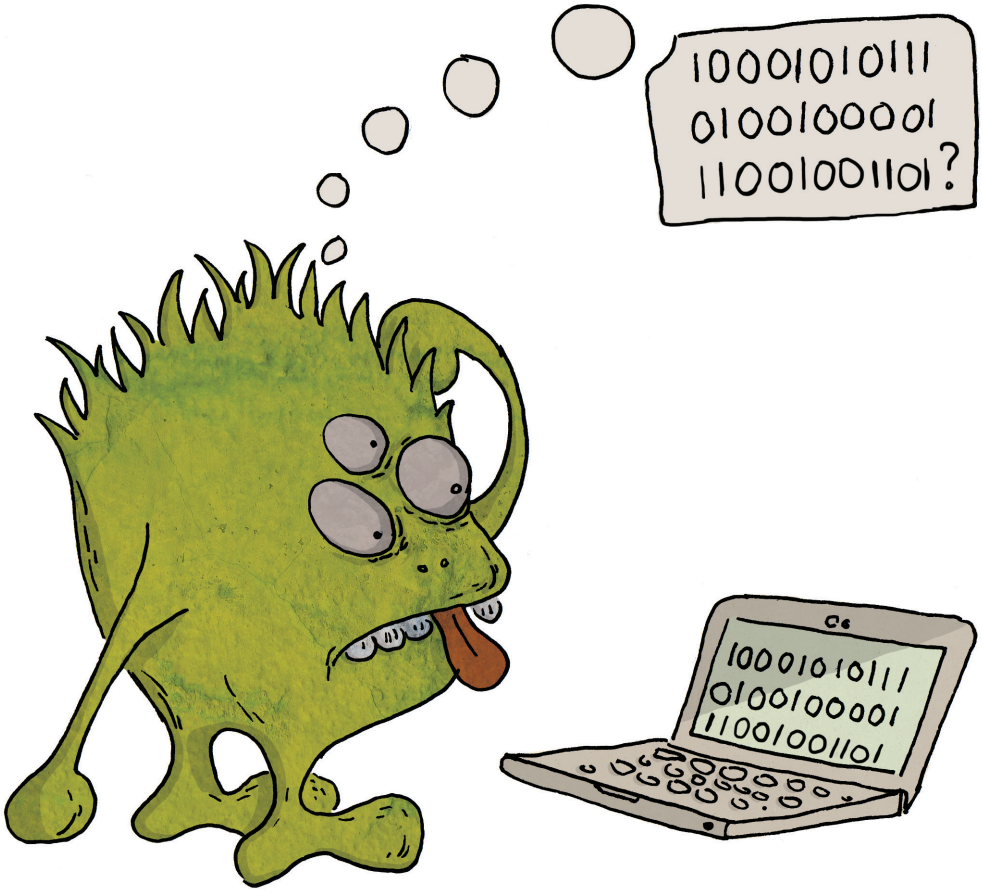
Willekeurige Getallen

Creatie en gebruik van willekeurige getallen door computers



Eindwerk voorgedragen door **Frederik De Paepe** tot het behalen van het diploma Bachelor in Grafische en Digitale Media afstudeerrichting Multimediaproductie.

Interne promotor: **Pieter Vandaele**
Externe promotor: **Walter Van Hove**

Arteveldehogeschool
Bachelor in de Grafische en Digitale Media
Industrieweg 232
9030 Mariakerke

2008 - 2009

Woord vooraf

Het onderwerp van dit werk mag dan op het eerste zicht weinig te maken hebben met de afstudeerrichting Multimedia, toch is het vanwege het feit dat deze afstudeerrichting op meerdere kennisgebieden onderricht geeft dat beide verscheidene raakpunten hebben.
Dit onderwerp, en dan meerbepaald *willekeur*, is iets wat mij al sinds lange tijd weet te boeien. Ik heb deze kans dan ook met beide handen aangegrepen om mij hierin, in het kader van een schoolopdracht, nog verder te verdiepen.
Ik hoop dan ook dat de personen die dit werk lezen door één of meerdere punten geboeid of geïnteresseerd geraken. Hoewel het werk over dit onderwerp een vrij volledig beeld geeft mag er geen twijfel over bestaan dat er nog heel wat te ontdekken valt over de verschillende onderwerpen die hier en daar ter sprake komen. Laat dit werk dan ook een kennismaking zijn met een niet zo voor de hand liggend onderwerp. Hopelijk kan het werk de lezer op sommige momenten doen verbazen en boeien door een uitweiding over één of meerdere van de raakpunten met het dagelijkse leven dat dit onderwerp rijk is.

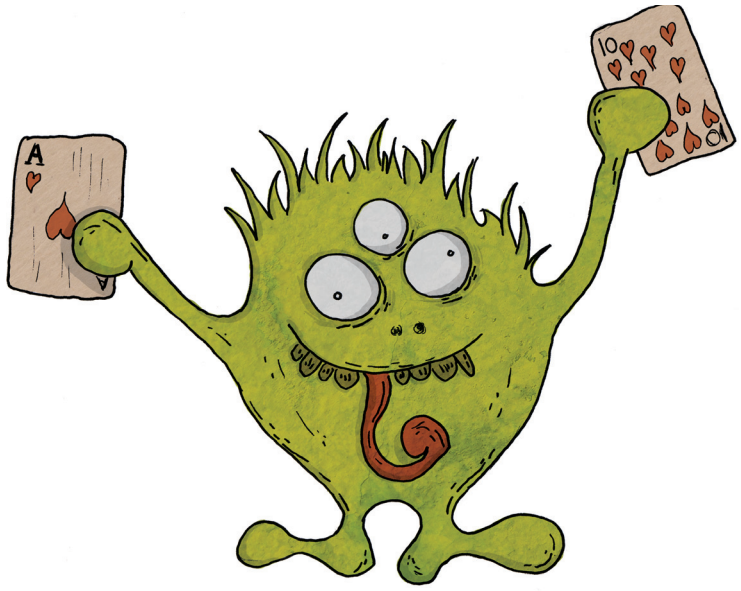
Graag had ik hierbij mijn dank geuit aan mijn interne promotor, Pieter Vandaele, alsook aan mijn externe promotor, Walter Van Hove, die beiden de moed hadden zich in dit onderwerp te verdiepen. Zij hebben mij ten gepaste tijde geduid op onduidelijkheden, fouten of gebreken die tijdens het tot stand komen van dit werk wel eens durfden opduiken.

Verder gaat mijn dank uit naar mijn broer, Niels De Paepe, die bereidt was de nodige illustraties te maken om het werk ook visueel wat interessanter te maken en naar Jan De Wit, die het werk grondig doorgenomen heeft op zoek naar taalfouten.

Inhoudsopgave

1. Inleiding

Zoals de titel zegt is het onderwerp van dit werk er één dat misschien niet zo voor de hand ligt. Misschien gaat het over iets waar weinig mensen weet van hebben, laat staan even stil bij staan, maar toch zal u er versteld staan van in welke mate dit onderwerp in ons dagelijkse leven aan bod komt.



In dit werk worden een aantal verschillende elementen behandeld. Centraal in dit werk staan getallen, meer bepaald *willekeurige getallen*. Er wordt besproken wat willekeur is, hoe we er mee omgaan en welke zoektocht ondernomen moet worden om tot echte willekeur te komen. Niettegenstaande de voor ons misschien voor de hand liggende willekeur, blijkt het in praktijk niet zo gemakkelijk om met een computer willekeurig getallen te creëren. En omdat de computer een steeds grotere rol speelt in ons dagelijkse leven, is het geen slecht idee eens stil te staan bij het hoe en waarom van deze beperking. Uiteraard wordt ook de 'oplossing' hiervoor behandeld, al is er geen sprake van slechts één, maar van meerdere oplossingen.
We zullen zien dat er bij het creëren van willekeurige getallen twee invalshoeken zijn: de ene is die van de *True Random Number Generators* en de andere van de *Pseudo Random Number Generators*. Het is dus mogelijk om *echte* of *pseudo* willekeurige getallen te maken. Maar wat is het nut van pseudo willekeurige getallen, als er ook echte aangemaakt kunnen worden? We zullen zien dat de aanmaak van deze echte willekeurige getallen niet zo voor de hand ligt, en dat in vele gevallen deze pseudo willekeurige getallen volstaan.

Los van dit creëren van willekeurige getallen wordt ook stilgestaan bij de verschillende toepassingen die hiervan gebruik maken. Waar en wanneer wordt een willekeurig getal gebruikt en in welke gedaante komt het voor? Wat is het belang van de willekeurige getallen en moeten ze echt willekeurig zijn, of maakt dit niet zoveel uit? De belangrijkste toepassingsgebieden van willekeurige getallen zijn cryptografie, simulatie, computerspelen, beslissingen nemen en kunst. Maar hoe kunnen we nu bepalen of getallen echt willekeurig zijn? We zullen zien dat er een aantal tests bestaan die de mate van willekeur kunnen bepalen. Deze tests worden uitgevoerd op een reeks willekeurige getallen en wanneer deze tests doorstaan worden, wordt de reeks als willekeurig genoeg beschouwd om gebruikt te worden bij de meest uiteenlopende toepassingen.

In het praktisch deel worden een aantal elementen uit het theoretisch gedeelte nader behandeld. Er worden enkele toepassingen uitgewerkt die elk een ander aspect van het gegeven *willekeurige getallen* verduidelijken. De toepassingen zijn variabel in toegankelijkheid voor een gebruiker die geen weet heeft van het theoretisch gedeelte: voor het ene zal de theoretische achtergrond nodig zijn, voor het andere niet.
De verschillende toepassingen werden uitgewerkt in o.a. Flash, XHTML, CSS en PHP. Twee ervan zijn online te bekijken.

Hoewel het onderwerp van dit eindwerk in eerste instantie erg wiskundig mag lijken, werd toch getracht om de hoeveelheid wiskunde te beperken. Dit wil zeggen dat er slechts enkele formules in het werk voorkomen en dat alles op een zo toegankelijk mogelijke manier uitgelegd wordt. De voorbeeldcode uitgewerkt in PHP kan voor een leek misschien in eerste instantie ietwat onduidelijk zijn, toch werd deze code in de mate van het mogelijke voorzien van de nodige commentaar als hulp ter interpretatie ervan.

2. Willekeur, (on)voorspelbaarheid en willekeurige getallen

2.1 Inleiding

Hoe ervaren we de fenomenen die zich rondom ons openbaren? Wat ervaren we als willekeur en wat als orde? Is het weer ordelijk? Het verkeer in de avondspits? Het menselijk gedrag?
Het is moeilijk om een maatstaf te vinden wanneer het om wanorde en orde gaat. Wat een persoon als orde beschouwt kan voor een ander persoon als wanorde ervaren worden. Maar toch is ieder persoon op een bepaald niveau op zoek naar orde en structuur. Kijk om u heen en u zal zien dat de mens de eigenschap heeft alles te willen structureren. Met deze structuur trachten we ons dagelijkse leven in goede banen te leiden. Kijk maar naar onze dagindeling, onze uurroosters, ons wegennet, de structuur van onze steden en ga zo maar door. Met de komst van de computer is dit structureren alleen maar toegenomen: een enorme hoeveelheid gegevens wordt vandaag de dag door computers verwerkt en bijgehouden in ontelbare databases, en dit alles om onze dagelijkse handelingen efficiënter en dus sneller te laten verlopen.
We zouden kunnen stellen dat de mens niet zo gesteld is op onvoorspelbaarheid. Uiteraard houden we allemaal wel eens van een verrassing en sommige onvoorspelbare factoren zijn niet zo belangrijk als andere. Maar neem nu bijvoorbeeld het weer, hét voorbeeld van een bijna onvoorspelbaar fenomeen. Voorspellingen voor een paar dagen zijn redelijk accuraat, maar meer dan een week vooruit een juiste voorspelling maken is quasi onmogelijk. Toch zouden we het aangenaam vinden indien we ook het weer voor de komende weken, zoniet voor een nog langere periode zouden kennen.
Maar niettegenstaande deze drang naar structuur, orde en voorspelbaarheid blijkt er toch een moment te zijn waarop men weer op zoek gaat naar wanorde en willekeur. Men ontdekt dat deze wanorde, die we zo trachten te vermijden, ook zijn nut kan hebben. Hierbij een voorbeeld: wanneer er een beslissing genomen moet worden, wordt een willekeurige bron als zijnde objectief beschouwd. Zo kan men bijvoorbeeld willekeur gebruiken bij de selectie van een rechter en juryleden in een rechtszaak.
De computer wordt voor steeds meer taken ingezet. Het mag dan ook niet verbazen dat men ook de computer wil gebruiken voor de creatie van deze willekeur. Maar niettegenstaande het feit dat de computer een ontzettend krachtig en veelzijdig apparaat is, blijkt het nog niet zo eenvoudig het willekeur te laten produceren. Over het aanmaken van willekeurige getallen door computers echter later meer. Laat ons eerst eens kijken naar wat willekeur nu eigenlijk is.

2.2 Willekeur

Onder willekeur -de Engelstalige term is *randomness*- verstaan we het onvoorspelbaar karakter van iets. Men stelt dat men niet bij machte is om het gedrag van dat iets af te leiden uit zijn huidige en voorgaande staat. Willekeur is iets dat zich op verschillende niveaus manifesteert, waarmee bedoeld wordt dat de hoeveelheid willekeur niet steeds het zelfde is, of toch niet ogenschijnlijk. Om terug te keren naar het voorbeeld van het weer en de voorspelbaarheid ervan: men is in staat om het weer op korte termijn te voorspellen omdat men in zijn onvoorspelbare karakter, in de chaos, vaste patronen heeft ontdekt. Het is willekeur die zich geordend aan ons voordoet en natuurlijk weten we maar al te goed hoe we met orde om moeten gaan.

Wat betreft deze onvoorspelbare verschijnselen bestaan er een tweetal filosofieën van hoe deze verschijnselen benaderd worden: er is de kwantumtheorie en de chaostheorie, waarvan de laatste een wetenschappelijk-filosofische theorie is die zijn hoogtepunt in populariteit kende omstreeks de jaren ‘90. Het is moeilijk uit te maken welke van deze twee theorieën de (meest) juiste is, maar alles heeft eigenlijk te maken met ons geloof in hoe het hele universum werkt. De belangrijkste vraag hierbij is of het universum deterministisch is of niet, met andere woorden: of alles reeds vast ligt sinds de oerknal. We bekijken deze theorieën iets meer in detail, omdat deze als de bouwstenen beschouwd kunnen worden bij het creëren van willekeurige getallen.

Kwantummechanica is een onderdeel van de theoretische fysica die tracht het universum te beschrijven op het atomisch en subatomisch niveau. Het stelt dat subatomische deeltjes zich willekeurig gedragen in bepaalde omstandigheden. Om de toestand van een deeltje te bepalen moet men zowel beschikken over de plaats van het deeltje alsook zijn impuls (snelheid). De onzekerheidsrelatie stelt echter dat de onzekerheid met de bepaling van de plaats, vermenigvuldigd met de onzekerheid van de bepaling van de impuls nooit kleiner kan zijn dan een bepaalde waarde. Wordt de onzekerheid van de ene factor kleiner, dan wordt de onzekerheid van de andere groter. Dit staat in contrast met de klassieke natuurkunde, die stelt dat we alles in het universum exact kunnen meten, als we maar genoeg metingen kunnen doen en deze ook nauwkeurig genoeg zijn. De onzekerheid ontstaat dus niet door de onnauwkeurigheid van de gebruikte apparatuur, maar is fundamenteel aanwezig. Men kan deze onzekerheid uit de kwantummechanica interpreteren als willekeur zodat de kwantummechanica in wezen stelt dat er een fundamentele willekeur is in de natuur om ons heen.
Er bestaat echter ook kritiek op deze theorie. Overtuigde aanhangers van het determinisme zullen argumenteren dat zelfs op dit subatomisch niveau alles voorbestemd is sinds de oerknal zoals alle andere dingen in het universum. Subatomische gebeurtenissen doen zich inderdaad willekeurig aan ons voor, maar er moet wel gezegd worden dat men nog niet in staat is deze te verklaren. Men gaat ervan uit dat dit willekeurig gedrag is, maar men is nog niet bij machte dit te verklaren[1].

