Perbandingan LCG dan LFSR Berdasarkan Penggunaannya dalam Permainan Minesweeper

Intan Nurjanah (13516131)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13516131@std.stei.itb.ac.id

***Abstract*—Random number generator* atau pembangkit bilangan acak merupakan alat yang dapat membangkitkan suatu bilangan angka secara acak. Ada dua jenis pembangkit bilangan acak, yaitu *pseudo-random number generator* dan *true-random number generator*. Perbedaan dari kedua jenis tersebut terdapat pada bagaimana menghasilkan bilangan acak, misalkan dengan pola tertentu seperti *pseudo-random mumber generator*. Terdapat beberapa algoritma pembangkit bilangan acak. Sebagai contoh, *Linear Congruential Generator* (LCG) dan *Linear Feedback Shift Register* (LFSR). Kedua algoritma tersebut dapat diterapkan pada berbagai pemodelan dan simulasi, salah satunya permainan Minesweeper. Untuk menemukan algoritma yang lebih cocok untuk permainan Minesweeper, dilakukan perbandingan berupa perbandingan jumlah iterasi untuk penempatan ranjau, perbandingan frekuensi ranjau tersisa, dan perbandingan *board* yang dihasilkan. Berdasarkan perbandingan tersebut, setiap algoritma memiliki kelebihan dan kekurangannya masing-masing.**

***Keywords*— LCG, LFSR, bilangan acak, ranjau, papan.**

I. PENDAHULUAN

Random number generator merupakan suatu alat yang dapat membangkitkan bilangan acak. Ada dua jenis pembangkit bilangan acak, yaitu *pseudo-random number generator* (PRNG) dan *true-random number generator*. *Pseudo-random number generator* memiliki pola tertentu dalam menghasilkan suatu bilangan acak sehingga dapat ditebak dengan melihat pola yang terdapat pada *pseudo-random number generator*. Bilangan acak hasil dari *true-random number generator* sulit untuk ditebak dan tidak memiliki pola tertentu untuk menghasilkannya.

Pseudo-random number generator atau disebut juga sebagai *deterministic random bit generator* menghasilkan urutan bit-bit dari sesuatu nilai inisial yang rahasia dinamakan *seed* [1]. *Deterministic random bit generator* (DRBG) dapat ditambahkan beberapa properti sehingga hasilnya tidak dapat diprediksi [1], hal ini juga dinamakan *cryptographic* DRBG. Hasil yang diberikan oleh DBRG bisa saja tidak diprediksi asalkan *seed* yang digunakan terjaga kerahasiaannya dan algoritma yang digunakan juga bagus [2].

Salah satu aspek yang perlu diperhatikan untuk menggunakan suatu pembangkit nilai acak yaitu bagaimana hasilnya tidak dapat diprediksi, tidak hanya hasilnya bahkan hingga urutannya tidak dapat diprediksi [3]. Selain itu juga perlu adanya sifat *random (randomness)*, sebagai contoh koin yang memiliki kemungkinan dihasilkan sisi atas maupun bawah sebesar 1/2. Probabilitas dihasilkan salah satu sisi tersebut tidak mempengaruhi perlakuan berikutnya sehingga dapat disebutkan saling *independent* [3].

Dalam pemanfaatannya, pembangkit bilangan acak semu cocok untuk digunakan dalam aplikasi di mana banyak dibutuhkan bilangan acak. Selain itu, pembangkit bilangan acak semu juga berguna untuk membangkitkan kembali deret bilangan yang sama dengan mudah. Contoh dari algoritma pembangkit bilangan acak semu adalah *Linear Congruential Generator* (LCG) dan *Linear Feedback Shift Register* (LFSR). Setiap algoritma memiliki keunikannya masing-masing. Pada pengujian ini, akan dilakukan perbandingan algoritma LCG dan LFSR berdasarkan penggunaannya untuk menempatkan ranjau pada permainan Minesweeper. Untuk menemukan algoritma yang lebih cocok untuk permainan Minesweeper, akan dilakukan berbagai pengujian.

