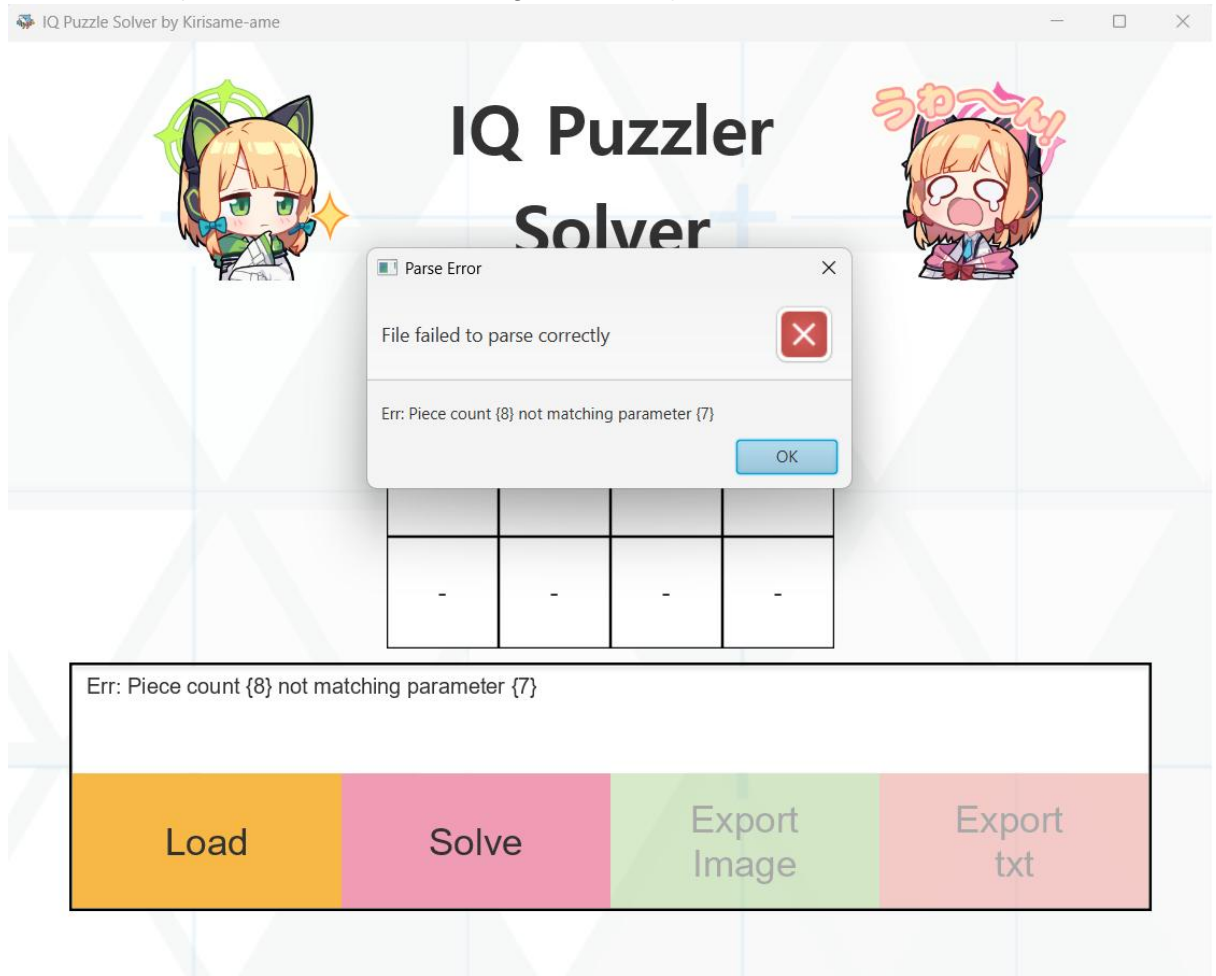
D. Test Case 4 (Invalid Shape Type)



E. Test Case 5 (Non Unique Piece Letter)




F. Test Case 6 (Piece Count Not Matching Parameter)



G. Test Case 7 (Box and Diagonals)


```
SSSSSSS  
SIUUUIS  
SUIUIUS  
SUUIIUS  
SUIUIUS  
SIUUUIS  
SSSSSSS  
Solution Found  
Execution time: 0ms  
Permutations tried: 131
```

IQ Puzzle Solver by Kirisame-ame



IQ Puzzler Solver

by [kirisame-ame](#)



S	S	S	S	S	S	S
S	I	U	U	U	I	S
S	U	I	U	I	U	S
S	U	U	I	U	U	S
S	U	I	U	I	U	S
S	I	U	U	U	I	S
S	S	S	S	S	S	S

Solution Found
Execution time: 0ms
Permutations tried: 131

Load

Solve

Export Image

Export txt

S	S	S	S	S	S	S
S	I	U	U	U	I	S
S	U	I	U	I	U	S
S	U	U	I	U	U	S
S	U	I	U	I	U	S
S	I	U	U	U	I	S
S	S	S	S	S	S	S