

Penyelesaian *Cyberpunk 2077 Breach Protocol* dengan Algoritma *Brute Force*

Disusun oleh
Farhan Nafis Rayhan - 13522037



(Sumber: <https://rockpapershotgun.com>)

Deskripsi Masalah

1. Cyberpunk 2077

Semenjak dirilis pada tahun 2020 lalu, Cyberpunk 2077 telah menjadi salah satu game paling populer pada masa ini. Game ini telah dikembangkan oleh *CD Projekt Red* dari Polandia selama hampir 8 tahun, dan sukses membangun antusiasme berbagai kalangan dari seluruh dunia. Dengan genre *Action-RPG*, game ini mengusung konsep seorang tentara bayaran bernama V yang menjelajahi

kota *Night City*, pada waktu yang jauh lebih modern dibandingkan sekarang. Salah satu hal yang banyak menarik perhatian dari *Cyberpunk 2077* ini adalah gaya bermain yang unik bagaikan seorang *hacker*, berbeda dengan kebanyakan permainan aksi lainnya.

2. Breach Protocol Minigame

Sebagai proyek Video Game yang besar tentunya *Cyberpunk 2077* juga dilengkapi dengan berbagai minigame, salah satu yang paling tidak bisa dilupakan adalah *Breach Protocol*. Minigame ini memberikan pengalaman seakan kita meretas jaringan lokal *ICE (Intrusion Countermeasures Electronics)*. Permainan ini cukup sederhana karena hampir seluruhnya terdiri atas 1 hal yang sama, yaitu *token*. *Token* disini adalah 2 buah karakter alfanumerik yang tergabung menjadi satu. Selain itu terdapat pula *sequence*, yaitu kombinasi lebih dari 2 *token* yang diurutkan menjadi suatu barisan tertentu, tepatnya kode yang harus kita bentuk agar dapat meretas musuh. *Sequence* tersebut dapat kita bentuk dengan cara mengumpulkan beberapa *token* dari matriks yang disediakan dengan aturan tertentu. Dan terakhir ada pula *buffer*, sebagai pembatas banyaknya *token* yang dapat kita kumpulkan.

Aturan permainan ini pun terbilang cukup sederhana. Permainan dimulai dengan memilih satu buah *token* yang berada pada baris paling atas matrik. Selanjutnya pemain dapat memilih suatu *token* lainnya yang terletak pada satu kolom dengan *token* yang dipilih sebelumnya. Pemain bergerak dengan arah awal vertikal, lalu menjadi selang-seling antara horizontal dan vertikal, dimana kita memilih 1 *token* pada setiap langkah. Hal ini terus dilakukan pemain sampai *buffer* yang ia miliki sudah penuh.

Tujuan dari *minigame* ini adalah membentuk *buffer* dengan sebanyak mungkin *sequence* didalamnya. Perlu disadari bahwa setiap *sequence* memiliki *reward*, atau bobot, yang variatif. Sehingga, setiap *sequence* akan memiliki prioritas yang berbeda. Setiap *token* yang telah disusun pada *buffer* dapat dicocokkan pada lebih dari 1 *sequence*. Solusi yang optimal dari *minigame* ini adalah *buffer* yang memberikan total bobot terbesar, dimana *buffer* ini bisa jadi sudah penuh, tetapi lebih optimal apabila melibatkan lebih sedikit *token*.

Algoritma

1. Brute Force

Brute Force adalah salah satu strategi yang sering digunakan untuk menyelesaikan persoalan dalam dunia Informatika. Sering disebut dengan strategi *Straightforward*, atau lempeng, brute force biasanya merupakan strategi yang paling mudah untuk ditemukan juga diimplementasikan. Strategi *Brute force* mengandalkan kekuatan komputer untuk memeriksa setiap kemungkinan secara sistematis, sampai ditemukan sebuah kemungkinan yang paling memenuhi persyaratan. Kelemahan utama strategi ini adalah *Brute Force* biasanya bukanlah strategi paling efisien, sehingga memiliki kompleksitas waktu yang tinggi. Namun bagi kasus yang memerlukan input tidak terlalu besar, strategi ini tetap berguna karena cenderung lebih mudah ditemukan dibanding strategi lainnya.

2. Algoritma Brute Force dalam penyelesaian Breach Protocol Minigame

Untuk menemukan solusi yang paling optimal dengan strategi *brute force*, kita perlu memeriksa setiap kemungkinan dan melihat siakah yang memberikan bobot tertinggi. Hal ini bisa kita lakukan dengan cara membangkitkan semua kemungkinan *buffer* dari panjang 1 sampai maksimal lalu kita bandingkan satu-persatu *reward* ia berikan. Pemilihan *buffer* dilakukan dengan cara memilih *buffer* pertama yang mungkin dilakukan. Lalu, kita lanjut dengan *buffer* selanjutnya dengan cara menaikkan *token* terakhir pada *buffer* sampai *token* terdepan pun sampai kemungkinan terakhir. Perlu diperhatikan bahwa pada setiap langkah, token harus memiliki baris atau kolom yang sama dengan token sebelumnya. Juga bisa diamati bahwa pada satu *buffer* tidak mungkin ada 2 atau lebih token yang sama.

Secara detail, algoritma yang digunakan penulis adalah sebagai berikut:

1. Membuat array *Current Sequence* berisi koordinat “x-y”, melambangkan token matrix pada baris x dan kolom y. Isi dengan elemen pertama “0-0”

2. Mengisi *Current Sequence* sampai penuh dimana pada setiap langkahnya diisi dengan koordinat pertama (secara baris/kolom) yang belum dipilih, tetapi ia memiliki angka baris/kolom yang sama dengan koordinat yang sebelumnya dan belum ada koordinat tersebut dalam array.
3. *Sequence Token* dari array diatas diperoleh dengan cara melihat *Token* apakah yang ada pada koordinat tersebut dalam matrix.
4. Dilakukan *pattern matching* dengan setiap *pattern* yang diberikan. Apabila *Sequence* memiliki huruf yang sama dengan *token* awal pada *pattern*, maka kita membandingkan setiap *token* satu persatu dengan pattern sampai dipastikan apabila ia cocok atau tidak.
5. Kecocokan pattern menghasilkan *reward* yang dapat dihitung, lalu dibandingkan dengan *reward* maksimum yang sudah disimpan. Apabila kasus ini lebih besar maka jadikan ia maksimum yang baru, jika sama boleh kita bandingkan mana yang memiliki panjang *sequence* terpendek.
6. Untuk membangkitkan *sequence* selanjutnya, kita akan menaikkan baris/kolom dari koordinat pada akhir array *Current Sequence*.
7. Apabila kita sudah mencapai ujung dari koordinat (tidak bisa naik lagi baris/kolomnya) maka kita perlu menghapus koordinat yang sudah mencapai ujung tersebut secara satu-persatu dari ujung belakang array sampai kita memperoleh koordinat pertama yang bukan di ujung. Maka kita bisa menaikkan koordinat ini dan memasukkan kembali semua koordinat setelahnya, tetapi mulai dari awal kembali (baris/kolom terkecil yang mungkin).
8. Biarkan algoritma ini terus berjalan hingga pada suatu saat array menjadi kosong karena koordinat pertama pun sudah mencapai ujung. Hal ini berarti algoritma sudah selesai. Sehingga *reward* maksimal yang disimpan juga *Sequence* terpendek merupakan solusi yang optimal.

Implementasi dalam bahasa Java

1. Repository Program

Repository program ini dapat diakses melalui pranala berikut:

https://github.com/Farhannr28/Tucill_13522037

Penulis memilih untuk menggunakan Bahasa Java karena performanya yang cukup cepat dibutuhkan untuk persoalan Brute Force, selain itu Java juga dilengkapi dengan berbagai package yang bisa memudahkan pemrograman. Salah satu package yang digunakan pada program ini adalah Java Swing guna membuat GUI. Java Swing dikenal sebagai GUI yang mudah dipelajari dan cukup lengkap sehingga penulis merasa cocok digunakan pada program ini.

