RANDOM NUMBERS FOR GENERATION IN WEB GAMES
And how the quality of them effects the end user.

Bachelor Degree Project in information technology
Basic level, 30 high school points
Spring term 2019

Linus Jacobsson

Supervisor: Yacine Atif
Examiner: Henrik Gustavsson
Summary

Today games are moving more and more towards being mobile and available at all times. But as mobile as a platform have bigger limitations than PC or gaming consoles have, less storage, worse connection etc. A great solution for this is to generate content on the fly for the user, this often requires the use of random numbers. There are several ways of getting random numbers today, but how does the choice of random number method impact the user and the game? In this work we test just that and finds that in most cases it wont really matter what kind of method you use, but there is still a possible impact that future work will have to be further explored.

Keywords: Procedural content generation, mobile games, random numbers, web games
Table of content

1 Introduction.................................................................................................................. 1
2 Background.................................................................................................................. 2
  2.1 Pseudo-random number generator......................................................................... 2
    2.1.1 Common randomness ...................................................................................... 2
    2.1.2 Cryptographically strong randomness in Java................................................ 3
  2.2 Platformers and levels............................................................................................ 4
    2.2.1 Procedural Content Generation...................................................................... 4
    2.2.2 Rhythm-based level generation....................................................................... 4
3 Problem and hypothesis............................................................................................... 5
  3.1 Problem statement................................................................................................... 5
  3.2 Hypothesis................................................................................................................ 5
  3.3 Related work............................................................................................................ 7
4 Methodology ................................................................................................................ 8
  4.1 Runtime measurements – Experiment................................................................. 8
  4.2 User-experience – Case study................................................................................ 9
  4.3 Ethical issues............................................................................................................ 9
5 Implementation ............................................................................................................ 10
  5.1 Case study............................................................................................................... 10
  5.2 User study................................................................................................................ 13
    5.2.1 Questionnaire.................................................................................................. 13
    5.2.2 Interview.......................................................................................................... 14
  5.3 Progression.............................................................................................................. 15
  5.4 Pilot study............................................................................................................... 16
  5.5 Literature study...................................................................................................... 17
6 Result ........................................................................................................................... 19
  6.1 Analysis.................................................................................................................... 19
    6.1.1 Experiment...................................................................................................... 19
    6.1.2 User study....................................................................................................... 21
  6.2 Findings.................................................................................................................... 25
    6.2.1 Discussion....................................................................................................... 26
  6.3 Future work.............................................................................................................. 26
7 References.................................................................................................................. 28
1 Introduction

The gaming industry is bigger than ever and just keeps growing. With a growing industry, the publisher’s competition in the market gets harder, and the games standards gets higher and higher. With this trend, the content creation for games is becoming a bottleneck, as content takes a lot of time to develop.

A lot of content can also be a problem as gaming is growing more and more on the mobile platform that usually have a lot less computing power and storage. With mobile developer also tend to get their games to we web-based as that would make the game available anywhere, from any device.

As the amount of content that developers have to create to reach up to current industries competition is ever-rising, interests for generating content are increasing. There are several ways for generating different kinds of contents for a game, but none of them takes randomization methods into consideraion. What if we want to use high-quality random-numbers for the generation? How does that impact the game and gameplay?

No method of creating randomness is created equal, they all have a specific problem that they are trying to solve. But why do we need so many? Are they not all trying to solve the basically same problem? They all generate numbers that we humans interpreted as random, and also to us as humans, they all do it at an amazing speed. But if we build huge applications that are all communicating with the same server, these small differences in speed of the methods can become significant. Is it worth to sacrifice the “quality” of randomness for more speed? In some cases yes, but in others, the quality of the randomness is by far more important than its generation speed. Like in the e-gaming industry where low quality randomness would cause you to lose your online gambling license, not allowing you to run your business until the quality is improved and reevaluated.

Do the method chosen for the random generation impact the game in a way that the users can experience a difference? And if so in what kind of way? These are all questions that this report tries to answer by building a prototype game that lets a user play two levels, which are generated with exactly the same logic, but with different methods of creating randomness.
2 Background

2.1 Pseudo-random number generator

Often referred to in its short form, PRNG, or Pseudo Random Number Generator, this computational random number generator function does not however deliver true randomness, but is in some way deterministic. These functions available in various programming languages work by following a long list of numbers and applying them to an algorithm to create numbers that are interpreted as random to the user. Because of this the PRNG methods often have a hard-time to pass statistical tests to determine the quality of their “randomness” (Like the DIEHARDER tests (Brown 2019) or TestU01 (L’Ecuyer & Simard 2007)), and how well they perform in the test is almost entirely decided by the length of the list of generated numbers.