II. LINEAR CONGRUENTIAL GENERATOR (LCG)

Pembangkit bilangan acak *linear congruential generator* (LCG) adalah PRNG yang memiliki bentuk sebagai berikut.

$$X_n = (aX_{n-1} + b) \bmod m$$

dengan

X_n = bilangan acak ke-n dari deretnya
X_{n-1} = bilangan acak sebelumnya
a = faktor pengali
b = *increment*
m = modulus

Kunci pembangkit adalah X_0 yang disebut sebagai *seed*. LCG mempunyai periode tidak lebih besar dari m dan pada kebanyakan kasus periodenya kurang dari itu. Keunggulan dari LCG terlihat dari sedikitnya operasi bit yang dibutuhkan. Selain itu, LCG juga unggul dari segi kecepatannya [4].

III. LINEAR FEEDBACK SHIF REGISTER (LFSR)

Linear feedback shift register adalah register dari bit-bit yang melakukan operasi langkah diskrit berupa:

1. Menggeser (*shift*) semua bit satu posisi ke kiri dan
2. Mengganti bit yang dikosongkan dengan *exclusive or* dari bit yang digeser dan bit pada posisi *tap* yang diberikan di register.

LFSR mempunyai tiga parameter yang mencirikan deretan bit yang dihasilkannya, yaitu jumlah bit N, *initial seed* (deretan bit yang menginisialisasi register), dan posisi *tap*. LFSR dapat diimplementasikan dalam *hardware* untuk aplikasi yang

membutuhkan pembangkitan deret *pseudo-random* dengan cepat, seperti *direct-sequence spread sprectum radio* [5].

IV. HASIL PERBANDINGAN

Perbandingan terhadap dua pembangkit bilangan acak semu ini akan dilakukan berdasarkan tiga kategori, yaitu perbandingan jumlah iterasi untuk penempatan ranjau, perbandingan frekuensi ranjau tersisa, dan perbandingan papan yang dihasilkan. Sebelum melakukan perbandingan tersebut, akan dijelaskan bagaimana algoritma PRNG untuk LCG dan LFSR yang digunakan beserta cara penempatan ranjau pada papan permainan.

Berikut adalah algoritma PRNG LCG dalam bahasa Java.

```
int cells = size * size; // jumlah petak dalam papan
int m = cells - 1;

SecureRandom rand = new SecureRandom();
int n = rand.nextInt(16);
for (int i = 0; i < mines; i++) {
    if (n > m) {
        i--;
        System.out.println("lebih besar");
    } else {
        int_random_numbers[i] = n;
    }

    int a = rand.nextInt(16);
    int b = rand.nextInt(16);
    n = ((a * a) + b) % m;
}
```

Jumlah bilangan acak semu yang dibangkitkan adalah jumlah ranjau yang akan ditempatkan pada papan permainan. Parameter a dan b akan dibangkitkan menggunakan fungsi `secureRandom` bawaan dari Java. Bilangan acak semu yang dihasilkan kemudian dimodulokan dengan jumlah petak yang ada pada papan permainan untuk menghasilkan nilai baru yang merepresentasikan index petak pada papan permainan.

Berikut adalah algoritma PRNG LFSR dalam bahasa pemrograman Java.

```
public int generateNumber() {
    // initial fill
    boolean[] a = {
        false, true, false, false, false,
        false, true, false, true, true, false
    };
    SecureRandom rand = new SecureRandom();
    int trials = rand.nextInt(40); // jumlah step
    int n = a.length; // panjang register
    int TAP = 8;  // tap position

    // Simulate operation of shift register.
    String result = "";
    for (int t = 0; t < trials; t++) {
        // Simulate one shift-register step.
        // Compute next bit.
        boolean next = (a[n - 1] ^ a[TAP]);
        for (int i = n - 1; i > 0; i--) {
            a[i] = a[i - 1]; // Shift one position.
        }
        a[0] = next; // Put next bit on right end.
        if (next) {
            result += "1";
        } else {
            result += "0";
            }
    }
    int dec = binaryToInteger(result);
    return dec % (size * size); // mod dgn petak
}

public void generateArray() {
    for (int i = 0; i < mines; i++) {
        random_numbers[i] = generateNumber();
    }
}
```

Mula-mula dibangkitkan bilangan acak semu dengan jumlah step dibangkitkan dengan fungsi `secureRandom` bawaan dari Java. Kemudian bilangan acak semu tadi dimodulokan dengan jumlah petak pada papan permainan untuk menghasilkan nilai yang merepresentasikan index petak pada papan permainan.