2. Struktur Data

Utamanya, program menyimpan struktur data yang merupakan masukkan dalam **class Main.java** yang meliputi *buffer* (ukuran *buffer*), matriks, dan *sequence list*. Terdapat **class Sequence.java** yang terdiri atas *strArr* yaitu array berisi string *Token* yang membentuk *sequence*, juga *rew* sebagai besarnya *reward* yang ia miliki. Sedangkan **class Matrix.java** merupakan kelas yang menyimpan matriks masukkan dari pengguna dan memiliki atribut *width & height* untuk menyimpan lebar dan tingginya juga tab sebagai *String* array 2 dimensi yang menyimpan seluruh *token*.

Implementasi algoritma yang saya jelaskan sebelumnya memanfaatkan sebuah **class bernama Coordinate.java** dengan atribut *col* dan *row* yang menunjukkan kolom dan baris mana yang ia lambangkan. Keseluruhan algoritma *brute force* ditulis dalam **Solver.java** dimana algoritma yang telah dijelaskan sebelumnya dijalankan pada sebuah array *Coordinate* bernama *route*. Sadari bahwa pada program setiap *Coordinate* yang dimasukkan pada *route* haruslah unik, karena pengecekan keunikan dilakukan berulang kali penulis memilih untuk memanfaatkan struktur data *Set* dengan nama *visitedSet* agar lebih efisien. *Set* memiliki keunikan dimana hanya elemen unik yang boleh dimasukkan ke dalamnya sehingga pengecekan keunikan bisa dilakukan dengan perulangan mencoba memasukkan dan menaikan koordinat, apabila metode *visitedSet.add()* mengeluarkan *true* maka koordinat sudah unik dan berhasil dimasukkan.

3. Source Code

3.1. Main.java

```
package src;

import java.util.Random;
public class Main{
    static Frame frame;
    static Solver solver;
    public static int buffer;
    public static Matrix matrix;
    public static Sequence[] sequenceList;
    public static int maxReward;
    public static Coordinate answerRoute[];
    public static Random rng;
    public static double time;
    public static String fileName;

    public static void main(String args[]){
        long currentTime = System.currentTimeMillis();
        rng = new Random(currentTime);
        solver = new Solver();
        frame = new Frame();
    }
}
```

3.2. Matrix.java

```
package src;

public class Matrix {

    String[][] tab;
    int width;
    int height;

    Matrix(int h, int w, String[][] table){
        this.tab = table;
        this.height = h;
        this.width = w;
    }
}
```

3.3. Sequence.java

```
package src;

public class Sequence {
    int len;
    String[] strArr;
    int rew;

    Sequence(int length, String[] stringArray, int reward){
        this.len = length;
        this.strArr = stringArray;
        this.rew = reward;
    }
}
```

3.4. Coordinate.java

```
package src;

public class Coordinate implements Comparable<Coordinate> {
    int col;
    int row;

    Coordinate(int c, int r){
        this.col = c;
        this.row = r;
    }

    @Override
    public int compareTo(Coordinate other) {
        int result = Integer.compare(this.col, other.col);
        if (result != 0) {
            return result;
        }
        return Integer.compare(this.row, other.row);
    }
}
```

3.5. Solver.java

```
package src;

import java.util.SortedSet;
import java.util.TreeSet;

public class Solver {
    static Coordinate[] route;
    static String[] currentBuffer;
    static SortedSet<Coordinate> visitedSet;
    static int pointer;
    static int caseCount;
    static int size;
    static int minimumLength;

    public static void solve(){
        // Initiate
        size = Main.buffer;
        route = new Coordinate[size];
        Main.maxReward = 0;
        currentBuffer = new String[size];
        visitedSet = new TreeSet<>();
        pointer = 0;
        caseCount = 0;
        Coordinate end = new Coordinate(Main.matrix.width-1,
Main.matrix.height-1);
        Coordinate temp = new Coordinate(0, 0);

        route[0] = temp;
        visitedSet.add(temp);
        currentBuffer[0] = Main.matrix.tab[0][0];
        patternMatch();

        // Fill
        // pointer odd, go vertical, same col
        boolean addPrevious;
        while (pointer < size){
            addPrevious = false;
            while ((pointer%2==0 && route[pointer].col>end.col) ||
(pointer%2==1 && route[pointer].row>end.row)){
                visitedSet.remove(route[pointer]);
                pointer--;
                addPrevious = true;
            }
            if (addPrevious)
                route[pointer] = temp;
            else
                route[pointer] = currentBuffer[pointer];
            currentBuffer[pointer] = Main.matrix.tab[pointer][0];
            patternMatch();
        }
    }
}
```

```

        }

        if (addPrevious) {
            while(!visitedSet.add(route[pointer])){
                temp = new Coordinate(route[pointer].col,
route[pointer].row);

                route[pointer] = temp;

                if (pointer%2==0){
                    route[pointer].col++;
                } else {
                    route[pointer].row++;
                }
            }

            if ((pointer%2==0 && route[pointer].col>end.col) ||
(pointer%2==1 && route[pointer].row>end.row)){
                continue;
            }

            currentBuffer[pointer] =
Main.matrix.tab[route[pointer].row][route[pointer].col];

            patternMatch();
        } else {
            pointer++;

            if (pointer == size){
                break;
            }

            if (pointer%2==0){
                temp = new Coordinate(0, route[pointer-1].row);
            } else {
                temp = new Coordinate(route[pointer-1].col, 0);
            }

            while (!visitedSet.add(temp)){
                if (pointer%2==0){
                    temp.col++;
                } else {
                    temp.row++;
                }
            }

            route[pointer] = temp;

            if ((pointer%2==0 && temp.col>end.col) || (pointer%2==1
&& temp.row>end.row)){
                continue;
            }
        }
    }
}

```

```

        currentBuffer[pointer] =
Main.matrix.tab[temp.row][temp.col];
        patternMatch();
    }
}

pointer--;

// Start
while (pointer >= 0){
    while ((pointer%2==0 && route[pointer].col>end.col) ||
(pointer%2==1 && route[pointer].row>end.row)){
        visitedSet.remove(route[pointer]);
        pointer--;
    }
    if (pointer<0){
        break;
    }
    visitedSet.remove(route[pointer]);
    if (pointer%2==0){
        route[pointer].col++;
    } else {
        route[pointer].row++;
    }
    temp = new Coordinate(route[pointer].col,
route[pointer].row);
    route[pointer] = temp;
    while(!visitedSet.add(route[pointer])){
        if (pointer%2==0){
            route[pointer].col++;
        } else {
            route[pointer].row++;
        }
    }
    if ((pointer%2==0 && route[pointer].col>end.col) ||
(pointer%2==1 && route[pointer].row>end.row)){
        continue;
    }
    currentBuffer[pointer] =
Main.matrix.tab[route[pointer].row][route[pointer].col];
    patternMatch();
    pointer++;
    while (pointer < size){