Daar tegenover staat het chaotisch systeem, dat stelt dat kleine veranderingen in de begintoestand kunnen zorgen voor enorme veranderingen in het uiteindelijke gedrag van het systeem. Een bekend voorbeeld dat dit illustreert wordt “Het Vlindereffect” genoemd: dit stelt dat het slaan van de vleugels van een vlinder in bijvoorbeeld Brazilië in staat is een tornado boven Texas te veroorzaken. Niettegenstaande dit quasi-onvoorspelbare karakter spreken we

1 Wikipedia, nl.wikipedia.org/Kwantummechanica/

hier wel over een deterministisch systeem. Met andere woorden, indien we in staat zouden zijn op een gegeven moment de plaats en snelheid van alle moleculen in ons weersysteem te meten, en we hierbij ook in staat zouden zijn met deze onmetelijke hoeveelheid aan gegevens een berekening uit te voeren, dan zouden we inderdaad zulke voorspellingen kunnen maken. In zo'n scenario zou het dus onmogelijk zijn echte onvoorspelbare, willekeurige getallen te creëren. Maar natuurlijk is de mens niet in staat deze metingen en berekeningen uit te voeren, zoals best geïllustreerd kan worden door het feit dat weersvoorspellingen doorgaans bijzonder onnauwkeurig kunnen zijn.
Maar onafhankelijk van welke theorie, filosofie jouw voorkeur heeft, komt het er bij willekeur uiteindelijk op neer dat het niet voorspelbaar is door mensen. Of nu als bron voor een *Random Number Generator* een kwantumsysteem gebruikt wordt, of een deterministisch chaotisch systeem, dit maakt voor het huidige resultaat van gegenereerde willekeurig getallen geen verschil. Men is in staat om reeksen willekeurige getallen te creëren, die gebruikt kunnen worden bij de meest uiteenlopende toepassingen, door zowel gebruik te maken van kwantum- als van chaosfenomenen.

2.3 Willekeurige getallen

Wat kunnen we nu beschouwen als willekeurige getallen? Kan er sprake zijn van één willekeurig getal, is bijvoorbeeld 1 een willekeurig getal? Donald Knuth[2] stelt dat men niet kan spreken van één willekeurig nummer, maar wel van een reeks van onafhankelijke willekeurige getallen. Hierna volgt dan ook een uiteenzetting over wat gezien wordt als een reeks willekeurig getallen.

Laat ons een getallenreeks opbouwen op een manier waarbij we weten dat het gaat over echte willekeurige getallen: we maken gebruik van een dobbelsteen. We gooien de dobbelsteen en het getal 3 komt bovenaan te liggen. Dit is ons eerste getal in de getallenreeks. We nemen nu de dobbelsteen opnieuw op en gooien nogmaals: ditmaal komt het getal 5 bovenaan te liggen. De reeks bestaat momenteel uit 3 en 5. Dit proces kunnen we blijven herhalen tot wanneer we vinden dat de reeks getallen lang genoeg is, bijvoorbeeld: 3, 5, 6, 2, 1, 1, 4, ... We gaan nu terug naar onze eerste worp. Hierbij hadden we de kans om gelijk welk getal van 1 tot 6 als resultaat te krijgen, overeenkomstig met de zes zijden van de dobbelsteen. Het feit dat we een drie als resultaat kregen staat niet in de weg dat we bij de tweede worp niet ook een drie als resultaat kunnen krijgen. Ook bij de tweede worp blijft de kans op 1 van elk van de 6 cijfers even groot. Dit zal ook zo zijn bij de derde worp, de vierde, ...
Om te kunnen spreken over een reeks willekeurige getallen mag bij de aanmaak van een volgend getal geen invloed uitgeoefend worden door de eventuele voorgaande getallen. Het getal moet met andere woorden onafhankelijk zijn van de andere getallen in de reeks. Dus, wanneer we ons in onze voorbeeldreeks bij het getal 2 bevinden, dan mag de creatie van het getal 1 dat in deze reeks volgt, op geen enkele manier gebaseerd zijn op het getal 2 en alle daaraan vooraangaande getallen. Omdat er niet zoiets bestaat als een 'dobbelsteen met een geheugen en geweten' hoeven we ons hierom in dit geval, namelijk bij het gebruik van een dobbelsteen, geen zorgen te maken.

Hoewel dit voorbeeld van de dobbelsteen duidelijk aantoont wat bedoeld wordt met een reeks willekeurige getallen, zal in de praktijk bij aanmaak van willekeurige getallen nauwelijks

2 DONALD KNUTH, *The Art of Computer Programming - Semi numerical Algorithm. Vol 2.* Chapter 3 Random Numbers pg1-184, 1997.

gewerkt worden met een zesdelig stelsel, maar met een binair stelsel. Dit kan je vergelijken met het gooien van een muntstuk in de lucht: na het opvangen ervan wordt gekeken welke zijde bovenaan ligt. Bij afspraak wordt gesteld dat bijvoorbeeld kruis gelijk is aan 1 en munt gelijk aan 0. Het binair aanmaken van een reeks willekeurige getallen heeft natuurlijk alles te maken met de aanmaak ervan door computers, die slechts met 0 en 1 overweg kunnen. Indien we alsnog gebruik willen maken van een ander stelsel, zoals bijvoorbeeld het decimale, dan kunnen we ons eenvoudigweg baseren op de getallen die binair aangemaakt werden, we hoeven ze enkel maar om te zetten. Omdat we hierbij vertrokken zijn van een reeks volledig willekeurige binaire getallen zal ook het decimale resultaat even willekeurig zijn.

Een reeks willekeurige getallen is dus een getallenreeks waarbij de getallen op geen enkele manier één of andere volgorde van verschijnen hebben en nooit voldoen aan gelijk welk patroon dan ook.

2.4 Geschiedenis van het creëren van willekeurige getallen

Willekeurige en onvoorspelbare verschijnselen zijn er natuurlijk altijd al geweest, maar vanaf wanneer en waarom had men nood aan propere willekeurige getallen? Propere willekeurige getallen zijn getallen die afgelezen en gebruikt kunnen worden, en niet louter vervat zitten in een onvoorspelbaar verschijnsel zoals het weer.
Het mag duidelijk zijn dat de eerste creatie van een willekeurige getallenreeks niet gebeurde met behulp van een computer, maar met een alledaagse dobbelsteen of een muntworp.
De interesse naar willekeurige getallen ontstond bij de studie van verkeersopstoppingen en de studie naar telefoonaanvragen in de telefooncentrale. Bij deze studies trachtte men een model te bouwen aan de hand van simulaties. Op deze manier kon men aan de hand van de hoeveelheid 'input', in dit voorbeeld het aantal bellers, een studie maken die tot een manier kon leiden om efficiënter om te gaan met de input, om zo meer mensen op een zelfde moment een betere service te geven. Bij deze simulaties stond deze 'input' natuurlijk gelijk aan de mens en zijn gedrag. Maar omdat het gedrag van de mens niet uniform is (zo is de situatie waarin iedere mens op exact het zelfde moment zou bellen quasi onbestaande), is het nodig om een 'menselijke factor' toe te voegen, en dat is waarom willekeurige getallen bij deze studie voor het eerst aan bod kwamen.

In eerste instantie werden voor de creatie van deze getallenreeksen manuele methodes gebruikt, zoals het werpen van een munt (kruis of munt), dobbelsteen gooien, kaarten schudden en het draaien van het roulettewiel. Deze methodes zijn erg betrouwbaar wat willekeur betreft, maar ze zijn uiteraard erg traag voor algemeen gebruik en de getallenreeksen konden niet gereproduceerd worden. Deze methodes zijn tevens erg arbeidsintensief, wat ze ook erg duur maakt en ze zijn niet bruikbaar in een geautomatiseerde omgeving (bijvoorbeeld bij computers).

In 1927 publiceerde Cambridge University Press het boek *Random Sampling Numbers*, samengesteld door de statisticus Leonard Henry Caleb Tippett. Het bevatte 41.600 getallen die willekeurig gekozen waren uit volkstellingrapporten. In 1939 werd een tabel met 100.000 getallen gepubliceerd door Kendall en Babington-Smith. Deze tabel was de eerste die met behulp van een machine geproduceerd werd. Eén van de laatste van dit soort projecten is waarschijnlijk de publicatie door de RAND Corporation. Zij publiceerden hun "Een miljoen willekeurige getallen met 100.000 normale afgeleiden. Deze RAND tabel werd gemaakt door een elektrische

machine gebaseerd op een roulettewiel en was een echte doorbraak, omdat nog nooit zo'n lange en aandachtig voorbereide tabel was aangemaakt. Toch waren het gebrek aan snelheid, eigen aan een tabel, en het risico de tabel 'uit te putten' twee grote nadelen aan het gebruik van de tabellenmethodes. Deze tabellen konden gebruikt worden voor het creëren van simulaties, maar het feit dat deze tabellen openlijk gepubliceerd werden zorgde ervoor dat ze niet gebruikt konden worden als sleutels bij encryptie.

Bij de introductie van de computer omstreeks 1940 nam de zoektocht naar het genereren van willekeurige getallen alsmaar toe. Er werd onderzoek gedaan naar het ontwerp van elektronische apparaten en apparaten die zich baseerden op gegevens van natuurlijke fenomenen, zoals bijvoorbeeld radioactieve straling.

Willekeurige signalen uit elektronische ruis werden gebruikt om willekeurige getallen te genereren door een computer wanneer men deze nodig had. Aan de hand van dit systeem werd onder andere ERNIE (Electronic Random Number Indicator Equipment) gebouwd, een apparaat dat door de British Post Office gebruikt werd om de winnaars van de *Premium Savings Bonds lottery* uit te loten. Maar toch was het moeilijk om de willekeur van de getallen te blijven behouden en liepen de kosten van het apparaat hoog op omwille van het feit dat deze het tempo van geautomatiseerde apparaten zoals computers bij moest kunnen houden. De instabiliteit van de elektronische circuits en het gebrek aan de mogelijkheid de getallenreeks te hergenereren zorgden voor bijkomende nadelen.

Een alternatief voor het genereren van willekeurige getallen wanneer men ze direct nodig had was het opslaan van tabellen met willekeurige getallen op magnetische cassettes. Deze konden dan, wanneer nodig, door de computer geraadpleegd worden. Uiteraard was deze laatste methode ook erg traag en duur, wat leidde tot de ontwikkeling van eenvoudige en efficiënte rekenkundige *Pseudorandom Number Generators*.

Met de computers van vandaag is het uiteraard geen enkel probleem om ontzettend lange reeksen willekeurige getallen te genereren. Afhankelijk van de applicatie waarin deze nummers zullen gebruikt worden, wordt al dan niet gebruik gemaakt van een *True Random Number Generator*.

In de huidige software die gebruikt wordt op elk platform is de aanwezigheid van een random-functie bijna niet meer weg te denken. Deze functie, die ofwel de *Random Number Generator* van het besturingsysteem aanspreekt, of een eigen *Pseudorandom Number Generator* binnen de applicatie aanspreekt, is terug te vinden in tal van softwarepakketten. Uiteraard in 'cijfermatige'-softwarepakketten, zoals Excel, Maple, MatLab, ... maar deze functie is ook terug te vinden in elke programmeertaal, gaande van de serieuze programmeertalen zoals C++, C#, Java, ... tot scriptingtalen zoals PHP, JavaScript, Perl, Python, ... Maar ook in minder voor de hand liggende programma's, zoals Flash, Director en Photoshop, worden random-functies gebruikt.

Doordat er steeds meer 'gevoelige' informatie via informaticasystemen uitgewisseld wordt, is de vraag naar beveiliging van deze data erg toegenomen. Bij deze beveiliging worden willekeurige getallen als unieke sleutels gebruikt.
Eén van de grootste gebruikers van willekeurige getallen is de computerspelindustrie. De populariteit van computerspelletjes, in alle mogelijke vormen, is alleen maar toegenomen, maar daarmee ook de verwachtingen van de gebruiker. De eerste spelletjes waren zeer eenvoudig en het was gemakkelijk om als mens de computer te verslaan, omdat men al snel door had hoe de computer dacht, en men hem een stap voor kon zijn. De spelen bezaten nog geen artificiële intelligentie (A.I.), maar al gauw trachtten de makers een meer menselijk karakter aan de personages (of andere) te geven, en dit kon enkel door gebruik te maken van onvoorspelbaarheden, wat ons weer brengt tot de willekeurige getallen. Aan de hand van deze getallen worden de personages in het computerspel namelijk samengesteld, hun reacties en bewegingen krijgen zo een menselijk, onvoorspelbaar karakter.

Willekeurige getallen worden vandaag de dag op verschillende manieren aangeleverd en gebruikt. De manier waarop is natuurlijk afhankelijk van de toepassingen ervan. Fabrikanten van computerhardware hebben veel tijd en geld gespendeerd aan onderzoeken naar de juiste methoden om willekeurige getallen te genereren. In een *white paper*[3] met de titel *The Intel® Random Number Generator* staat een volledige test-analyse van de hardware *Random Number Generator* die ontwikkeld werd door Intel®. Deze *Random Number Generator* stelt het besturingssysteem in staat willekeurige getallen rechtstreeks door de hardware te laten genereren in plaats van zelf de nodige functies uit te laten voeren.
Zoals gezegd, alles is afhankelijk van de toepassing van de willekeurige getallen. De kans is heel groot dat bedrijven, voornamelijk financiële instellingen dan, waarbij encryptie en veiligheid van 'levensbelang' zijn, hun eigen server hebben die voor hen alleen de nodige getallen genereert. Maar er bestaan ook USB-sticks die een *Random Number Generator* bevatten. De prijs varieert mee met het aantal getallen en de snelheid waarmee deze gegenereerd kunnen worden. Laat ons zeggen dat deze eerder gemaakt zijn voor het 'thuisgebruik'. Er zijn ook nog de online diensten, die in een volgend deel besproken zullen worden. Deze leveren gratis getallen indien het gaat om persoonlijk gebruik en gebruik op kleine schaal, of vragen al dan niet een bijdrage wanneer er in grote hoeveelheid getallen aangeleverd moeten worden.

Het staat zonder meer vast dat het gebruik van willekeurige getallen alleen maar kan toenemen. Hiertoe kan de evolutie van het steeds meer willen digitaliseren (en dus ook te encrypteren) alsook het toenemende aantal intelligente computerspelletjes dan ook alleen maar bijdragen. En wie weet welke toepassingen waarbij willekeurige getallen gebruikt worden er in de toekomst nog zullen bijkomen?

3 BENJAMIN JUN, PAUL KOCHER, *The Intel® Random Number Generator,* White Paper, Cryptography Research Inc., 22 April 1999.

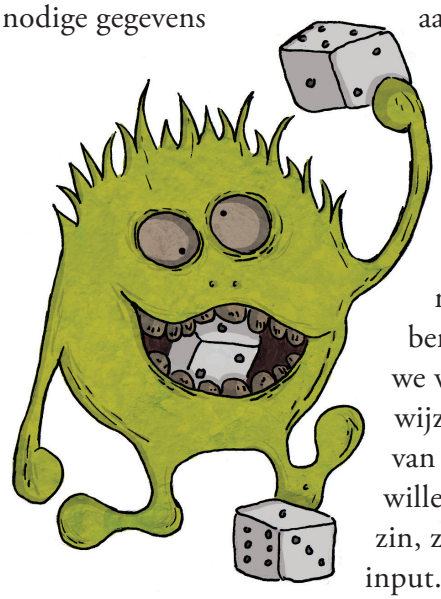
3. Pseudorandom Number Generators versus True Random Number Generators

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."
– John von Neumann

3.1 Inleiding

We behandelen in dit deel de vraag: wat is een *(Random) Number Generator*? In het Nederlands spreken we van een getallengenerator, maar in overeenstemming met de literatuur blijven we de Engelstalige term hanteren, afgekort met RNG. Verder gaan we ook in op het verschil tussen *True Random Number Generators* (TRNGs) en *PseudoRandom Number Generators* (PRNGs).

Onder een generator verstaan we een programma dat bepaalde gegevens of andere programma's aanmaakt. Dit programma kan verschillende vormen aannemen, maar telkens zal het hierbij de nodige gegevens aanmaken: namelijk willekeurige getallen.



Bij de eerste RNGs was nog geen sprake van een hardware- of softwarematig programma: het was een persoon die met dobbelstenen gooide of die aan het roulettewiel draaide. Hoewel het een goede oplossing was, was het uiteraard niet efficiënt in gebruik waardoor verder gezocht werd naar snellere alternatieven. Zo ontwikkelde men hardware die met behulp van elektronische circuits het zelfde resultaat kon bereiken, maar dan een stuk sneller. Onder een RNG verstaan we vandaag de dag een apparaat dat op een geautomatiseerde wijze willekeurige getallen kan genereren, weliswaar met behulp van een onvoorspelbare input. Uiteraard is een computer die willekeurige getallen genereert ook een hardware RNG in strikte zin, zolang er maar gebruik gemaakt wordt van die onvoorspelbare input. Is dit niet het geval, en worden de getallen softwarematig gegeneerd op basis van een voorspelbare input, dan is het resultaat ook niet meer onvoorspelbaar of willekeurig. In dit geval spreken we dan ook over een PRNG.

Maar hoe ziet het resultaat van deze generators er nu uit? Afhankelijk van de gebruikte RNG wordt een korte of lange reeks willekeurige getallen aangemaakt. Deze reeks kan bijgehouden worden in een bestand, een database, ... Maar in de meeste gevallen wordt een willekeurig

getal (of reeks) op het moment zelf aangemaakt. Op deze manier zal het getal meteen door de applicatie gebruikt worden die het getal opgevraagd heeft. Het is hierbij dan ook onnodig een getallenreeks te genereren die een stuk langer is dan eigenlijk nodig. Daarom stellen verschillende programmeertalen verschillende functies en methodes ter beschikking waarbij men parameters kan meegeven die de grootte van deze reeks al dan niet beperken. Men kan deze reeks zodanig klein kiezen dat er slechts sprake is van één getal.