Setiap bilangan acak semu yang dihasilkan, dengan LCG atau dengan LFSR, kemudian akan disimpan pada *array of integer* bernama `random_numbers`. Berikut adalah algoritma dari penempatan ranjau secara umum dalam bahasa pemrograman Java.

```
for (int i = 0; i < random_numbers.length; i++) {
    if (grid.get(random_numbers[i]) ==
          Minesweeper.BOMB) {
        count = count + 1;
    } else {
        grid.addMine(random_numbers[i]);
    }
}
System.out.println("sisa ranjau: " + count);
```

Sebelum menambahkan ranjau pada suatu petak i, dilakukan pemeriksaan apakah petak tersebut sudah terisi oleh ranjau atau belum. Bila sebelumnya petak sudah terisi ranjau, maka ranjau yang baru tidak akan ditempatkan. Sebagai gantinya, jumlah sisa ranjau `count` ditambahkan untuk menyimpan jumlah ranjau yang tidak dapat ditempatkan.

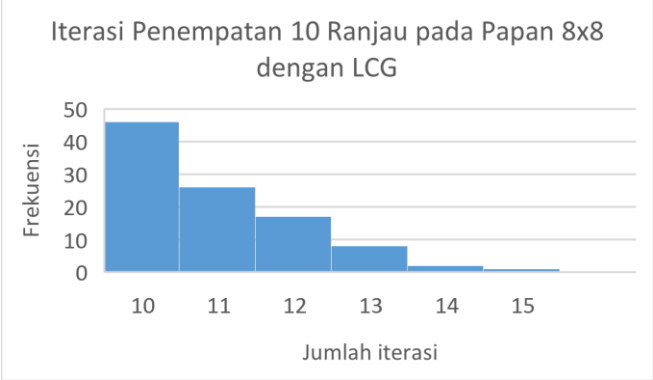
A. Perbandingan Jumlah Iterasi Penempatan Ranjau

Perbandingan dilakukan berdasarkan jumlah iterasi dalam menempatkan ranjau untuk menghasilkan papan permainan yang lengkap. Percobaan ini dilakukan dengan melakukan pembentukan papan dengan spesifikasi:

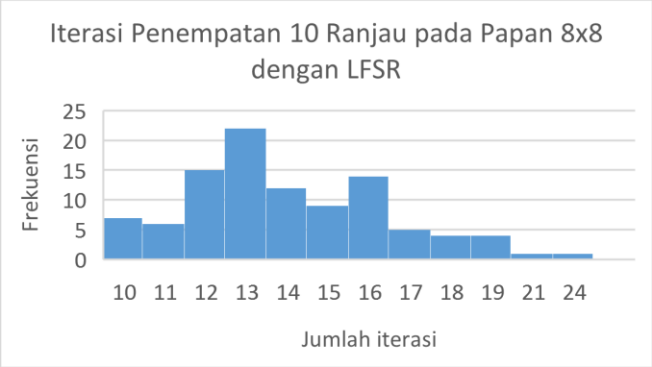
1. Papan 8x8 dengan 10 ranjau
2. Papan 16x16 dengan 40 ranjau
3. Papan 24x24 dengan 99 ranjau

Masing-masing percobaan dilakukan sebanyak 100 kali.