```

```

        if (pointer%2==0){
            route[pointer].col = 0;
            route[pointer].row = route[pointer-1].row;
        } else {
            route[pointer].row = 0;
            route[pointer].col = route[pointer-1].col;
        }
        temp = new Coordinate(route[pointer].col,
route[pointer].row);
        route[pointer] = temp;
        while(!visitedSet.add(route[pointer])){
            if (pointer%2==0){
                route[pointer].col++;
            } else {
                route[pointer].row++;
            }
            if ((pointer%2==0 && route[pointer].col>end.col) ||
(pointer%2==1 && route[pointer].row>end.row)){
                continue;
            }
            currentBuffer[pointer] =
Main.matrix.tab[route[pointer].row][route[pointer].col];
            patternMatch();
            pointer++;
        }
        pointer--;
    }
}

public static void patternMatch(){
    caseCount++;
    int sum = 0;
    int j;
    for (Sequence sq : Main.sequenceList){
        for (int i=0; i<=pointer; i++){
            if (currentBuffer[i].charAt(0) == sq.strArr[0].charAt(0)
&& currentBuffer[i].charAt(1) == sq.strArr[0].charAt(1)){
                j=1;
                while (j<sq.len && i+j<=pointer &&
currentBuffer[i+j].charAt(0) == sq.strArr[j].charAt(0) &&
currentBuffer[i+j].charAt(1) == sq.strArr[j].charAt(1)){

```

```

                j++;
            }
            if (j == sq.len){
                sum += sq.rew;
                break;
            }
        }
    }
    if (sum > Main.maxReward){
        Main.maxReward = sum;
        int i=0;
        minimumLength = pointer;
        while (i <= pointer){
            Main.answerRoute[i].row = route[i].row;
            Main.answerRoute[i].col = route[i].col;
            i++;
        }
        while (i < Main.buffer){
            Main.answerRoute[i].row = -1;
            Main.answerRoute[i].col = -1;
            i++;
        }
    } else if (sum == Main.maxReward){
        if (pointer < minimumLength){
            Main.maxReward = sum;
            int i=0;
            minimumLength = pointer;
            while (i <= pointer){
                Main.answerRoute[i].row = route[i].row;
                Main.answerRoute[i].col = route[i].col;
                i++;
            }
            while (i < Main.buffer){
                Main.answerRoute[i].row = -1;
                Main.answerRoute[i].col = -1;
                i++;
            }
        }
    }
}
}

```

3.6. FileReader.java

```
package src;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class FileReader {
    private static Scanner sc;
    public static void readFile(String path) {
        try {
            sc = new Scanner(new File(path));
            Main.buffer = Integer.parseInt(sc.nextLine());
            String[] temp = sc.nextLine().split(" ");
            int w = Integer.parseInt(temp[0]);
            int h = Integer.parseInt(temp[1]);
            String[][] tempMat = new String[h][w];
            for (int i=0; i<h; i++) {
                temp = sc.nextLine().split(" ");
                for (int j=0; j<w; j++) {
                    tempMat[i][j] = temp[j];
                }
            }
            Main.matrix = new Matrix(h, w, tempMat);
            int numSequence = Integer.parseInt(sc.nextLine());
            Main.sequenceList = new Sequence[numSequence];
            int reward;
            for (int i=0; i<numSequence; i++) {
                temp = sc.nextLine().split(" ");
                reward = Integer.parseInt(sc.nextLine());
                Main.sequenceList[i] = new Sequence(temp.length, temp,
reward);
            }
            sc.close();
        } catch (FileNotFoundException e) {
```

```
        System.out.println("File with path \'" + path + '\'' is not
found");
    }
}
}
```

3.7. WriteFile.java

```
package src;

import java.awt.Color;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class WriteFile {
    String filePath;
    int generated = 0;
    String content;

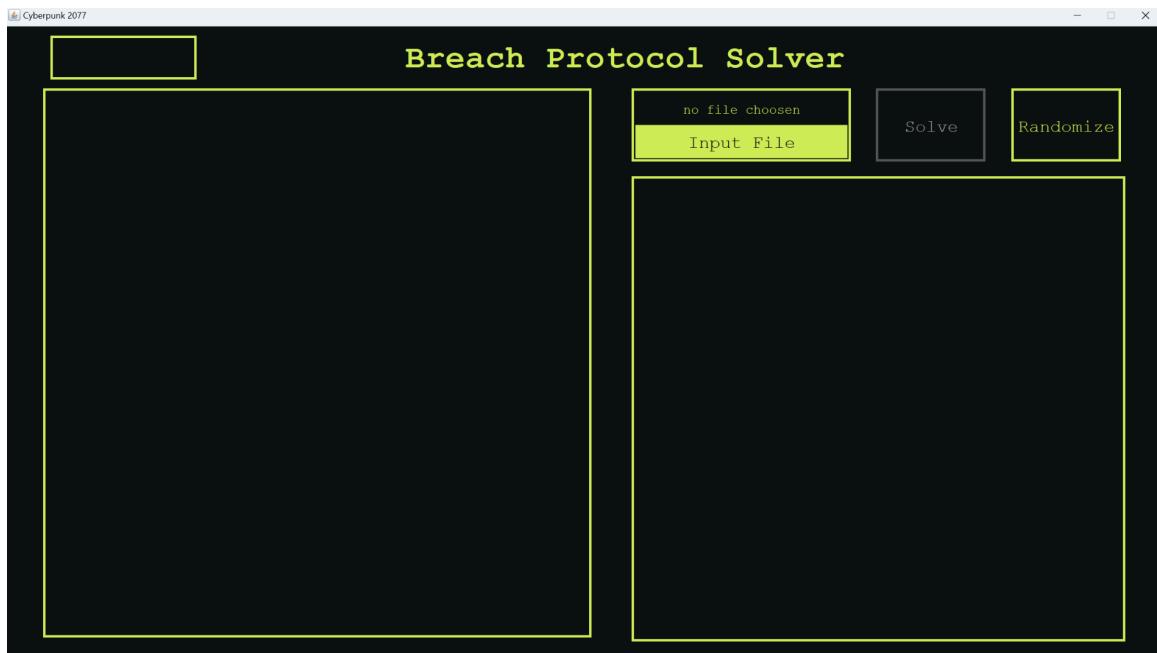
    WriteFile(){
        if (Main.fileName==null){
            filePath = "test/Solved_generated" + generated + ".txt";
            generated++;
        } else {
            filePath = "test/Solved_" + Main.fileName;
        }
        content = Main.maxReward + "\n";
        if (!AnswerFrame.notFound){
            content += AnswerFrame.optimalSequence + "\n";
            int i=0;
            while (i < Main.buffer && !(Main.answerRoute[i].col == -1 &&
Main.answerRoute[i].row == -1)){
                content += (Main.answerRoute[i].col+1) + "," +
(Main.answerRoute[i].row+1) + "\n";
                i++;
            }
        }
        content += "\n";
        content += Main.time + " ms";
        try{
            FileWriter fileWriter = new FileWriter(filePath);

