Projet Fil Rouge : Générateur de nombres (pseudo)-aléatoires basés chaos

2020

BECHAY Michael
CHAYÉ Guillaume
GIRAUDEAU David
NGUYEN Sophie
RAKOTOSON MALALANIRINA Tahina Lauren

Classe 12

Table des Matières

I. Introduction

Afin de conclure cette année scolaire 2019-2020, nous avons pour mission de réaliser un projet dans lequel nous sommes amenés à mettre en pratique les connaissances acquises durant toute l'année. Les matières principales concernées sont : les mathématiques, l'électronique, l'informatique, et l'anglais. Chaque groupe doit choisir parmi deux sujets imposés par les professeurs, l'un basé sur électronique, l'autre sur l'informatique. Nous avons choisi pour notre part le sujet basé sur l'informatique. En effet, le sujet correspondait à nos préférences, et de plus cela permettrait d'améliorer nos connaissances dans ce domaine.

Le but de ce projet est de mettre en place un générateur basé chaos afin de produire des suites de nombres (pseudo)-aléatoires.
Le générateur doit être basé soit sur :
• un système physique ou électronique simple possédant une dynamique chaotique,
• une simple fonction chaotique mathématique.

Nous avons pris la fonction physique. En effet, nous la trouvions plus pertinente. Nous avons au fur et à mesure fait évoluer notre fonction et avons fait des tests que nous détaillerons par la suite.

II. Définitions

Au début, nous avons fait des recherches afin de trouver une fonction suffisamment pertinente pour la réalisation du projet; nous sommes tombés sur des fonctions/suites mathématiques chaotiques basé sur la congruence mais aussi des algorithmes mathématiques tels que Xorshift, Xorwow mais au fil du temps nous avons décidé de créer nous-même notre algorithme et ne pas intégrer une fonction mathématique.

Nous avons dû en premier lieu chercher des définitions :

Une graine aléatoire ou germe aléatoire est un nombre utilisé pour l'initialisation d'un générateur de nombres pseudo-aléatoires. Toute la suite de nombres aléatoires produits par le générateur découle de façon déterministe de la valeur de la graine.

Aléatoire signifie imprévisible, lié au hasard.

Une fonction pseudo-aléatoire est une fonction dont l'ensemble des sorties possibles n'est pas efficacement distinguable des sorties d'une fonction aléatoire.

Chaotique signifie désordonné, confus, anarchique.

Un générateur de nombres aléatoires est un dispositif capable de produire une séquence de nombres pour lesquels il n'existe aucun lien déterministe (connu) entre un nombre et ses prédécesseurs, de façon que cette séquence puisse être appelée « suite de nombres aléatoires ».

Une séquence pseudo-aléatoire est une suite de nombres entiers x_0, x_1, x_2, ... prenant ses valeurs dans l'ensemble M={0, 1, 2, ..., m-1}. Le terme x_n (n>0) est le résultat d'un calcul (à définir) sur le ou les termes précédents. Le premier terme x_0 est appelé le germe (*seed* en anglais).

I. Explication du programme

Le but de l'algorithme est de générer des octets pseudo-aléatoires. Au début de la génération, on donne une graine au programme et celui-ci va générer une liste totalement différente selon la graine. Cependant cette liste a une période; c'est à dire qu'au bout d'un moment, elle va se répéter. On recherche donc la période la plus grande possible. Notre algorithme a une période estimée d'au moins 65 011 464.

A. Modélisation de l'algorithme

Notre algorithme peut être assimilé à une boîte constituée de 64 cases dont seulement 32 sont remplies et 32 sont vides. La valeur retournée par l'algorithme correspond à l'état d'une portion de 8 cases parmi les 64. Entre chaque tirage, la totalité des cases est mélangée.
Le fait de se focaliser sur seulement 8 cases sur 64 crée un phénomène chaotique. En effet, localement la disposition des cases varie énormément et de façon très désordonnée.

Informatiquement, on peut représenter la boîte par un entier de 64 bits composé de 32 '0' et 32 '1'. Entre chaque tirage, on mélange les bits en faisant attention à conserver l'équilibre entre les 0 et les 1. On retourne une portion de 8 bits parmi les 64 de l'entier.