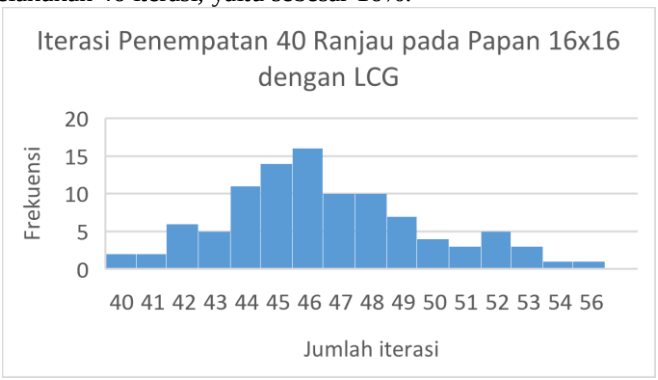
Pada LCG, penempatan 10 ranjau pada papan 8x8 dapat dilakukan dengan 10 kali iterasi, sama dengan jumlah ranjau. Artinya, bilangan acak yang dihasilkan dalam 10 kali pembangkitan tidak menghasilkan bilangan yang sama. Peluang hal ini terjadi adalah 46% atau dalam 46 dari 100 pembentukan papan yang dilakukan.



Sedangkan pada LFSR, penempatan 10 ranjau pada papan 8x8 juga dapat dilakukan dengan 10 kali iterasi. Hanya saja, peluang tersebut hanya terjadi dalam 7 dari 100 pembentukan papan yang dilakukan (7%). Peluang terbesar untuk membentuk papan permainan yang lengkap adalah dengan melakukan 13 iterasi, yaitu 22%.

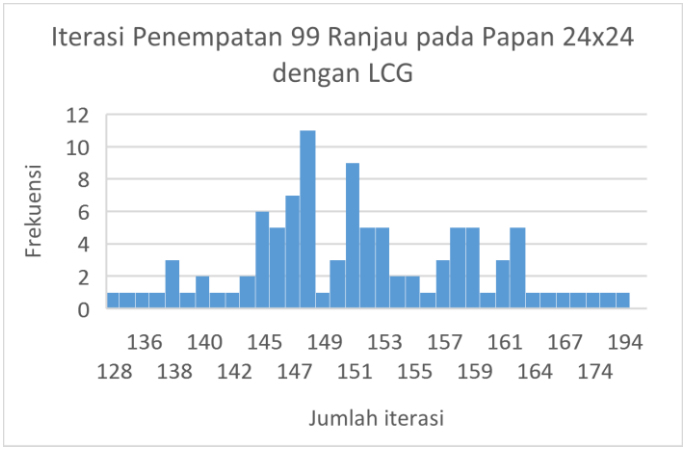


Untuk penempatan 40 ranjau pada papan 16x16, LCG berhasil membentuk papan yang lengkap dalam 40 iterasi. Akan tetapi, peluang hal tersebut terjadi hanya sebesar 2%. Peluang terbesar dalam pembentukan papan yang lengkap adalah dengan melakukan 46 iterasi, yaitu sebesar 16%.



Pada percobaan dengan LFSR, dalam waktu 30 menit pembangkitan bilangan acak yang dihasilkan masih memunculkan bilangan yang sama sehingga papan yang lengkap tidak dapat terbentuk.

Untuk penempatan 99 ranjau pada papan 24x24, LCG mampu menghasilkan papan yang lengkap dengan paling sedikit 128 iterasi dengan peluang 2%. Peluang terbesar dalam pembentukan papan yang lengkap adalah dengan melakukan 148 iterasi, yaitu sebesar 11%.



Sama seperti pada penempatan 40 ranjau pada papan 16x16, dalam waktu 30 menit pembangkitan bilangan acak yang dihasilkan masih memunculkan bilangan yang sama. Bahkan sampai waktu 1 jam pun hal tersebut masih terjadi sehingga papan yang lengkap tidak dapat terbentuk.

Dibandingkan dengan LFSR, LCG memiliki peluang yang lebih besar untuk menghasilkan papan yang lengkap dengan sekali pembangkitan bilangan acak untuk masing-masing ranjau. LFSR hanya dapat menghasilkan papan yang lengkap untuk ukuran papan 8x8 dengan 10 ranjau (Minesweeper level *beginner*). Pembentukan papan permainan dengan ukuran dan jumlah ranjau yang lebih banyak akan membutuhkan waktu lebih dari 1 jam.