```

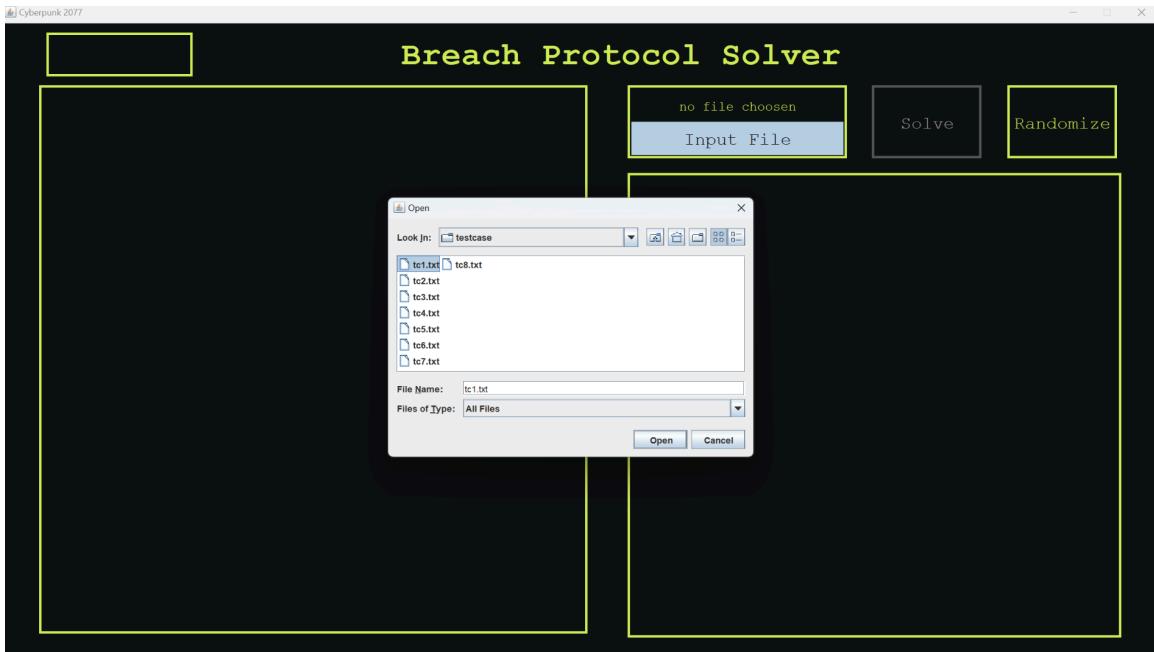
```
        BufferedWriter bufferedWriter = new  
BufferedWriter(fileWriter);  
        bufferedWriter.write(content);  
        bufferedWriter.close();  
        AnswerFrame.messageLabel.setText("<html>Solution successfully  
saved as<br>" + filePath + "</html>");  
        AnswerFrame.messageLabel.setForeground(Frame.Blue);  
    } catch(IOException e) {  
        AnswerFrame.messageLabel.setText("An error occurred while  
saving Solution");  
        AnswerFrame.messageLabel.setForeground(Color.RED);  
    }  
}
```

Masukkan dan Luaran

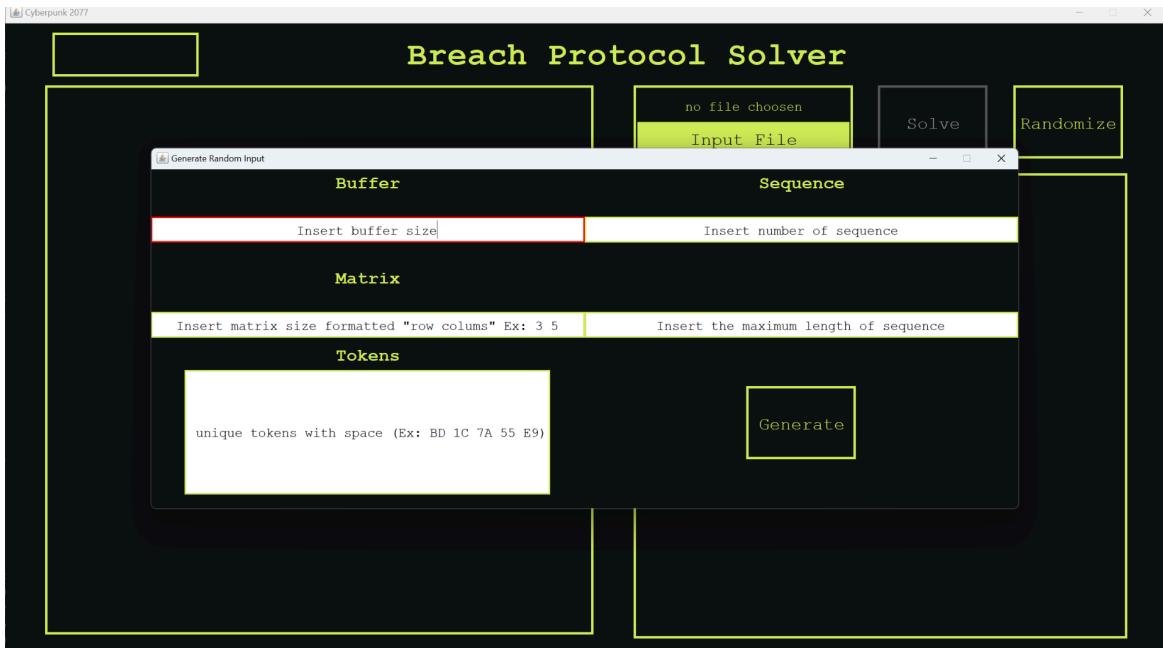
1. Tampilan GUI



(Tampilan Awal)



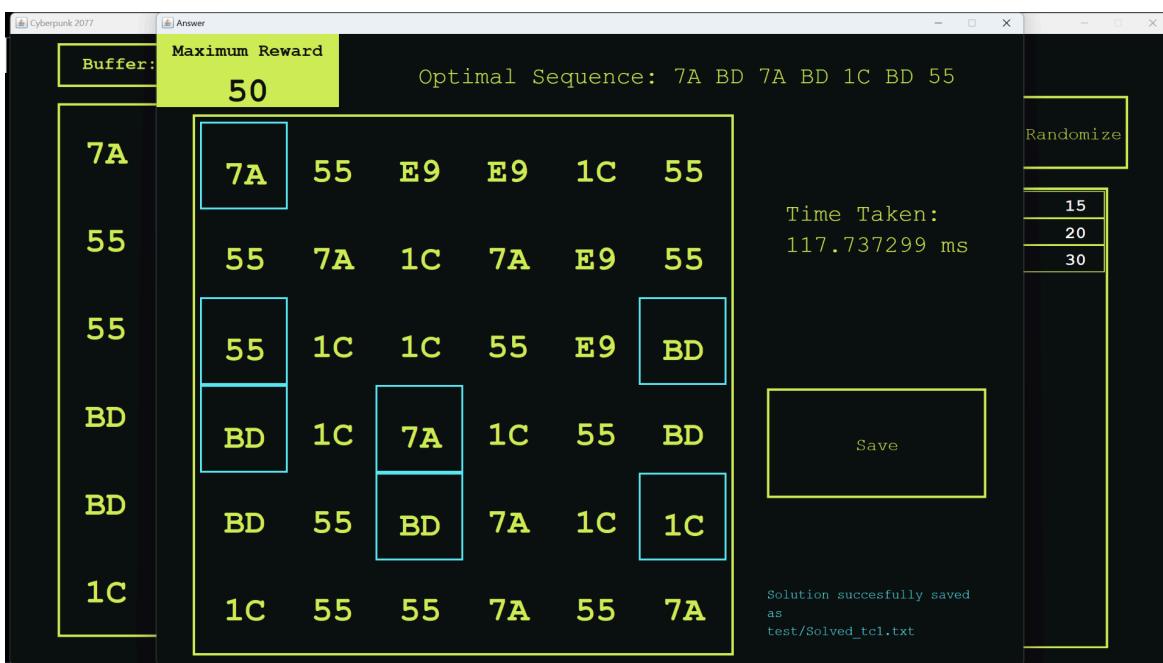
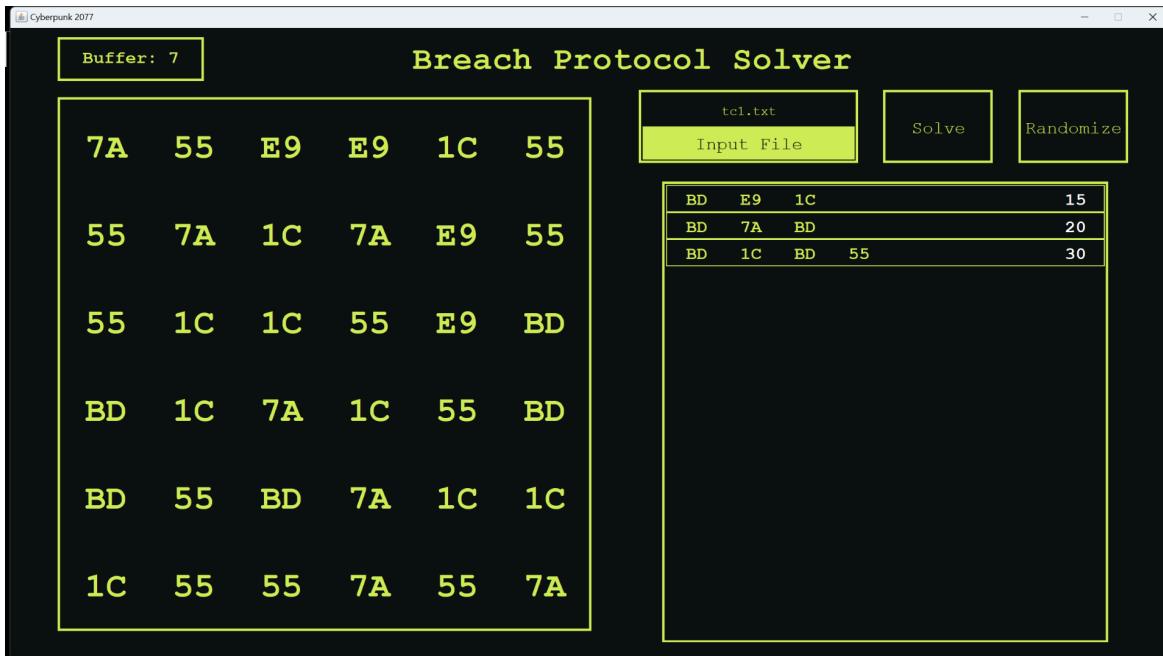
(Fitur Input File)



