IMPLEMENTASI *LINEAR CONGRUENTIAL GENERATOR* UNTUK SISTEM *UPGRADE* DALAM *GAME* BERGENRE *ROGUELIKE*

SKRIPSI

AVIN CHAILI SALIM

201401040

PROGRAM STUDI S-1 ILMU KOMPUTER

FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI

UNIVERSITAS SUMATERA UTARA

MEDAN

2024

IMPLEMENTASI *LINEAR CONGRUENTIAL GENERATOR* UNTUK SISTEM *UPGRADE* DALAM *GAME* BERGENRE *ROGUELIKE*

SKRIPSI

Diajukan untuk melengkapi tugas dan memenuhi syarat memperoleh ijazah Sarjana Ilmu Komputer

AVIN CHAILI SALIM

201401040

PROGRAM STUDI S-1 ILMU KOMPUTER

FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI

UNIVERSITAS SUMATERA UTARA

MEDAN

2024

PERSETUJUAN

Judul	:	IMPLEMENTASI *LINEAR CONGRUENTIAL GENERATOR* UNTUK SISTEM *UPGRADE* DALAM *GAME* BERGENRE *ROGUELIKE*
Kategori	:	SKRIPSI
Nama	:	AVIN CHAILI SALIM
Nomor Induk Mahasiswa	:	201401040
Program Studi	:	SARJANA (S-1) ILMU KOMPUTER
Fakultas	:	ILMU KOMPUTER DAN TEKNOLOGI INFORMASI UNIVRSITAS SUMATERA UTARA

Telah diuji dan dinyatakan lulus di Medan, 8 Juli 2024

Dosen Pembimbing II	Dosen Pembimbing I
Anandhini Medianty Nababan S.Kom., M. T.	Dr. Jos Timanta Tarigan S.Kom., M.Sc.
NIP 199304132021022001	NIP 198501262015041001

Diketahui/Disetujui Oleh
Ketua Program Studi S-1 Ilmu Komputer

Dr. Amalia, S.T., M.T
NIP. 19781221 201404 2 001

PERNYATAAN

IMPLEMENTASI *LINEAR CONGRUENTIAL GENERATOR* UNTUK SISTEM *UPGRADE* DALAM *GAME* BERGENRE *ROGUELIKE*

SKRIPSI

Saya mengakui bahwa skripsi ini adalah hasil karya saya sendiri, kecuali beberapa kutipan yang telah disebutkan sumbernya.

Medan, 8 Juni 2024

Avin Chaili Salim

201401040

PENGHARGAAN

Dengan rasa hormat, peneliti ingin memberikan penghargaan atas dukungan dari berbagai pihak yang telah membantu penelitian ini. Skripsi berjudul *"Implementasi Linear Congruential Generator Untuk Sistem Upgrade Dalam Game Bergenre Roguelike"* ini berhasil diselesaikan atas kerja sama, dukungan, dan bimbingan dari berbagai instansi.

Terima kasih kepada:

1. Sebagai Dekan Fakultas Ilmu Komputer Universitas Sumatera Utara, Dr. Maya Silvi Lydia B.Sc., M.Sc.
2. Dr. Amalia, ST., M.T., sebagai Ketua Prodi Program Studi Ilmu Komputer
3. Bapak Dr. Jos Timanta Tarigan S.Kom., M.Sc., selaku Ketua Laboratorium *Computer Vision* dan Multimedia Program Studi Ilmu Komputer dan juga Pembimbing I yang telah memberikan bimbingan yang berharga.
4. Ibu Anandhini Medianty Nababan S.Kom., M. T., selaku Pembimbing II yang telah memberikan dukungan dan bimbingan yang berharga.
5. Keluarga yang telah mendukung dan memberikan bantuan secara jasmani dan rohani selama penelitian.
6. Al Imamul Luthfi, Andrew Benedictus Jamesie, Ariel Matius Surbakti, dan Wilson sebagai sahabat yang telah memberikan dukungan dan motivasi yang besar kepada penulis selama penelitian.
7. *Playtester* yang telah secara sukarela meluangkan waktu untuk membantu penelitian ini, baik yang memainkan *game* yang telah dibuat maupun yang memberikan kritik dan saran yang membangun.
8. Teman-teman yang berada di program studi Ilmu Komputer yang telah memberikan bantuan yang berarti selama proses perkuliahan.

Peneliti juga mengucapkan terima kasih kepada semua pihak yang telah membantu proses penelitian ini yang tidak dapat disebutkan satu persatu. Semua kontribusi dalam segala bentuk sangat berarti untuk penyelesaian penelitian ini.

Dengan rendah hati, peneliti meminta maaf atas segala kekurangan yang mungkin terdapat dalam penulisan ini. Penulis berharap penelitian ini dapat memberikan kontribusi yang positif terhadap perkembangan ilmu pengetahuan. Terima kasih atas kesempatan ini.

Medan, 8 Juni 2024

Peneliti,

Avin Chaili Salim

201401040

ABSTRAK

Meningkatnya popularitas *game* bergenre *roguelike* belakangan ini menyebabkan diperlukannya sistem randomisasi yang cukup rancu, namun tidak terlalu mengekang pemain. Penelitian ini menggunakan algoritma *Linear Congruential Generator* dalam *Unity* sebagai algoritma randomisasi dasar untuk sistem *upgrade* yang terdapat dalam *game* yang diciptakan dalam penelitian ini. *Linear Congruential Generator* sudah dapat digunakan dalam berbagai aspek seperti *game puzzle* atau sebagai sistem pengacak soal ujian namun tidak dalam *roguelike*. Dengan mengintegrasikan sistem yang digunakan, penelitian ini bertujuan untuk menciptakan sebuah *game 3D action roguelike* yang berjudul *"Rudantara".* Hasil pengujian menunjukkan bahwa algoritma *Linear Congruential Generator* dapat menghasilkan randomisasi yang baik untuk berbagai aspek seperti randomisasi jatuhnya harta karun dari musuh, pemilihan *upgrade,* dan juga pemilihan *stats*. Dengan demikian, implementasi *Linear Congruential Generator* ke dalam *game* terutama dengan genre *roguelike* dapat menciptakan pengalaman yang variatif.

Kata kunci: *Linear Congruential Generator, Game Roguelike, Unity*

IMPLEMENTATION OF LINEAR CONGRUENTIAL GENERATOR FOR UPGRADE SYSTEM IN A ROGUELIKE GAME

ABSTRACT

The increasing popularity of Roguelike games in recent years necessitates a randomization system that, while random, does not held back the player a lot. This research uses Linear Congruential Generator in Unity as the basis for randomization, specifically the upgrade system, that is in the game. Linear Congruential Generator has been used in several aspects such as in puzzle games or randomizing test questions, but not in roguelike specifically. By integrating the system, this study aims to develop a 3D Action Roguelike titled “Rudantara”. Test results indicates that Linear Congruential Generator can be used to generate adequate randomized numbers which would be used for random drops from enemies, randomizing upgrades, and randomizing stats. Thus, the implementation of Linear Congruential Generator algorithm for games, more specifically, roguelike games creates a varying game experience.

Keywords : *Linear Congruential Generator, Roguelike, Unity*

DAFTAR ISI

PERSETUJUAN ii
PERNYATAAN iii
PENGHARGAAN iv
ABSTRAK vi
ABSTRACT vii
DAFTAR ISI viii
DAFTAR GAMBAR x
DAFTAR TABEL 1
BAB 1 2
1.1 Latar Belakang 2
1.2 Rumusan Masalah 4
1.3 Batasan Masalah 4
1.4 Tujuan Penelitian 5
1.5 Manfaat Penelitian 5
1.6 Metodologi Penelitian 5
1.7 Sistematika Penulisan 6
BAB 2 8
2.1 Roguelike 8
2.2 Linear Congruential Generator 9
2.3 Unity 11
2.4 Rudantara 12
BAB 3 14
3.1 Analisis Sistem 14
3.1.1 Analisis Masalah 14
3.1.2 Analisis Kebutuhan 14
3.2 Perancangan Sistem 15
3.2.1 Pendefinisian *stats* dan *upgrade* 15
3.2.2 Pembuatan sistem randomisasi 15
3.2.3 Pembuatan sistem untuk memberikan *upgrade* kepada pemain 17

3.3 Desain User Interface 19
3.4 Rencana Pengujian 19
3.4.1. Pengujian Tingkat Kerancuan Hasil Randomisasi 19
3.4.2 Pengujian Hasil Algoritma Oleh *Playtester* 19
BAB 4 21
4.1. Video Game "Rudantara" 21
4.2 Pengujian Program 23
4.2.1 Analisis Randomisasi 23
4.2.2 Hasil Pengujian Oleh Sampel 34
BAB 5 35
5.1 Kesimpulan 35
5.2 Saran 35
DAFTAR PUSTAKA 36
LAMPIRAN 37

DAFTAR GAMBAR

Gambar 1.1 *Game Roguelike* yang dirilis dari tahun ke tahun.....2
Gambar 2.1 Contoh *game roguelike* yang terkenal, *Hades*.....8
Gambar 2.2 Logo *Unity*.....11
Gambar 3.1 Flowchart cara kerja sistem.....18
Gambar 4.1 Tampilan *Main Menu Game*.....21
Gambar 4.2 Tampilan *gameplay* dari *game*.....22
Gamber 4.3 *UI* ketika pemain melakukan *pause* terhadap *game*.....22
Gambar 4.4 *UI* status pemain.....22
Gambar 4.5 Grafik hasil randomisasi *random drop*.....26
Gambar 4.6 Grafik hasil randomisasi pemilihan *rarity* dari *upgrade*.....29