Een praktisch voorbeeld uit de scriptingtaal PHP. De functie `rand()` geeft ons een willekeurige getal bestaande uit 9 tot 10 karakters. We kunnen het resultaat ook beperken. Stel dat we voor een bepaalde toepassing slechts vier mogelijkheden hebben waarvan we er één willekeurig uit willen kiezen, dan hebben we een willekeurig getal tussen 1 en 4 nodig. In dit geval kunnen we twee parameters meegeven aan de functie, namelijk een minimum- en een maximum waarde: `rand(int min, int max)`. Dit levert een getal op gelijk aan deze waarden of één van de waarden tussenin. Willen we een reeks creëren van enkel deze getallen, dan is het een kwestie van een eenvoudige lus aan te maken. Onderstaande voorbeelden uitgewerkt in PHP verduidelijken het gebruik van deze functie.

```
<?php

echo rand(); // Uitkomst bv: 821209699

// Enkel de getallen 4, 5, 6, 7 en 8 mogen voorkomen.

echo rand(4,8); // Uitkomst bv: 6.

/*
Voorbeeld van een reeks bestaande uit 5 karakters,
elk  karakter varieert van 0 tot en met 3.
*/

for ($i = 0; $i < 5; $i++)
{
     echo rand(0,3);
}

// Uitkomst bijvoorbeeld: 20332

?>
```

Op het niveau van de programmeertaal is alles uiteraard zo gebruiksvriendelijk mogelijk gemaakt. De programmeur wordt niet gehinderd door de technische aspecten van hoe een willekeurig getal nu echt aangemaakt wordt en welk algoritme er gebruikt wordt, maar de programmeur hoeft dit in vele gevallen ook niet te weten. De aanmaak van deze willekeurige getallen, hoogstwaarschijnlijk gegenereerd door een PRNG, zal in vele gevallen volstaan. Als deze standaard methode niet zou volstaan is het ook mogelijk een eigen methode te schrijven, met een formule naar keuze. En als ook dit nog niet zou volstaan is het ook mogelijk de interne hardwarematige RNG te gebruiken, als die aanwezig is uiteraard. Hardware RNGs

zijn vandaag de dag erg betrouwbaar[1]. Ze kunnen zich baseren op tal van elementen die als onvoorspelbare input kunnen dienen: netwerkactiviteit, schrijf- en leessnelheid van een harde schijf, de temperatuur van verschillende hardware-elementen, ... Ook gebruikersinput kan gebruikt worden, zoals intervallen tussen toetsaanslagen of de coördinaten van muisbewegingen. Met deze laatste methodes moet echter voorzichtig omgesprongen worden: het kan zijn dat de gebruikersinput bijgehouden wordt in een buffer en wanneer men deze kan benaderen kan men ook in staat zijn de uiteindelijk gegenereerde getallen te voorspellen.
Maar zoals we in volgende paragraaf zullen zien, zijn er ook methodes die gebruik maken van externe bronnen. Dit zijn voornamelijk internetdiensten, waarbij de gebruiker het gewenste aantal willekeurige getallen kan opvragen, maar indien het nodige kapitaal aanwezig is kan men zich ook van de nodige externe hardware voorzien (geigerteller, radiogolfontvanger, ...). Het voordeel van de internetdiensten is dat zij de nodige hardware hebben om echte willekeurige getallen te genereren. Het nadeel hierbij is wel dat men afhankelijk is van het internet bij elke real-time creatie, wat voor veel beveiligingstoepassingen ontoelaatbaar is.

3.2 True Random Number Generators

Een overzicht van een aantal diensten die TRN's aanbieden:

3.2.1 HotBits (radioactief materiaal)

WWW.FOURMILAB.CH

HotBits is een online dienst die echte willekeurige getallen aanbiedt op basis van het radioactief verval van elementen[2]. Als voorbeeld gebruiken we het atoom Cæsium-137. De halveringstijd van Cæsium-137 bedraagt 30,17 jaar, wat wil zeggen dat van een bepaalde hoeveelheid atomen er na 30,17 jaar nog slechts de helft van over zijn: de andere helft zal tot Barium-137 vervallen zijn. Stel dat we 100 miljoen Cæsium-137 atomen hebben, dan kunnen we met zekerheid zeggen dat er na 30,17 jaar nog slechts 50 miljoen over zijn en na nog eens 30,17 jaar slechts 25 miljoen, enz.... Gedurende de eerste periode zijn dus 50 miljoen atomen vervallen, alleen is het onmogelijk te voorspellen wanneer één bepaald atoom zal vervallen tot Barium. Je kan dus stellen dat er 50 % kans bestaat dat het bij één bepaald atoom zal gebeuren, maar dat is dan ook alles. Deze onvoorspelbaarheid wordt gebruikt om echte willekeurige getallen te genereren. Maar hoe zetten we nu dit verval van atomen om in willekeurige getallen, in enen en nullen?
Een geigerteller meet bij een hoeveelheid radioactief materiaal wanneer een deeltje uitgezonden wordt, dit is met andere woorden het moment dat, zoals in het voorbeeld, een atoom van Cæsium-137 vervalt tot Barium-137. Omdat het moment dat een atoom vervalt willekeurig is, is ook het interval ertussen willekeurig. Aldus wordt telkens het interval gemeten en krijgt elk interval in verhouding tot zijn lengte een 1 of een 0 toegewezen. Is het volgende interval langer, dan krijgt het een 1, is het korter, dan krijgt het een 0. Komt het voor dat het interval even lang is, dan wordt hier geen rekening met gehouden en gaat men verder naar een nieuwe meting. Om 'vervuiling' of beïnvloeding van het signaal tegen te gaan wordt gebruik gemaakt van een dedicated server, die rechtstreeks verbonden is met de HotBits hardware. Op die manier wordt

1 BENJAMIN JUN, PAUL KOCHER, *The Intel® Random Number Generator*, White Paper, Cryptography Research Inc., 22 April 1999.

2 HotBits, http://www.fourmilab.ch/hotbits/

het verzoek dat ontvangen werd door de web server doorgestuurd naar de dedicated server. De reden hiervoor is dat de dedicated server op geen enkele manier beïnvloed mag worden. Zo zou een ander CPU-proces, zoals bijvoorbeeld een eenvoudige screensaver, er voor kunnen zorgen dat er niet-willekeurige patronen in de reeks bits opduiken.
Om te kunnen voldoen aan verschillende verzoeken in een korte tijdsperiode (al dan niet op het zelfde moment) wordt er een buffer opgebouwd van zo'n 67 miljoen willekeurige bits (8192 Kb). Deze buffer wordt in de achtergrond terug aangevuld tussen de verschillende verzoeken door. Deze buffer stelt de makers van HotBits ook in staat om onderhoud door te voeren aan de dedicated server en de HotBits hardware.
Opmerkelijk is ook nog dat de volledige broncode van de HotBits generator open source is, d.w.z. dat ze dus gratis te downloaden is van de website.

3.2.2 RANDOM.ORG (atmosferische ruis)

WWW.RANDOM.ORG

RANDOM.ORG, een andere online dienst, biedt echte willekeurige getallen aan die gegenereerd zijn door gebruik te maken van atmosferische ruis. De website werd opgericht door Mads Haahr in 1998. Op de website kan men het gewenste aantal willekeurige getallen ingeven, dat dan voor u gegenereerd wordt. Voor persoonlijk, kleinschalig gebruik wordt deze dienst gratis aangeboden. Wanneer men de dienst regelmatig gebruikt en hierbij een groot aantal getallen in een keer nodig heeft kan men een betalende 'premium-account' aanmaken. Deze accounts krijgen voorrang bij het verkrijgen van willekeurige getallen wanneer er op een bepaald moment teveel vraag zou zijn en er niet genoeg willekeurige input voorhanden is (in dit geval dus atmosferische ruis).
Een technische uiteenzetting over de aanmaak van de getallen is niet terug te vinden op de website, wel enkele interessante uiteenzettingen over willekeurige getallen in het algemeen, alsook een pagina met testimonials waarop per categorie de reacties terug te vinden zijn van tevreden gebruikers.

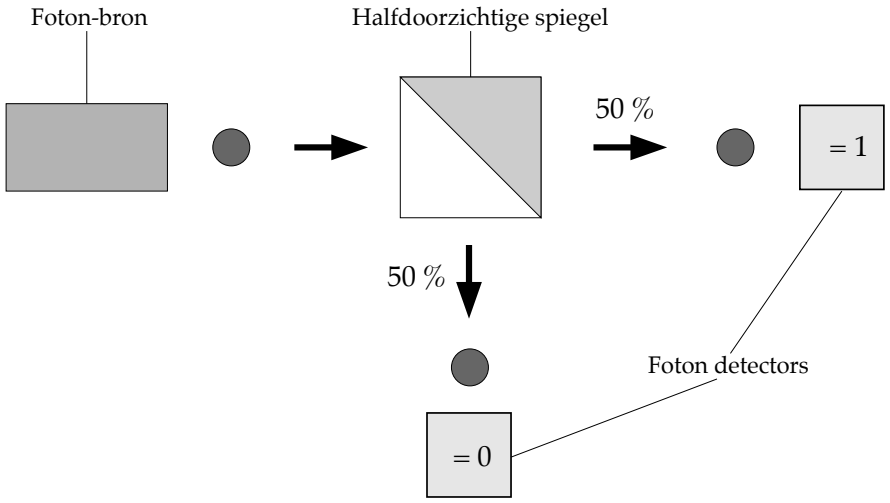
3.2.3 LavaRnd (fotoelektrische ruis)

WWW.LAVARND.ORG

LavaRnd is een goed voorbeeld van hoe creatief men kan zijn bij het zoeken naar een bron voor onvoorspelbare input. De lavarand generator werd ontwikkeld door Silicon Graphics en maakte gebruik van foto's van Lava-lampen als bron voor willekeurig materiaal. Lavarand wordt momenteel niet meer gebruikt, maar het idee werd wel verder uitgewerkt door één van de uitvinders ervan. In plaats van gebruik te maken van foto's van Lava-lampen, wordt nu een (digitale) foto genomen van een perfect donkere ruimte. De ruis die opgevangen wordt door de CCD van de camera wordt als bron van willekeur gebruikt. De gemaakte afbeelding bestaat uit 19.200 pixels, waarbij de lichtheid van de pixels gebruikt worden als random bits. Deze waarden kunnen echter niet zomaar afgelezen worden. In de afbeelding zitten namelijk nog niet-willekeurige patronen, die er door een aantal bewerkingen uitgefilterd kunnen worden.

3.2.4 idQuantique (kwantum bron)

idQuantique is een Zwitsers bedrijf dat verschillende oplossingen biedt voor het aanmaken van TRNs. Het verkoopt hardware elementen die aan de computer aangesloten kunnen worden, zoals een PCI-kaart of een USB-module. Afhankelijk van de gekozen hardware kunnen getallen aangeleverd worden met een snelheid tot 16Mbit/sec. Op de website www.randomnumbers.info kunnen getallen opgevraagd worden die gegenereerd werden door de Quantis PCI-kaart.
De Quantis toepassingen maken gebruik van volgend principe[3] : er wordt gebruik gemaakt van kleine deeltjes licht, fotonen, die in bepaalde situaties een onvoorspelbaar karakter vertonen. Zo'n situatie is er één waarbij de fotonen door een halfdoorzichtige spiegel getransporteerd worden. Het foton zal ofwel op de spiegel afkaatsen, of zal door de spiegel gaan. Afhankelijk hiervan zal het verkregen bit in de reeks een nul of een één zijn.



Figuur 1: Illustratie van de werking van Quantis

3.3 Pseudorandom Number Generators

3.3.1 Wat is een Pseudorandom Number Generator?

Een PRNG is een RNG die *ogenschijnlijke* willekeurige getallen genereert. Het eigenlijke genereren van de getallen gebeurt met behulp van een wiskundige formule, een algoritme, dat door een computer uitgevoerd wordt. Omdat een computer slechts uitvoert wat hem opgedragen wordt, in dit geval de berekening van een bepaalde formule, is het niet mogelijk op deze manier tot echte willekeurige getallen te komen. Vandaar het woord *pseudo*. Het lijkt alsof het echte willekeurige getallen zijn, maar het zijn er geen.
Het gebruik van PRNGs heeft ook zo zijn voordelen, niettegenstaande het feit dat het niet om echte willekeurige getallen gaat. Zo is het mogelijk om aan de hand van dezelfde formule en de zelfde input de getallenreeks opnieuw te genereren. Dit lijkt in eerste instantie misschien niet wat we willen bereiken, maar toch zijn er situaties waarin dit bruikbaar kan zijn. Een voorbeeld

3 IDQUANTIQUE, *Random Numbers Generation using quantum physics,* White Paper, http://www.idquantique.com/products/files/quantis-whitepaper.pdf, 2004.

hiervan is bij het spelen van een computerspel over een netwerk: de situatie kan zich voordoen dat men op de verschillende computers nood heeft aan een reeks willekeurige getallen. Deze getallen worden dan door beide computers op het zelfde moment gegenereerd, zonder daarbij het netwerk onnodig te belasten door het uitwisselen van een reeks willekeurige getallen van de ene naar de andere computer. Een ander voorbeeld is het testen van applicaties die gebruik maken van een reeks willekeurige getallen. Omdat dezelfde reeks getallen telkens terugkeert kan men op deze manier gemakkelijk eventuele fouten opsporen: de uitkomst gebaseerd op de willekeurige input is telkens het zelfde, zodat men zich kan concentreren op de overige code in de applicatie.
De dag van vandaag is het mogelijk om, met de beste computers nu beschikbaar, willekeurige getallenreeksen te creëren die zich gegarandeerd niet zullen herhalen in een tijdsperiode die miljoenen keer langer is dan de leeftijd van het heelal.

3.3.2 Pi als willekeurige getallenreeks

Het getal π is de verhouding tussen de diameter en de omtrek van een cirkel. Het is een irrationaal getal, wat wil zeggen dat het niet als een verhouding van twee gehele getallen te schrijven is en dat in de decimale voorstelling geen zich herhalend patroon voorkomt. Het aantal cijfers achter de komma is oneindig lang.
Uit deze getallenreeks kan een kleinere getallenreeks gekozen worden die dienst doet als bron van willekeur. Onderzoek[4] wees uit dat de getallenreeksen bestaande uit cijfers van π een aanvaardbare bron voor willekeur zijn, maar dat deze niet zo efficiënt aangemaakt kunnen worden als sommige PRNGs doen. Bij het onderzoek werd wel duidelijk vermeld dat niets erop wijst dat er een patroon in π aanwezig is. Na het uitvoeren van een aantal tests doorstond de gebruikte getallenreeks van π deze meestal met glans. Toch scoorden sommige andere PRNGs af en toe nog iets beter.
Hoewel π dus gebruikt kan worden als bron van willekeur, zal dit in vele gevallen niet volstaan. Het feit dat het een getallenreeks is die door iedereen berekend kan worden zorgt er bijvoorbeeld al voor dat ze niet gebruikt kan worden bij toepassingen waar veiligheid van groot belang is.

3.3.3 Lineaire Congruentiemethode van Lehmer

Dit is een van de oudste en tevens meest gekende PRNGs. De methode is eenvoudig toe te passen en is snel in gebruik. Toch wordt afgeraden deze methode te gebruiken in gevallen waarbij 'kwalitatieve' willekeurige getallen vereist zijn, zoals bijvoorbeeld bij encryptie.

Het algoritme:

Kies: $m > 0$
$0 < a < m$
$0 \leq c < m$
$0 \leq X_0 < m$

$X_{n+1} = (aX_n + c) \bmod m$

4 SCIENCE DAILY, *Pi Seems A Good Random Number Generator - But Not Always The Best*, www.sciencedaily.com, 27 April 2005.

We bekijken even in detail hoe dit algoritme te werk gaat:
De waarde X_0 is de beginwaarde, ook wel de *seed* genoemd, voor X_n. Deze waarde wordt slechts eenmaal gebruikt. De uitkomst van de berekening met deze waarde wordt dan op zijn beurt gebruikt als *seed* voor het volgende getal. Dit herhalingsproces wordt een *iteratie* genoemd en kan zoveel keer uitgevoerd worden als men wenst. Met de mod-operator wordt de rest bij een gehele deling van een bepaald getal berekend. Bijvoorbeeld: 8 mod 3 = 2, want 2 * 3 + 2 = 8.

We illustreren het algoritme met volgende waarden: $a = 7$, $c = 0$, $m = 32$ en $X_0 = 1$.
Bij een eerste iteratie krijgen we:

$$X_1 = (7 * 1 + 0) \bmod 32 = 7$$

Bij volgende iteraties krijgen we:

$$X_2 = (7 * 7 + 0) \bmod 32 = 17$$
$$X_3 = (7 * 17 + 0) \bmod 32 = 23, \text{ enz...}$$

Na x-aantal iteraties krijgen we volgende reeks: 7, 17, 23, 1, 7, 17, 23, 1, 7, 17, 23 ... We zien duidelijk een patroon dat herhaald wordt. De lengte van dit patroon noemen we de *periode*. Wanneer we de waarde voor *a* nu veranderen naar $a = 5$, dan krijgen we volgende reeks: 5, 25, 29, 17, 21, 9, 13, 1, 5, ... Hierbij is de periode verdubbeld van 4 naar 8. Het mag dan ook duidelijk zijn dat we een zo'n groot mogelijke periode wensen te verkrijgen, die alle getallen genereert tussen 0 en *m*.