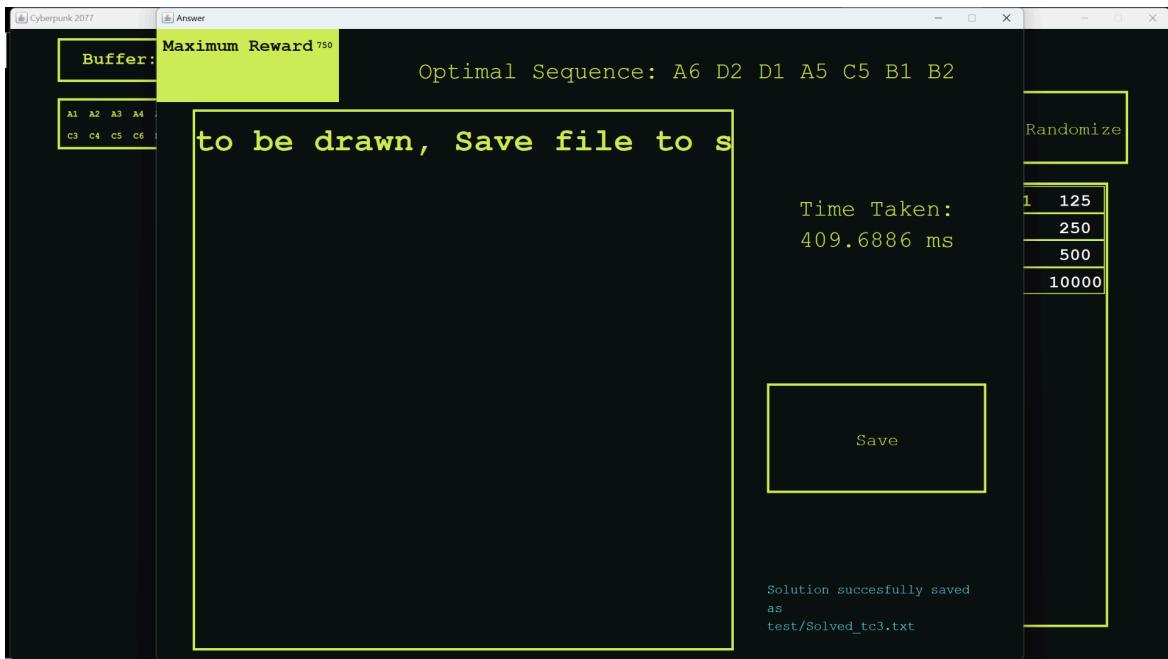
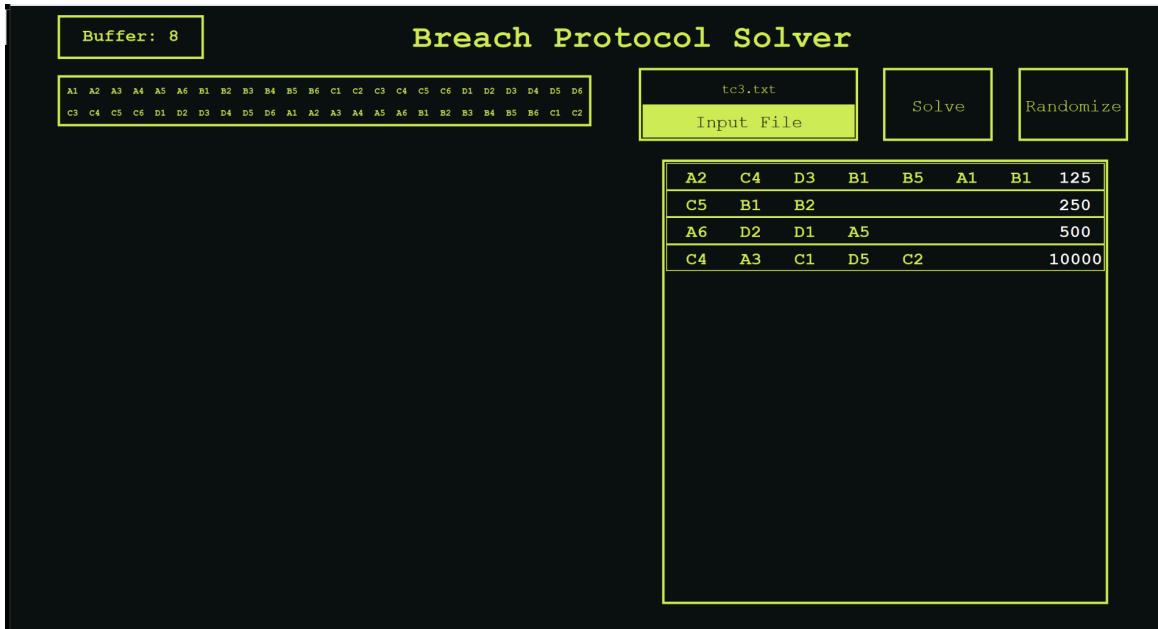
(Fitur membangkitkan Input secara acak)