DAFTAR TABEL

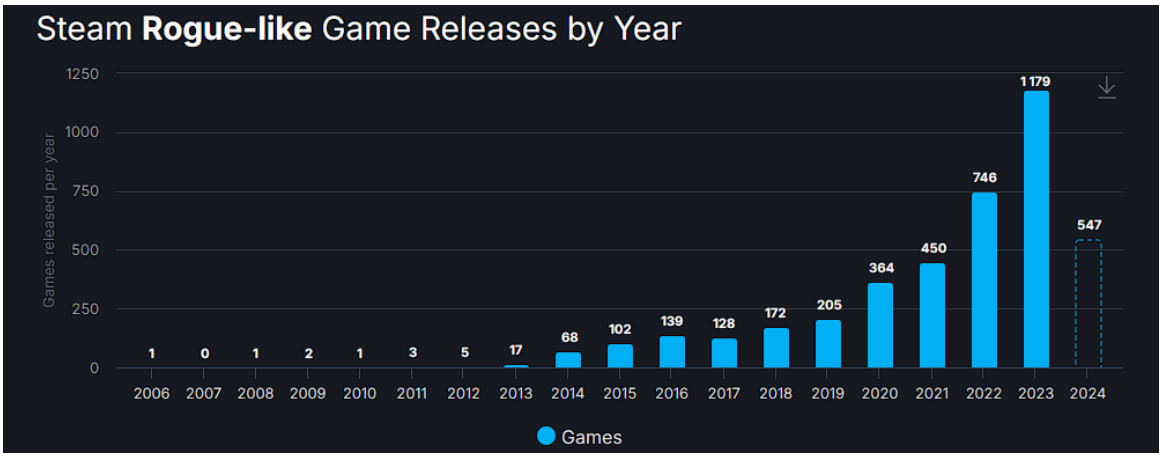
Tabel 4.1 Daftar *upgrade* yang ada dalam *game* 24
Tabel 4.2 Hasil uji coba *random drop* dengan X0 = 395413116 25
Tabel 4.3 Hasil uij coba penentuan *rarity* dengan X0 = 449064552 28
Tabel 4.4 Urutan *upgrade* dengan tingkatan *common* 29
Tabel 4.5 Urutan *upgrade* untuk tingkatan *rare* 30
Tabel 4.6 Hasil uij coba penentuan *upgrade* dengan X0 = 17 30
Tabel 4.7 Frekuensi kemunculan setiap *upgrade* 31
Tabel 4.8 Hasil uij coba penentuan *stats* dari *upgrade* dengan X0 = 17 32
Tabel 4.9 Frekuensi kemunculan nilai dari *stats* untuk *upgrade* 33
Tabel 4.10 Hasil kuesioner implementasi *Linear Congruential Generator* 34

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Sejak tahun 2018, jumlah *game* bergenre *Roguelike* yang dirilis di platform Steam terus menerus meningkat hingga pada tahun 2023, lebih dari 1000 *game* dengan genre ini sudah dirilis di platform tersebut seperti yang terlihat pada gambar 1.1. *Roguelike* adalah genre dalam *game* yang dikarakterisasikan dengan kekalahan berulang-ulang yang disebabkan oleh tingkat kesulitan dalam *game* yang tinggi sehingga pemain diharuskan untuk memulai *game* dari awal secara berulang-ulang (Szabados et al., 2022). Genre ini sudah menjadi genre utama dalam *game* yang terkenal saat ini seperti Demon's Souls. Salah satu hal yang harus diperhatikan dalam *game*, terutama *Roguelike*, adalah nilai *replayability* dari suatu *game*. Untuk meningkatkan nilai *replayability* tersebut, diperlukan suatu cara agar pengalaman yang dialami pemain berbeda dengan pengalaman yang sudah dialaminya. Dalam genre *Roguelike*, nilai tersebut ditingkatkan dengan mengandalkan *Random Number Algorithm* (Witney, 2019).



Gambar 1.1 *Game Roguelike* yang dirilis dari tahun ke tahun

Salah satu cara untuk memperoleh randomisasi adalah dengan menggunakan *Pseudo Random Number Generator* (PRNG). PRNG adalah algoritma perangkat lunak yang menghasilkan deretan nilai yang perkiraannya mirip dengan nilai acak (Okada et al., 2023). Salah satu algoritma PRNG adalah *Linear Congruential Generator* (LCG). Algoritma ini adalah sudah digunakan dalam berbagai aspek seperti dalam *game puzzle*, randomisasi urutan soal yang muncul dalam ujian, maupun untuk keamanan. Namun, belum ada data yang menunjukkan bahwa algoritma ini pernah digunakan dalam *game roguelike*.

Algoritma ini akan digunakan di dalam *game roguelike* yang bernama Rudantara. Dalam *game* ini, pemain bermain sebagai petualang didalam sebuah peta level yang dibentuk dengan *procedural generation*. Dalam peta ini pemain diberikan waktu lima menit untuk mengalahkan bos lantai untuk mencapai lantai selanjutnya. Permainan berakhir ketika pemain kehabisan HP (*Health Point*. Pemain dapat mengalahkan musuh yang muncul secara acak untuk meningkatkan kemampuan pemain melalui *upgrade* atau *equip* yang dapat muncul secara acak ketika pemain mengalahkan musuh.

Randomisasi dari *upgrade* atau *equip* yang didapatkan pemain dalam *game* ini akan menggunakan algoritma *Linear Congruential Generator* (LCG). Semua *upgrade* yang memungkinkan didapat pemain akan dimasukkan ke dalam sebuah daftar dan kemudian *upgrade* tersebut diberikan nomor sesuai urutannya dalam daftar tersebut. LCG kemudian dijalankan untuk mendapatkan sebuah nilai dimana nilai yang didapatkan akan merujuk pada salah satu dari *upgrade* yang berada di dalam daftar tersebut, Kemudian, LCG akan dilakukan sekali lagi untuk memberikan nilai terhadap upgrade yang akan diberikan kepada pemain. Hal ini dilakukan sebanyak tiga kali untuk memberikan kepada pemain pilihan yang bisa diambil agar pemain merasa mendapatkan kendali dalam pemilihan dari *upgrade* yang pemain inginkan.

Ada berbagai bukti yang menunjukkan kemampuan LCG sebagai algoritma randomisasi untuk digunakan dalam berbagai hal. Pada penelitian (Safitri et al., 2021), algoritma LCG berhasil digunakan untuk menghasilkan nilai acak dalam

membuat *game puzzle* yang bertemakan budaya. Begitu juga dengan penelitian (Siahaan & Hendrik, 2022) yang menunjukkan LCG dapat juga digunakan sebagai algoritma yang mengacak kemunculan soal dalam *game* edukasi pembelajaran angka dan alfabet. Pada penelitian (Limbong & Matondang, 2021), sebuah *game* dimana pemain memilih salah satu koper berisis uang yang tersedia. Nilai yang ada dalam setiap koper diacak menggunakan algoritma LCG. Pada penelitian (Desliyanti & Fahry, 2023), sebuah aplikasi pembelajaran Bahasa Inggris dibuat dengan menggunakan LCG untuk merandomisasikan urutan soal dalam aplikasi tersebut, dimana soal-soal yang muncul berasal dari kumpulan soal yang banyak dan kemudian dirandomisasikan untuk mengambil 10 soal diantaranya, Pada penelitian (Makmur et al., 2019), sebuah aplikasi ujian berbasis elektronik dibuat dengan menggunakan LCG untuk merandomisasikan urutan soal yang muncul dari setiap siswa satu sama lain untuk menurunkan kemungkinan Tindakan kecurangan.

Dalam penelitian ini, algoritma LCG akan digunakan sebagai algoritma utama untuk menghasilkan dan menentukan seberapa kuat *upgrade* yang akan diterima pemain dalam *game* bergenre *Roguelike*, sehingga akan dilakukan lebih dari satu randomisasi secara beruntun di dalam penelitian ini.

1.2 Rumusan Masalah

Game bergenre *Roguelike* membutuhkan algoritma randomisasi yang baik, fleksibel, dan dapat dimodifikasi sesuai kebutuhan dengan berkembangnya *game* tersebut. Algoritma randomisasi yang disediakan Unity tidak bisa dimodifikasi sama sekali sehingga dibutuhkan algoritma yang dibuat langsung oleh *developer* sendiri. *Linear Congruential Generator* yang sudah digunakan dalam berbagai hal untuk randomisasi diharapkan dapat digunakan sebagai algoritma randomisasi dasar yang memenuhi kebutuhan yang sudah disebutkan.

1.3 Batasan Masalah

Beberapa batasan masalah yang terdapat dalam penelitian ini adalah sebagai berikut,

1. Menggunakan *Linear Congruential Generator* pada *game* dengan genre *Roguelike*.
2. *Game* yang digunakan dibuat dalam platform Unity dan menggunakan Bahasa C#.
3. Yang akan dirandomisasikan adalah sistem *Upgrade* dan nilainya dalam aplikasi.
4. *Output* yang dihasilkan adalah tiga buah pilihan *Upgrade* yang dapat dipilih pemain.

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah menciptakan sebuah *game* bergenre *roguelike* dengan menggunakan algoritma *Linear Congruential Generator* sebagai algoritma randomisasi dasar untuk sistem *upgrade* dalam *game* untuk menentukan apakah *Linear Congruential Generator* dapat digunakan sebagai algoritma randomisasi dalam *game* dengan genre *roguelike*.

1.5 Manfaat Penelitian

Adapun beberapa manfaat yang diharapkan dari penelitian ini adalah sebagai berikut,

1. Memberikan informasi mengenai penerapan algoritma *Linear Congruential Generator* dalam *Game Development* berbasis Unity.
2. Menjadi referensi yang berguna dalam pengembangan sistem randomisasi dalam *Game Development* berbasis Unity.

1.6 Metodologi Penelitian

Di bawah ini merupakan metodologi penelitian yang dilakukan dalam penelitian ini dapat dilihat pada Gambar 1 berikut ini.

- Studi Pustaka

  Pada tahap ini, penelitian dimulai dengan melakukan studi literatur untuk mencari referensi dari berbagai sumber dan melakukan pengumpulan buku, jurnal, *e-book*, artikel ilmiah, makalah, buku instruksi manual, ataupun situs

internet yang membahas tentang *Linear Congruential Generator (LCG)* dan pengimplementasiannya, serta *game* dengan genre *Roguelike.*

- Analisis dan Perancangan Sistem

  Berdasarkan ruang lingkup penelitian, pada tahap ini penulis melakukan proses analisis terhadap apa saja yang diperlukan dalam penelitian dan merancangnya dalam sebuah diagram alir (*flowchart*).

- Implementasi

  Pada tahap ini, dilakukan pembuatan sebuah sistem randomisasi menggunakan *Linear Congruential Generator* ke dalam *game* dengan menggunakan Bahasa pemrograman C# dalam aplikasi Unity untuk menghasilkan *output* sesuai dengan sistem yang telah dirancang.

- Pengujian dan Dokumentasi

  Pada tahap ini, dilakukan uji coba terhadap penelitian yang telah dilaksanakan dan hasil dari penelitian didokumentasikan dalam bentuk laporan akhir.