Het selecteren van de waarden voor *a*, *c* en *m* is van cruciaal belang voor de creatie van een goede willekeurige uitkomst. Zo nemen we *m* best zo groot mogelijk, zodat er een reeks met willekeurige getallen aangemaakt kan worden die lang genoeg is. Hierbij wordt meestal gekozen voor een getal waarmee een computer gemakkelijk overweg kan, namelijk 2^{32} of 2^{64}. De waarden *a* en *c* bepalen of de generator een lange periode zal hebben en of deze vlugger herhaald zal worden. Experimenten hebben uitgewezen dat slechts een paar waarden voldoen om een resultaat op te leveren dat willekeurig genoeg is. Zo'n waarde voor *a* is bijvoorbeeld $a = 7^5 =$ 16.807, die gevonden werd door IBM en gebruikt werd in hun IMB 360–reeks van computers. Het lineaire congruentie algoritme is in staat een reeks van willekeurige getallen te genereren die niet te onderscheiden is van een reeks echt willekeurige getallen, maar het blijft natuurlijk een algoritme, waarbij enkel de waarde voor X_0 bij het begin willekeurig gekozen wordt. Wanneer een persoon weet dat dit algoritme gebruikt wordt, kan deze met de kennis van slechts een paar parameters die gebruikt worden een zelfde reeks getallen aanmaken. Wanneer deze persoon dan één waarde kent kan hij zonder problemen de voorgaande alsook volgende getallen berekenen. Om er toch voor te zorgen dat er een onvoorspelbaar element als beginwaarde gekozen wordt bij elke generatie van een getal, kan het huidige tijdstip als beginwaarde genomen worden. Hiervoor wordt een *UNIX-timestamp* gebruikt. Dit is het aantal seconden sinds 1 januari 1970. Op deze manier zal de reeks bij elke generatie anders zijn.

In het praktisch deel werden een paar voorbeelden uitgewerkt die het gedrag van dit algoritme illustreren.

3.3.4 Blum Blum Shub

Deze PRNG werd genoemd naar zijn uitvinders: Lenore Blum, Manuel Blum en Michael Shub, die de PRNG uitvonden in 1986. Deze PRNG wordt algemeen beschouwd als één van de meest veilige PRNGs en kan ook toegepast worden bij cryptografie.

Het algoritme:

$$X_{n+1} = (X_n)^2 \bmod m$$

Hierbij is m het product van twee priemgetallen, p en q, die elk een rest hebben van 3 wanneer ze gedeeld worden door 4. Dit kan genoteerd worden als

$$m = p * q$$

met $p \bmod 4 = q \bmod 4 = 3$

Hierbij mag de *seed* X_0, de eerste waarde voor X_n, niet deelbaar zijn door p of q.
Meestal wordt een reeks bits B_i verkregen door de *least significant bit* te gebruiken van het gegenereerde getal. Dit kan men verkrijgen door na de uitvoering van voorgaand algoritme volgende bewerking uit te voeren:

$$B_i = X_{n+1} \bmod 2$$

wat telkens een één of een nul als resultaat zal hebben.

We illustreren het algoritme met volgende waarden: $X_0 = 101355$, $p = 383$ en $q = 503$ zodat $m = 192649$.

Bij een eerste iteratie krijgen we:

$$X_1 = (101355)^2 \bmod 192649 = 20749 \text{ en } B_1 = 20749 \bmod 2 = 1$$

Bij volgende iteraties krijgen we:

$$X_2 = (20749)^2 \bmod 192649 = 143135 \text{ en } B_2 = 143135 \bmod 2 = 1$$
$$X_3 = (143135)^2 \bmod 192649 = 177671 \text{ en } B_3 = 177671 \bmod 2 = 1$$
$$X_4 = (177671)^2 \bmod 192649 = 97048 \text{ en } B_4 = 97048 \bmod 2 = 0, \text{ enz...}$$

De reden waarom deze PRNG als veilig beschouwd wordt ligt in het feit dat het moeilijk is het getal m te ontbinden tot de waarden p en q die hiervan aan de basis liggen.

3.4 Analyse en overzicht

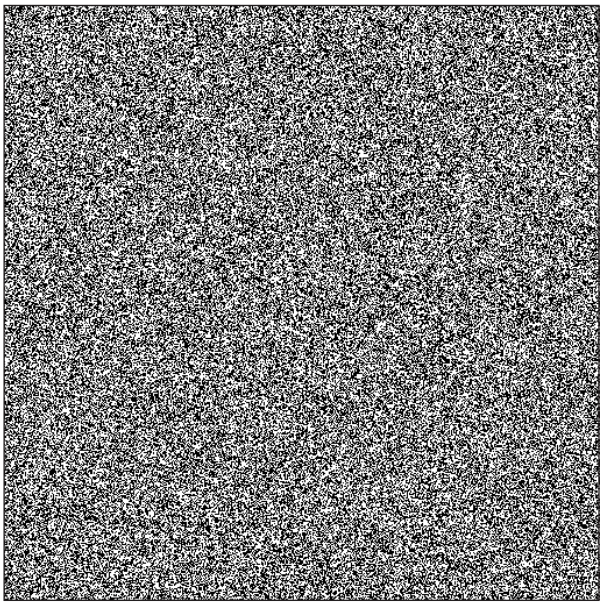
In voorgaande delen werden de voornaamste eigenschappen en werking van TRNGs en PRNGs besproken. Hier worden ze tegenover elkaar geplaatst, worden hun belangrijkste eigenschappen opgesomd en hun sterke en zwakke punten met elkaar vergeleken.

3.4.1 Overzicht

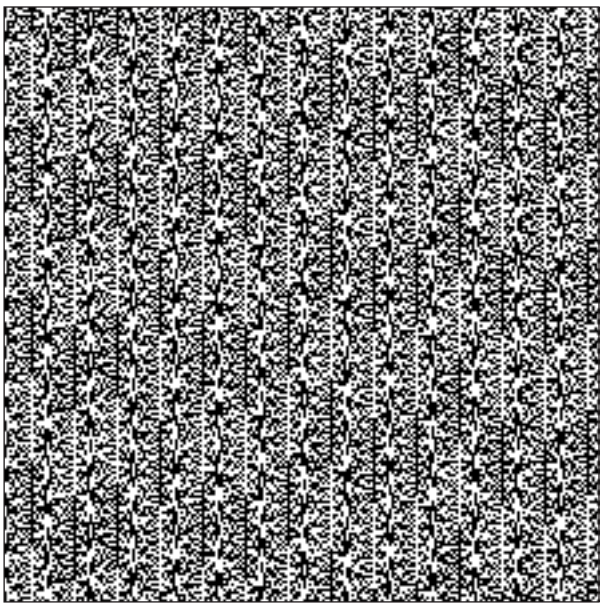
	TRNGs	PRNGs
Kenmerken	• Hardware • Chaotisch- of kwantum fenomeen als bron • Niet-deterministisch • Niet-periodiek	• Software • Algoritme • Deterministisch • Periodiek
Sterke en zwakke punten	• Traag en inefficiënt • Er moet genoeg externe willekeur voor handen zijn. Bij veel vraag kan de buffer uitgeput geraken. • Kostelijk • Hoog veiligheidsgehalte • De getallenreeks kan niet opnieuw gegenereerd worden • Geen mogelijkheid tot voorspellen van volgende getallen op basis van de voorgaande.	• Snel en efficiënt • Geen behoefte aan een buffer • Goedkoop • Geen veiligheid gegarandeerd • Mogelijkheid tot hergenereren van de zelfde getallenreeks. • Mogelijkheid tot voorspellen van volgende getallen op basis van de voorgaande.

3.4.2 Een opmerkelijke vergelijking

Een goed voorbeeld van een vergelijking tussen PRNs en TRNs wordt geïllustreerd op de website van web-developer Bo Allen[5]. Daarop is een visuele vergelijking terug te vinden tussen de getallen gecreëerd met een PRNG, meer bepaald de `rand()` functie uit PHP, en de TRNG van RANDOM.ORG. De bits uit de getallenreeks worden omgezet in zwarte of witte pixels. De afbeelding die het resultaat was van de getallenreeks van RANDOM.ORG als input leverde een te verwachten resultaat op, wat je kan zien hieronder:



Maar tot Bo Allen's grote verrassing leverde de gegenereerde afbeelding met getallen uit de `rand()` functie van PHP het volgende resultaat op:



5 BO ALLEN, *Pseudo Random vs. True Random – A Simple Visual Example,* http://www.boallen.com/random-numbers.html

Nader onderzoek, waarvan men details kan terugvinden op de blog Codifies[6], verduidelijkte dat zo'n opvallend patroon slechts weinig voorkomt, maar dat het in dit geval aan de ongelukkige combinatie lag van programmeertaal (PHP), besturingssysteem (Windows) en functie (`rand()`). Dezelfde afbeelding werd nogmaals aangemaakt maar dan in een *Linux* omgeving, en dit leverde geen zo'n zichtbaar patroon op. Op *Windows* werd dezelfde afbeelding nogmaals aangemaakt, maar dan gebruik makende van de `mt_rand()` functie, die gebruik maakt van een andere PRNG, namelijk de Mersenne Twister en deze had ook geen zichtbaar patroon.

Deze toch wel opmerkelijke visuele vergelijking toont aan dat men toch voorzichtig moet omspringen met het gebruik van een random-functie. Zoals aangetoond werd kan door een verkeerde combinatie te maken het resultaat toch niet zo willekeurig zijn zoals men misschien gehoopt had. Het onderzoek wees uiteindelijk uit dat de `rand()` functie van PHP geen optimaal gebruik maakt van de random-functie van het besturingssysteem. Het is dan ook aangeraden de `mt_rand()` functie te gebruiken. En het mag dan ook duidelijk zijn dat zo'n zichtbaar visueel patroon onmogelijk kan opduiken bij een TRNG zoals RANDOM.ORG, HotBits, ...

6 Codifies, *PHP rand(0,1) on Windows < OpenSSL rand() on Debian*, http://cod.ifies.com/2008/05/php-rand01-on-windows-openssl-rand-on.html

4. Toepassingen van Random Numbers

Willekeurige getallen worden ingezet bij tal van toepassingen, zoals cryptografie, simulatie, computerspelen, beslissingen nemen en kunst. Hoewel de meest voorkomende toepassingen hieronder vermeld worden, mag men hieruit niet besluiten dat dit een definitieve lijst is met een overzicht van alle bestaande toepassingen.

4.1 Cryptografie en beveiliging

Cryptografie is de kunst of wetenschap van het versleutelen van een bericht in een ogenschijnlijk willekeurige warboel die alleen met een bepaalde sleutel te ontcijferen is, zodat het oorspronkelijke bericht opnieuw geïnterpreteerd kan worden. Cryptografie werd reeds door Caesar in de tijd van de Romeinen toegepast, maar is sinds de komst van de computer en het internet steeds belangrijker geworden. Door de versnelde rekenkracht van de computer kunnen versleutelde berichten sneller gekraakt worden. Daarom zijn de algoritmen om berichten te versleutelen steeds complexer geworden. Aan de basis van elke versleuteling ligt dan ook de creatie van een sleutel, en deze sleutel bevat liefst zo onvoorspelbaar mogelijke elementen, zoals uiteraard willekeurige getallen.
Voorbeelden van encryptiealgoritmen zijn het RSA-algoritme, PGP, ...
Wanneer de gebruiker op internet bijvoorbeeld een webwinkel bezoekt, wordt er op de server een *Session Key* aangemaakt. Dit is een tijdelijke sleutel die aan een gebruiker gekoppeld wordt. Omdat er uiteraard meerdere gebruikers op het zelfde moment met de server in verbinding staan is het nodig dat deze *Session Key* zo uniek mogelijk is. Er wordt een RNG gebruikt bij de creatie hiervan.
Wanneer een verbinding gelegd wordt tussen twee computers -in vaktermen is dit een *handshake* tussen twee *peers*- wordt een unieke *identifier* bij elke afgelegde weg meegestuurd. Deze *identifier* moet telkens uniek en moeilijk te raden zijn door anderen. Door willekeur te gebruiken zal het onrechtmatig identificeren van de *identifier* zo goed als onmogelijk zijn.

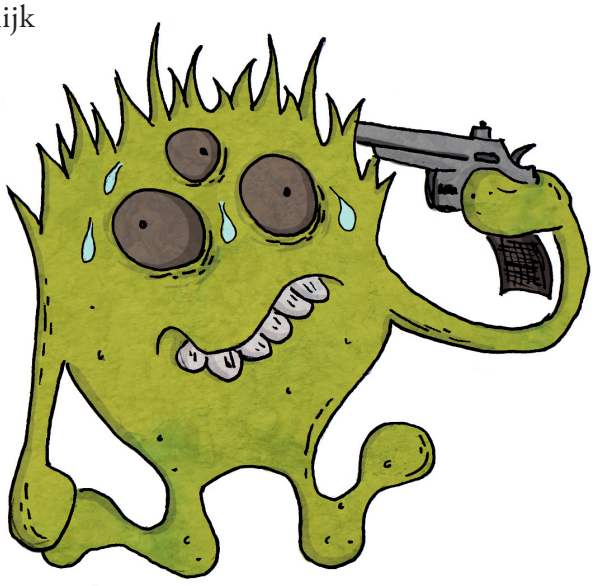
4.2 Simulatie

Simuleren is het recreëren van een complex fenomeen, een omgeving of een ervaring, weliswaar in een (sterk) vereenvoudigde vorm. Een simulatie stelt de gebruiker ervan in staat tot nieuwe inzichten te komen die bij rechtreeks onderzoek van het origineel (waarop de simulatie gebaseerd is) niet mogelijk zouden zijn. Een simulatie laat ook toe om zaken te onderzoeken, te proberen,

aan te leren zonder dat dit in praktijk uitgevoerd moet worden en zo voor eventueel gevaar zou kunnen zorgen (vb: vliegsimulator, simulatie van een nieuw verkeerssysteem, ...).
Wanneer er een computer gebruikt wordt om natuurlijke fenomenen te simuleren, dan doet men beroep op willekeurige getallen om de simulatie realistischer te maken. Dit is zeker noodzakelijk wanneer men menselijk of dierlijk gedrag probeert na te bootsen.
Een gedetailleerdere uitleg over het gebruik van willekeurige getallen wordt gegeven aan de hand van een simulatie methode die de *Monte Carlo methode* genoemd wordt.

De Monte Carlo methode

Monte Carlo methodes, ook wel Mont Carlo simulaties genoemd, zijn een verzameling van algoritmes die berusten op herhaalde berekeningen met willekeurige getallen. Het gebruik ervan werd geïntroduceerd in de jaren 40 in het Los Alamos Nationaal Laboratorium door fysici zoals John von Neumann, Enrico Fermi en Stanislaw Ulam.
De methodes worden voornamelijk gebruikt bij studies van systemen waarbij een grote mate van vrijheid aanwezig is, zoals bijvoorbeeld bij vloeistoffen of het gedrag van neutronen. Bij dit laatste wordt de Monte Carlo methode gebruikt bij nucleaire reactors om het gedrag te voorspellen van neutronen die doorheen verschillende materialen gaan. In de reactor botsen neutronen, vermenigvuldigen ze zich, worden ze geabsorbeerd of ontsnappen ze, elk met verschillende waarschijnlijkheden. Het is onmogelijk om de staat van elk neutron op een bepaald moment te kennen. De Monte Carlo methode wordt gebruikt om een hoeveelheid mogelijke kettingreacties te genereren. Deze mogelijke kettingreacties berekenen niet exact de uiteindelijke uitkomst, maar geven er wel een goede indicatie van. Wanneer men hierbij genoeg mogelijkheden genereert kan men hiervan het gemiddelde nemen. Dit gemiddelde zal erg nauw in de buurt liggen van het werkelijke resultaat.
In het praktisch deel werd een voorbeeld uitgewerkt ter illustratie van het gebruik van de Monte Carlo methode. Het gaat hier om het berekenen van de oppervlakte van een cirkel zonder de waarde van π te kennen. Een benadering van de waarde van π wordt dus door middel van willekeurige getallen gevonden. Laat het wel duidelijk zijn dat dit slechts een zeer eenvoudig voorbeeld is en dat Monte Carlo methodes gebruikt worden bij veel complexere probleemstellingen.



4.3 Computer- en gokspelen

Gokspelen

Iedereen kent ze wel: de gokautomaten op de kermis of in een café. Flikkerende lichtjes, geluiden, bedragen die te winnen zijn, ... kortom alles om mensen te overtuigen hun kans eens te wagen. Maar we spreken hier inderdaad over een *kans*, want eigen aan deze gokspelen is dat ze de input van de speler herleiden tot slechts louter amusementswaarde en geen eigenlijke inbreng in het spel. De kern van dit spel is namelijk dat de RNG bepaalt welke kant het spel uitgaat, zodat dit ertoe kan leiden dat de speler met een hoop geld terug naar huis keert, of met juist helemaal niets.