2. Input dari File

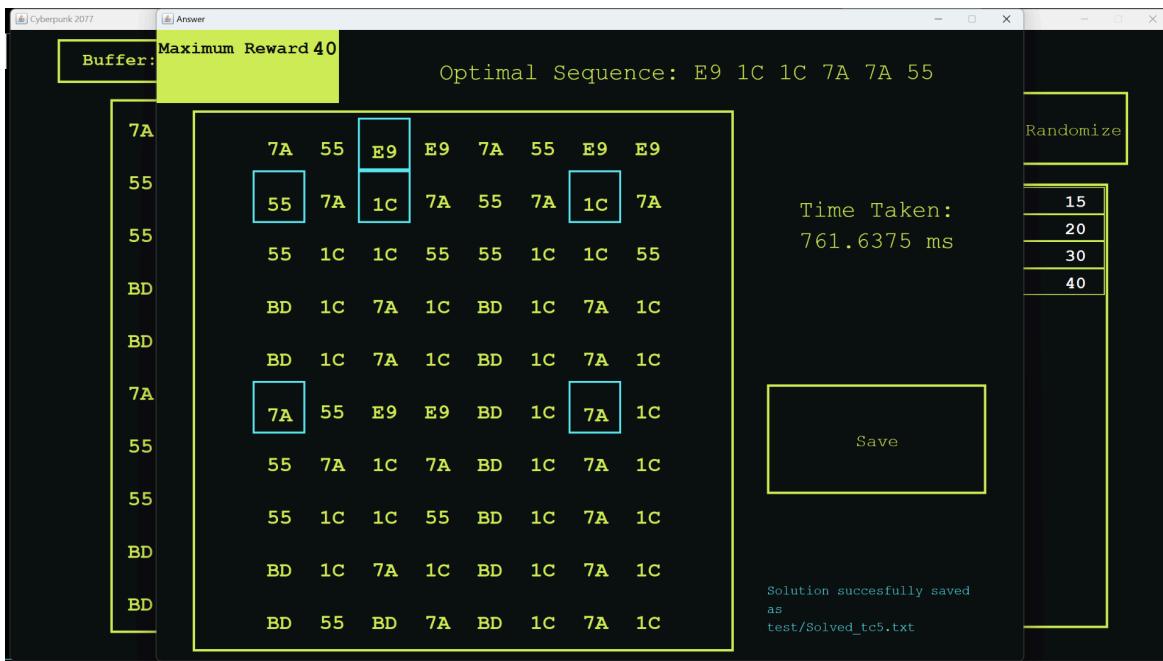
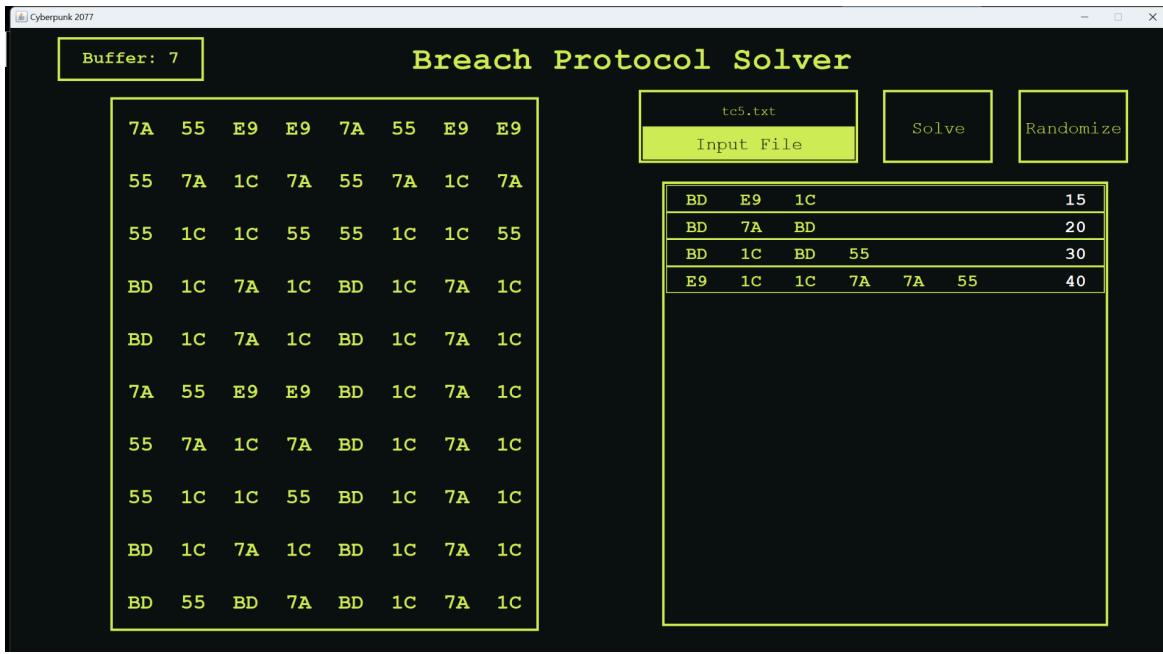
2.1. Tc1.txt



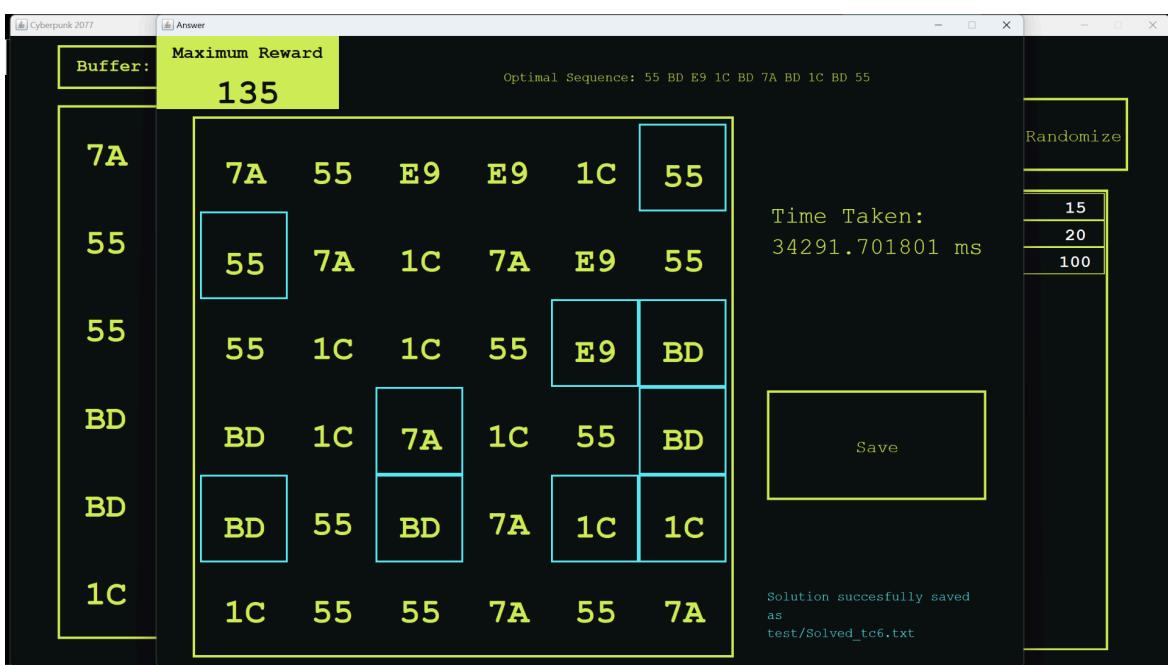
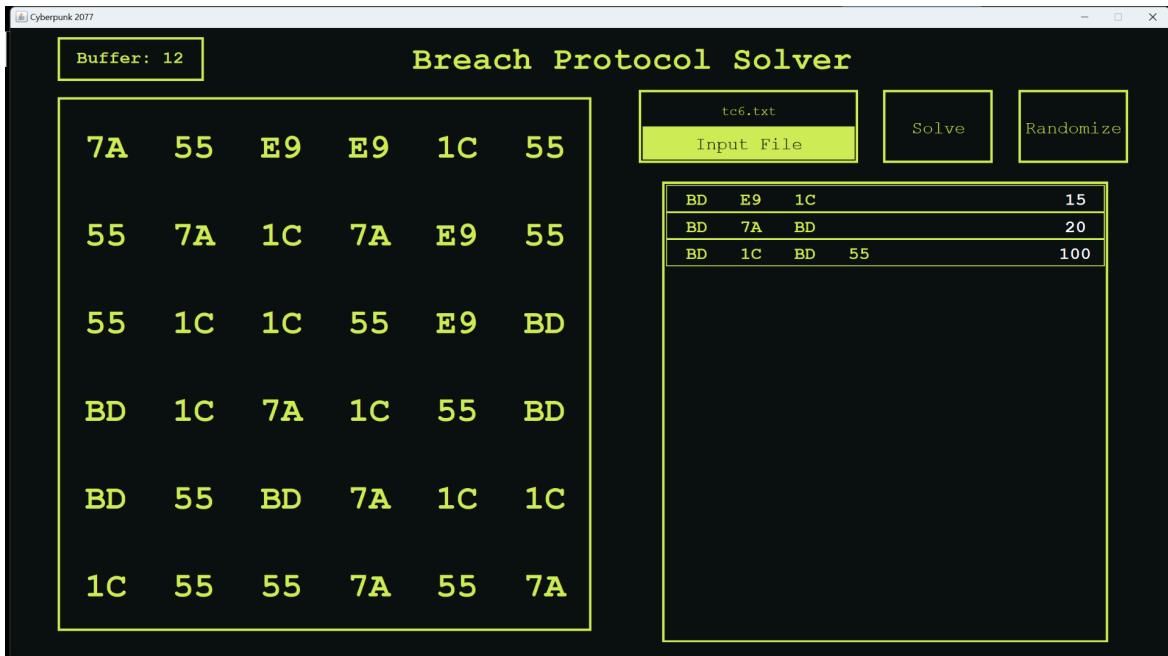
2.2. Tc3.txt