1.7 Sistematika Penulisan

Sistematika penulisan dari skripsi ini terdiri dari lima bab, yakni:

BAB 1 PENDAHULUAN

Bab ini berisi segala aspek yang berkaitan dengan penelitian, termasuk rumusan masalah, Batasan masalah, tujuan penelitian, manfaat penelitian, metode penelitian, dan struktur penulisan, diuraikan dengan mendetail dan komprehensif dalam bab ini.

BAB 2 LANDASAN TEORI

Bab ini berisi tinjauan teoritis yang berkaitan dengan *Linear Congruential Generator, game,* dan *Roguelike* dibahas dalam bab ini.

BAB 3 ANALISIS DAN PERANCANGAN SISTEM

Bab ini berisi analisis masalah dan sistem yang dibangun dibahas pada bab ini, yang kemudian diikuti dengan perancangan sistem dengan menggunakan *Linear Congruential Generator*.

BAB 4 IMPLEMENTASI DAN PENGUJIAN SISTEM

Bab ini berisi implementasi dan pengujian dari sistem yang telah dibuat dan dibangung dalam tahapan analisis dan perancangan.

BAB 5 KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan dari penelitian yang sudah dilaksanakan dan saran untuk penelitian selanjutnya.

BAB 2

LANDASAN TEORI

2.1 Roguelike

Roguelike sebagai sebuah genre masih belum memiliki konsensus untuk mendefinisikan sebuah game termasuk *roguelike* atau tidak. Nama '*Roguelike*' sendiri berasal dari sebuah *game* yang bernama 'Rogue', sehingga *roguelike* berarti game yang mirip dengan 'Rogue'. 'Rogue' dikarakterisasikan dengan struktur *level* yang dihasilkan dengan *procedural generator*, randomisasi, dan *permadeath*. Karena *roguelike* sebagai genre didasari dari 'Rogue', maka setiap alur permainan yang satu dengan yang lain selalu berbeda karena bentuk *level*, peletakan musuh, dan juga senjata maupun *upgrade* yang bisa didapatkan pemain setiap alur selalu berbeda. Hal ini menciptakan dorongan kepada pemain untuk terus menerus mencoba dan mengulang untuk mendapatkan semua kombinasi yang bisa dicoba. (N et al., 2022). Salah satu *game roguelike* yang terkenal adalah "Hades" yang dapat dimainkan pada platform *Steam*, seperti pada gambar 2.1.

Gambar 2.1 Contoh *game roguelike* yang terkenal, *Hades*

Roguelike memiliki beberapa faktor dasar yang harus diperhatikan dalam pembuatannya, yakni *gameplay loop* dan *replayability*. *Roguelike* dibangun berdasarkan satu elemen yang wajib dipenuhi oleh setiap *roguelike*: pemain akan kalah secara berulang-ulang untuk memulai kembali dari awal, kehilangan semua hasil kerja keras yang sudah dicapai. Dampak yang dipandang ekstrem ini menyebabkan *roguelike* secara garis besar tidak menembus pasar besar. Namun, *roguelike* memang dibangun berdasarkan aturan ini. Melalui kekalahannya, pemain akan belajar mengenai cara bermain yang optimal, pilihan terbaik dari randomisasi yang diberikan. maupun kesalahan yang tidak boleh diulangi, menciptakan perulangan yang terjadi terus menerus hingga pemain menguasai *game* yang dimainkan. Inilah *gameplay loop* dasar dari setiap *roguelike*.

Meski pemain sudah berhasil menamatkan *game* tersebut, *roguelike* yang baik haruslah mempunyai konten yang tidak bisa didapat hanya dengan sekali penamatan. Hal-hal ini dapat memiliki banyak bentuk seperti *upgrade* maupun *item* baru, *level* yang berbeda dari sebelumnya, atau bahkan cerita tambahan. Walaupun konten-konten baru ini tentu saja meningkatkan *replayablity* dari sebuah *game,* membuat pemain merasakan ada banyaknya kombinasi-kombinasi yang dapat mereka lakukan hanya dengan konten dasar saja dapat menjadi daya tarik sendiri untuk pemain memainkan kembali *game* yang sudah mereka tamatkan dalam genre ini (Bycer, 2021).

2.2 Linear Congruential Generator

Menghasilkan deretan angka yang acak dapat berasal dari dua cara, *True Random Number Generator* (TRNG) dan *Pseudo Random Number Generator* (PRNG). TRNG terjadi dengan adanya aksi fisik yang secara bawaan memanglah acak. TRNG tidak cocok untuk menghasilkan deretan angka acak yang panjang karena proses yang lama dari melakukan aksinya dan kemudian mengubah aksi tersebut dari bentuk fisik ke bentuk digital.

Sementara itu, PRNG menggunakan algoritma untuk menciptakan deretan angka acaknya. Algoritma ini menggunakan nilai awal yang disebut *seed* dan menggunakannya

secara berulang-ulang untuk menciptakan deretan angka yang panjang. Oleh karenanya, PRNG mudah digunakan di dalam *hardware* maupun *software* (Krishnamoorthi et al., 2021).

Linear Congruential Generator (LCG) adalah salah satu algoritma PRNG yang banyak digunakan untuk menghasilkan deretan angka acak. Algoritma ini ditemukan oleh Dr. Lehmer pada tahun 1951 dimana nilai dari hasil randomisasi dari algoritma bergantung pada nilai awal yang disebut *seed* (Elveny et al., 2020). Berikut adalah perhitungan untuk LCG:

$$X_n = (aX_{n-1}) + c \; mod \; m$$

Dimana:

Xn = Angka acak ke-n

Xn-1 = Angka acak sebelumnya

a = Pengali

c = Penambah

m = Modulus

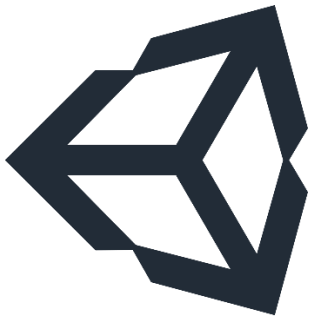
Nilai-nilai awal yang dihasilkan melalui LCG tidak akan melebih modulus (m). Ketika nilai acak sudah melebihi modulus ada kemungkinan terjadinya nilai repetitif setelah melewat beberapa periode tertentu. Untuk menghindari nilai repetitif ini, penentuan nilai-nilai konstan dalam perhitungan (a, b, dan m) sangatlah penting untuk menciptakan angka-angka acak yang terlihat tidak repetitif dalam beberapa kali pengujian. (Makmur et al., 2019)

Untuk memperoleh periode yang lengkap dimana semua nilai yang mungkin didapatkan (m-1), maka ada beberapa kondisi yang harus dipenuhi yakni:

- c relatif prima terhadap m,
- a-1 dapat dibagi dengan semua faktor prima dari m,
- m lebih besar dari a, c, dan x0 (*seed*), dan
- a dan c lebih besar dari 0. (Desliyanti & Fahry, 2023)

2.3 Unity

Unity, dengan logonya terdapat pada gambar 2.2, adalah sebuah *cross-platform development engine* yang awalnya diciptakan untuk pembuatan *game* namun sekarang digunakan untuk berbagai hal seperti arsitektur, seni, manajemen informasi, edukasi, hiburan, dan sebagainya. *Unity* mengurangi kompleksitas dari proses pembuatan *game* dan pengalaman interaktif lainnya, mengurus hal-hal tersebut dibelakang layar sehingga pengguna bisa langsung mendesain dan menciptakan *game* yang mereka inginkan. Hal-hal yang kompleks yang langsung diurus oleh *Unity* sendiri adalah seperti *graphic rendering*, *world physics*, dan *compiling*. *Game* yang diciptakan dalam *Unity* dibuat melalui *Unity Editor* dan C# sebagai Bahasa pemrogramannya (Alves, 2020).



Gambar 2.2 Logo *Unity*

Beberapa fitur *Unity* yang dipertimbangkan untuk digunakan dalam projek ini adalah sebagai berikut.

a. *Physics*

Dalam Unity, ada komponen yang sudah disediakan sejak awal untuk mengelola simulasi fisika dalam permainan, termasuk deteksi gerakan objek dan tabrakan antara objek. Dengan adanya fitur bawaan ini, pengembang dapat lebih fokus pada pembuatan sistem untuk meningkatkan randomisasi dalam permainan mereka.

b. *Unity Asset Store*

Unity Asset Store adalah *platform* yang disediakan oleh *Unity Technologies* yang menjadi tempat para pengembang *game* untuk melakukan transaksi terhadap aset-aset yang bisa digunakan dalam pengembangan suatu *game*, baik itu model 2D maupun 3D, tekstur,

hingga audio maupun *visual effects* (VFX). *Platform* ini menghemat waktu dan tenaga yang diperlukan untuk membangun *game* sehingga pengembangan randomisasi *Linear Congruential Generator* dapat difokuskan dalam penelitian.

c. *ScriptableObject*

ScriptableObject adalah *class* yang sudah disediakan oleh *Unity Engine*. *ScriptableObject* memiliki karakteristik dari *class* C# pada umumnya seperti membuat objek baru dan menyimpan data. Penggunaan *ScriptableObject* dalam penelitian ini mempermudah penerapan sistem *upgrade* dan randomisasi *Linear Congruential Generator*.

2.4 Rudantara

"Rudantara" adalah nama dari *game* yang dikembangkan dalam penelitian ini. *Game* ini adalah sebuah *game 3D action roguelike* yang berfokus pada petualangan sepasang gadis di dalam hutan yang berubah-ubah. Dalam *game* ini, pemain akan memainkan karakter utama ditemani oleh pendamping kepercayaannya sebagai pengelana hutan. Di dalam hutan tersebut, pemain dan pendampingnya akan berhadapan dengan musuh-musuh yang mencoba menghalangi mereka yang diciptakan oleh hutan itu sendiri.

Dalam *game* ini, pemain mengendalikan karakter dengan menggunakan *keyboard* dan *mouse*. Tombol untuk menggerakkan pemain W untuk berjalan maju, S untuk berjalan mundur, A untuk berjalan ke kiri, dan D untuk berjalan ke kanan. Pemain dapat menyerang dengan menekan tombol kiri pada *mouse*. Pemain juga dapat mengaktifkan *skill* dengan menekan tombol 1 atau 2 pada *keyboard*. Pemain memiliki *Health Point* (HP) yang apabila habis maka pemain akan kalah dan *Mana* dimana pemain akan menghabiskan sejumlah *mana* untuk menggunakan *skill*. Jika *mana* tidak cukup, maka pemain tidak dapat menggunakan *skill*. Pendamping pemain akan bergerak sendiri sesuai dengan AI yang mengendalikannya.