To create something random with a computer that is not only pseudo-random, one can use special a hardware that uses quantum physics to create randomness (L’Ecuyer & Simard 2007). These are often referred to as HRNG or TRNG (Hardware/True Random Number Generator) and looks at different kinds of noise from photons to create randomness.

2.1.1 Common randomness

Most randomness in computer programs uses a linear congruential algorithm. It is deterministic, meaning that if you were to create two instances of a class like this, and call the same methods from each instance in the same order, the “randomness” generated will be exactly the same (Oracle 2019a). Although it is generally enough for most applications (Coddington, Mathew & Hawick 1999), it is however not cryptographically safe as its a pseudo-random generation. This means it is not fit for cryptographic use as it is deterministic and thus ciphers could be guessed. For an in-depth explanation of this linear congruential algorithm, the reader can look at D. Knuth’s book “The art of computer programming, Volume 2” (Knuth 1997).

One of the most used pseudo random number generators for java is util.Random. Mostly because the more commonly used Java.Math.Random creates an instance of this generator. The method uses a predefined algorithm to produce the numbers within the main class, which can extended into a subclass, as shown in figure 1. However, you can’t overwrite or change the algorithm inside of the main class, this is mainly because Java wants to maintain absolute portability (Coddington, Mathew & Hawick 1999; Oracle 2019a).

To create random numbers, the algorithm builds on a linear congruential generation that modifies a seed. Thus, a seed number is required to generate the random sequence, which is why the generated random numbers are not truly random. The method allows for a seed up to 48-bit, however all the numbers generated are 32-bit (Coddington, Mathew & Hawick 1999; Oracle 2019a). The theory as well as the optimal selection of a seed are beyond the scope of this document. However, the reader may consult more in-depth theoretical resources (Fontaine 2011).
2.1.2 Cryptographically strong randomness in Java

This is a native Java subclass of the Java.util.Random class made to have the possibility to use cryptographically strong randomness in Java (Oracle 2019a). This means that it meets the federal information processing standards, which specifies the security requirements for cryptographic modules through several different kinds of statistical tests (National Institute of Standards and Technology 2001).

The method however doesn’t automatically give the user cryptographically-strong numbers, but is dependent on the actual implementation of the class and the employed algorithm. The default is a so called PRGN (pseudo-random number generator), similar but not equal to the class it is based on. To override the default algorithm, one can set it up to use a specific algorithm from a list supplied ones by Java as shown in Figure 2 (Oracle 2019b).

If the default set of algorithms are not sufficient for the intended use, just like Java.util.Random, you can supply your own algorithms by subclassing (Coddington, Mathew & Hawick 1999).

```java
import java.security.SecureRandom;

public class MyExample {
    public static void main(String[] argv) {
        SecureRandom sr = new SecureRandom.getInstance("SHA1PRNG");
        System.out.println(sr.nextInt(101));
    }
}
```

Figure 2 - Feeds SecureRandom with a specific algorithm and prints a number between 1 and 100
2.2 Platformers and levels

Platformers are games that often, but not always are in 2D and is seen from the side. The games are often easier to understand without any complex mechanics and got a huge focus on making the levels interesting to the user (Smith, G. et al. 2011). Some more known characteristics for platformers are hidden areas, enemies, collectible items and of course platforms (Smith, Gillian, Treanor, Whitehead & Mateas 2009).

The game often challenges the user in different kind of ways. Some more usual challenges are advanced platforming sections or puzzles the user has to solve to make progress.

On of the biggest challenges with mobile games on the web according to Santanché et al is peoples concern about performance, as mobile platforms have usually less performance and storage to work with. This usually leads to developers having to release and maintain several versions of a game if they are to release it to mobile as well as other platforms. However web technologies is a promising approach to this as its already something that most platforms share (Santaché A et al 2013).

2.2.1 Procedural Content Generation

Content for games takes a lot of time to create, but if we instead generate the content, it can greatly reduce the development time and also opens up new doors to explore in the design of the game. As the average amount of content in games during the last years have gone up, the content that needs to be designed is quickly becoming a bottleneck in the gaming industry (Hendrikx, Meijer, Van Der Velden & Iosup 2013). The topic of generating this kind of content is more commonly known as procedural content generation.

There are several reasons to look at this kind of approach more than decreasing development time and cost. Some of the cool features this technique offers are: the game could adapt to the players play style or situation, creating a unique playthrough for every player. A game with an infinitely large world generated by an algorithm could still have a small size on the hard drive. The game could generate an infinite amount of different items the player can interact with and find new possibilities in the game (Smith, Gillian, Treanor, Whitehead & Mateas 2009; Hendrikx et al. 2013).

The things that could be generated with procedural content generation are currently placed into six different categories by M Hendrikx in his survey “Procedural content generation for games - A survey”: Derived content, game design, game scenarios, game systems, game space and game bits. If you want to read more about the possibilities with procedural content generation, this book is highly recommended and goes into depth with each category, talking about its possibilities and problems (Hendrikx et al. 2013).