2.3. Tc5.txt



2.4. Tc6.txt

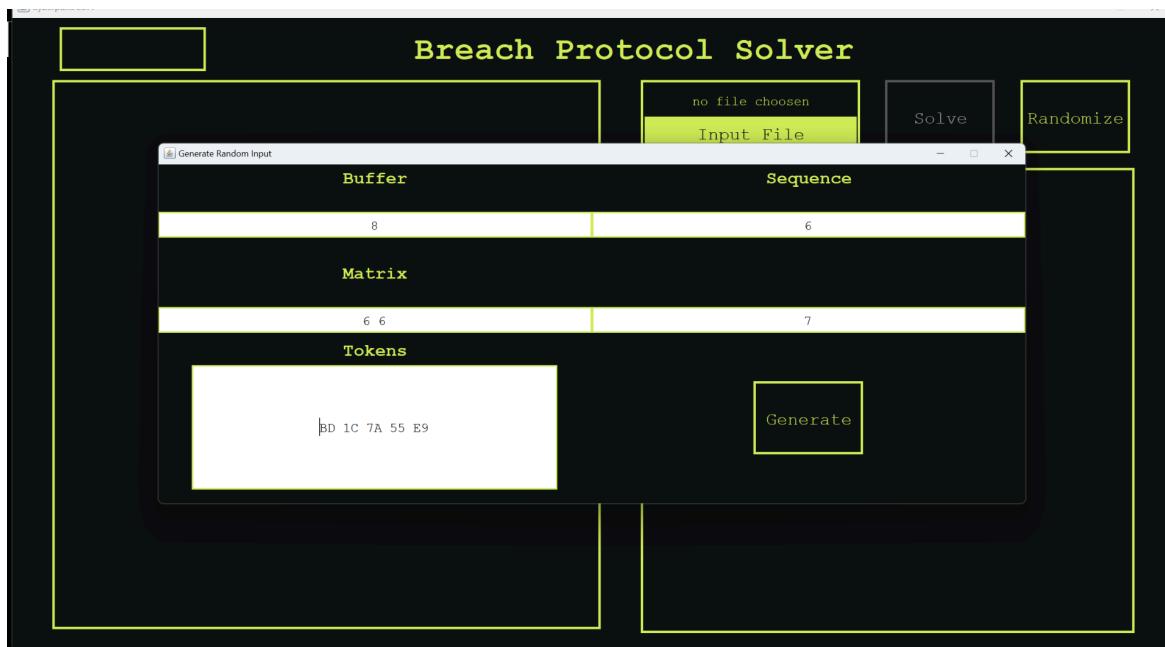
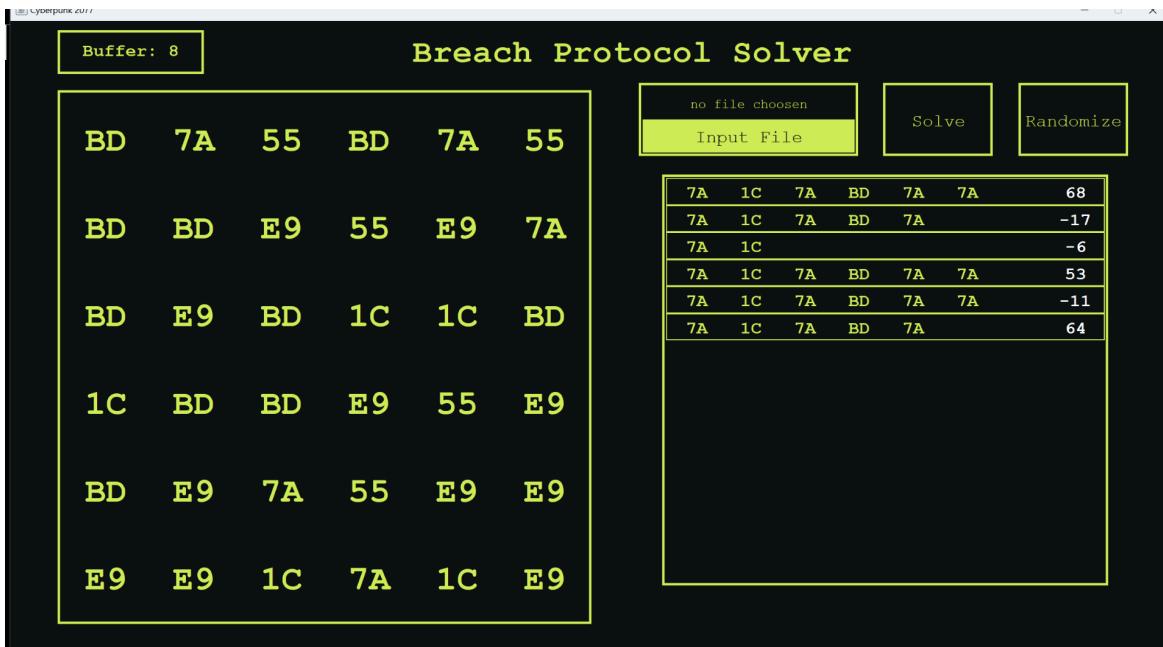