Musuh dalam permainan ini akan muncul secara acak pada peta. Musuh hanya akan berkeliling saja menunggu pemain memasuki jarak pandangnya. Ketika pemain memasukinya, maka musuh akan mengejar pemain untuk menyerang pemain. Pemain dapat mengalahkan musuh untuk mendapatkan poin pengalaman yang dapat menaikkan *level* pemain yang akan memperkuat pemain dan apabila pemain beruntung, pemain juga

berkemungkinan mendapatkan sebuah kotak harta karun yang akan memberikan pemain *upgrade* yang akan semakin memperkuat pemain.

BAB 3

ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Sistem

Pada penelitian ini, sebuah sistem randomisasi untuk sistem *upgrade* dan hal-hal yang berhubungan dengan *upgrade* diciptakan untuk *game roguelike*. Sebelum sistem dibangun, diperlukan analisa terhadap masalah dan juga kebutuhan. Analisis masalah diperlukan untuk mengidentifikasi penyebab dan dampak dari suatu masalah dan analisis terhadap kebutuhan diperlukan untuk mengidentifikasi proses-proses yang diperlukan untuk merancang suatu sistem.

3.1.1 Analisis Masalah

Pada penelitian ini, masalah terdapat pada pembuatan *game roguelike* dengan randomisasi yang memadai untuk memberikan tantangan kepada pemain. Randomisasi untuk membuat konten dalam *game* adalah salah satu lamgkah yang paling membingungkan dalam mendesain sebuah *game*. Randomisasi digunakan untuk menentukan hasil dari suatu kejadian agar hasil dari kejadian tersebut tidak sama setiap saat. Penggunaan randomisasi dalam *game* dapat menentukan menang atau kalahnya pemain. Oleh karenanya, menciptakan sistem randomisasi yang adil tapi tetap menantang pemain diperlukan untuk menciptakan *game roguelike* yang menyenangkan untuk dimainkan.

3.1.2 Analisis Kebutuhan

Berdasarkan masalah di atas, kebutuhan yang perlu dipenuhi dalam pengembangan *game roguelike* dengan *Linear Congruential Generator* sebagai dasar sistem *upgrade* dalam *game* tersebut adalah sebuah sistem randomisasi yang dapat memberikan pemain kemampuan yang dapat membuat pemain dapat menyelesaikan tantangan dengan lebih baik, namun tidak terlalu signifikan sehingga pemain tidak merasakan adanya tantangan yang berasal dari *game* yang dimainkan sehingga memungkinkan pemain merasa bosan. Pemain juga harus diberikan bantuan berupa peningkatan kemungkinan diberikannya *upgrade* kepada pemain

3.2 Perancangan Sistem

Pada tahapan ini, gambaran rencana cara kerja sistem *upgrade* yang menggunakan *Linear Congruential Generator* sebagai dasar randomisasinya untuk *game roguelike* akan dijelaskan. Sistem bekerja melalui berbagai tahap dan menggunakan beberapa komponen secara bersamaan untuk membuat sistem *upgrade* yang baik.

3.2.1 Pendefinisian *stats* dan *upgrade*

Sebelum berbagai *upgrade* yang dapat dipilih pemain dibuat, diperlukan *stats* yang akan menjadi parameter yang dilekatkan kepada karakter yang digunakan pemain. *Stats* yang ada dalam pemain meliputi apa saja *stats* tersebut dan bagaimana *stats* yang dimiliki dapat mempengaruhi maupun dipengaruhi oleh sumber lain.

Stats yang akan dipengaruhi oleh *upgrade* dalam *game* yang akan dibuat adalah *Attack* (kekuatan serangan), *Defense* (kekuatan bertahan), *Speed* (kecepatan berlari), *Accuracy* (ketepatan serangan), *Health Regen* (kecepatan penambahan darah pemain per detik), dan *Mana Regen* (kecepatan penambahan *mana* pemain per detik). *Upgrade* yang didapat pemain akan meningkatkan atau terkadang menurunkan salah satu atau lebih *stats* dari daftar yang ada di atas.

Setiap *upgrade* yang didapatkan pemain akan memiliki sebuah nama, gambar, deskripsi, tingkat kelangkaan (*rarity*), dan *stats* yang akan dipengaruhi. Untuk *rarity* akan ada *common* dan *rare*, dimana tingkatan *rare* akan lebih jarang muncul namun memberikan penambahan *stats* yang lebih besar. Untuk *stats* yang akan dipengaruhi, nilai penambahan yang diberikan kepada pemain akan dirandomisasikan maka akan ada batas atas dan batas bawah agar *upgrade* yang didapatkan pemain tidak memiliki nilai yang begitu besar maupun begitu kecil. Penambahan *stats* juga dapat bertipe sekadar penambahan atau perkalian persen.

3.2.2 Pembuatan sistem randomisasi

Pembuatan sistem randomisasi menggunakan algoritma *Linear Congruential Generator* untuk menentukan *rarity* dari *upgrade*, *upgrade* yang terpilih, dan juga *stats* dari *upgrade* yang terpilih serta *random drop* yang menjatuhkan *upgrade* tambahan untuk didapatkan pemain. Berikut adalah penjelasan mengenai alur pembuatan sistem randomisasi:

a) Variabel Penting:

- *seed:* Nilai awal perhitungan algortima *Linear Congruential Generator*.
- a: Nilai dari pengali, yang menggunakan jumlah dari *upgrade* dengan *rarity common*.
- c: Nilai dari penambah. yang menggunakan jumlah dari *upgrade* dengan *rarity rare.*
- m: Nilai modulus, yang menggunakan jumlah dari seluruh *upgrade.*

b) Menentukan *rarity* :

Rarity ditentukan dengan melakukan *Linear Congruential Generator*, yang kemudian hasilnya dibandingkan dengan jumlah dari *upgrade* dengan *rarity common* atau *rare*. *Seed* akan diinisialisasikan terlebih dahulu menggunakan `System.DateTime.Now.Ticks` sebagai nilai awal yang merepresentasikan X0. Dikarenakan menggunakan jumlah dari seluruh *upgrade* sebagai m, maka hasil dari algoritma *Linear Congruential Generator* akan dibandingkan dengan jumlah dari banyak *upgrade* dengan *rarity common*. Apabila nilai hasil perhitungan lebih besar, maka *upgrade* akan dipilih dari *rarity rare,* dan jika tidak maka akan dipilih dari *common*.

c) Menentukan *upgrade*

Setelah *rarity* dari *upgrade* didapatkan, maka *upgrade* akan dipilih dengan menggunakan *Linear Congruential Generator*. Nilai dari a, c, dan m kembali menggunakan nilai yang sama, namun nilai dari *seed* akan bergantung dari *seed* yang digunakan untuk *rarity* untuk pertama kalinya. Setelah algoritma LCG dilakukan, maka hasil dari algoritma tersebut, *upgradeSeed*, kemudian di modulus dengan menggunakan jumlah dari semua *upgrade* yang berada dalam tingkatan *rarity* yang terpilih. Hasil akhir inilah yang akan menentukan *upgrade* yang dipilih sesuai dengan urutannya dalam *database* yang berisi semua *upgrade* dengan *rarity*.

d) Menentukan *stats* dari *upgrade*

Namun sebelum *upgrade* dimasukkan ke dalam daftar *upgrade* yang bisa dipilih pemain, *stats* yang ada di dalam *upgrade* terlebih dahulu dirandomisasikan menggunakan *Linear Congruential Generator*. Nilai dari *seed* akan menggunakan

nilai *seed* dari pemilihan *rarity* untuk nilai pertamanya dan seterusnya akan menggunakan hasil dari algoritma LCG untuk pemilihan *stats* yang disebut *statSeed*. Untuk nilai a dan c akan menggunakan nilai yang sama dengan pemilihan *rarity* maupun *upgrade*.

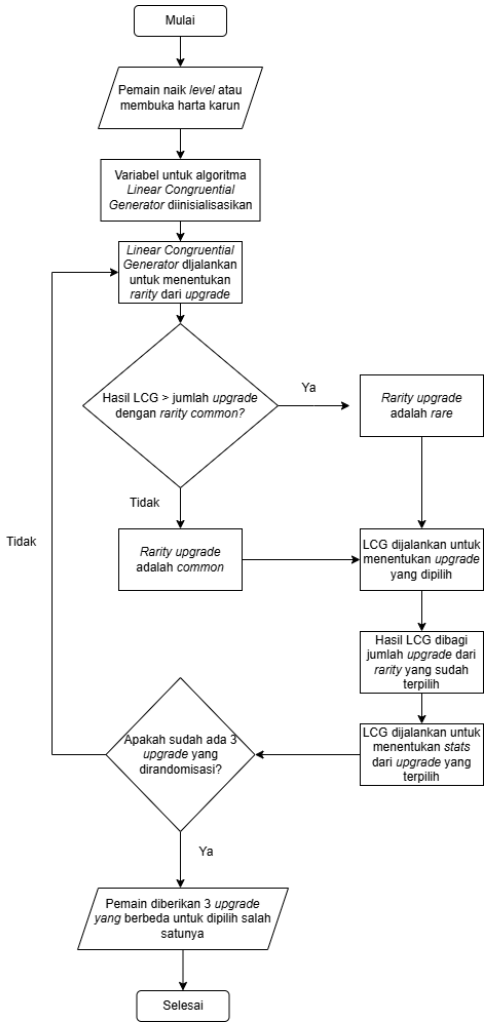
Untuk nilai m, perlu diketahui dalam pemilihan *stats* akan ada batas atas dan batas bawah untuk tiap *stats* dari *upgrade*. Hal ini dilakukan untuk mencegah nilai yang terlalu tinggi maupun terlalu rendah dari suatu *upgrade* yang menyebabkan ketidakseimbangan dalam *game*. Nilai batas atas dari suatu *stat* + 1 akan digunakan sebagai m untuk algoritma LCG penentuan *stats*. Sementara itu, batas bawah akan digunakan apabila hasil dari LCG terlalu rendah dan nilai dari *stats* langsung menggunakan batas bawah dari suatu *stats*. Setelah semua nilai dari *stats* telah dirandomisasi, barulah *upgrade* dimasukkan ke dalam daftar *upgrade* yang dapat dipilih pemain. Kemudian algoritma dilakukan dari awal lagi sebanyak tiga kali, sebab tiga *upgrade* akan diberikan kepada pemain agar pemain bisa memilih salah satu dari *upgrade* yang diberikan untuk memperkuat pemain.