B. Perbandingan Frekuensi Ranjau Tersisa

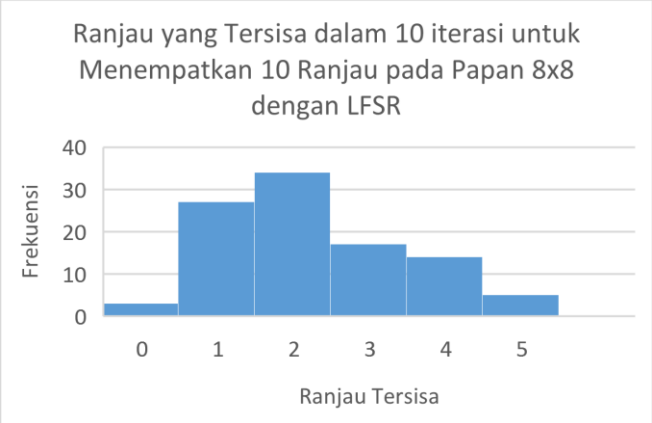
Perbandingan dilakukan berdasarkan frekuensi ranjau tersisa dalam n iterasi untuk jumlah ranjau n. Percobaan ini dilakukan dengan melakukan pembentukan papan dengan spesifikasi:

1. Papan 8x8 dengan 10 ranjau
2. Papan 16x16 dengan 40 ranjau
3. Papan 24x24 dengan 99 ranjau

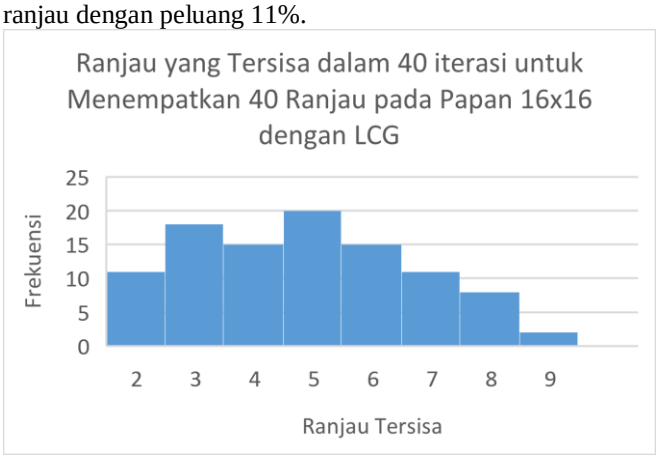
Masing-masing percobaan dilakukan sebanyak 100 kali.

Pada penempatan 10 ranjau pada papan 8x8, LCG mampu untuk tidak menyisakan ranjau. Artinya, bilangan acak yang dihasilkan dalam 10 kali pembangkitan tidak menghasilkan bilangan yang sama. Peluang hal ini terjadi adalah 46% atau dalam 46 dari 100 pembentukan papan yang dilakukan. Peluang terbesar untuk menyisakan ranjau adalah 36%, dengan banyaknya ranjau tersisa adalah 1. Jumlah ranjau paling banyak tersisa adalah 3, dengan peluang hanya sebesar 3%.

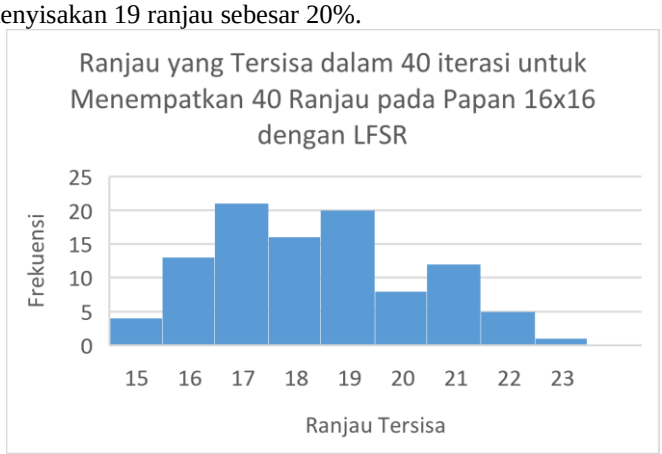
Sedangkan dengan LFSR, peluang untuk tidak menyisakan ranjau hanya sebesar 3%. Peluang terbesar adalah menyisakan sebanyak 2 ranjau, yaitu 34%; disusul dengan peluang menyisakan 3 ranjau sebesar 27%.