Gezien de grote aanwezigheid van gokautomaten en de serieuze gevolgen van het gebruik ervan, mag het geen verrassing zijn dat er een reglementering opgelegd is door de overheid. Deze gokautomaten moeten voldoen aan bepaalde opgelegde vereisten. Zo'n vereiste is bijvoorbeeld dat de gebruikte RNG wel echte willekeurige getallen genereert en dat ze op de juiste manier gebruikt worden. U kan best begrijpen dat een gokautomaat die getallen genereert die telkens net iets voordeliger uitkomen voor de eigenaar van de automaat, niet voldoet aan deze vereisten.

Gokspelen hebben de komst van het internet niet aan zich voorbij laten gaan. Het is dan ook een markt die zich wijd verspreid heeft over het internet: er zijn ettelijke online gokspelen te vinden en dit in alle mogelijke variaties. Ook hier liggen RNGs aan de basis, al dan niet gebruik makende van TRNGs. Gezien het vaak nogal vage juridische karakter van het internet (en dit afhankelijk van land tot land) is het zo goed als onmogelijk te achterhalen hoe de getallen gegenereerd worden en kunnen de opgelegde vereisten door de overheid variabel of zelfs onbestaande zijn.

Computerspelen

De markt van computerspelen is er één die een enorme groei gekend heeft en waarvan het einde nog lang niet in zicht is. De variatie in genres is enorm en de spelen zijn er in alle vormen voor zowel jong als oud. Hoewel niet alle spelen gebruik maken van willekeurige elementen, zal dit toch wel het geval zijn bij het merendeel van deze spelen. Gezien het feit dat er soms erg veel en vooral snel willekeurige getallen beschikbaar moeten zijn, ligt het voor de hand dat hierbij gebruik gemaakt wordt van een PRNG.

Om het gebruik ervan ietwat te verduidelijken worden een aantal mogelijke toepassingen opgenoemd:

- **Het creëren van een karakter:** dit kan gaan van visuele eigenschappen, zoals lichaamsbouw, kleren, ..., tot persoonlijke eigenschappen die bij het personage horen. De gebruiker zal meestal zijn eigen karakter samenstellen, de computer zal willekeurig alle (of een aantal elementen ervan) andere personages samenstellen.
- **"Bewegingen" van de tegenstander:** wanneer een spel gespeeld wordt tegen een computer kan het de taak zijn van de computer om zo onvoorspelbaar mogelijk gedrag te vertonen. Deze graad van onvoorspelbaarheid staat wel in verhouding tot de gekozen moeilijkheidsgraad. Zo kan men verwachten dat bij een hogere moeilijkheidgraad de bewegingen van de computer het meest onvoorspelbaar zullen zijn.
- **Creatie van het landschap, plattegrond, traject:** telkens een unieke spelsessie creëren door één van deze elementen willekeurig op te bouwen. Op deze manier kan men het spel boeiend houden voor de speler: hij is in staat bij het spel elke spelsessie anders te beleven.

4.4 Beslissingen nemen

Willekeur, en dus ook een reeks willekeurige getallen, wordt algemeen aanvaard als zijnde een rechtvaardige beslissingnemer. Het kan hierbij gaan om louter een ja of nee beslissing, of het uitkiezen van één of meerdere waarden uit een reeks getallen.
Wanneer dus een volledig onbeïnvloedbare en gelijk opgaande keuze gemaakt moet worden kan

men gebruik maken van een (al dan niet hardwarematige) TRNG. Enkele voorbeelden hierbij zijn het selecteren van rechters bij rechtspraak, het uitkiezen van personen die hun militaire dienst moeten vervullen of het selecteren van één of meerdere werknemers ter controle op drugsgebruik. En natuurlijk is er ook nog de (commerciële) loterij, in alle mogelijke variaties. Hierbij zal het beeld van de hardware RNG als glazen bol, met daarin de balletjes die een nummer bevatten, ons allen wel bekend voorkomen.

4.5 Kunst

Het mag dan misschien een niet zo voor de hand liggend toepassingsgebied zijn, toch worden vandaag de dag willekeurige getallen gebruik bij het creëren van kunst. Hierbij enkele voorbeelden: willekeurige input wordt gebruikt om een uniek geluid te creëren, dat op die manier elke keer het afgespeeld wordt anders klinkt. De Amerikaanse muziekgroep Technician[1] gebruikt willekeurige getallen voor het aanmaken van unieke hoezen voor hun cd's.
Het is het onvoorspelbare karakter waartoe de kunstenaar zich aangetrokken voelt. Hiermee is hij in staat unieke resultaten te creëren, waarmee hij zich kan onderscheiden van anderen.

4.6 Diverse toepassingen

Los van deze grootste groepen van toepassingen, zijn er natuurlijk ook nog een heel aantal kleine toepassingen waarbij beroep gedaan wordt op één of meerdere willekeurige getallen.

- **Websites**: om een website een dynamischer karakter te geven wordt vaak op de hoofdpagina een element geplaatst dat willekeurig uit een database werd geselecteerd. Dit kan bijvoorbeeld "De quote van de dag" zijn, een bericht uit een blog-archief of een foto uit een fotobibliotheek.
- **Captcha**: dit is een afkorting voor Completely Automated Public Turingtest to tell Computers and Humans Apart. Het wordt gebruikt op internetpagina's om bij het invullen van een formulier een onderscheid te kunnen maken tussen een mens en een *spamrobot*. Hiervoor worden vaak willekeurig gegenereerde karakters in een afbeelding geplaatst, die de gebruiker dan over moet typen. Wanneer de ingegeven karakters overeenkomen met die uit de afbeelding zal het formulier verder verwerkt worden. Om het de *spamrobots* zo moeilijk en onvoorspelbaar mogelijk te maken wordt gebruik gemaakt van een RNG.
- **(Interactieve) Presentaties:** bepaalde elementen kunnen willekeurig aan de presentatie toegevoegd worden. Zo kunnen bijvoorbeeld geanimeerde grafische elementen aan de achtergrond toegevoegd worden, waarbij x- en y-coördinaten gekozen worden die in het bereik liggen van de breedte en hoogte van het document.
- **Shuffle-functie**: bij het afspelen van muziek wordt willekeurig het volgende nummer uit een lijst van nummers geselecteerd. Hier is echter geen sprake van echte willekeur, omdat moest dit wel zo zijn de nummers die reeds gespeeld geweest zijn ook opnieuw gekozen zouden kunnen worden. Omdat de luisteraar dit uiteraard minder aangenaam vindt zullen de gekozen waarden bijgehouden worden, opdat deze nummers niet meer aan bod zouden komen (bij het selecteren van het nummer wordt verder gezocht tot er een nummer gevonden is dat zich niet in deze "lijst" bevindt).

1 Technician, http://technician.org

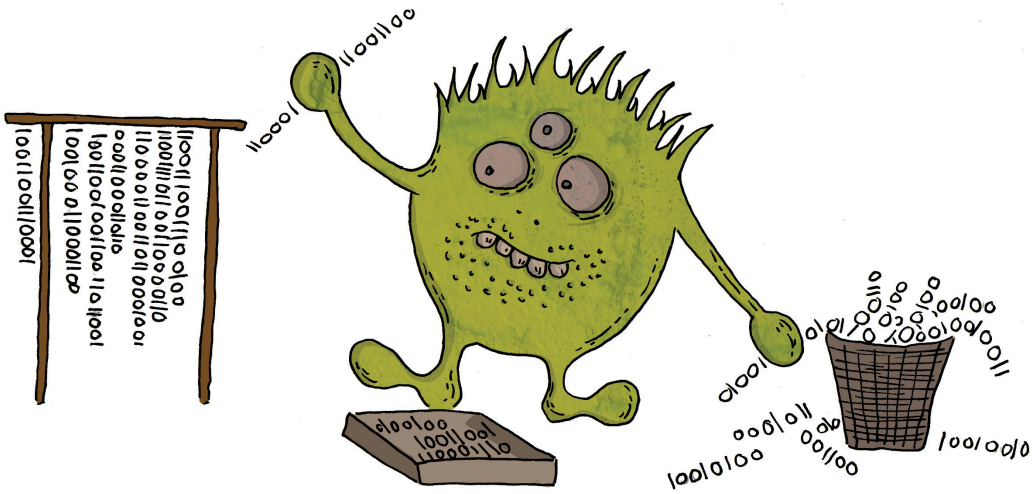
4.7 Overzicht van de gebruikte RNG

Toepassing	Best gekozen RNG
Cryptografie en beveiliging	TRNG
Computerspelen	PRNG
Gokspelen	TGNR
Simulatie	PRNG
Beslissingen nemen	TRNG
Kunst	Afhankelijk van de toepassing

5. Hoe aantonen dat Random Numbers echt willekeurig zijn?

5.1 Inleiding

Wanneer men zoveel investeert in het aanmaken van echte willekeurige getallen, dan mag het geen verrassing zijn dat men nu ook wil aantonen dat deze getallen echt willekeurig zijn. Maar hoe kan men bewijzen dat deze getallen echt willekeurig zijn? Is dit dan hoegenaamd wel mogelijk? Het antwoord hierop is eenvoudig: nee, het is niet mogelijk met een waterdicht bewijs aan te tonen dat deze getallen echt willekeurig zijn. Maar men heeft wel een aantal tests ontworpen die moeten nagaan in welke mate de getallen (en dus ook de PRNG) willekeurig zijn. Hoe meer van zulke tests de PRNG kan doorstaan, hoe groter het vertrouwen in de PRNG wordt.



Omdat we verwachten dat in de getallenreeksen soms ook niet-willekeurige reeksen voorkomen, kunnen we ons voornemen dat sommige reeksen op zijn minst enkele van deze tests niet doorstaan. Als blijkt dat de getallenreeks echter meerdere tests niet doorstaat moeten we op onze hoede zijn. Dit is te vergelijken met het testen of er met een dobbelsteen geknoeid is of niet. Als we verschillende malen met de dobbelsteen gooien en een bepaald getal komt frequenter voor dan de andere, dan is er een reden om de betrouwbaarheid van de dobbelsteen in vraag te stellen. Toch wil het niet zeggen dat wanneer een gegeven getallenreeks een bepaalde test niet doorstaat dit geen willekeurige getallen zijn. Het kan goed zijn dat bijvoorbeeld het getal 9 zes keer na elkaar voorkomt. Theoretisch gezien is dit perfect mogelijk, elk getal staat namelijk volledig

los van het vorige en komende getal, alleen zal de test deze reeks tegenhouden omwille van de bruikbaarheid ervan (al is dit afhankelijk van de toepassing). Niettegenstaande het feit dat deze getallen dus wel degelijk willekeurig zijn kan men ervoor kiezen ze toch niet te gebruiken. Criteria hiervoor zijn deze tests.
De tests worden uiteraard uitgevoerd op een reeks willekeurige getallen en het aantal getallen die nodig zijn is afhankelijk van welke test uitgevoerd wordt. Hierbij moet ook verduidelijkt worden dat de tests uitgevoerd worden op binaire getallen en niet op decimale. Dit is vrij logisch omdat men er kan vanuit gaan dat de getallen gegenereerd werden door een computer (los van het feit of er al dan niet echte willekeurige input gebruikt werd), die uiteraard slechts enkel met enen en nullen overweg kan.
Het is ook aangeraden, wanneer men de goede kwaliteit van een RNG wil aantonen, de tests meerdere keren uit te voeren op verschillende gegenereerde reeksen willekeurige getallen. Het kan voorkomen dat zelfs een absoluut willekeurige getallenreeks een bepaalde test niet doorstaat. In dat geval spreekt men over een *Type I error*. Dit is natuurlijk zeer normaal: omdat het juist een absoluut willekeurige getallenreeks is kan het zijn dat er elementen in aanwezig zijn die voor de test als niet-random beschouwd worden.
De uitkomsten van alle tests resulteren telkens in een bepaalde waarde die tussen een bepaalde onder- en bovengrens ligt. Deze grenzen kunnen aangepast worden. Zo kunnen "slechte" RNGs sneller opgespoord worden, maar zullen langs de andere ook minder "goede" RNGs de tests doorstaan. Met het bepalen van deze grenzen moet omzichtig omgegaan worden.

5.2 Overzicht van enkele tests

We bespreken kort enkele tests [1] [2] die gebruikt kunnen worden om na te gaan in welke mate willekeurige getallen willekeurig zijn. Deze tests behandelen telkens één specifiek aspect van de reeks willekeurige getallen. We beperken ons tot het bespreken van deze specifieke eigenschappen.

Het is zo dat er verschillende 'pakketten' van zulke tests bestaan. Hierbij wordt niet specifiek ingegaan op welk pakket juist welke tests bevat, maar zal een aantal van de belangrijkste tests uit de verschillende pakketten aan bod komen, zodat men zich toch een beeld kan vormen wat nu specifiek gecontroleerd wordt bij een gegenereerde reeks willekeurige getallen. Hierbij wel even een kort overzicht van enkele van de meest gebruikte pakketten:

- **Knuth**: Donald Knuth behandelt in zijn boek *The Art of Computer Programming, Volume 2* (1969) de creatie van willekeurige getallen en voorzag hierbij ook een aantal tests. Deze tests zijn vandaag de dag echter verouderd, waarmee bedoeld wordt dat bepaalde PRNGs de tests zouden doorstaan die door andere (nieuwere) testpakketten tegengehouden zouden worden.
- **Diehard**: dit pakket werd ontwikkeld door George Marsaglia, die in 1995 een aantal verbeterde tests introduceerde. Deze speelden beter in op de ondertussen enorm toegenomen rekenkracht van computers (en dus ook het aantal willekeurige getallen dat berekend kon worden).

1 LOUISE FOLEY, *Analysis of an online random number generator*, Trinity College Dublin, April 2001.

2 KENNY CHARMAINE, *Random Number Generators: An evaluation and comparison of Random.org and some commonly used generators*, Trinity College Dublin, April 2005.

- **Crypt-X**: dit is een commercieel softwarepakket dat ontwikkeld werd door de mensen van het *Information Security Research Center* aan de *Queensland University of Technology* in Australië.
- **NIST**: Het *National Institute of Standards and Technology* bracht een eigen pakket uit in 2001. Het bestaat uit zo'n 16 tests en wordt vandaag de dag als de standaard gezien wat betreft het testen van willekeurige getallen.
- **ENT**: Dit pakket werd ontwikkeld door John Walker in 1998. Het kan specifiek gebruikt worden bij het testen van de resultaten van PRNGs, voornamelijk dan bij encryptie.

Bepaalde tests kunnen uitgevoerd worden specifiek voor een *bepaalde* toepassing waarbij de willekeurige getallen gebruikt worden. Deze tests kunnen ook geoptimaliseerd worden voor die specifieke toepassing. Toch is het aangeraden de nodige tests uit te voeren die het gebruik van de getallenreeks bij *elke* soort toepassingen kunnen verzekeren.

Frequency test (monobit)

Het doel van deze test is het nagaan of het aantal keren dat een nul en een één voorkomen in de getallenreeks wel overeenkomt met wat verwacht kan worden van een echte reeks willekeurige getallen . Verwacht wordt dat deze twee zo goed als aan elkaar gelijk zijn. Er wordt aangeraden dat elke getallenreeks minstens 100 bits bevat. Wanneer de getallenreeks deze test niet doorstaat is de kans erg groot dat de getallenreeks ook de andere tests niet zal doorstaan. Om deze reden wordt deze test dan ook meestal eerst uitgevoerd.

Frequency test (block)

Deze test komt overeen met voorgaande monobit frequentie test, alleen wordt nu de getallenreeks opgedeeld in verschillende blokken, waarvan gecontroleerd wordt of het aantal enen en nullen ongeveer gelijk is.

Runs test

Met deze test wordt gezocht naar het aantal *runs* in een getallenreeks, waarbij een *run* staat voor een ononderbroken opeenvolging van dezelfde bits. Met deze test wordt nagegaan of het aantal *runs* van zowel enen als nullen overeenkomt met wat verwacht kan worden van een echte willekeurige getallenreeks. Deze test gaat meer bepaald na of de afwisseling tussen enen en nullen niet te snel of te traag is. Ook hierbij wordt een minimum van 100 bits aangeraden.
Deze test wordt ook uitgevoerd op verschillende blokken, zoals bij de block frequentie test.

Binary Matrix Rank test

Deze test plaatst alle getallen uit een getallenreeks in verschillende matrices met gelijke dimensies. Met deze test wordt onderzocht of de verschillende subreeksen (de matrices) genoeg verschillen met elkaar vertonen. Er wordt aangeraden om matrices te gebruiken van 32 op 32, met een getallenreeks van minimum 38.912 bits.

Discrete Fourier Transform test

Deze test gaat na of er repetitieve patronen naast elkaar optreden. Er wordt gewerkt met een vooropgestelde grenswaarde waarvan aangenomen wordt dat deze een indicatie is van een reeks echte willekeurige getallen. Afhankelijk van het aantal 'pieken' boven of onder deze waarde wordt de reeks al dan niet als willekeurig beschouwd. Bij deze test wordt aangeraden met minstens 1000 bits te werken.