3.2.3 Pembuatan sistem untuk memberikan *upgrade* kepada pemain

Upgrade dapat didapatkan pemain melalui dua cara: menaikkan *level* dan membuka harta karun yang dijatuhkan oleh musuh. Sistem untuk menaikkan *level* menggunakan suatu perhitungan dimana pemain akan mendapatkan *experience* selama permainan dijalankan dalam jumlah kecil, dan jumlah tersebut akan meningkat apabila pemain sedang bertarung. Mengalahkan musuh juga akan memberikan *experience* kepada pemain. Setelah *experience* pemain melebihi batas yang ditentukan, maka tingkatan *level* dari pemain akan meningkat, mendapatkan *upgrade* pilihannya dan juga sedikit tambahan ke beberapa *stats* lain.

Sementara itu, sistem bagi musuh untuk menjatuhkan harta karun dilakukan dengan menggunakan randomisasi *Linear Congruential Generator*. Dengan menggunakan nilai a, m, c yang sama dan nilai *seed* juga akan menggunakan System.DateTime.Now.Ticks sebagai nilai awalnya. Setelah nilai dari LCG didapatkan, nilai tersebut akan dibandingkan dengan jumlah seluruh *upgrade* dibagi 4. Apabila hasil

perhitungan lebih kecil, maka harta akan dijatuhkan dan dapat dibuka oleh pemain dan apabila tidak maka tidak akan ada yang dijatuhkan oleh musuh. Untuk setiap musuh yang tidak menjatuhkan harta, maka akan ada nilai yang menyimpan jumlah musuh yang tidak menjatuhkan harta tersebut yang disebut *higherChance*, Nilai ini kemudian akan digunakan untuk mengurangi nilai dari perhitungan LCG ketika dibandingkan guna meningkatkan kemungkinan pemain untuk mendapatkan harta. Jika pemain sudah mendapatkan harta, nilai *higherChance* akan dikembalikan menjadi 0. Lebih jelasnya dapat dilihat pada gambar 3.1.



Gambar 3.1 Flowchart cara kerja sistem

3.3 Desain User Interface

Karena penelitian ini berfokus pada pemilihan *upgrade* yang akan dipilih pemain untuk memperkuat pemain, maka diperlukan suatu tampilan yang akan memberi tahu pemain untuk memilih salah satu dari *upgrade* yang ditemukan. Dengan memberikan sebuah *interface* yang sederhana maka pemain dapat dengan mudah memilih *upgrade* yang mereka inginkan dan langsung kembali ke *game*. Pada saat *interface upgrade* muncul, *game* akan berhenti sejenak untuk memberikan pemain peluang untuk memikirkan *upgrade* yang akan dipilih.

Pemain juga dapat melihat *upgrade* yang telah diambil dalam menu *pause* dengan memilih pilihan *upgrade* maupun dengan menekan tombol *'Your Upgrades'* ketika pemilihan *upgrade* guna memberikan pemain informasi yang ia perlukan untuk memilih *upgrade* yang diinginkan. Informasi mengenai *upgrade* yang telah dipilih akan muncul ketika ikon dari *upgrade* yang informasinya ingin dilihat di-*hover* menggunakan *mouse*.

3.4 Rencana Pengujian

Pengujian akan dilakukan menggunakan kuesioner. Pengujian akan dibagi ke dalam dua tahap, yakni:

3.4.1. Pengujian Tingkat Kerancuan Hasil Randomisasi

Pengujian dilakukan untuk menguji apakah hasil dari randomisasi menggunakan algoritma *Linear Congruential Generator* sudah benar-benar rancu dengan menganalisa semua *rarity, upgrade,* dan *stats* yang muncul serta kecenderungan jatuhnya harta karun dari musuh dalam satu kali *game* dijalankan untuk mendapatkan pengaruh dari konstanta a. c. dan m yang dipilih.

3.4.2 Pengujian Hasil Algoritma Oleh *Playtester*

Pengujian akan dilakukan melalui kuisioner *online* dengan target orang-orang yang sudah berpengalaman dalam *video game* secara umu, dan *game roguelike* secara khusus dan telah memainkan *game* yang dibuat peneliti. Pertanyaan di dalam kuisioner akan mencakup penilaian mengenai apakah *rarity* dari *upgrade* yang didapat sudah cukup tersebar dengan baik dimana tingkat *rare* tidak cukup sering muncul, apakah *upgrade* yang didapat pemain cukup beragam, dan apakah *stats* dari *upgrade* yang didapat pemain

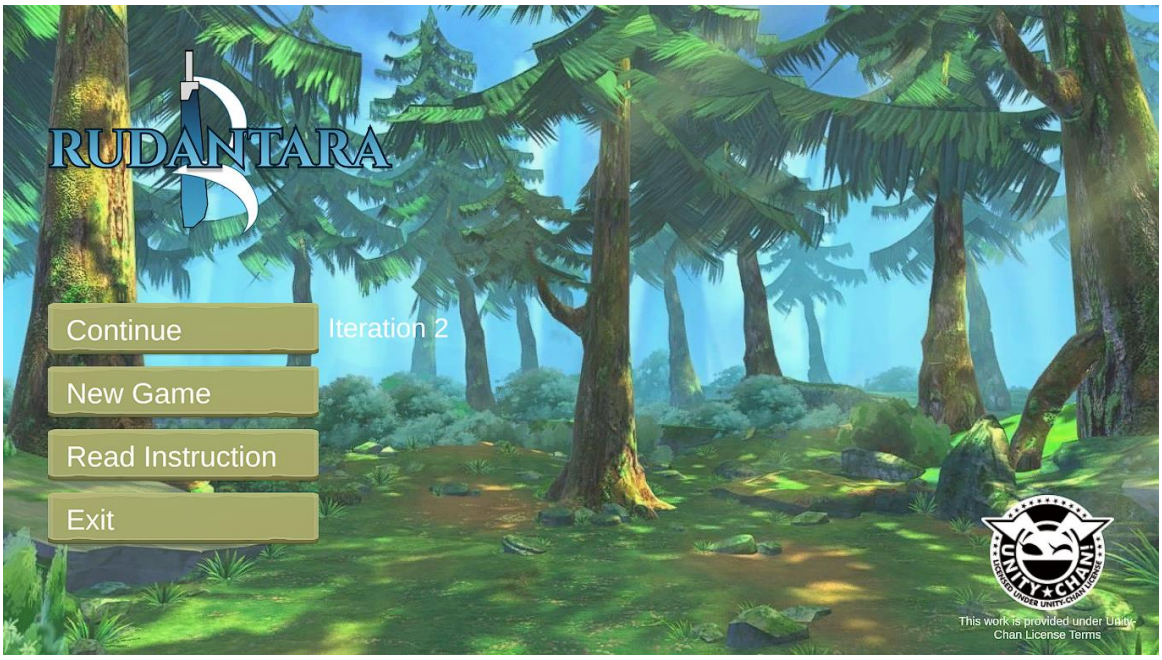
juga berbeda dari *upgrade* serupa yang didapatkan sebelumnya serta apakah harta jatuh dari musuh dalam jangka yang cukup tak tertebak.

BAB 4

IMPLEMENTASI DAN PENGUJIAN SISTEM

4.1. Video Game "Rudantara"

Rudantara adalah judul dari *game* yang dikembangkan oleh peneliti dengan menggunakan *Linear Congruential Generator* untuk sistem *upgrade* yang ada di dalam *game* tersebut. *Game* ini memiliki genre *Roguelike Action RPG*.

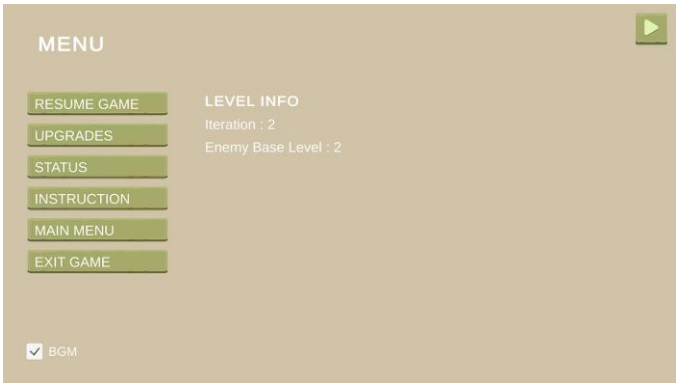


Gambar 4.1 Tampilan *Main Menu Game*

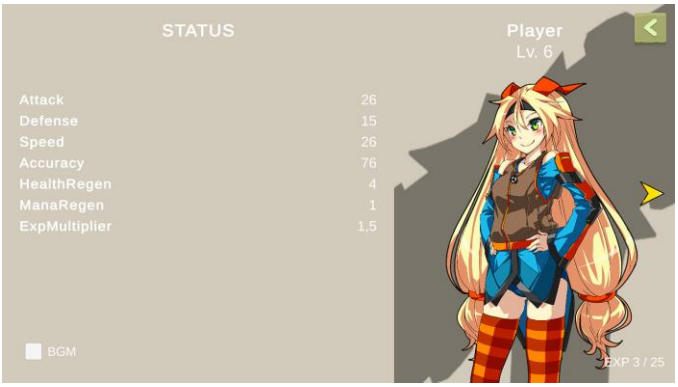
Setiap kali permainan dimulai, tatanan peta akan dibuat menggunakan *procedural generation* untuk menciptakan level selalu berbeda-beda setiap iterasi sehingga pemain tidak merasa bosan. Dalam setiap iterasi, tujuan pemain adalah untuk mengalahkan *boss* yang akan muncul setelah 60 detik sejak pemain muncul untuk menuju iterasi selanjutnya. Dalam setiap iterasi pemain akan menemui musuh yang akan menghadang dan mencoba mengalahkan pemain. Meski begitu, pemain akan ditemani oleh pendamping yang akan

membantu pemain. Pemain akan selalu berlanjut hingga ke berbagai iterasi sampai *Health Point (HP)* pemain mencapai 0.