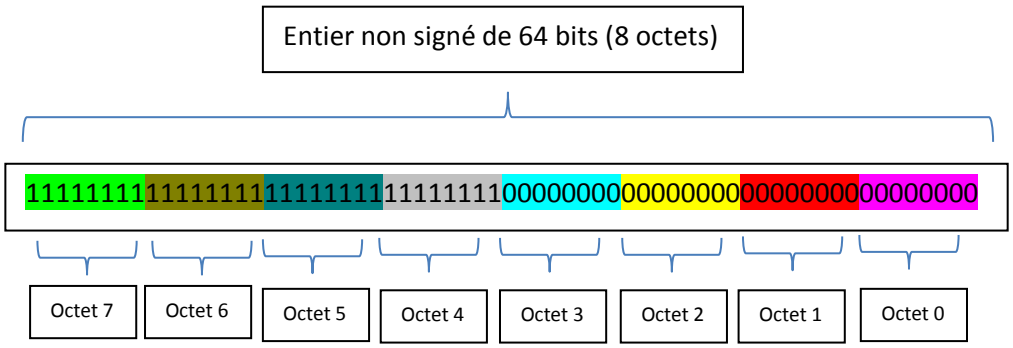
Le mélange est fait de façon à ce que les octets soient générés de façon équilibrée et soient compliqués à prédire. En effet, l'utilisateur connaît seulement un octet parmi les 8 utilisés par l'algorithme.

La période est extrêmement compliquée à calculer de façon mathématique, vue la manière dont fonctionne notre générateur. Cependant, nous avons pu la déterminer dans certains

cas en laissant tourner notre programme jusqu'à ce qu'il trouve une répétition. La période a été estimée pour certaines graines à 65 011 464. Dans d'autre cas elle est bien supérieure.

B. Les fonctions principales

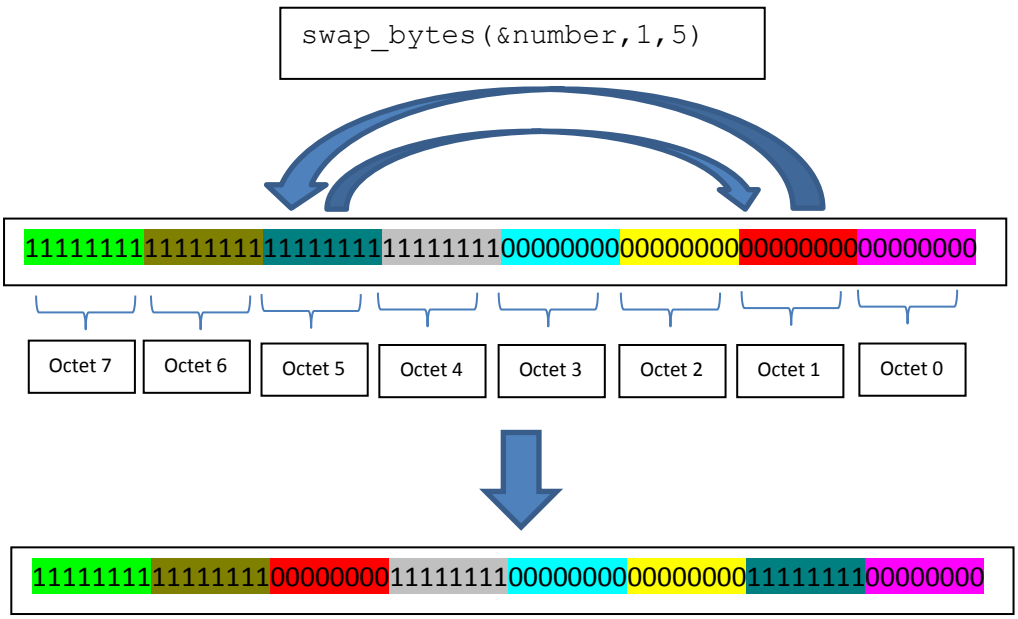
Pour réaliser les tirages, notre programme s'appuie sur un entier non signé de 64 bits.