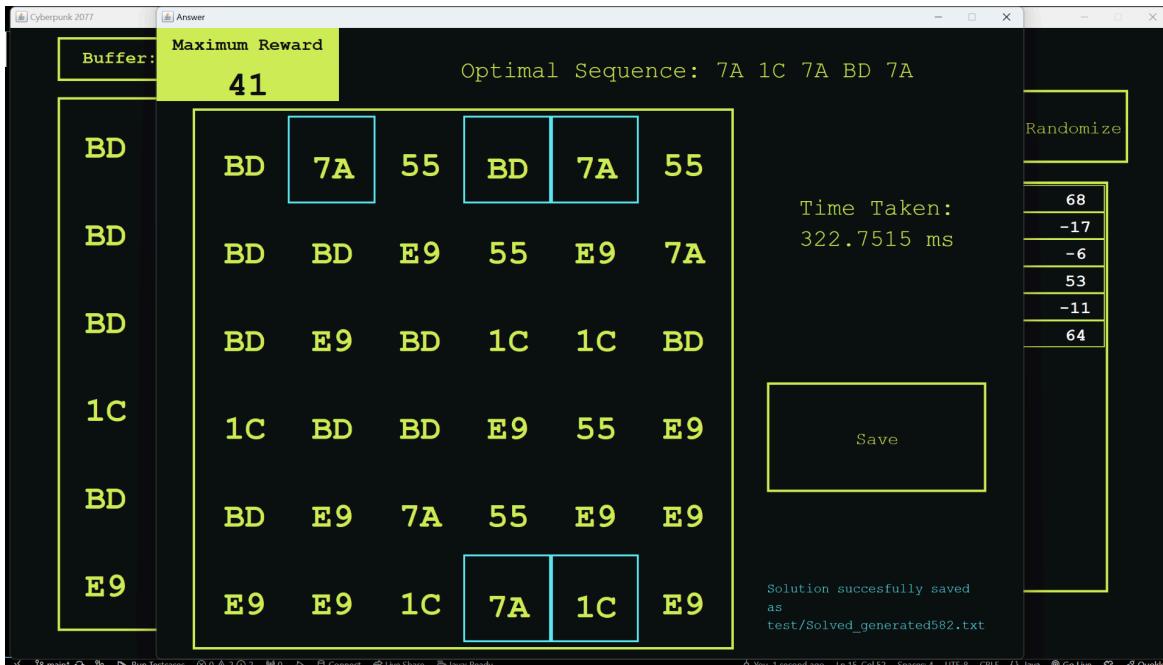
2.5. Tc7.txt



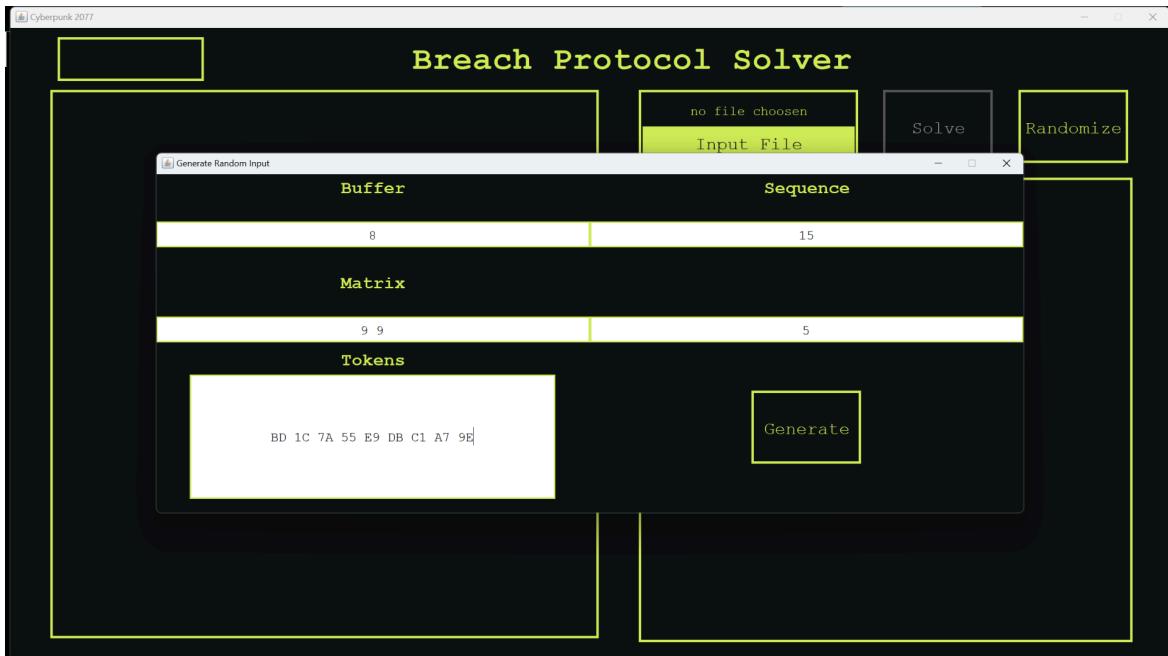
3. Input otomatis

3.1. Percobaan Pertama





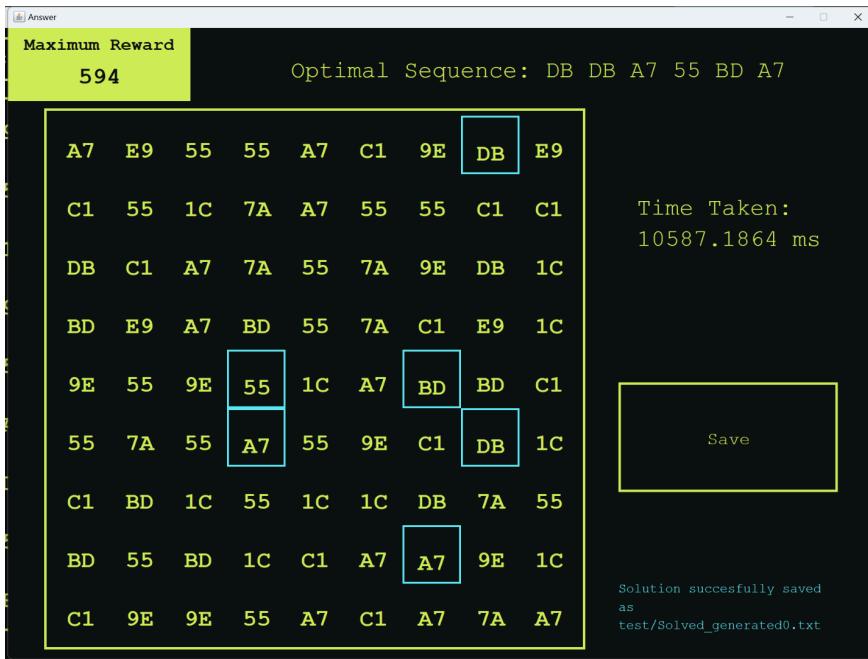
3.2. Percobaan Kedua



Buffer: 8	Breach Protocol Solver									
	A7	E9	55	55	A7	C1	9E	DB	E9	
C1	55	1C	7A	A7	55	55	55	C1	C1	
DB	C1	A7	7A	55	7A	9E	DB	1C		
BD	E9	A7	BD	55	7A	C1	E9	1C		
9E	55	9E	55	1C	A7	BD	BD	C1		
55	7A	55	A7	55	9E	C1	DB	1C		
C1	BD	1C	55	1C	1C	DB	7A	55		
BD	55	BD	1C	C1	A7	A7	9E	1C		
C1	9E	9E	55	A7	C1	A7	7A	A7		

On the right side of the interface, there is a table of results:

DB A7	-17
DB A7 55 BD	57
DB A7 55	79
DB A7 55 BD	31
DB A7 55	23
DB A7 55 BD	60
DB A7	37
DB A7	74
DB A7	-18
DB A7 55	37
DB A7 55 BD	45
DB A7 55 BD A7	3
DB A7 55	88
DB A7	90
DB A7	5



Penutup

1. Kesimpulan

Dari hasil percobaan diatas, terlihat bahwa strategi brute force berhasil menemukan setiap solusi yang ada dari permainan *Breach Protocol*. Benar dengan anggapan bahwa brute force adalah strategi yang mudah karena implementasi dan merancang solusi pada program ini sesungguhnya tidak sulit. Namun, terbukti juga bahwa strategi brute force tergolong algoritma yang lambat, terbukti pada test case 3.2 dan 2.4 yang membutuhkan waktu puluhan detik untuk menemukan solusi. Kompleksitas waktu strategi ini yang sangat tinggi mengakibatkan solusi brute force tidak efisien untuk kasus dengan input yang besar.

Penulis meyakini bahwa program yang ditulis sudah berjalan dengan baik namun masih banyak ruang untuk diperbaiki. Sebut saja GUI yang masih belum konsisten terutama untuk Scroll bar dan graphics garis yang seharusnya muncul pada solusi. Selain itu, implementasi strategi brute force ini mungkin masih belum yang optimal dan penulis sangat terbuka terhadap berbagai implementasi lainnya yang mungkin lebih efisien.

2. Lampiran

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	✓	
2. Program berhasil dijalankan	✓	
3. Program dapat membaca masukan berkas .txt	✓	
4. Program dapat menghasilkan masukan secara acak	✓	
5. Solusi yang diberikan program optimal	✓	
6. Program dapat menyimpan solusi dalam berkas .txt	✓	
7. Program memiliki GUI	✓	

Repo:

https://github.com/Farhannr28/Tucil1_13522037