Gambar 4.2 Tampilan *gameplay* dari *game*



Gambar 4.3 *UI* ketika pemain melakukan *pause* terhadap *game*



Gambar 4.4 *UI* status pemain

Berikut adalah *stats* pemain ketika memulai game:

- *Health* = 150
- *Mana* = 50
- *Attack* = 20
- *Defense* = 12
- *Speed* = 25
- *Accuracy* = 75
- *Health Regen* = 5
- *Mana Regen = 1*

Sementara itu, untuk *stats* musuh yang ada dalam *game* adalah sebagai berikut:

- *Health* = 50
- *Attack* = 20
- *Defense* = 10
- *Speed* = 10
- *Accuracy* = 75

Untuk pengujian terhadap *Linear Congruential Generator* dalam *game* ini, pemain dapat mengalahkan musuh untuk memulai randomisasi. Randomisasi untuk penentuan *random drop* dan *upgrade* yang dapat dipilih pemain akan bekerja setiap kali pemain mengalahkan musuh.

4.2 Pengujian Program

4.2.1 Analisis Randomisasi

Randomisasi yang ada dalam berbagai aspek dalam *game* ini menggunakan *Linear Congruential Generator*. Sebelum randomisasi dimulai, beberapa nilai konstanta, yakni a, c, dan m, harus ditentukan terlebih dahulu untuk melakukan randomisasi. Penentuan nilai a, c, dan m adalah sebagai berikut:

a = Nilai jumlah dari seluruh *upgrade* dengan tingkat *rarity common*

c = Nilai jumlah dari seluruh *upgrade* dengan tingkat *rarity rare*

m = Nilai jumlah dari seluruh *upgrade* yang ada

Berikut adalah semua *upgrade* yang ada dalam *game* ini.

Tabel 4.1 Daftar *upgrade* yang ada dalam *game*

Nama *Upgrade*	*Rarity*	*Stats* dengan batas atas, batas bawah, dan tipe penambahan
Ruda Blade	*Common*	*Attack Add* (1 – 5)
Fog Blade	*Common*	*Attack Add* (1 – 8), *Defense Add*(-4 - -2)
Fog Boots	*Common*	*Speed Add* (1 – 8), *Defense Add*(-4 - -2)
Rhodonite Shield	*Common*	*Defense Percent Add* (1% - 2%)
Rhodonite Blade	*Common*	*Attack Percent Add* (1% - 2%)
Ruda Boots	*Common*	*Speed Add* (1 – 5)
Ruda Essence	*Common*	*Health Regen Add (1 – 5)*
Ruda Mana	*Common*	*Mana Regen Add* (1 – 5)
Ruda Scope	*Common*	*Accuracy Add* (1 – 5)
Ruda Shield	*Common*	*Defense Add* (1 – 5)
Soul Essence	*Common*	*Health Regen Percent Add* (1% - 2%)
Soul Mana	*Common*	*Mana Regen Percent Add* (1% - 2%)
Winged Boots	*Common*	*Speed Percent Add* (1% - 2%)
Amethyst Blade	*Rare*	*Attack Add* (6 – 10)
Amethyst Boots	*Rare*	*Defense Add* (6 – 10)
Amethyst Essence	*Rare*	*Health Regen Add* (6 – 10)
Amethyst Mana	*Rare*	*Mana Regen Add* (6 – 10)
Amethyst Shield	*Rare*	*Defense Add* (6 – 10)

Berdasarkan tabel di atas, dapat disimpulkan jika nilai dari a adalah 13, c bernilai 5, dan m bernilai 18. Sementara itu, untuk nilai dari X0 dimana belum dilakukan perulangan akan ditentukan menggunakan System.DateTime.Now.Ticks. Berikut adalah uji coba terhadap beberapa aspek yang akan dirandomisasikan dengan menggunakan *Linear Congruential Generator*:

1. *Random Drop*

Setelah melakukan randomisasi menggunakan *Linear Congruential Generator* diatas, maka ditentukan terlebih dahulu apakah nilai berada dalam jangkauan nilai yang dibutuhkan untuk harta karun dijatuhkan oleh musuh. Rumus untuk menentukan musuh menjatuhkan harta karun atau tidak adalah:

$$drop = dropChance > (lcgResult - moreChance)$$

Drop = Nilai *boolean* yang menentukan apakah harta karun jatuh atau tidak. Jika *true*, maka harta karun akan jatuh.

dropChance = Batas atas yang harus dilewati untuk menentukan apakah harta karun jatuh atau tidak. Nilai dari *dropChance* ditentukan dengan menggunakan jumlah dari seluruh *upgrade* yang ada dibagi 4, maka 18 / 4 = 4,5.

lcgResult = Nilai hasil *LCG* yang dihitung terlebih dahulu.

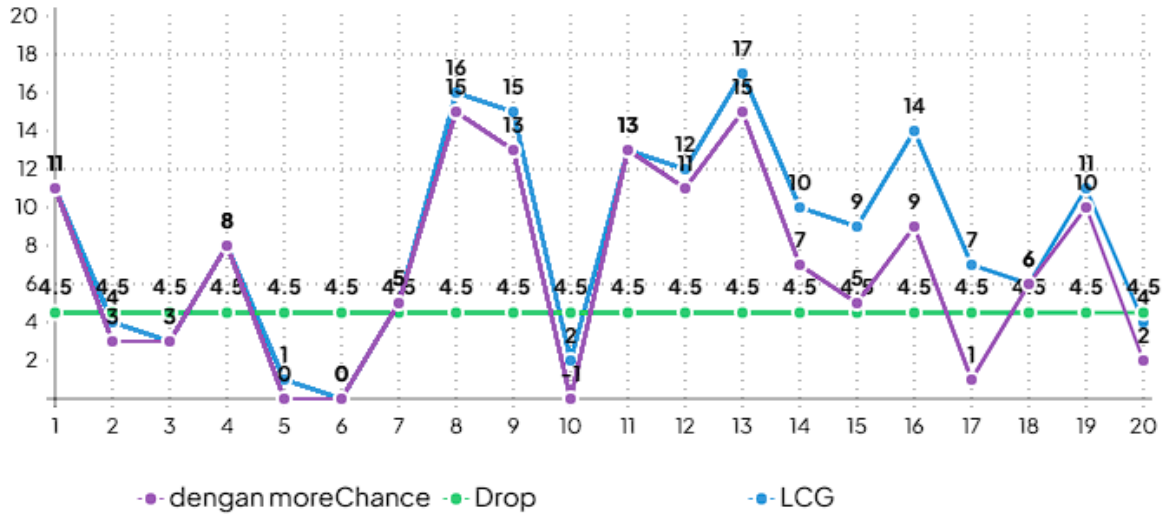
moreChance = Angka pengubah yang selalu meningkat apabila pemain tidak mendapat harta karun dan kembali ke 0 apabila pemain mendapat harta.

Untuk uji coba, digunakan nilai sembarang dari System.DateTime.Now.Ticks. yang menghasilkan 395413116. Berikut hasil uji coba untuk mendapatkan nilai *random drop*.

Tabel 4.2 Hasil uji coba *random drop* dengan X0 = 395413116

Iterasi	Perhitungan *LCG*	Nilai *moreChance*	Harta jatuh atau tidak
1	$(13 * 395413116 + 5)\ mod\ 18 = 11$	0	4.5 > (11 – 0) =Tidak
2	$(13 * 11 + 5)\ mod\ 18 = 4$	1	4.5 > (4 – 1) = Jatuh
3	$(13 * 4 + 5)\ mod\ 18 = 3$	0	4.5 > (3 – 0) = Jatuh
4	$(13 * 3 + 5)\ mod\ 18 = 8$	0	4.5 > (8 – 0) = Tidak
5	$(13 * 8 + 5)\ mod\ 18 = 1$	1	4.5 > (1 – 1) = Jatuh
6	$(13 * 1 + 5)\ mod\ 18 = 0$	0	4.5 > (0 – 0) = Jatuh
7	$(13 * 0 + 5)\ mod\ 18 = 5$	0	4.5 > (5 – 0) = Tidak

Iterasi	Perhitungan *LCG*	Nilai *moreChance*	Harta jatuh atau tidak
8	$(13 * 5 + 5)\ mod\ 18 = 16$	1	4.5 > (16 – 1) = Tidak
9	$(13 * 16 + 5)\ mod\ 18 = 15$	2	4.5 > (15 – 2) = Tidak
10	$(13 * 15 + 5)\ mod\ 18 = 2$	3	4.5 > (2 – 3) = Jatuh
11	$(13 * 2 + 5)\ mod\ 18 = 13$	0	4.5 > (13 – 0) = Tidak
12	$(13 * 13 + 5)\ mod\ 18 = 12$	1	4.5 > (12 – 1) = Tidak
13	$(13 * 12 + 5)\ mod\ 18 = 17$	2	4.5 > (17 – 2) = Tidak
14	$(13 * 17 + 5)\ mod\ 18 = 10$	3	4.5 > (10 – 3) = Tidak
15	$(13 * 10 + 5)\ mod\ 18 = 9$	4	4.5 > (9 – 4) = Tidak
16	$(13 * 9 + 5)\ mod\ 18 = 14$	5	4.5 > (14 – 5) = Tidak
17	$(13 * 14 + 5)\ mod\ 18 = 7$	6	4.5 > (7 – 6) = Jatuh
18	$(13 * 7 + 5)\ mod\ 18 = 6$	0	4.5 > (6 – 0) = Tidak
19	$(13 * 6 + 5)\ mod\ 18 = 11$	1	4.5 > (11 – 1) = Tidak
20	$(13 * 11 + 5)\ mod\ 18 = 4$	2	4.5 > (4 - 2) = Jatuh



Gambar 4.5 Grafik hasil randomisasi *random drop*

Dari hasil randomisasi dan visualisasi di atas, didapatkan dari 20 percobaan, harta karun akan jatuh sebanyak 7 kali. Hal ini menunjukkan jika kemungkinan untuk harta karun adalah sebesar:

$$\frac{7}{20} * 100\% = 35\%$$

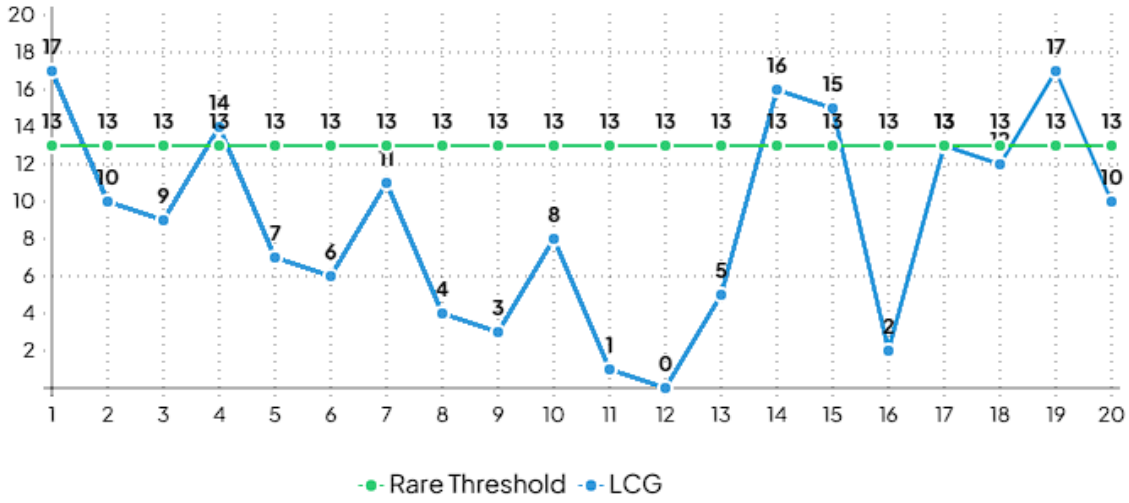
2. *Rarity* dari *Upgrade*

Setelah melakukan algortima randomisasi *Linear Congruential Generator,* maka untuk menentukan nilai *rarity* dari suatu *upgrade* dilihat dari hasilnya kemudian dibandingkan dengan jumlah dari banyak *upgrade* dengan tingkatan *rarity common*. Apabila hasil *LCG* lebih rendah dari jumlah *rarity common*, maka *upgrade* yang diberikan adalah *common*. Jika tidak, maka *rare*.