2.2.2 Rhythm-based level generation

There are some common problems when generating a level for a game using procedural content generation algorithms. In the end, many methods end up generating one or more unbeatable levels that obviously creates a bad user-experience. The levels also feel random, like there is no thought process behind objects placements and how the user beats it. (Smith, Gillian et al. 2009)
One solution to these problems is to first generate how the user should give inputs to beat the level, and then make a level that can be beaten with that sequence of inputs. The program can now make sure that the level is beatable. If we also add rhythm as a variable when making the sequence, we can make challenges that feel like a bit more thought have been placed behind the generation of the level (Smith, G. et al. 2011).

3 Problem and hypothesis

The interest to automatically generate levels for platform games is on the rise. To be able to generate levels is an attractive feature, since it reduces the cost of production, saves storage for users’ end devices, this being extra important now when gaming is leaning more and more towards web-technologies and mobile (Santanché A et al 2013), and adds a possibility for the game to adapt to the user’s playing style (Smith, Gillian et al. 2009; Shaker et al. 2011). One simple approach to this problem is to base the generation on rhythm and randomness (Smith, Gillian et al. 2009; Smith, G. et al. 2011). Since Java works great for running several concurrent threads in one game and given that most RNG methods are synchronized (i.e. thread-safe), a slow generation of the random numbers could easily become a bottleneck for the game. This can be a problem if the levels are generated in real-time. (Coddington & Newell 2004).

There is a huge amount of different random generators to choose from. Some create “better” randomness than others, while others are faster. The choice of a random number generator method is highly relevant to the kind of project that is in development, especially if the generation happens simultaneously to other important calculations (L’Ecuyer & Simard 2007).

3.1 Problem statement

Creating levels for a platformer game with the help of randomness is something that can be challenging. One of the most common problem with this kind of approaches is that the levels are often impossible for a player to beat. This has led to using randomness for usually creating open landscapes, or choosing the order of already made puzzle pieces to patch together a complete level. A solution to this problem is to randomize a rhythm-based pattern, and from that build the level (Smith, Gillian et al. 2009). The randomly generated numbers used in the game have to be generated without any noticeable impact for the user, at the same time as the levels are interpreted as interesting and unique (Smith, G. et al. 2011).

More specifically, the problem is to compare the speed and quality of random-generation operations along with perceived user-experiences in gaming when employing util.Random vs. security.SecureRandom for generating platform-levels in a game.

3.2 Hypothesis

H1: The hypothesis for this work is that Java SecureRandom and util.Random are both sufficient for creating games.

H2: Even if everything points towards SecureRandom being slower, both are fast enough to not affect the gaming experience.
H3: The spread of the random numbers in both methods is good enough so that a human user would not notice a difference.
3.3 Related work

There is an earlier work somewhat related to this where a student at Högskolan i Skövde did a comparison of two different rhythm based generation algorithms for a platformer called “Procedurell generering av nivåer för plattformsspel: En jämförelse av två algoritmer”, written by Johan Elmquist. Johan did a survey with users over the Internet where they could play a game that used both algorithms. (Elmquist 2016)

In 2010, there was a level generation competition held by IEEE. Their report about the competition brings up a lot of interesting ideas about a topic called PCG (procedural content generation) and also a great way to judge between two different level generation algorithms with a user survey and small prototype. The competition is to raise awareness about this field of AI, since during its time the most AI that was developed was for so called non-playing characters. And AI that could generate good content would be great from a lot of different aspects for the game industry (Shaker et al. 2011).
4 Methodology

To measure and compare the quality of the generated numbers, there are several established methods that are used, two of them are called DIEHARDER and TestU01 (L’Ecuyer & Simard 2007). All of them are however heavily statistical and do not create representative results for how they would work when used in a game. A case study would be best to collect data about the user-experience for the two different methods in a simple prototype together with a questionnaire (Wohlin et al. 2012). In a prototype, data about the generation can also be collected while running the tests and then studied to see whether the level generation is a bottleneck for the runtime of the game.

A platformer game will be created that can use both RNG methods for the level generation as a mid-fi prototype. This will be a playable version so that user-experience can be tested in a realistic setting that ends with a questionnaire. The game will be a minimum viable product with just enough game elements to make up the game and make it interesting. During the gameplay the prototype also collects data about the generation and gameplay. For example, the prototype collects how long time did the game spend to generate a new part of the level while the user is playing. Utilizing this, we can relate specific numbers from the experiment to the answers to the questions in the questionnaire collected from each user.