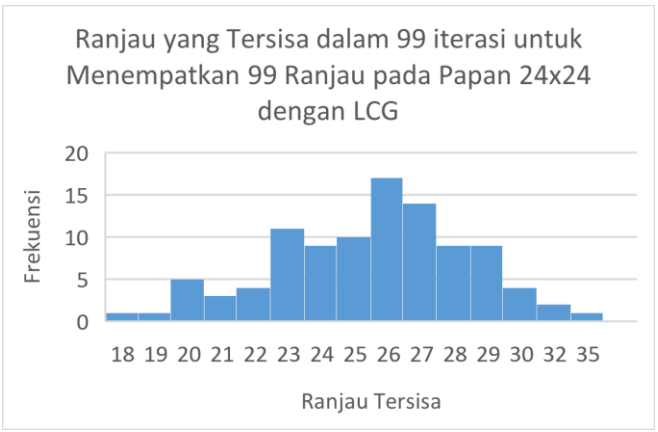
Untuk sisa ranjau pada penempatan 40 ranjau dalam 40 iterasi pada papan 16x16, LCG paling banyak menyisakan 9 buah ranjau dengan peluang 2%; dan paling sedikit menyisakan 2 ranjau dengan peluang 11%.

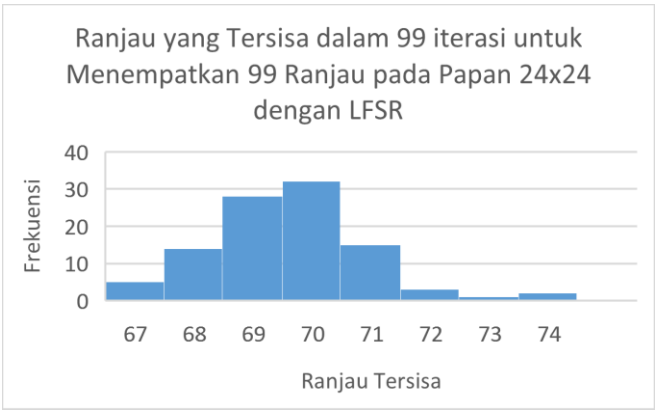


Sedangkan dengan LFSR, jumlah ranjau yang tersisa lebih banyak. Ranjau tersisa paling sedikit adalah 15 dengan peluang kejadian sebesar 4% dan ranjau tersisa paling banyak adalah 23 dengan peluang kejadian sebesar 1%. Peluang terbesar adalah menyisakan 17 ranjau, yaitu 21%; disusul dengan peluang menyisakan 19 ranjau sebesar 20%.



Terakhir, sisa ranjau pada penempatan 99 ranjau dalam 99 iterasi pada papan 24x24, LCG paling banyak menyisakan 35 buah ranjau dan paling sedikit menyisakan 18 ranjau dengan peluang masing-masing sebesar 1%. Peluang terbesar adalah menyisakan 26 ranjau, yaitu 17%; disusul dengan peluang menyisakan 27 ranjau sebesar 14%.





Sedangkan dengan menggunakan LFSR, jumlah ranjau yang tersisa jauh lebih banyak. Ranjau tersisa paling sedikit adalah 67 dengan peluang kejadian sebesar 5% dan ranjau tersisa paling banyak adalah 74 dengan peluang kejadian sebesar 2%. Peluang terbesar adalah menyisakan 70 ranjau, yaitu 32%; disusul dengan peluang menyisakan 69 ranjau sebesar 28%.