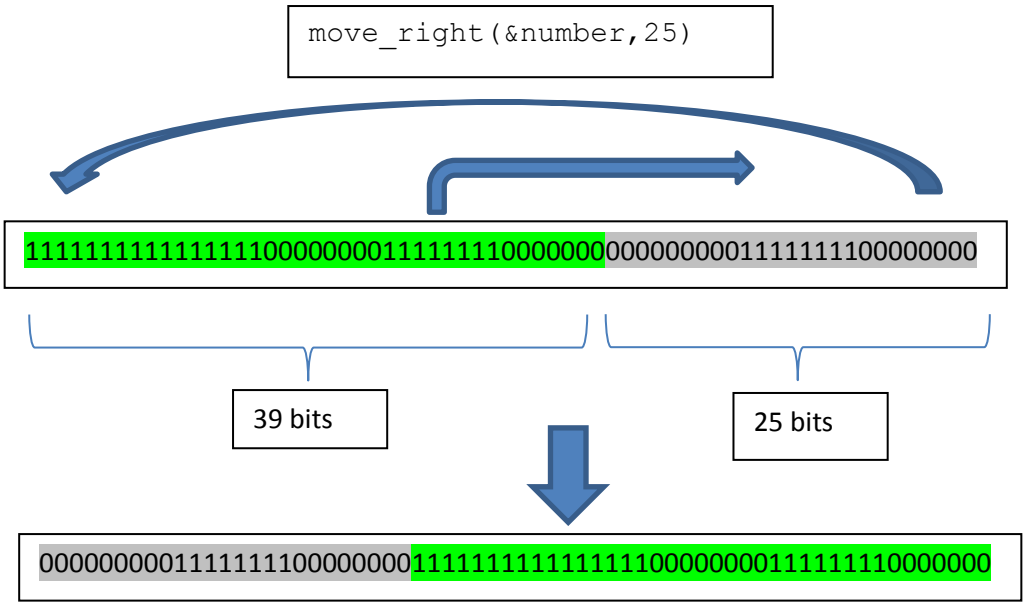
Le programme va mélanger les bits de ce nombre lors de chaque tirage à l'aide de deux fonctions : `swap_bytes` et `move_right`.

1. La fonction swap_bytes

`swap_bytes` est une fonction qui permet d'inverser l'ordre des octets dans un entier de 8 octets. Dans l'exemple en-dessous, l'octet 1 (rouge) prend la place de l'octet 5 et l'octet 5 la place de l'octet 1.