4.1 Runtime measurements – Experiment

During the tests with users the prototype will gather data about its runtime. When it starts generating a part of the level, it will save the system time as a variable, and then compare it to the system time when the generation is done to get an overview of how long it takes to generate new content. It also saves all the numbers that will be generated for the method so that we can look at the quality of randomness from the methods. The varying factor of the experiments will be the method used for generation, and the variables we gather from the experiment is its generation time, and the numbers that are generated and then used in the level generation. Parts of the experiment results from this project will be quantitative that will have to be analyzed to find any kind of patterns or trends (Wohlin et al. 2012).

The collected data include the time-cost consumed by both Java random-generation functions involved in the experiment, and discussed in earlier sections. Samples are collected for various contents across game levels. Quantitative analysis is conducted to evaluate the variation of random generation across different levels of the prototype. The user perception of this difference is also evaluated as discussed in the next section.
4.2 User-experience – Case study

To measure the user-experience of the different methods, a case study is performed in the form of a prototype with both methods implemented. A case study is a method that focuses on a single entity of information in a real life situation. (Wohlin et al. 2012). The approach is similar to how the Mario AI competition was judged where the user first gets to try a level generated by one method, and then a level generated by the other. A user gets to test the prototype and after the test, they get to fill out a questionnaire about the test to report their thoughts and experience about each level (Shaker et al. 2011). The results will then be studied to see whether the user could experience any difference. This qualitative analysis complements our previous quantitative evaluation to assess user perception of the random-generation variations.

What's interesting to gather here is data about the users’ perceived experience. Or more specifically, whether any method was considered more fun than the other, to ascertain whether the users will notice a significant difference between the levels, and whether the user could notice any lag or interruptions while the level is generated. This approach is more quality focused than quantity focused, as every test will give the exact data that the test wants to bring forth without having to break it down and study the results. However, the disadvantage with case studies is that the information gathered will not cater to other situations. We will only know if the user is experiencing a difference in this specific situation within this specific setup (Wohlin et al. 2012).

4.3 Ethical issues

Since the empirical research involves humans for the survey the research must take its ethics into consideration (Wohlin et al. 2012). And also, since the research is also going publish all data the is gathered from the survey to be as open as possible, the survey will have to get consent from every test subject, and keep it totally free of personal information (Sieber 2001). The consent comprises the elements talked about in Wohlin’s book at chapter 2.11 (Wohlin et al. 2012). This however introduces another ethical problem. Since it doesn't gather any personal data, it's hard for any reader to see whether its focused on any special group of people. And the diversity of the test group will be impossible to verify.

The entire research results, survey reports, and the source code for the prototype will be hosted on github so that the research could be replicated as closely as possible. Opening up for a replication that is as close as possible still keeps the possibility to make a differentiated replication.
5 Implementation

The test is split into three different parts to be able to triangulate the collected data from different samples. This chapter explains how they came about and how the important parts for the pilot-study artifact is built.

5.1 Case study

The first part of the pilot-study starts with a platformer game written in legacy java without any frameworks. This is to give as much control over the program flow as possible. The experiment starts from a regular platforming-game that was modified to fit the test needed in this part of the pilot-study. The prototype game is created so that the game can gather data about the RNG methods while the user is playing the two different levels.

The user first gets to play a level generated with cryptographically-strong randomized numbers for three minutes. Then the game resets and generates a level using the normal randomization instead for an additional three minutes. This can be done easily since both methods can be an instance of the java “Random” class. To do this, the experiment uses a singleton class to handle all randomization. The singleton class contains code that allows switching between its RNG method as shown in Figure 3.1

```java
private static void setState(State state) {
    Randomizer.state = state;
    switch (Randomizer.state) {
        case NORMAL:
            random = new Random();
            break;
        case CRYPTO:
            try {
                random = SecureRandom.getInstance("SHA1PRNG");
            } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
                random = new SecureRandom();
            }
            break;
        default:
            random = new Random();
            Randomizer.state = State.NORMAL;
            break;
    }
}
```

Figure 3 - the function that is used to switch what method is being used for randomization.

1 [https://github.com/d16linja/Examensarbete/pull/4](https://github.com/d16linja/Examensarbete/pull/4)
Now that the experiment has the ability to switch between the two different RNG methods, it also needs to be able to time the generation of a part of the level, so that the experiment can determine what method makes the fastest generation. To achieve this, we developed a function that can generate a new part (or chunk as it is called in the program code) that is used as a time-measurement segment. When we call that function, it starts by storing the systems time in nanoseconds, then it generates the (level) chunk and adds it to the game, and yet again stores the systems time in nanoseconds. Then we use another object that transfers time-measurements to the drive, (its functionality is explained further down in this section). In Figure 4, we can see a code example of how this timing of the generation works.²

```java
public void generateNewChunk()
{
    Randomizer.clearData();
    long startTime = System.nanoTime();
    List <WorldObject> world = generateWorld();
    List <AgentObject> agents = generateAgents(world);
    new Chunk;world, agents;
    long endTime = System.nanoTime();
    dataHandler.writeData(chunkList.size()-1, Randomizer.getState(), Randomizer.getData(), endTime-startTime);
}
```

Figure 4 - generating and timing a new chunk to finally write the data to a file.