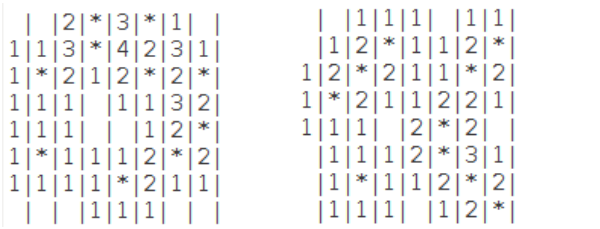
LFSR memiliki peluang lebih besar untuk menyisakan ranjau. Selain itu, jumlah ranjau yang disisakan hampir dua kali lebih banyak dibanding jumlah ranjau yang disisakan LCG. Hal ini terjadi bukan karena bilangan acak yang dihasilkan dari LFSR selalu berulang, tetapi hasil dari modulo pada LFSR untuk dipetakan ke dalam posisi ranjau pada papan menghasilkan nilai yang relatif sama.

C. Perbandingan Papan yang Dihasilkan

Perbadingan kali ini dilakukan dengan meninjau hasil dari penempatan ranjau dari pembangkitan bilangan acak. Spesifikasi papan yang ditinjau, adalah sebagai berikut.

1. Papan 8x8 dengan 10 ranjau
2. Papan 16x16 dengan 40 ranjau
3. Papan 24x24 dengan 99 ranjau

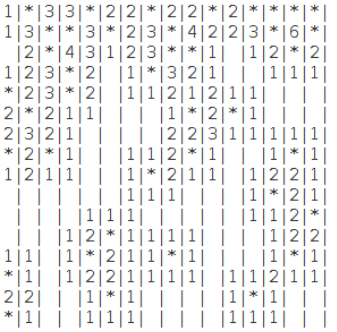
Masing-masing akan dilakukan pembentukan sebanyak 50 kali.



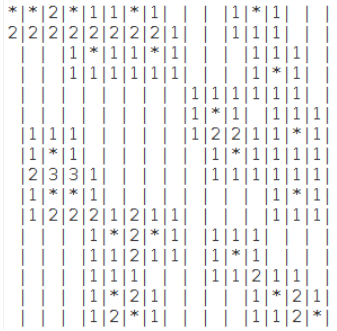
(a) (b)

Gambar 1. Hasil penempatan 10 ranjau dalam papan 8x8 dengan (a) LCG dan (b) LFSR.

Dari Gambar 1, tidak begitu terlihat perbedaan dalam persebaran posisi ranjau. Hal ini karena interval (ukuran papan) dan jumlah bilangan acak (ranjau) yang dibangkitkan masih relatif rendah.



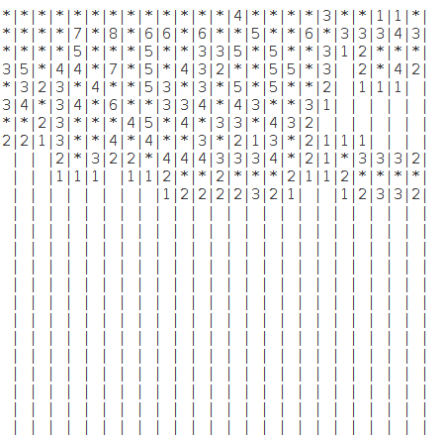
Gambar 2. Hasil penempatan 40 ranjau dalam papan 16x16 dengan LCG.



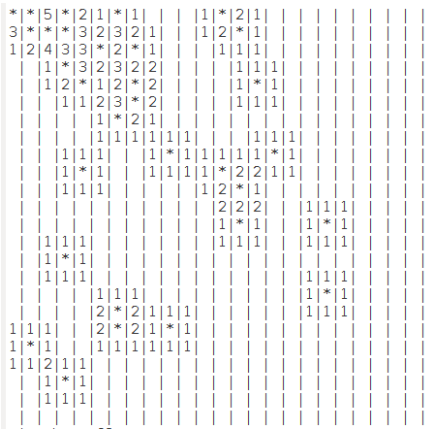
Gambar 3. Hasil penempatan 40 ranjau dalam papan 16x16 dengan LFSR.