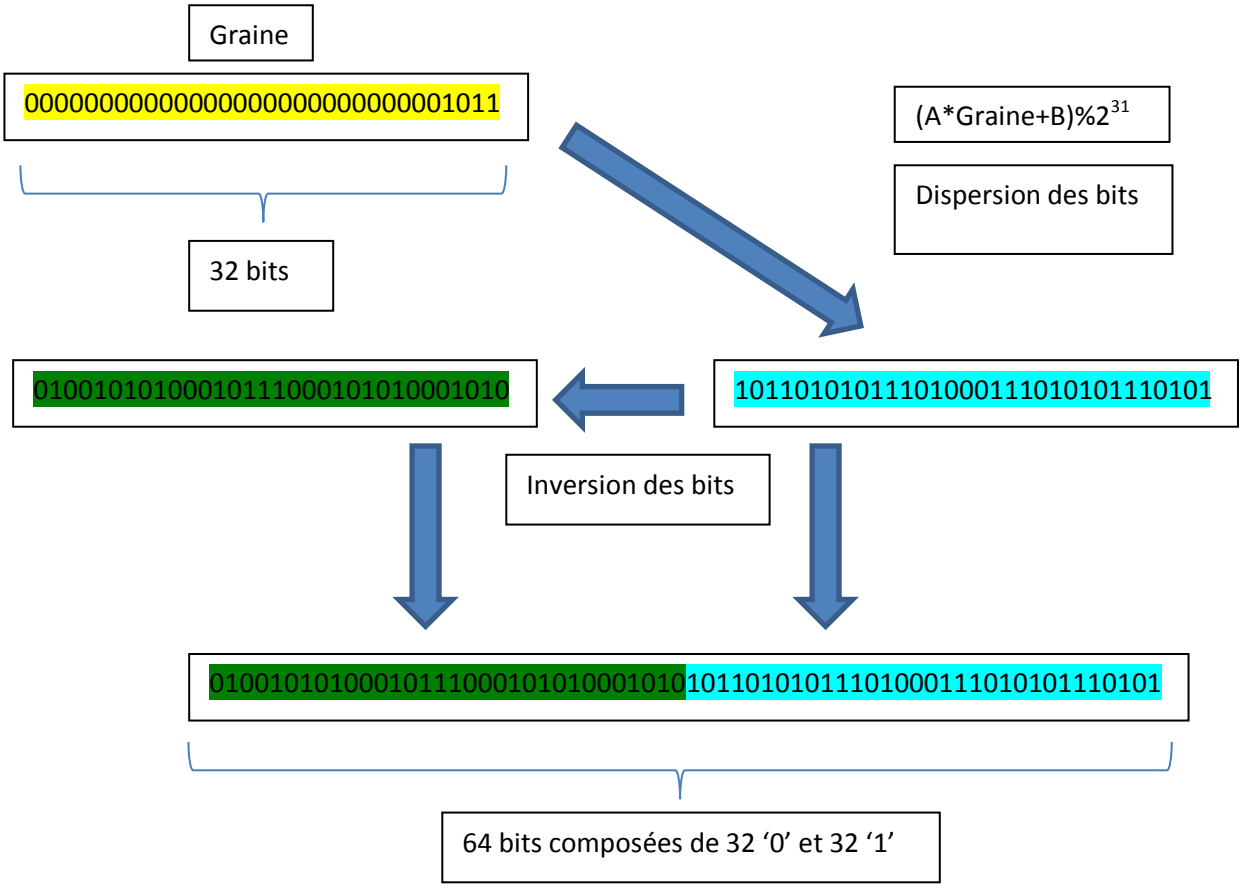
2. La fonction move_right



`Move_right` est une fonction qui permet de décaler d'un nombre de case voulu vers la droite. Plus précisément, cette fonction va extraire la partie a droite d'un nombre voulu (dans cet exemple 25). Puis elle va insérer cette partie extraite le plus à gauche et ensuite ramené à droite le reste de la représentation de notre chiffre. Cette technique à ses avantages car ce type d'opération fait partie des opérations simples à calculer.

C. Tirage d'un nombre pseudo-aléatoire

Pour générer des octets pseudo-aléatoires, il faut d'abord initialiser le générateur avec une graine. Chaque graine va produire une série d'octets différents de période très longue. Dans notre programme, la fonction `init_random` va permettre d'initialiser le l'entier de 64 bits sur lequel notre programme va s'appuyer pour générer des séquences d'octet. Cette fonction prend en argument une graine entière non signée de 32 bits. L'entier de 64 bits va être très différent selon les graines passées en argument.