Untuk uji coba, digunakan nilai sembarang dari System.DateTime.Now.Ticks. yang menghasilkan 449064552. Berikut hasil uji coba untuk mendapatkan nilai *random drop*.

Tabel 4.3 Hasil uij coba penentuan *rarity* dengan X0 = 449064552

Iterasi	Perhitungan *LCG*	*Rarity*
1	$(13 * 449064552 + 5)\ mod\ 18 = 17$	17 > 13 = *Rare*
2	$(13 * 17 + 5)\ mod\ 18 = 10$	10 < 13 = *Common*
3	$(13 * 10 + 5)\ mod\ 18 = 9$	9 < 13 = *Common*
4	$(13 * 9 + 5)\ mod\ 18 = 14$	14 > 13 = *Rare*
5	$(13 * 14 + 5)\ mod\ 18 = 7$	7 < 13 = *Common*
6	$(13 * 7 + 5)\ mod\ 18 = 6$	6 < 13 = *Common*
7	$(13 * 6 + 5)\ mod\ 18 = 11$	11 < 13 = *Common*
8	$(13 * 11 + 5)\ mod\ 18 = 4$	4 < 13 = *Common*
9	$(13 * 4 + 5)\ mod\ 18 = 3$	3 < 13 = *Common*
10	$(13 * 3 + 5)\ mod\ 18 = 8$	8 < 13 = *Common*
11	$(13 * 8 + 5)\ mod\ 18 = 1$	1 < 13 = *Common*
12	$(13 * 1 + 5)\ mod\ 18 = 0$	0 < 13 = *Common*
13	$(13 * 0 + 5)\ mod\ 18 = 5$	5 < 13 = *Common*
14	$(13 * 5 + 5)\ mod\ 18 = 16$	16 > 13 = *Rare*
15	$(13 * 16 + 5)\ mod\ 18 = 15$	15 > 13 = *Rare*
16	$(13 * 15 + 5)\ mod\ 18 = 2$	2 < 13 = *Common*
17	$(13 * 2 + 5)\ mod\ 18 = 13$	13 = 13 = *Rare*
18	$(13 * 13 + 5)\ mod\ 18 = 12$	12 < 13 = *Common*
19	$(13 * 12 + 5)\ mod\ 18 = 17$	17 > 13 = *Rare*
20	$(13 * 17 + 5)\ mod\ 18 = 10$	10 < 13 = *Common*



Gambar 4.6 Grafik hasil randomisasi pemilihan *rarity* dari *upgrade*

Dari hasil randomisasi dan visualisasi di atas, didapatkan dari 20 percobaan, harta karun akan jatuh sebanyak 6 kali. Hal ini menunjukkan jika kemungkinan untuk mendapatkan *upgrade* dengan tingkatan *rare* adalah sebesar:

$$\frac{6}{20} * 100\% = 30\%$$

3. *Upgrade* yang dipilih

Setelah *rarity* dari *upgrade* terpilih, maka dilakukan randomisasi menggunakan *LCG* untuk memilih *upgrade*. Setelah hasil dari LCG didapatkan, hasil tersebut kemudian dibagi menggunakan jumlah *upgrade* dari *rarity* tersebut untuk dicari hasil modulusnya yang kemudian digunakan untuk memilih *upgrade*.

Setiap *upgrade* yang ada di dalam *game* dan urutannya untuk *common* dan *rare* adalah sebagai berikut:

Tabel 4.4 Urutan *upgrade* dengan tingkatan *common*

Upgrade	ID
Ruda Blade	0
Fog Blade	1
Fog Boots	2

Upgrade	ID
Rhodonite Shield	3
Rhodonite Blade	4
Ruda Boots	5
Ruda Essence	6
Ruda Mana	7
Ruda Scope	8
Ruda Shield	9
Soul Essence	10
Soul Mana	11
Winged Boots	12

Tabel 4.5 Urutan *upgrade* untuk tingkatan *rare*

Upgrade	ID
Amethyst Blade	0
Amethyst Boots	1
Amethyst Essence	2
Amethyst Mana	3
Amethyst Shield	4

Dengan menggunakan nilai X0 = 17 untuk uji coba, berikut adalah hasil uji coba untuk mendapatkan *upgrade*.

Tabel 4.6 Hasil uij coba penentuan *upgrade* dengan X0 = 17

Iterasi	Perhitungan *LCG*	*Rarity*	*Upgrade* yang terpilih
1	$(13 * 17 + 5)\ mod\ 18 = 10$	*Rare*	$10\ mod\ 5 = 0\ (Amethyst\ Blade)$
2	$(13 * 0 + 5)\ mod\ 18 = 5$	*Common*	$5\ mod\ 13 = 5\ (Ruda\ Boots)$
3	$(13 * 5 + 5)\ mod\ 18 = 16$	*Common*	$16\ mod\ 13 = 3\ (Rhodonite\ Shield)$
4	$(13 * 3 + 5)\ mod\ 18 = 8$	*Rare*	$8\ mod\ 5 = 3\ (Amethyst\ Mana)$
5	$(13 * 3 + 5)\ mod\ 18 = 6$	*Common*	$6\ mod\ 13 = 6\ (Ruda\ Essence)$

Iterasi	Perhitungan *LCG*	*Rarity*	*Upgrade* yang terpilih
6	$(13 * 6 + 5)\ mod\ 18 = 11$	*Common*	$11\ mod\ 13 = 11$ (*Soul Mana)*
7	$(13 * 11 + 5)\ mod\ 18 = 4$	*Common*	$4\ mod\ 13 = 4$ (*Rhodonite Blade)*
8	$(13 * 4 + 5)\ mod\ 18 = 3$	*Common*	$3\ mod\ 13 = 3$ *(Rhodonite Shield)*
9	$(13 * 3 + 5)\ mod\ 18 = 8$	*Common*	$8\ mod\ 13 = 8$ (*Ruda Scope)*
10	$(13 * 8 + 5)\ mod\ 18 = 1$	*Common*	$1\ mod\ 13 = 1$ (*Fog Blade)*
11	$(13 * 1 + 5)\ mod\ 18 = 0$	*Common*	$0\ mod\ 13 = 0$ (*Ruda Blade)*
12	$(13 * 0 + 5)\ mod\ 18 = 5$	*Common*	$5\ mod\ 13 = 5$ (*Ruda Boots)*
13	$(13 * 5 + 5)\ mod\ 18 = 16$	*Common*	$16\ mod\ 13 = 3$ (*Rhodonite Shield)*
14	$(13 * 3 + 5)\ mod\ 18 = 8$	*Rare*	$8\ mod\ 5 = 3$ (*Amethyst Mana)*
15	$(13 * 3 + 5)\ mod\ 18 = 8$	*Rare*	$8\ mod\ 5 = 3$ (*Amethyst Mana)*
16	$(13 * 3 + 5)\ mod\ 18 = 8$	*Common*	$8\ mod\ 13 = 8$ (*Ruda Scope)*
17	$(13 * 8 + 5)\ mod\ 18 = 1$	*Rare*	$1\ mod\ 5 = 1$ (*Amethyst Boots)*
18	$(13 * 1 + 5)\ mod\ 18 = 0$	*Common*	$0\ mod\ 13 = 0$ (*Ruda Blade)*
19	$(13 * 0 + 5)\ mod\ 18 = 5$	*Rare*	$5\ mod\ 5 = 0$ (*Amethyst Blade)*
20	$(13 * 0 + 5)\ mod\ 18 = 5$	*Common*	$5\ mod\ 13 = 5$ (*Ruda Boots)*

Berikut adalah jumlah *upgrade* yang muncul dalam 20 percobaan, yakni:

Tabel 4.7 Frekuensi kemunculan setiap *upgrade*

Upgrade	Frekuensi Kemunculan
Amethyst Blade	2
Ruda Boots	3
Rhodonite Shield	3
Amethyst Mana	3
Ruda Essence	1
Soul Mana	1
Rhodonite Blade	1
Ruda Scope	2
Fog Blade	1
Ruda Blade	2
Amethyst Boots	1

Dari hasil di atas dapat diketahui jika dalam 20 percobaan, tidak semua *upgrade* yang ada terpilih, terutama yang memiliki tingkatan *rarity rare*. Beberapa *upgrade* terpilih sebanyak 3 kali termasuk *Amethyst Mana* yang memiliki tingkatan *rare*. Hal ini menunjukkan randomisasi untuk pemilihan *upgrade* sudah cukup acak, namun masih belum seacak mungkin untuk memilih semua *upgrade* dalam satu iterasi yang merupakan hal yang cukup baik untuk memberikan pemain motivasi lebih untuk memainkan kembali *game* untuk melihat semua *upgrade* yang ada.

4. Penentuan *stats* untuk *upgrade* yang dipilih

Setelah *upgrade* terpilih, akan dilakukan randomisasi LCG untuk setiap *stats* yang ada dalam *upgrade* tersebut. Berbeda dengan pemilih *rarity* dan *upgrade* yang menggunakan jumlah seluruh *upgrade* sebagai m, nilai m pada pemilihan *stats* adalah nilai batas atas + 1.