To easily read the data collected from the experiments part, it needs to have a suitable structure, so we know what data came from what test and what method was used. This structure needs to correlate with the answers from the questionnaire and interviews, to complete the triangulation process. The experiment was made so that it puts all the data in a folder structure where the first run of the test gets a folder named “0” and then the next run of the test would put its data in the next folder “1”. In this folder, it then creates two new folders, one for the data from the level generated with “normal” RNG and one for cryptographically strong RNG called “NORMAL” and “CRYPTO” respectively.

² https://github.com/d16linja/Examensarbete/pull/4
Figure 5 - The experiments data structure
In the data files for each chunk, we save two different kinds of data. The first row in each data file is the time in nanoseconds it took to generate that specific chunk, and then the following rows represent every number that was made with the RNG method for creating that chunk. This makes it possible to get a mean time of each generation and also a mean time for each randomization instance. The experiment also leaves the possibility to look at the numbers randomization quality.

```java
public void writeData(int block, Randomizer.State state, List<Double> data, List<Double> time) {
    File dataFile = new File(currentRuntimePath + "/" + state + "/" + time);
    dataFile.mkdirs();

    dataFile = new File(dataFile.getAbsolutePath() + "/" + state + "/" + block + ".txt");
    dataFile.setWritable(true);

    try {
        dataFile.createNewFile();
        FileWriter writer = new FileWriter(dataFile);
        writer.write((int) time + "n");
        for (Double data : data) {
            writer.write(data + "n");
        }
        writer.close();
    } catch (IOException e) {
        System.err.println(e); System.err.println("Error writing testdata, needs manual correction at ");
        System.err.println(data.getAbsolutePath() + "/" + state + "/" + block + ".txt");
        System.exit(0);
    }
}
```

Figure 6 - writing the data according to the specified structure

5.2 User study

5.2.1 Questionnaire

The questionnaire we made to collect data about the users’ perceived experience of the game. The most important data to gather here is if the user’s input towards different kind of game levels and whether they felt any time lag. To measure these perceptions towards the different levels the questionnaire, we used a likert scale of 1-6 (Sauro & Dumas 2009). And to see whether the user experienced any lag during any of the levels, the questionnaire simply asks it. The questionnaire also gathers the users’ experience with computers to be able to evaluate the results against any impacts of the answers which might be a possible margin of error. It also helps to adapt the questions for the subsequent individual interviews. The questionnaire was made using Google forms as illustrated next.

---

3 https://github.com/d16linja/Examensarbete/pull/4
5.2.2 Interview

The interview is used to discuss the perceived differences during the test. This step does not contribute to collect the main underlying data for our pilot study, but it is used to identify possible patterns among users, whenever some point of data stands out from the crowd, we could get their feedback in the interview to get a deeper understanding of that data as another source of our triangulation methodology (Cros, Chaumon & Cuvillier 2015). The interview does not consist of predetermined questions, but has the goal of bringing-up what the user thinks was different or what they felt different.

Figure 7 - an example where a Likert scale was used to measure attitude
5.3 Progression

During the implementation, the level generation was changed from being rhythm-based to a more simplified generation logic. This will affect the results of the measurements, but will not matter for the sake of the experiment as both methods still use the same level generation logic. The magnitude of the difference between them will not be as big since the logic is simpler, but will still be visible in the same way as if the rhythm-based.

```
private List<WorldObject> generateWorld() {
    List<WorldObject> list = new ArrayList<WorldObject>();

    boolean makeHole = false, holePossible = true;
    if (chunkList.size() == 0) {
        for (int i = 9; i <= 11; i++) {
            list.add(wf.getWorldObject('s', i, 25));
        }
        for (int i = 0; i < 40; i++) {
            list.add(wf.getWorldObject('s', 0, i));
        }
    }

    for (int i = 0; i < 60; i++) {
        if (chunkList.size() == 0 && i == 0) continue;

        if ((Randomizer.get() > 0.95 || makeHole) && holePossible) {
            if (makeHole) {
                makeHole = false;
                holePossible = false;
                continue;
            } else {
                makeHole = true;
                continue;
            }
        }

        if (Randomizer.get() > 0.3) {
            list.add(wf.getWorldObject('g', i, 39));
        } else {
            list.add(wf.getWorldObject('s', i, 39));
            int k = 1;
            for (double j = Randomizer.get(); j > 0.5 && k < 6; j = Randomizer.get()) {
                list.add(wf.getWorldObject('s', i, 39-k));
                k++;
            }
        }

        holePossible = true;
    }

    return list;
}
```

Figure 8 - The simplified generation logic

*https://github.com/d16linja/Examensarbete/pull/5*
An interview was also added when during our research of user studies, the term “triangulation” came up as a method to analyze complex problem. This will make the analysis of the data more comprehensive to see why users answered the questionnaire as they did, and correlate this feedback with empirical measurements collected throughout the experiment. The interview will be used to further be able to study points of data that sticks out.