Non-overlapping Template Matching test

Het doel van deze test is na te gaan of er in de getallenreeks een bepaald patroon (een niet-periodiek patroon) niet teveel voorkomt. Een x-bit 'venster' wordt gebruikt voor het zoeken naar een x-bit patroon. Wanneer het patroon niet gevonden wordt, m.a.w. de bits komen niet overeen, dan schuift het venster één bit op. Indien het patroon wel gevonden wordt schuift het venster op tot vlak na het aantal bits van het gevonden patroon en gaat het vanaf daar verder. De *Overlapping Template Matching test* berust op quasi het zelfde principe, alleen zal het venster hierbij niet met het volledige patroon mee opschuiven, maar slechts één bit, net zoals wanneer het patroon niet gevonden zou zijn (uiteraard wordt het aantal overeenkomsten wel bijgehouden).

Maurer's "Universal Statistical" test

Bij deze test wordt gekeken naar het aantal bits tussen overeenkomende patronen. Het doel van deze test is het nagaan of het mogelijk is de getallenreeks opmerkelijk veel te comprimeren zonder verlies van informatie. Een reeks die aanzienlijk gecomprimeerd kan worden wordt gezien als niet-willekeurig.

Linear Complexity Test

Deze test gaat na of de getallenreeks wel genoeg complexiteit bevat om te voldoen aan de complexiteit die geassocieerd wordt met een reeks echt willekeurige getallen. Bij deze test wordt een reeks van minimum 1.000.000 getallen aanbevolen.

Dit zijn slechts een aantal van een hele lijst tests. Het is moeilijk om te bepalen welke van deze tests nu doorslaggevend zijn in de bepaling van de willekeurigheid van een getallenreeks, maar het is in ieder geval aan te raden zoveel mogelijk tests uit te voeren. Het uitvoeren van deze tests kan soms erg rekenintensief zijn, afhankelijk van het soort test. Tests waarbij een groot aantal getallen gebruikt worden die een aantal complexe berekeningen ondergaan kunnen dan ook al snel meerdere uren rekenwerk opleveren voor een computer.

Hoewel reeds vermeld werd dat het onmogelijk is met absolute zekerheid aan te tonen dat een RNG echte willekeurige getallen genereert, hebben deze tests in praktijk reeds bewezen dat ze een handig hulpmiddel kunnen zijn, dat in quasi alle gevallen volstaat. RNGs die één van deze testpakketten, maar voornamelijk dan het NIST-pakket, succesvol doorstaan, hebben de dag van vandaag reeds bewezen dat ze als volwaardige bron van willekeurige getallen gebruikt kunnen worden bij de meest uiteenlopende toepassingen.

6. Praktische uitwerkingen

In dit praktische deel werken we 3 toepassingen verder uit die besproken werden in het theoretisch gedeelte. Elk van deze toepassingen verduidelijkt een andere aspect van het gegeven *willekeurige getallen*.

1. In de eerste plaats wordt de Linear Congruential Generator van Lehmer uitgewerkt in een interactieve webapplicatie. Deze uitwerking behandelt een mogelijke creatie van willekeurige getallen.
2. Een tweede uitwerking is een toepassing van een Monte Carlo methode. We zullen een eenvoudige simulatie uitvoeren die gebruik maakt van willekeurige getallen. Deze uitwerking behandelt een mogelijkheid voor het gebruiken van willekeurige getallen.
3. De derde uitwerking is ook een toepassing van willekeurige getallen, doch hier worden een aantal creatieve elementen geïntegreerd. Er wordt een interactief spelletje in Flash uitgewerkt, waarbij aangetoond wordt hoe nuttig willekeurige getallen in sommige gevallen kunnen zijn. Er wordt ook aangetoond hoe men gebruik kan maken van *True Random Numbers* afkomstig van één van de online diensten die eerder besproken werden.

6.1 Webapplicatie rond de Linear Congruential Generator

Deze webapplicatie stelt de lezer in staat zelf op zoek te gaan naar hoe de LCG werkt en welke waarden gebruikt kunnen worden voor een optimaal resultaat.

Het resultaat is te bekijken op **http://cientouno.be/random/** [1]

Structuur van de website:

Manuele input

Hier kunnen alle waarden manueel ingevuld worden. Merk op dat er een validatie uitgevoerd wordt bij het versturen van het formulier. Deze validatie kijkt in eerste plaats na of de velden niet leeg zijn, maar kijkt daarna ook of de ingegeven waarden voldoen aan de voorwaarden die opgelegd zijn door de formule. De gebruiker kan het aantal getallen kiezen, gaande van 10 tot

1 Om een optimale werking te verzekeren wordt aangeraden **Firefox** als webbrowser te gebruiken.

10.000. Ook de *output*, hoe de getallen weergegeven worden, kan gekozen worden: de originele getallen, als getallen van 0 tot 1, als getallen van 0 tot 10 en als enen en nullen.
Het resultaat wordt opgedeeld in drie delen: de getallenreeks, het aantal getallen en een overzicht van de gebruikte getallen. De *getallenreeks* spreekt voor zich, dit is uiteraard de reeks van getallen die verkregen werd na de berekening. Het *aantal getallen* is het aantal unieke getallen dat in de reeks voorkomt. Het *overzicht van de gebruikte getallen* geeft een overzicht van alle unieke getallen en vermeldt hoeveel keer elk getal voorkomt.

Optimale input

Hierbij kan enkel gekozen worden uit een aantal vooraf ingestelde waarden. Bij de waarde voor X_0 werd wel de keuze *PHP time()* toegevoegd. Dit stelt de gebruiker in staat telkens een unieke reeks getallen te creëren door als input voor X_0 de functie `time()` te gebruiken, die het aantal seconden sinds 1 januari 1970 genereert. De weergave van het resultaat is identiek aan die van de manuele input.

Visueel

De gebruiker kan ook hier weer kiezen voor een aantal waarden die gebruikt zullen worden bij de berekening. In plaats van echter louter een reeks getallen weer te geven, wordt ditmaal een afbeelding gegenereerd op basis van de getallen. Deze afbeelding is 400 op 400 pixels groot, wat dus betekent dat een reeks van 160.000 getallen gegenereerd moet worden! Deze reeks bestaat uit bits, zijnde enen en nullen. Deze worden één voor één uitgelezen: is de waarde een 1 dan wordt er een witte pixel geplaatst, is de waarde een 0, dan wordt er een zwarte pixels geplaatst. Dit gebeurt van linksboven tot rechts onderaan.
Naast het resultaat wordt de formule weergegeven met daarbij de gebruikte waarden. Merk op dat wanneer gekozen wordt voor *PHP time()*, dit enkel zal uitgevoerd worden wanneer de hoogste waarde voor *m* gekozen wordt. In alle andere gevallen wordt een één in de plaats gebruikt. Dit gebeurt om te blijven voldoen aan de voorwaarden die opgelegd worden bij de formule. Om deze zelfde reden wordt ook de waarde voor *a* automatisch toegewezen afhankelijk van de geselecteerde waarde voor *m*.

Overzicht toepassingen

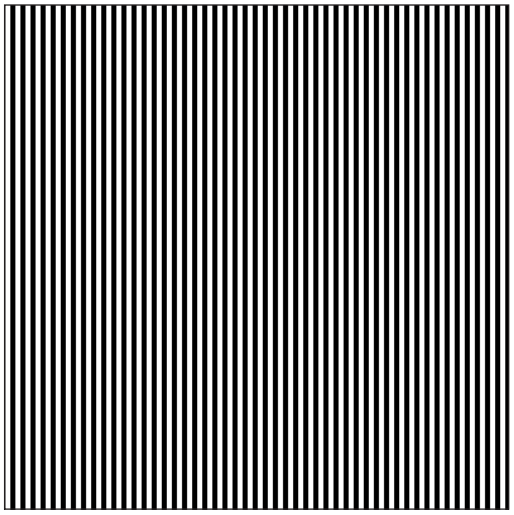
Hier wordt een overzicht gegeven van welke waarden bij welke (populaire) programmeeromgeving gebruikt worden.

Analyse van de resultaten met verschillende waarden

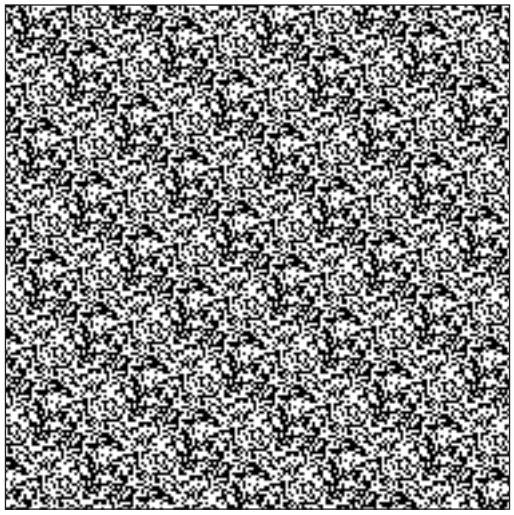
Stel dat we volgende waarden ingeven: $X_0 = 2$, $a = 8$, $c = 3$, $m = 10$, dan krijgen we na berekening een reeks getallen, namelijk 9, 5, 3, 7, 9, 5, 3, 7, 9, ... Het is duidelijk dat de vier unieke getallen in een cyclus terugkomen. De lengte van deze unieke reeks noemen we de periode *p* van de generator. Wanneer *p* gelijk zou zijn aan *m* dan heeft de generator een volledige periode, wat wil zeggen dat er geen "gaten" optreden in de getallenreeks.
Omwille van het feit dat alle gegenereerde getallen binnen de waarde *m* liggen, is het aangeraden om *m* zo groot mogelijk te nemen, zodat er genoeg potentieel aanwezig is voor de creatie van een lange reeks willekeurige getallen. Daarom wordt in de meeste gevallen gekozen voor een getal

waarmee een computer gemakkelijk overweg kan, namelijk 2^{32} of 2^{64}.
Tests hebben uitgewezen dat wanneer m een priemgetal is en $c = 0$, bij bepaalde waarden voor a de periode van de generator gelijk is aan $m - 1$, waarbij enkel de waarde 0 ontbreekt. Zo'n getal is $2^{31} - 1 = 2147483647$, de waarde die gebruikt werd voor m in het deel *optimale input*.
Bij kleine waarden voor m wordt al snel duidelijk dat de reeks zich herhaalt, dit is gemakkelijk uit de getallenreeks af te leiden. Om toch in staat te zijn op een efficiënte manier eventuele herhalingen op te sporen kunnen we gebruik maken van de visuele weergave van de getallenreeks.

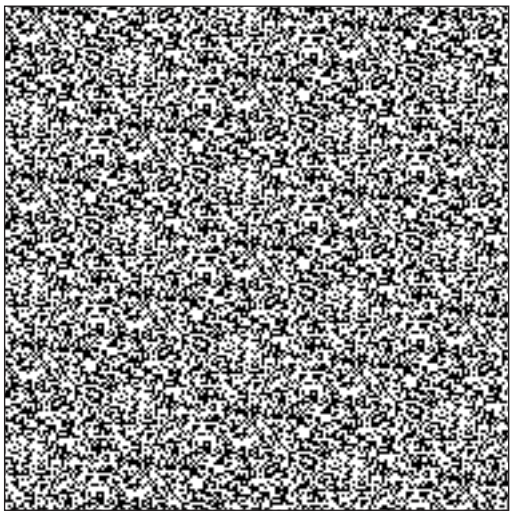
Wanneer men voor de waarde $m = 32$ kiest is de periode maar al te duidelijk waar te nemen. Bij $m = 3566$ zien we al wat meer willekeur opduiken. Toch is ook hier duidelijk dat een vast patroon zich herhaalt. Dit is ook zo bij $m = 35687$, maar we merken hier dat de periode een stuk groter geworden is. Een patroon blijft echter duidelijk zichtbaar. Met $m = 2^{31} - 1$ kunnen 160.000 willekeurige getallen gegenereerd worden waarbij geen zichtbaar patroon optreedt. Wat we zien is een willekeurige brij van witte en zwarte puntjes, wat ons duidt op een reeks willekeurige getallen die voor menige toepassing zal volstaan.



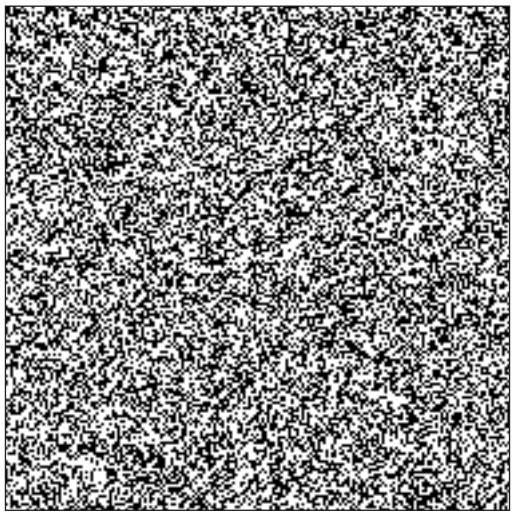
$m = 32$



$m = 3566$



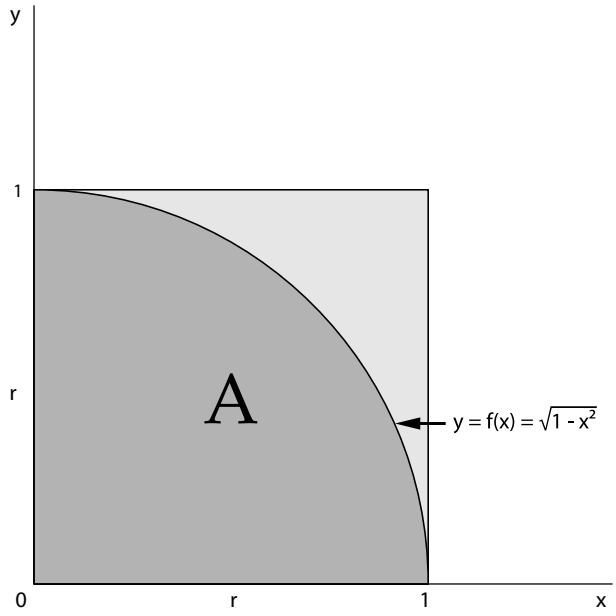
$m = 35687$



$m = 2^{31} - 1$

6.2 Illustratie Monte Carlo methode

Hieronder volgt een voorbeeld[2] van hoe de waarde van π benaderd kan worden door gebruik te maken van een Monte Carlo methode:
We gaan uit van een situatie waarbij de oppervlakte van een cirkel berekend moet worden, zonder dat we hierbij de waarde voor π kennen. De straal is r = 1, dus het oppervlakte van de cirkel is π.



Op de afbeelding is te zien dat we een vierkant R hebben met daarin de oppervlakte A. Dit wordt afgebakend door de functie $y = f(x) = \sqrt{1-x^2}$, de x-as en de verticale lijnen op volgende plaatsen: x = 0 en x = 1. De oppervlakte R van het vierkant is r*r = 1. De oppervlakte A (een kwadrant van de cirkel = π/4) zal berekend worden door middel van een simulatie.