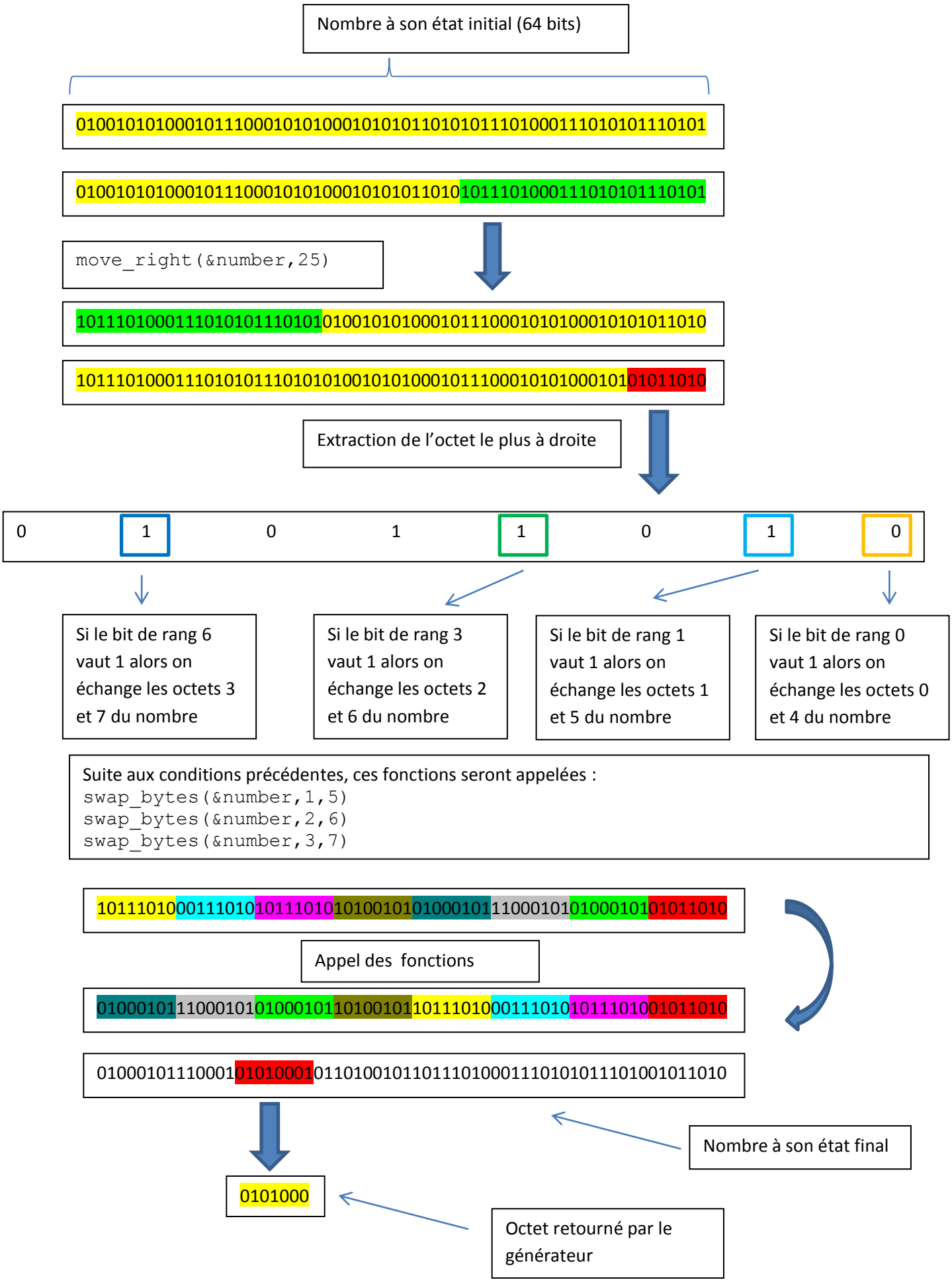


Comme vous pouvez le constater, la fonction va premièrement changer la valeur de la graine à l'aide d'une fonction chaotique : il s'agit de l'application linéaire congruentielle :

$$f(x) = (Ax + B)\%M$$

Nous avons choisi de faire cette transformation pour résoudre un problème de notre générateur pseudo-aléatoire. En effet lorsque les graines se suivaient, les premiers octets générés par notre programme se ressemblaient car les entiers de 64 bits étaient quasiment identiques. Pour résoudre ce problème, nous avons rajouté cette transformation qui permet de générer un entier de départ très différent selon les graines. Une fois cette transformation faite, on affecte à l'entier de 64 bits le résultat. A ce stade seulement 32 des 64 bits sont remplis. On complète les 32 bits restant de gauche avec l'inversion bit à bit des 32 bits de droite. De cette manière, on obtient un entier de 64 bits ayant bien 32 '0' et 32 '1'. Le système est donc très sensible aux conditions initiales

A chaque tirage, le programme va effectuer une transformation sur l'entier de 64 bits. Cette transformation s'appuie sur les fonctions présentées précédemment :


Nombre à son état initial (64 bits)
0100101010000101110001010100010101011010101110100011101010111010 1
0100101010000101110001010100010101011010101110100011101010111010 1
move_right(&number,25)
1011101000111010101110101010010101000101110001010100010101011010
1011101000111010101110101010010101000101110001010100010101011010
Extraction de l'octet le plus à droite
0 1 0 1 1 0 1 0
Si le bit de rang 6 vaut 1 alors on échange les octets 3 et 7 du nombre
Si le bit de rang 3 vaut 1 alors on échange les octets 2 et 6 du nombre
Si le bit de rang 1 vaut 1 alors on échange les octets 1 et 5 du nombre
Si le bit de rang 0 vaut 1 alors on échange les octets 0 et 4 du nombre
Suite aux conditions précédentes, ces fonctions seront appelées :
swap_bytes(&number,1,5)
swap_bytes(&number,2,6)
swap_bytes(&number,3,7)
1011101000111010101110101010010101000101110001010100010101011010
Appel des fonctions
0100010111000101010001011010010110111010001110101011110100101101 0
0100010111000101010001011010010111011100011101010111010010110 10
Nombre à son état final
0101000
Octet retourné par le générateur

La transformation se fait en plusieurs étapes (dans la fonction random_uint8) : décalage des bits vers la droite avec la fonction move_right, puis échange d'octets si certaines conditions sont respectées avec la fonction swap_bytes. Ensuite la fonction retourne 8 bits parmi les 64.