5.4 Pilot study

A pilot study was performed to evaluate the feasibility of data collection in support of our experiments. However, some irregularities were observed that will have to be analyzed in the subsequent step of our experiment study. Since the first chunks of the level is created at the start of the game, the first few chunks always take much longer to build than the subsequent ones, going down to a stable number after the first five chunks have been created (these are the initially made chunks).

![Figure 9 - An example of when startup effects the timing](image)

The same pattern is not seen when the level changes, as much less computing power is needed for that operation. This is not a problem for the test, but the first five chunks generated may have to be cut out of the result. An example of the first few lines of data from running the game can be seen in Figure 8, where the first row is the creation time of that chunk, and the subsequent rows are numbers from the RNG method that were used to generate that chunk.
Figure 10 - an example of the data gathered from playing the game

There were no problems with the questionnaire during the pilot study and the user thought it was very clear and easy to answer. Google forms allow us to export the answers to a so called google sheet (google's version of excel) which in turn can be exported to a format suitable for analysis. Finally, the interview did not give anything noteworthy from this first run of the test. However, subsequent candidates for the interview have been identified to participate in further experiments that augment our test data and complement our triangulation methodology.

5.5 Literature study

No random number generator method is equal, some even suggest that methods that can't create good enough randomness should simply be thrown away, never to be used again (L'Ecuyer 1998). But you could make the argument that the method that you use is entirely up to what its purpose is, and you could also make arguments about the empirical tests overall. For example, the Java.math.random class is just a normal pseudo random number generator, but scores great on the TestU01 empirical tests (L'Ecuyer & Simard 2007). However, according to its documentation and Prof. J. R. Weiss, it's made to uniformly spread its number, which would indicate that the method is not very random at all, even if it passes empirical tests (Weiss 2004; Oracle 2019c). An easy way to test if your number are uniformly spread or not with floats or doubles as this work uses, is to look at the numbers generated and see if there is a somewhat clear bias leaning towards 0 or 1 (Weiss 2004).

The idea of using a questionnaire and building prototypes to evaluate the users' attitude towards different levels come from the Mario AI competition where the competition for year 2010 was to build level generation for a Mario game. To evaluate the users' attitude towards different levels that was generated the users got to play two levels made by to different methods and then answer a questionnaire. As the competition was sponsoring a IEEE conference the documentation about it is very good, and the most interesting part for this research is very well explained, the scoring system. They however use a more simplistic questionnaire that only has one enforced two-alternative question: "Which level was better" (Shaker et al. 2011). Barnum also agrees with this testing method, where you follow up the testing scenario with a post-test questionnaire (Barnum 2011). Having the questionnaire immediately after the scenario, gives the user a low risk of forgetting important aspects or experiences during the test and their answers are more reliable for the study.
We use a Likert scale to collect data about the users perceived experience (Sauro & Dumas 2009; Barnum 2011) and also try not to influence the users while answering the questionnaire by using words like “difficult” or “easy” in the questions itself, only in the Likert scale that serves as an answer from the user (Barnum 2011). Barnum also recommends not writing your own post-questionnaires, but to use a standard post-test questionnaire. However, while looking at the two most popular ones (SUS, System Usability Scale and CSUQ, Computer System Usability Questionnaire) we can quickly determine that while they could be used, more specific questions could more easily form a result for our tests as the generic standard questionnaires are, well, too generic (they are probably amazing while testing for usability and not perceived experience). However, SUS does bring up an interesting way of using the Likert scale to calculate a “SUS-score”, something that could be applied to any questionnaire using Likert scales (Barnum 2011).
6 Result

6.1 Analysis

Before the summarization of the analysis we need to look at its statistics as objective as possible. So for the analysis of the data from the experiment a confidence interval was used together with anova as well as comparing graphs of the data between the two methods. And for the user study we compare the users’ answers to the data from the experiment and then lastly we do a qualitative analysis of the interview data.

6.1.1 Experiment

First we need to see whether our data have a statistically significant difference. We can do this by calculating the methods’ average margin of error by supplying a confidence level, the datasets standard deviation and its size. We want to be confident in our result, so a confidence level of 95% was set. The graphs can then be placed next to each other to visualize how much their confidence interval intercepts each other. If they don't intercept each other we can be at least 95% sure that there is difference between the two methods. If they do intercept, it's still not guaranteed that there is no difference. We can do an Anova (analysis of variation) test for the dataset to study the difference further.