Wanneer we een punt P willekeurig laten "vallen" op R, dan is het voor elk punt in het gebied R even waarschijnlijk dat het punt P op dat punt valt. De waarschijnlijkheid dat P op een punt in A valt is dus:

$$p = A / (r * r) = A / (1*1) = A = \pi/4 \quad (1)$$

We kunnen de waarschijnlijkheid p ook nog op een andere manier berekenen.Wanneer we punt P N-keer laten vallen en het daarbij K-keren binnen A valt, dan krijgen we:

$$p = K / N \quad (2) \text{ waarbij } N \rightarrow \infty$$

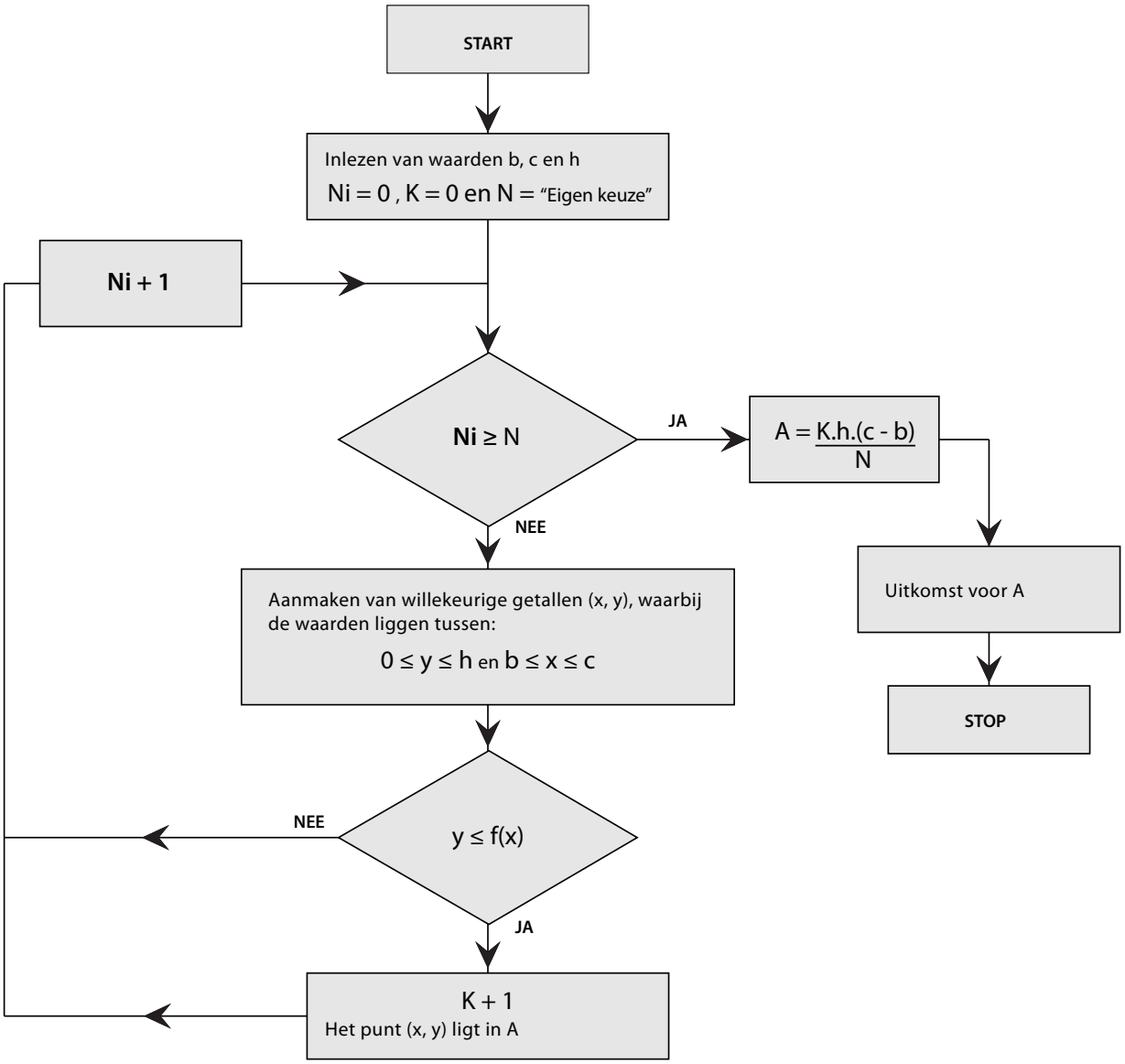
Uit (1) en (2) volgt dan:

$$\pi/4 = K/N$$

$$\pi = 4K / N = \frac{4 * \text{aantal keren dat P binnen A valt}}{\text{aantal keren dat P op R valt}}$$

2 ALLEN KENT, JAMES G.WILLIAMS, *Encyclopaedia of Microcomputers: Volume 7*, Marcel Dekker Inc., pg 441-444, 1991.

Onderstaande flowchart toont de werking van het algoritme:



Praktische uitwerking in PHP

Dit algoritme werd uitgewerkt in PHP. Om na te gaan in welke mate de verkregen waarden afwijken van de echte waarde van π werd omheen dit algoritme een lus geplaatst. Deze lus zorgt ervoor dat er 100-keren een waarde voor π berekend wordt.
Hieronder worden een aantal zaken verduidelijkt uit de PHP-code. De nummers verwijzen naar regelnummers van de pagina `montecarlo.php` [Bijlage 1].

10: Gekozen waarde voor N. Deze waarde zal in verder onderzoek verhoogd worden.
15: Start van de lus die 100-maal herhaald wordt.
18: Waarde voor K is nul, er zijn met andere woorden nog geen punten die in A liggen.
22: Start van de lus die N-keer herhaald wordt.
27 en **28**: Aanmaak van de x- en y-waarden. Dit zijn waarden die tussen 0 en 1 liggen, precies

tot 4 cijfers na de komma. Merk op dat voor de functie `mt_rand()` gekozen werd in plaats van `rand()`.
32: Nakijken of de coördinaten in het oppervlak A liggen. Dit doen we door de waarden van x en y in de functie in te voeren.
35: Wanneer dit het geval is wordt de waarde van K met 1 verhoogd (++ wil zeggen + 1).
41: De lus is voltooid en de waarde voor A kan berekend worden. Dit doen we door K te delen door N. Merk op dat we hierbij ook de waarden voor *(c – b)* en *h* mee rekenen. Omdat deze in ons geval alle twee 1 zijn heeft het echter geen nut deze hierin te verwerken.
51: De waarde voor A wordt toegevoegd aan een Array, die op het einde 100 waarden zal bevatten.
55: Er wordt geteld hoeveel keer een bepaalde waarde voorkomt. Op die manier wordt een nieuwe Array aangemaakt die enkel de unieke waarden bevat, met daarbij het aantal keren dat de waarde voorkomt in voorgaande Array.
57: De nieuwe Array wordt oplopend gesorteerd op de waarde voor A, met daarbij het aantal keren dat deze waarde voorkomt.
59: Deze lus wordt gebruikt om alle waarden uit de Array uit te lezen en weer te geven in het document.

Met dit stukje code zijn we in staat om een lijst van honderd benaderingen voor π te berekenen. In de voorbeeldcode zien we dat de waarde N = 2.500 gebruikt werd. Hiermee krijgen we inderdaad getallen die π benaderen, maar het resultaat is erg onnauwkeurig. Om te kijken of we het resultaat nauwkeuriger kunnen krijgen wordt de waarde voor N in enkele stappen verhoogd: N = 2.500, N = 10.000, N = 100.000 en N = 1.000.000.
De berekening worden drie keer uitgevoerd per waarde voor N. Vermits we gebruik maken van willekeurige getallen is hierbij elke uitkomst uiteraard verschillend.

Onderstaande tabel geeft de resultaten weer:

N	Hoogste waarde	Laagste waarde	Verschil	Gemiddelde	Verschil met π
2.500	3,1856	3,0896	0,0960	3,13760	0,00399
	3,2064	3,0800	0,1264	3,14320	-0,00161
	3,2024	3,0672	0,1352	3,13480	0,00679
10.000	3,1820	3,106	0,0760	3,14400	-0,00241
	3,1828	3,1024	0,0804	3,14260	-0,00101
	3,1816	3,1084	0,0732	3,14500	-0,00341
100.000	3,1560	3,1316	0,0244	3,14380	-0,00221
	3,1593	3,1243	0,0350	3,14180	-0,00021
	3,1535	3,1288	0,0247	3,14115	0,00044
1.000.000	3,1456	3,1375	0,0081	3,14155	0,00004
	3,1459	3,1371	0,0088	3,14150	0,00009
	3,1454	3,1369	0,0085	3,14115	0,00044

Opmerkingen en conclusies

Ter verduidelijking hierbij de echte waarde van π: 3,141592653.

We zien duidelijk dat de hoogste en laagste waarden bij een kleine waarde voor N relatief ver uit elkaar liggen.
We bekijken ook de getallen die verkregen werden na de berekening van het *verschil*. Ter vereenvoudiging nemen we hier per waarde voor N telkens het gemiddelde van de drie metingen. Bij N = 2.500 zien we dat het gemiddelde van het *verschil* 0,12 bedraagt. Bij N = 10.000 is dit 0,076, bij N = 100.000 is dit 0,028 en bij N = 1.000.000 is dit 0,0084. Dit toont duidelijk aan dat naarmate N groter wordt de gemeten waarden voor π steeds minder ver uit elkaar liggen.
Het *gemiddelde* vertelt ons in welke mate de getallen π benaderen en hoe nauwkeurig ze dat doen. Ook hier is duidelijk dat naarmate N groter wordt de nauwkeurigheid toeneemt. Waar er bij een kleine waarde voor N wel nog eens speling zit rond de tweede waarde na de komma, is het duidelijk dat deze naarmate N groter wordt steeds vaster op zijn plaats blijft staan. Meer nog, ook het derde cijfer na de komma neemt de juiste waarde aan, zoals uiteindelijk ook het vierde zal doen, enz....

Hiermee hebben we duidelijk gemaakt dat we dus best een zo groot mogelijke waarde nemen voor N. Toch moeten we wel stilstaan bij het feit dat dit veel meer rekenkracht vergt van de computer. Waar de berekening bij een waarde voor N = 2.500 slechts een fractie van een seconde in beslag nam, duurde de zelfde berekening bij een waarde voor N = 1.000.000 een goede 3 minuten en 13 seconden! En dit is slechts een eenvoudig voorbeeld. Men kan zich dus wel voorstellen dat het bij complexe simulaties wikken en wegen is tussen snel het resultaat kunnen zien of een nauwkeuriger resultaat te verkrijgen.
We mogen ook niet vergeten dat de kwaliteit van de RNG zeer belangrijk is. Deze moet namelijk zo goed mogelijk gelijk verdeelde getallen genereren. Dit wil zeggen dat elke waarde die hierbij voor x en y gecreëerd wordt tussen 0 en 1 moet liggen en dat het aantal keren dat het getal voorkomt ongeveer gelijk ligt met de andere gegenereerde getallen. Dit zorgt ervoor dat elk punt p in R een even grote kans heeft te bestaan.

6.3 Gebruik maken van een Online Random Number Service en de uitwerking van een interactief spelletje in Flash.

Het resultaat is te bekijken op: **http://cientouno.be/randy/** [3]

Dit is een kleine applicatie die in Flash uitgewerkt werd. De gebruiker krijgt bij een bezoek aan de website een afbeelding te zien van een monster (“Randy”), dat willekeurig samengesteld werd. De gebruiker kan een nieuwe verschijning van het monster creëren of kan met de huidige creatie verder gaan. De gebruiker wordt daarna voor de keuze gesteld het spel te spelen met of zonder willekeur. Het spel zelf is het bekende spelletje Pong, waarbij het gecreëerde monster dienst doet als bal die heen en weer gegooid wordt.

3 Om een optimale werking te verzekeren wordt aangeraden **Firefox** als webbrowser te gebruiken.

True Random Numbers ophalen van de online dienst RANDOM.ORG

Hiervoor werd beroep gedaan op een gratis PHP-Klasse die ontwikkeld werd door Jonathon Reinhart [4]. Deze klasse maakt de benodigde verbinding met de website RANDOM.ORG. Met een aantal verschillende methodes kunnen zo de gewenste getallen(reeksen) opgevraagd worden. Hierbij moet wel vermeld worden dat de hoeveelheid verzoeken beperkt is: bij teveel aanvragen zal de toelevering van getallen dan ook geblokkeerd worden.

Met het bestand `numbers.php` [Bijlage 2] wordt een XML-document aangemaakt. Dit zal later gebruikt worden in de Flash-applicatie. De nummers verwijzen naar de regelnummers van het bestand:

14: Laden van de klasse `RandDotOrg.Class.php`.
17: Het bestand zich laten voordoen als een XML-bestand.
24: De `methode get_integers(aantal_cijfers, minimum_waarde, maximum_waarde)` wordt aangesproken. Deze bezorgt ons een Array met 7 getallen, telkens bestaande uit een nul of één.
25: Gebruik maken van de zelfde methode, maar deze keer slechts één getal, gaande van 0 tot 5, als resultaat.
26: Zelfde als het voorgaande, maar dan met slechts 4 mogelijkheden.
29 – 48: Het XML-document wordt verder opgemaakt, waarbij telkens de benodigde waarde uitgelezen wordt uit de Array. Er zijn verschillende *nodes*: zes voor de armen, één voor de staart, één voor de ogen en één voor de mond.

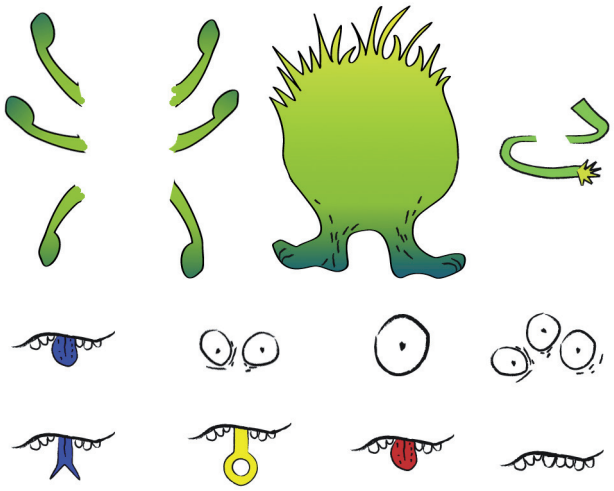
Het bestand `numbers.php` wordt op de server geplaatst, zodat door de Flash-applicatie het XML-document benaderd kan worden. Telkens dat dit bestand benaderd wordt zal een XML-document gegenereerd worden waarbij elke *node* een willekeurig getal bevat dat afkomstig is van RANDOM.ORG.

De verdere uitwerking van de figuur

Bij de verdere uitwerking in Flash wordt de XML ingeladen door de juiste url op te geven naar het bestand `numbers.php` op de server.
Er wordt steeds vertrokken van een basislichaam zonder armen, ogen, staart of mond. Hieraan worden dan met behulp van enkele eenvoudige *if-statements* de andere items toegevoegd. Zo wordt elke waarde voor een arm bekeken: is het een 1, dan wordt de arm toegevoegd, anders niet. Zo ook bij de mond, al zijn er hierbij 5 mogelijkheden waaruit minstens één gekozen moet worden. Eenmaal alle elementen bepaald zijn wordt de figuur zichtbaar gemaakt. Bij een klik op de knop “New Randy” wordt het XML-document opnieuw opgevraagd en begint de hele opbouw opnieuw. Op deze manier wordt telkens een “Randy” gecreëerd waarvan de opbouw onmogelijk te voorspellen is. Uiteraard is het aantal combinaties wel beperkt en zal na heel veel klikken een identieke “Randy” tevoorschijn komen.

4 Jonathon Reinhart - http://software.onthefive.com/RandDotOrg/

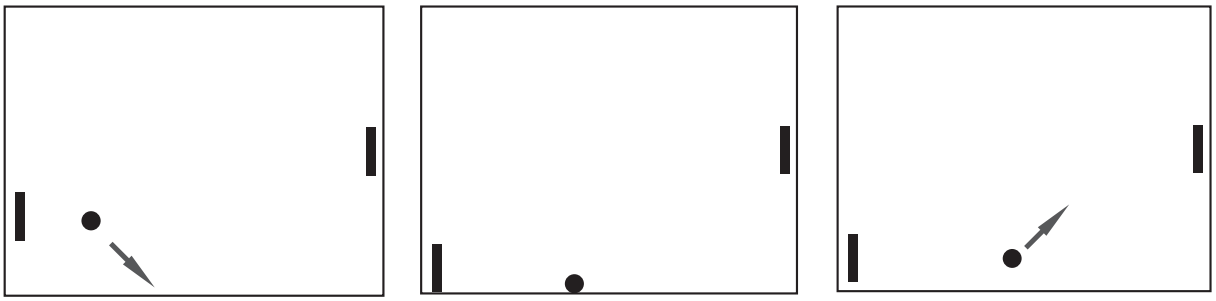
De verschillende lichaamsdelen van "Randy"



Uitwerking van het spelletje Pong

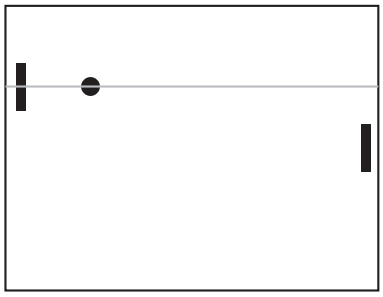
De opzet van dit spel is erg eenvoudig: het is de bedoeling een bal voorbij de andere speler te krijgen. Om te voorkomen dat de tegenstander wint is de speler dan ook genoodzaakt de bal die in zijn eigen richting komt af te weren.

In een **eerste versie** van het spel wordt geen gebruik gemaakt van de `random()` functie, er wordt met andere woorden geen PRNG aangesproken. De bal wordt voortbewogen door telkens dat een frame afgespeeld wordt een aantal pixels bij de x- en y-coördinaten te tellen. Doordat het aantal frames snel genoeg afgespeeld wordt krijgen we een vloeiende beweging van de bal te zien. Het aantal pixels dat bij de x en y waarden geteld wordt is voor beiden gelijk. Dit wil zeggen dat de bal in een hoek van 45° voort zal bewegen. Wanneer de bal bij de boven- of onderkant komt wordt het teken van de bijgetelde y-waarde (plusY) omgekeerd: op die manier zal de bal "terugkaatsen".