Dengan menggunakan nilai X0 = 17 untuk uji coba, berikut adalah hasil uji coba untuk mendapatkan *stats* dari *upgrade*.

Tabel 4.8 Hasil uij coba penentuan *stats* dari *upgrade* dengan X0 = 17

Iterasi	*Upgrade*	*Stats* dan batas atas	Perhitungan *LCG*
1	*Amethyst Blade*	*Attack* (10)	$(13 * 17 + 5)\ mod\ 11 = 6$
2	*Ruda Boots*	*Speed* (5)	$(13 * 6 + 5)\ mod\ 6 = 5$
3	*Rhodonite Shield*	*Defense* (2)	$(13 * 5 + 5)\ mod\ 3 = 1$
4	*Amethyst Mana*	*Mana Regen* (10)	$(13 * 1 + 5)\ mod\ 11 = 7$
5	*Ruda Essence*	*Health Regen* (5)	$(13 * 7 + 5)\ mod\ 6 = 0$
6	*Soul Mana*	*Mana Regen* (2)	$(13 * 0 + 5)\ mod\ 3 = 1$
7	*Rhodonite Blade*	*Attack* (2)	$(13 * 1 + 5)\ mod\ 3 = 0$
8	*Rhodonite Shield*	*Defense* (2)	$(13 * 0 + 5)\ mod\ 3 = 2$
9	*Ruda Scope*	*Accuracy* (5)	$(13 * 2 + 5)\ mod\ 6 = 1$
10	*Fog Blade*	*Attack* (8)	$(13 * 1 + 5)\ mod\ 9 = 0$
11	*Fog Blade*	*Defense* (-2)	$(13 * 0 + 5)\ mod(-1) = 0$
12	*Ruda Blade*	*Attack* (5)	$(13 * 0 + 5)\ mod\ 6 = 5$

Iterasi	*Upgrade*	*Stats* dan batas atas	Perhitungan *LCG*
13	*Ruda Boots*	*Speed* (5)	$(13 * 5 + 5)\ mod\ 6 = 4$
14	*Rhodonite Shield*	*Defense* (2)	$(13 * 4 + 5)\ mod\ 3 = 0$
15	*Amethyst Mana*	*Mana Regen* (10)	$(13 * 0 + 5)\ mod\ 11 = 5$
16	*Amethyst Mana*	*Mana Regen* (10)	$(13 * 5 + 5)\ mod\ 11 = 4$
17	*Ruda Scope*	*Accuracy* (5)	$(13 * 4 + 5)\ mod\ 6 = 3$
18	*Amethyst Boots*	*Speed* (10)	$(13 * 3 + 5)\ mod\ 11 = 0$
19	*Ruda Blade*	*Attack* (5)	$(13 * 0 + 5)\ mod\ 6 = 5$
20	*Amethyst Blade*	*Attack* (10)	$(13 * 5 + 5)\ mod\ 11 = 4$

Berikut adalah tingkat kemunculan setiap angka:

Tabel 4.9 Frekuensi kemunculan nilai dari *stats* untuk *upgrade*

Angka	Frekuensi Kemunculan
0	6
1	3
2	1
3	1
4	3
5	4
6	1
7	1

Dari hasil percobaan di atas terlihat nilai m yang berubah-ubah sesuai dengan nilai batas atas dari suatu *stats* mempengaruhi nilai yang didapat. Nilai m yang rendah menyebabkan *stats* yang didapatkan pemain adalah batas bawah dari *stats* untuk tiap *upgrade* dikarenakan apabila hasil randomisasi lebih rendah dari batas bawah maka nilai tersebut akan ditambah dengan batas bawah dari *stats*. Hal ini menunjukkan diperlukan adanya nilai yang akan membantu pemain untuk mendapatkan *stats* yang lebih baik.

4.2.2 Hasil Pengujian Oleh Sampel

Peneliti memberikan akses terhadap *game* ke beberapa *playtester* untuk mencoba memainkan *game* yang telah dibuat. 8 *playtester* yang mencoba *game* familiar dalam dunia *video game*. Setelah mamainkan *game* selama 5 sampai 10 menit, semua *playtester* diberikan kuesioner mengenai randomisasi untuk *upgrade* yang telah dibuat.

Kuesioner menanyakan kepada *playtester* apakah randomisasi untuk harta karun yang dijatuhkan musuh, pilihan *upgrade* yang didapatkan pemain, dan penambahan *stats* yang didapat dari *upgrade* sudah cukup acak atau tidak.

Tabel 4.10 Hasil kuesioner implementasi *Linear Congruential Generator*

Yang diacak dalam *game*	Hasil				Total poin	Rata-rata	Kesimpulan
	Sangat Kurang (1)	Kurang (2)	Baik (3)	Sangat Baik (4)			
Harta karun dari musuh	0	1	2	5	28	3,5	Baik
Pemilihan *upgrade* dan *rarity*	0	1	4	3	26	3,25	Baik
Distribusi *stats*	0	1	4	3	26	3,25	Baik

Dari hasil kuesioner ini, dapat diketahui jika randomisasi terhadap berbagai aspek dalam *game* seperti harta karun yang jatuh dari musuh, pemilihan *rarity* dan *upgrade*, serta *stats* yang diberikan kepada pemain memiliki hasil **baik**. Hal ini membuktikan penggunaan *Linear Congruential Generator* sebagai algoritma randomisasi untuk sistem *upgrade* **sukses**.

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan penelitian pengimplementasian *Linear Congruential Generator* untuk sistem *upgrade* dalam *game* bergenre *roguelike*, dapat disimpulkan:

1. Implementasi *Linear Congruential Generator* sebagai sistem dasar untuk randomisasi pemilihan *upgrade* baik dalam memberikan hasil yang tidak dapat ditebak pemain.
2. Implementasi *Linear Congruential Generator* sebagai sistem yang mengatur musuh untuk menjatuhkan harta karun ketika dikalahkan sudah baik untuk menjatuhkan harta karun tidak terlalu sering agar memberikan pemain ketertarikan untuk berinteraksi dengan musuh yang ada.

5.2 Saran

Adapun beberapa saran yang bisa diberikan setelah penelitian ini, yakni:

1) Penelitian selanjutnya dapat meningkatkan jumlah *upgrade*, *rarity*, dan juga *stats* yang dapat dirandomisasikan dalam program.
2) Menggunakan *Linear Congruential Generator* dalam aspek randomisasi lain seperti tempat munculnya musuh, nilai *experience* yang didapatkan pemain setiap mengalahkan musuh, ataupun memilih *stage* setelah mengalahkan suatu *level*.

DAFTAR PUSTAKA

Bycer, J. (2021). *Game Design Deep Dive: Roguelikes*.

Desliyanti, D., & Fahry, A. (2023). *APLIKASI PEMBELAJARAN BAHASA INGGRIS DENGAN MENGGUNAKAN ALGORITMA LINEAR CONGRUENT GENERATOR BERBASIS ANDROID*. *5*(1). https://jurnal.ikhafi.or.id/index.php/jusibi

Elveny, M., Syah, R., Jaya, I., & Affandi, I. (2020). Implementation of Linear Congruential Generator (LCG) Algorithm, Most Significant Bit (MSB) and Fibonacci Code in Compression and Security Messages Using Images. *Journal of Physics: Conference Series*, *1566*(1). https://doi.org/10.1088/1742-6596/1566/1/012015

Krishnamoorthi, S., Jayapaul, P., Dhanaraj, R. K., Rajasekar, V., Balusamy, B., & Islam, S. H. (2021). Design of pseudo-random number generator from turbulence padded chaotic map. *Nonlinear Dynamics*, *104*(2), 1627–1643. https://doi.org/10.1007/s11071-021-06346-x

Limbong, T., & Matondang, Z. A. (2021). IMPLEMENTATION OF THE LINEAR CONGRUENT METHOD IN INTERACTIVE QUIZ GAMES APPLICATION. *JURNAL INFOKUM*, *10*(1). http://infor.seaninstitute.org/index.php/infokum/index

Makmur, F., Daniawan, B., & Wijaya, A. (2019). Computerized Semester Exams by Randomization Order of the Questions with Linear Congruential Generator Methods (Study Case: Agathos Vocational High School). *Bit-Tech*, *1*(3). http://jurnal.kdi.or.id/index.php/bt

N, S. G., É, B. B., & Sz, O. G. (2022). Roguelike Games-The way we play. *International Journal of Engineering and Management Sciences (IJEMS)*, *7*(4). https://doi.org/10.21791/IJEMS.2022.4.7

Okada, K., Endo, K., Yasuoka, K., & Kurabayashi, S. (2023). Learned pseudo-random number generator: WGAN-GP for generating statistically robust random numbers. *PLoS ONE*, *18*(6 June). https://doi.org/10.1371/journal.pone.0287025

Safitri, D. N., Fitri, I., & Nuraini, R. (2021). *Implementasi Metode Linear Congruential Generator Pada Game Puzzle Kesenian Tari* (Vol. 8, Issue 1). http://jurnal.mdp.ac.id

Siahaan, A. M., & Hendrik, J. (n.d.). *BULLETIN OF COMPUTER SCIENCE RESEARCH Perancangan Aplikasi Edukasi Pembelajaran Alfabet dan Angka Berbasis Android dengan Metode Linear Congruential Generator (LCG)*. https://doi.org/10.47065/bulletincsr.v3i1.223

Witney, O. (2019). *Usage of Random Number Generators and Artificial Intelligence to improve replayability of a Roguelike Game*.

LAMPIRAN

KUESIONER RANDOMISASI MENGGUNAKAN *LINEAR CONGRUENTIAL GENERATOR*

No	Pertanyaan	1	2	3	4
1	Apakah harta karun yang jatuh dari musuh tidak dapat tertebak jatuhya? (Tidak ada pola kapan jatuh, tidak terlalu sering maupun terlalu jarang)				
2	Apakah pilihan *upgrade* yang dapat dipilih ketika membuak harta karun sudah cukup random? (Tidak ada pola, *upgrade* dengan tingkatan *rare*/berwarna ungu terkadang muncul tapi tidak terlalu sering)				
3	Apakah penambahan *stats* dari *upgrade* sudah cukup random? (Tidak ada pola, nilai penambahan *stats* berubah setiap *upgrade*)				

TABULASI JAWABAN RESPONDEN

Responden	1	2	3
A	4	4	4
B	3	3	4
C	4	3	3
D	4	4	4
E	4	4	3
F	4	3	3
G	2	3	3
H	3	2	2