![Confidence interval graph](image)

Figure 11 - the confidence interval graph
From the confidence interval graph we can clearly see that their confidence intervals intercepts. And we cannot say for sure that there is a difference in the datasets or not. To study this further a anova test was performed via excel. But before we do an anova test, we must check if we can see any form of normal distribution in the data. If we can’t, we must take the parametric tests with a “grain of salt” as they are known to be sensitive towards non-normal distribution. It could produce a false positive with datasets that don’t have normal distribution. As the Java.util.Random method is made to have a uniformed distribution, we should not expect a normal distribution from it.

![Distribution Graph](image)

**Figure 12 - our two datasets distribution**

Anova table is shown below:

**Table 1 – results from anova test**

<table>
<thead>
<tr>
<th>Source of Variation</th>
<th>SS</th>
<th>Df</th>
<th>MS</th>
<th>F</th>
<th>P-value</th>
<th>F crit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between Groups</td>
<td>0.051297442</td>
<td>1</td>
<td>0.051297442</td>
<td>0.616971289</td>
<td>0.432175861</td>
<td>3.841501566</td>
</tr>
<tr>
<td>Within Groups</td>
<td>18087.80442</td>
<td>217548</td>
<td>0.08314397</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

There are a few things we can gather from the anova. Since the confidence level was chosen as 95%, the P-value will need to be less than 0.05 for us to say that the null-hypothesis is false. All our statistical analysis of the data from the experiment points towards there is not any significant statistical difference between the two methods.

As we also stated in our hypothesis that one of the methods would perform slower than the other when used for generation, this is also something that we need to analyze and confirm. To do that, we first summarize all the time each method spent generating all of its chunks, than
we count all of the random numbers that were used, so that we get a mean of time spent per number used. Later we could also compare it to the users perceived lag to see whether they correlate.

![ns per random number used](image)

**Figure 13 - A block graph that displays each method's mean time per number**

From this data we can conclude that the crypto method used 53.8% more time for each of its numbers during the generation.

6.1.2 User study

Since we used a likert scale to gather the users thoughts about how much they liked each level there are a few things we can do to analyze the answers. The most obvious and easiest to utilize is to calculate the mean of each question and look if one is greater than the other. This won’t say for sure that one of the methods made better levels than the other. To conclude that we would also have to look at the spread of the answers and compare them with the interview studies.

![Perfered level likert scale mean](image)

**Figure 14 - the difference in the user’s opinion of each level**
The graph tends towards that there was not any significant difference in what the users felt about each level, at least not when observed as a group. However, the distribution of these would give us a more clear view of their possible differences. If there is a close to no difference, the distribution graphs should almost line up.

![User opinion distribution graph](image)

**Figure 15 - The distribution of the user’s opinion of each level**

The distribution of the user's answer shows a much bigger difference than what the mean of their answers did. The opinions about the crypto level is far more spread than the opinions of the normal level.

So far it's hard to make the conclusion with confidence if the users perceived a difference, however we do have an interview question that answers just that.

![Users perceived a difference](image)

**Figure 16 - times a user mentioned any perceived difference in the interview**

In the questionnaire the test also asks the user if they experienced any lag during the test on either level. They could choose from picking one of the levels, no level and both levels. To
evaluate the users perceived lag of each level we simply add up all the times a user said that a level lagged (so the answer “both levels” would add up both levels total). Than we can use another bar chart to clearly see the difference.

![Times users perceived lag](image)

**Figure 17 - times a user said that they perceived lag for each level**

From this graph, it's very clear that the users did not perceive any lag more on one level or the other. Later in the findings chapter this will be compared to the actual lag to answer one of the hypotheses.

Next we start looking at the interview and prepare to make a qualitative analysis. To do this we first read through all of the interviews, and try to take notice of anything significant or of interest. When we have read through all of the interviews and found some significant points made during them we have to read all of the interviews again. This time with all the points of significance in mind. Next we try to summarize key subjects that were mentioned in the interview that we later can connect the interviews to. The key subjects that was identified was: (x being any of the levels)

- There was any difference
- X was more flat
- X had less distribution of things
- X had more holes
- X had more enemies
- X had more coins
- X had a better variation of things
- X had clumped obstacles
Next we read through all of the interview one more time. However this time we have made a table where each row is an interview, and each column is one of the identified keywords, lastly we fill what level was mentioned about what key point for each interview. This gives a good overview of our interviews that can be used to make conclusions.