Dari Gambar 2 dan Gambar 3, perbedaan dalam persebaran posisi dari ranjau mulai terlihat. Meskipun pada Gambar 4 ranjau yang berhasil ditempatkan hanya 22 ranjau, tetapi dilihat dari persebarannya, hasil penempatan ranjau dengan LFSR lebih merata dibandingkan dengan LCG.

Gambar 4 dan Gambar 5 di bawah ini lebih jelas lagi memperlihatkan perbandingan persebaran posisi ranjau.



Gambar 4. Hasil penempatan 99 ranjau dalam papan 16x16 dengan LCG.



Gambar 5. Hasil penempatan 99 ranjau dalam papan 16x16 dengan LFSR.

Posisi ranjau hasil dari pembangkitan bilangan acak LCG lebih banyak mengisi petak di bagian atas, sedangkan posisi ranjau hasil dari pembangkitan bilangan acak LFSR lebih merata karena mencakup pada petak bagian bawah.

Dari ketiga hasil perbandingan yang telah dilakukan sebelumnya, didapatkan bahwa:

1. LCG memiliki peluang yang lebih besar untuk menempatkan ranjau dengan sekali pembangkitan bilangan acak untuk masing-masing posisi dibanding LFSR.
2. LCG memiliki peluang lebih kecil untuk menyisakan ranjau dan jumlah ranjau yang disisakan lebih sedikit dibanding LFSR.
3. LFSR menghasilkan persebaran ranjau yang lebih merata dibanding LFSR.

V. KESIMPULAN

Hasil yang diberikan oleh kedua PRNG yang diuji memberikan kesimpulan bahwa dalam penggunaannya untuk menempatkan ranjau dalam permainan Minesweeper, LCG memberikan kecepatan yang lebih baik dibandingkan LFSR. Akan tetapi, berdasarkan persebaran posisinya, LFSR menghasilkan persebaran yang lebih merata dibanding PRNG. Sehingga algoritma yang lebih cocok dilihat berdasarkan kriteria yang dibutuhkan. Jika dibutuhkan waktu yang lebih

singkat untuk menempatkan jumlah ranjau yang banyak, LCG lebih direkomendasikan dibanding LFSR. Sedangkan jika dibutuhkan persebaran ranjau yang merata pada papan permainan, LFSR lebih direkomendasikan.

Percobaan ini dapat dilanjutkan dengan mengubah parameter dalam algoritma pembangkitan bilangan acak semu dengan komputasi yang lebih disesuaikan dengan lingkungan permainan Minesweeper. Percobaan ini juga dapat dilanjutkan dengan melakukan pengujian lebih banyak, tidak terbatas pada 100 kali iterasi seperti yang dilakukan pada pengujian ini.

VI. UCAPAN TERIMA KASIH

Penulis mengucapkan syukur kepada Allah S.W.T, karena atas rahmat-Nya, penulis dapat menyelesaikan makalah ini. Selain itu, penulis juga mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T selaku dosen pengampu kuliah IF4020 Kriptografi yang telah memberikan ilmu yang bermanfaat kepada penulis.

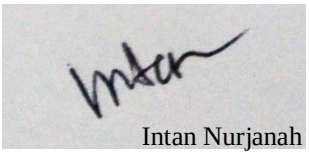
REFERENSI

[1] B. Elaine, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators", NIST, Juni 2015
[2] B. Elaine, "Recommendation for Key Management, Part 1: General", NIST, Januari 2016
[3] R. Andrew, dkk., "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications", NIST, April 2010
[4] Rinaldi Munir, *Pembangkit Bilangan Acak*, ITB, 2018
[5] Robert Sedgewik dan Kevin Wayne, *Linear Feedback Shift Register*, Yale Education, 2012

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 10 Mei 2019



Intan Nurjanah
13516131