Een gelijkaardig iets gebeurt wanneer de bal de speler of de tegenstander raakt: hierbij wordt het teken van de bijgetelde x-waarde (plusX) omgekeerd.
De speler bestuurt het spel door met de muis in het kader te bewegen: de y-coördinaten van de muis worden gebruikt om de positie van de speler te bepalen. Is de speler niet op tijd om de bal af te laten kaatsen, dan raakt de bal de rechterkant, wordt een punt bij de score geteld van de

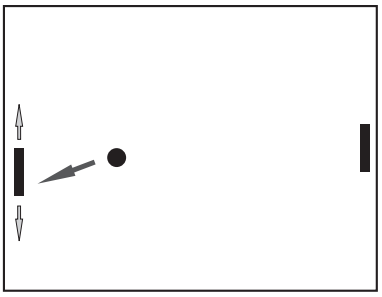
tegenstander en wordt de bal opnieuw vanuit het midden gelanceerd.
De tegenstander wordt bestuurd door de computer. Dit gebeurt door rekening te houden met de positie van de bal. Wanneer de bal omhoog gaat zal de tegenstander omhoog gaan en omlaag wanneer de bal naar beneden gaat.
Omdat de positie van de bal elk frame in de gaten gehouden wordt door de computer, zal ook de positie van de tegenstander elk frame exact bepaald worden, namelijk telkens op de zelfde hoogte als de bal. Op deze manier is het dan ook onmogelijk voor de speler om te winnen. De computer is veel te slim voor de speler: hij weet op elk moment waar de bal zich bevindt en kan hierdoor de speler telkens te slim af zijn door een simpele berekening.



TegenstanderY = BalY

Bij de **tweede versie** wordt op verschillende plaatsen gebruik gemaakt van de `random()` functie. Eerst en vooral wordt bij het starten van de bal gebruik gemaakt van variabele waarden die bijgeteld worden bij de x- en y-coördinaten. Dit zorgt ervoor dat de bal zich niet louter in een hoek van 45° graden kan verplaatsen, maar alle richtingen uit kan. Om het spel ook eerlijker te maken naar beide spelers toe wordt bij het lanceren van de bal ook willekeurig bepaald naar welke kant deze uit gaat. Telkens dat de bal de speler of de tegenstander raakt wordt een nieuwe willekeurige snelheid berekend, zodat de bal plotseling kan vertragen of versnellen.
Dit alles zou er echter nog steeds niet toe leiden dat de tegenstander niet kan verliezen: ook nu is de computer nog steeds in staat de exacte positie van de bal te berekenen in elk frame. Om de speler toch de mogelijkheid te geven het spel te winnen is het nodig de computer zich dommer voor te laten doen dan dat hij eigenlijk is. Dit doen we door hem te misleiden bij het plaatsen van de tegenstander tijdens het spel: telkens wanneer de bal zich boven of onder de tegenstander bevindt worden een aantal pixels bij zijn positie bijgeteld of afgetrokken. Dit gebeurt echter door gebruik te maken van de `random()` functie, waarbij het kan voorkomen dat er een grote of een kleine waarde toegevoegd wordt. Wanneer deze waarden enkele frames na elkaar klein zijn, dan zal de tegenstander niet op tijd tot bij de bal geraken, raakt de bal de linkerkant en heeft de speler een punt gescoord.

De positie van de tegenstander wordt mede bepaald door willekeur.



Door gebruik te maken van de `random()` functie werd het spel een stuk dynamischer gemaakt en is de speler in staat om het spel te winnen. Dit is uiteraard slechts een eenvoudig voorbeeld van hoe het gebruik van willekeurige getallen tot een aangenamer speelervaring kan leiden. Toch is dit de basis van hoe willekeur gebruikt wordt bij interactieve spelletjes.

7. Besluit

Het beoogde doel van dit werk was de lezer kennis te laten maken met het gegeven *Willekeurige Getallen*. Er werd getracht een zo volledig mogelijk beeld te geven van wat deze getallen zijn en waar ze zoal toegepast worden. Met de praktische uitwerkingen werd getracht enkele aspecten, zoals de aanmaak en het gebruik van deze getallen, duidelijker te illustreren.

Het is duidelijk dat er heel veel manieren zijn om willekeurige getallen aan te maken. Maar we weten ondertussen dat de kwaliteit van de reeks willekeurige getallen erg variabel kan zijn, alsook de snelheid waarmee deze aangemaakt wordt. De keuze tussen getallen afkomstig van een TRNG of een PRNG ligt geheel bij de gebruiker, maar is uiteraard ook weer afhankelijk van de toepassing. In praktijk zullen PRNGs meer gebruikt worden vanwege hun snelheid en eenvoud in toepasbaarheid. Wanneer een echte objectieve willekeur vereist is doet men beroep op een TRNG. Afhankelijk van de toepassing zal men voor een online dienst of een hardwarematige oplossing kiezen. Worden de willekeurige getallen gebruikt als sleutel bij het encrypteren van belangrijke gegevens, dan heeft het weinig zin beroep te doen op een online dienst, vermits de gegenereerde bits onderschept kunnen worden tijdens hun transport van de *server* naar de *client*. In dit geval is de hardwarematige RNG van idQuantique een betere oplossing. Voor dagdagelijks gebruik bij de meest uiteenlopende softwarepakketten volstaat het gebruik van een PRNG. Hierbij kan dan eventueel het huidige tijdstip als variabele input gebruikt worden om zo telkens een unieke –maar wel voorspelbare- reeks getallen te creëren.

Opmerkelijk is toch wel het feit dat men ondanks de zoektoch naar willekeur niet tevreden is met elke vorm van willekeur. Hiermee wordt geduid op de *Type I Error*: de melding die gegeven wordt wanneer een echte reeks willekeurige getallen een test op willekeurigheid niet doorstaat. Hoewel het hier om de zuiverste vorm van willekeur gaat, heeft men toch beslist hier geen gebruik van te maken. Dit vanwege de bruikbaarheid ervan. We verwachten een willekeurig gedrag, wat wil zeggen dat er genoeg variatie moet zitten in de getallenreeks, maar soms kan het voorkomen dat de echte reeks willekeurige getallen een sequentie vertoont van een aantal dezelfde cijfers na elkaar. Dit is perfect mogelijk -want alle getallen staan los van elkaar-, toch zal deze reeks in de praktijk minder bruikbaar zijn. Hoewel de kans op een *Type I Error* erg klein is (dit hangt natuurlijk nauw samen met de *strengheid* van de test), bestaat ze zeker en wordt ze afhankelijk van de toepassing al dan niet genegeerd.
Men is dus op zoek naar willekeur die op sommige momenten dan toch niet zo willekeurig mag zijn als men zou gedacht hebben.

De uitgewerkte toepassingen werden gecreëerd om een aantal aspecten van *Willekeurige Getallen* duidelijker te maken aan de lezer. Los daarvan hebben ze ook een erg verduidelijkende waarde gehad voor de auteur van dit werk: het stelde de auteur in staat om tijdens het uitwerken ervan een beter inzicht te krijgen in de aanmaak en het gebruik van willekeurige getallen. Hierbij werd alles stap voor stap uitgewerkt en getest, zodat proefondervindelijk het uiteindelijke resultaat bekomen werd. Bij de uitwerking van de webapplicatie rond de Linear Congruential Generator werd in eerste instantie enkel het niet-visuele gedeelte uitgewerkt. Omdat dit voor de gebruiker echter niet zo verhelderend is, werd een extra inspanning gedaan door ook het visuele gedeelte uit werken. Dit stelt de gebruiker in staat om sneller en eenvoudiger conclusies te trekken uit de resultaten van de LCG.
Het werd als een onmisbaar aspect gezien dat er in het praktische deel zowel gebruik gemaakt werd van een PRNG als van een TRNG. Daarom werd in de laatste uitwerking beroep gedaan op de dienst RANDOM.ORG. Hoewel het bij de uitgewerkte toepassing in wezen weinig uitmaakt of de getallen *True Random* zijn of niet, gaat het hier uiteraard om het idee en het feit dat een mogelijkheid getoond werd hoe *True Random Numbers* aangesproken kunnen worden. De uitwerkingen van het spelletje Pong toont de basis van hoe willekeurige getallen gebruikt kunnen worden om de spelervaring bij een computerspel een stuk interessanter te maken.

Bij de uitwerking van dit werk werden verschillende onderwerpen en elementen uit verwante onderzoeksgebieden bestudeerd. Hierbij werd door de auteur de materie in de mate van het mogelijke geïnterpreteerd en verwerkt. Op sommige momenten werd -gezien het complexe karakter van de materie- het bewuste onderwerp iets oppervlakkiger uitgewerkt. Met name de gedeelten waar de wiskunde van een hoger niveau is, werden door de auteur met veel inspanning, maar met helaas weinig resultaat, onderzocht. Los van een klein aantal aspecten van het werk die vatbaar zijn voor een gedetailleerder uitwerking, werden de overige aspecten zo grondig mogelijk uitgewerkt. Hierdoor kan de lezer zich een zo volledig mogelijk beeld vormen van wat *Willekeurige Getallen* nu juist zijn, hoe we ze kunnen aanmaken en wat we er kunnen mee doen.

8. Bronnenlijst

- ALLEN KENT, JAMES G.WILLIAMS, *Encyclopaedia of Microcomputers: Volume 7*, Marcel Dekker Inc., 1991.
- BENJAMIN JUN, PAUL KOCHER, *The Intel® Random Number Generator, White Paper,* Cryptography Research Inc., 22 April 1999.
- BO ALLEN, *Pseudo Random vs. True Random – A Simple Visual Example*, http://www.boallen.com/random-numbers.html
- CODIFIES, *PHP rand(0,1) on Windows < OpenSSL rand() on Debian*, http://cod.ifies.com/2008/05/php-rand01-on-windows-openssl-rand-on.html
- DONALD KNUTH, *The Art of Computer Programming - Semi-numerical Algorithm*. Vol 2. Chapter 3 Random Numbers, 1997.
- HOTBITS, http://www.fourmilab.ch
- HOWSTUFFWORKS, *How can a totally logical machine like a computer generate a random number?*, http://www.howstuffworks.com, 1998-2009.
- IAN STEWART, *Over sneeuwkristallen en zeebrastrepen: De wereld volgens de wiskunde*, Uniepers/Davidsfonds/Natuur & Techniek, 2002.
- IDQUANTIQUE, *Random Numbers Generation using quantum physics*, White Paper, http://www.idquantique.com/products/files/quantis-whitepaper.pdf, 2004.
- KENNY CHARMAINE, *Random Number Generators: An evaluation and comparison of Random.org and some commonly used generators*, Trinity College Dublin, April 2005.
- LAVARND, http://lavarnd.org
- LOUISE FOLEY, *Analysis of an online random number generator*, Trinity College Dublin, April 2001.
- MADS HAAHR, *Random.org*, http://www.random.org, 1998-2009.
- SCIENCE DAILY, *Pi Seems A Good Random Number Generator - But Not Always The Best*, www.sciencedaily.com, 27 April 2005.
- WIKIPEDIA, http://en.wikipedia.org/ → *Randomness, Random Number Generators, Pi, Blum Blum Shub, Modulo, Linear Congruential Generator, Kwantummechanica*.
- WILLIAM STALLINGS, *Cryptography and Network Security: Principles and Practices*, Pearson Education, 2006.

9. Bijlagen

9.1 Bijlage 1: montecarlo.php

```php
<?php

/*
* Eindwerk "Willekeurige getallen" - Frederik De Paepe
* Uitleg van de werking van de Monte Carlo Methode
* aan de hand van het berekenen van de waarde voor Pi
*
*/

$N = 2500;
$K = 0;

// dit is de lus die telkens een benadering van Pi berekend

for ($k = 0; $k < 100; $k++)
{

    $K = 0;

    // dit is de lus die N-keer herhaald wordt

    for ($i = 0; $i < $N; $i++)
    {

        // aanmaken van waarden voor x en y. Deze bevinden zich tussen 0 en 1

        $y = mt_rand(0,10000)/10000;
        $x = mt_rand(0,10000)/10000;

        //nagaan of de waarde binnen het oppervlakte ligt of niet

        if ($y <= sqrt(1 - pow($x, 2)))
        {
            // De waarde ligt binnen het oppervlak, dus K wordt met 1 verhoogd
            $K++;
        }
    }

    // Berekenen van het oppervlakte voor A

    $A = $K/$N;

    // We weten dat Pi/4 gelijk is aan het oppervlak, dus daarom:

    $pi = $A * 4; // de benadering van Pi

    // verdere bewerkingen dienen louter voor de weergave in een browser.

    (int)$pi_int = (int)($pi*10000);

    $data[] = $pi_int;

}

$count = array_count_values($data);

ksort($count);

while (list ($key, $value) = each ($count))
{
    echo ($key/10000) . ' &#8594 '. $value . "<br/>\n";
}

?>
```

9.2 Bijlage 2: numbers.php

```php
<?php

/*
*   Willekeurige Getallen - eindwerk MMP Arteveldehogeschool
*   Frederik De Paepe
*   3MMPa
*   The Randy Game
*   numbers.php => XML-file
*/


//laden van de Class gemaakt door Jonathon Reinhart
//gedownload van http://software.onthefive.com/RandDotOrg/
include_once('RandDotOrg.class.php');

//XML-header meegeven aan het document
header('Content-type: application/xml');

$rnd = new RandDotOrg();

//Een Array vullen met een getallenreeks die 9 karakters
//lang is, bestaande uit nullen en enen

$numbers = $rnd->get_integers(7, 0, 1);
$ogen = $rnd->get_integers(1, 0, 5);
$mond = $rnd->get_integers(1, 0, 3);

//opmaak van het XML document
echo '<?xml version="1.0" encoding="UTF-8"?>'."\n";

echo '<randy>'."\n";

echo "\t".'<romp>'."\n";

echo "\t\t".'<arm1>'.$numbers[0]."</arm1>\n";
echo "\t\t".'<arm2>'.$numbers[1]."</arm2>\n";
echo "\t\t".'<arm3>'.$numbers[2]."</arm3>\n";
echo "\t\t".'<arm4>'.$numbers[3]."</arm4>\n";
echo "\t\t".'<arm5>'.$numbers[4]."</arm5>\n";
echo "\t\t".'<arm6>'.$numbers[5]."</arm6>\n";
echo "\t\t".'<staart>'.$numbers[6]."</staart>\n";

echo "\t</romp>\n";

echo "\t".'<hoofd>'."\n";
echo "\t\t".'<ogen>'.$ogen[0]."</ogen>\n";
echo "\t\t".'<mond>'.$mond[0]."</mond>\n";
echo "\t".'</hoofd>'."\n";

echo '</randy>'."\n";

?>
```

9.3 Bijlage 3: Logboek

Datum	Actie	Inhoud	To do	Opmerkingen
09/10/2008	Briefing 1	afspraken rond keuze eindwerkonderwerp	Onderwerkkeuze uitwerken op basis van briefing	Deadline 3 november
18/11/2008	gesprek externe mentor	Een kennismakingsgesprek met mijn externe mentor		
14/11/2009	interne mentor	ondertekenen van contract door interne mentor		na de examens eens samenkomen
22/01/2009	Bib	Boeken ontlenen uit bibliotheek Universiteit Gent	boeken doornemen op zoek naar bruikbaar materiaal	
16/02/2009	Gesprek interne promotor	Afspraken over communicatie onderhouden, bespreken inhoud eindwerk	Eerste pagina's neerschrijven, informatie inwinnen bij Dhr Neuttiens	
18/03/2009	Eerste inzending	Alles wat reeds uitgewerkt is doorsturen, dit zowel naar externe als interne promotor		Dit telt mee voor een tussentijdse evaluatie
31/03/2009	Mail uit	Contact opnemen met Niels De Paepe ivm illustraties die gebruikt zullen worden in de lay-out van het werk.	Nadenken over welke illustraties juist nodig zijn.	
03/04/2009	Mail in	Int. promotor stuurt mijn eerste inzending terug met daarin een aantal opmerkingen.	Opmerkingen overlezen en aanpassingen doorvoeren	
03/04/2009	Mail uit	Mijn reactie op de opmerkingen + de link naar een uitgewerkte online applicatie die deel uitmaakt van het praktische deel.		
14/04/2009	Mail in	Ext. promotor stuurt opmerking op mijn eerste inzending door	Opmerkingen en taalfouten verwerken	
20/04/2009	Mail uit	Doorsturen van de volledige inhoud van het werk naar beide promotors	Een deel uit het praktisch gedeelte en het besluit afwerken	
23/04/2009	Mail uit	Het werk wordt opgestuurd naar Jan De Wit, die het zal nalezen op zoek naar taalfouten		
25/04/2009	Mail in	Jan De Wit stuurt het werk terug door	Taalfouten in het werk verbeteren	
25/04/2009	Mail in	Een eerste lading illustraties van Niels De Paepe ontvangen	Illustraties nakijken en opmerkingen doorsturen.	
26/04/2009	Mail uit	Doorsturen van de laatste uitgewerkte delen naar beide promotors		De links naar de online uitgewerkte delen wordt meegestuurd
27/04/2009	Mail in	Int. promotor stuurt vorige inzending terug door.	Opmerkingen en taalfouten verwerken	

Datum	Actie	Inhoud	To do	Opmerkingen
27/04/2009	Mail in	Ext. promotor stuurt vorige inzending terug door.	De laatste aanpassingen doorvoeren en aan de lay-out beginnen.	
27/04/2009	Mail in	Niels De Paepe stuurt alle afgewerkte illustraties door.	De illustraties in de lay-out opnemen.	
29/04/2009	Mail uit	Doorsturen van een opgemaakte versie van het eindwerk in Pdf formaat naar beide promotors.	Alle gebruikte bestanden bij de praktische uitwerkingen verzamelen en op een cd-rom branden.	