<table>
<thead>
<tr>
<th>Difference</th>
<th>Hat</th>
<th>Less distribution</th>
<th>More holes</th>
<th>More enemies</th>
<th>More coins</th>
<th>&quot;Better&quot; variation</th>
<th>Clumped obstacles</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 18 - overview of the data pulled from the interviews (the darker background indicates that it was the second level played by the user)

6.2 Findings

H1: The hypothesis for this work is that SecureRandom and util.Random are both sufficient for creating games.

Confirmed, both are very much sufficient as a RNG method, however one should take the significance of the random numbers context into consideration. Also take into consideration that the cryptographically strong method proved to take 53,8% longer time when used for this specific generation.

H2: Even if everything points towards SecureRandom being slower, both are fast enough to not affect the gaming experience.

Confirmed, the analysis of the data proves that the SecureRandom method took 53,8% longer time when used for generation in the game. If the 53,8% was a significant enough to introduce lag the users would experience it. However the analysis of the questionnaire proves that the users could not experience any noticeable lag more in one level or the other.

H3: The spread of the random numbers in both methods is good enough so that a human user would not notice a difference.

Refuted, the data collected from the interview clearly states that 75% of the users noticed some kind of difference in the two levels. The comparison of distribution with the users are also quite different, which would also imply that the users noticed some difference. Although it's hard to
say that its 100% disconfirmed as well, since the statistical analysis of the number collected from the generation quite clearly points towards there being no clear difference.

6.2.1 Discussion

About the findings for H3. If there is an actual difference or just a perceived difference is hard to say. There are a few things that I can think of that could explain this, but more work would definitely have to be done to give an absolute answer. What I think might have had an impact is the question in the interview. The question “What do you think was different between the levels” implies that something was different, and could possibly impact the users answers. However the comparison of distribution from the likert scale in the questionnaire also points towards a perceived difference. That would support that the users noticed a difference. To add to that point the users also filled the questionnaire before knowing much about the test or having the interview, where a difference could be seen as being implied.

There are also some other interesting points to be made from the interview, however most of them are just patterns that are starting to show themselves at most. To be sure about the data gathered from the interview the pattern has to be either very clear, like if any difference at all was noticed, or a much larger scale of the test will have to be performed with a lot more tests. The points that can be made from the interview are things like “The users perceived the normal level to have more clumped obstacles” from it being 15% of the users mentioning that as a perceived difference. 15% might seem low, but seeing as it’s something that multiple users have mentioned about that level, and no users mentioned it about the other level, it’s very possible that it’s a pattern that is starting to emerge.

6.2.2 Social benefit

The biggest benefit from this work is a method to evaluate what random number generator that should be used for the type of game you are trying to create. As the gaming industry is leaning more and more towards web-based applications and games and also the mobile platform have exploded into gaming over the recent years the size of the game you are creating becomes more important. You need to have a fast and sufficient RNG method, but also know how it impacts the user in other ways than just performance speed.

This work can also be used in relation to the TAS (Tool-assisted speed run) community. This community creates scripts or recording that can beat games as fast a possible, often manipulating random numbers to their advantage. However with high quality randomness the numbers get harder or in some cases impossible (with todays knowledge and technology) to manipulate, hindering TAS to be made, to in a way creating another challenge for the TAS community.

6.3 Future work

The anomaly between the users’ perceived difference and the empirical study is definitely something that is worth looking into. To get a good understanding of this anomaly you would have to focus more on the interview and especially gather a lot more answers to be able to see patterns more clearly. With the pattern clearer, it might give some sense to why this perceived difference exist, or if it’s not just perceived but an actual difference.
If it proves that there is a clear perceived difference with results that can be reproduced. Work should definitely be spent on testing more kinds of random number generators. There is a lot of work here that could be done since there are a lot of different random number generators. The most interesting category of method to try would be hardware random number generators, as they are the most different from the logical pseudo-random methods. If we try a lot of methods we could also cross examine them with the automatic empirical tests like TestU01 and DIEHARD. And then see if there is a connection between the quality of randomness and the users perceived difference.

The possibilities of procedural content generation seems endless, and we live in a world that is going more and more towards apps and programs being entirely on the internet. Procedurally generated content is definitely of huge interest when it comes to making games. The smaller sized games, due to not having all content pre-made, would make the loading times bearable for devices like mobile connected over the mobile network. And if procedurally generated content is the way to do this, some work will definitely have to be done to see what kind of randomness method would be most feasible for this usage. If the users really can notice a difference, the user perceived difference will have to be taking into account.

Mobile phones also usually have a significant less amount of storage and performance than other platforms where games are played. This makes any kind of optimization very relevant to its market. Java is often used when making games for mobile because of its platform independency, but a lot of games is made with premade frameworks and libraries that some times supplies their own RNG method that of course needs testing.
7 References


Mario AI Championship: Level Generation Track. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(4), pp. 332–347