La complexité en temps de la fonction random_uint8 est dans le pire comme dans le meilleur cas de 0(1). Lorsqu'on réalise n tirages, la complexité totale sera donc linéaire.

La fonction random_uint8 retourne des entiers entre 0 et 255 dont le nombre d'apparitions sur un nombre élevé de tirage (au moins 10 000) est très proche. La moyenne de tous les entiers générés tend vers 127,5. La variance du nombre d'occurrence de chaque valeur est très faible.

Nous avons fait passer à notre programme le test de NIST dans plusieurs conditions. D'abord des tests de fréquences que l'algorithme a passées avec succès. Puis d'autres tests plus poussé pour analyser les séries de bits générées. Ces résultats étaient dans l'ensemble assez positifs même si certains tests ont échouée.

La fréquence d'apparitions des octets et des bits est excellente. La visualisation en image noir et blanc et couleurs permet bien de vérifier le côté aléatoire de notre générateur.

II. L'interface graphique et l'affichage

Nous avons utilisé le langage python, plus précisément l'interface Tkinter pour l'affichage des valeurs chaotiques et des graphiques pour mieux visualiser le phénomène.

Tkinter nous a permis de créer une interface qui permet d'afficher toutes les informations liées à notre programme, c'est aussi la seule interaction possible entre le programme et l'utilisateur.

L'interface graphique rend l'utilisation du générateur plus facile et agréable pour un utilisateur. Le programme python se charge de lancer la génération des nombres. En effet, il lance le programme C puis récupère les informations.

III. Organisation du travail de groupe

Tout au long du projet, nous nous sommes organisées afin de pouvoir rendre notre travail avant la date limite. Nous nous sommes concertés dès le premier jour afin de pouvoir choisir

le système (physique ou mathématique) et aussi de mettre en place nos idées. Nous avons pu discuter entre nous de chaque idée qui nous venait en tête qu'elles valent la peine ou non, sans quoi cela ne nous aurait pas fait avancer dans le projet.

Nous avons eu chacun un rôle important dans la mise en place du projet, même si certaines personnes du groupe ont eu plus de facilité à comprendre que d'autres. Le fait que nous soyons cinq dans le groupe nous a permis d'être actifs afin que nous nous empêchions une quelconque déconcentration. Nous avons mis en avant nos connaissances, reliées au sujet qui nous a été imposé.

- Guillaume : Chef de groupe. Apport d'idées au projet, distribution des tâches, connaissances avancés sur le projet, recherche d'informations, rédaction et mise en page du rapport, montage de la vidéo.
- Michael : Apport d'idées au projet, recherche d'informations, rédaction du rapport, script de la vidéo, enregistrement de la vidéo.
- David : Apport d'idées au projet, recherche d'informations et de modèles concluants, rédactions du rapport, script de la vidéo, enregistrement de la vidéo.
- Lauren : Apport d'idées au projet, recherche d'informations, rédaction du rapport, script de la vidéo, enregistrement de la vidéo.
- Sophie : Apport d'idées au projet, recherche d'informations, rédaction du rapport, enregistrement de la vidéo.

IV. Difficultés rencontrées

Nous avons eu en tête l'idée de programmer notre algorithme. Au cours du projet, nous avons eu des difficultés afin de déterminer si notre algorithme était bien chaotique comme nous le voulions.

Nous avons donc calculé la variance des occurrences, la période, ce qui constituent des éléments importants à la mise en place d'un phénomène chaotique. M. François nous a ensuite conseillé de tester les résultats obtenus avec notre algorithme dans le test NIST. Nous avons eu quelques difficultés lors de l'exécution du test ; néanmoins, il s'est avéré concluant : nous avons obtenu des résultats positifs lors de ce test.

Conclusion

Ce projet fil rouge nous a permis de nous perfectionner en C, notamment sur la manipulation directe des bits. Egalement en python avec l'utilisation de l'interface Tkinter encore inconnue pour la plupart d'entre nous avant ce projet. Le projet nous a permis aussi d'améliorer notre maîtrise des classes en python.

Ainsi, nous pouvons voir que ce projet a été le premier projet "hybride" de l'année où nous avons pu programmer à la fois en langage C et en langage Python, ce qui fut très intéressant.

De plus, le travail de groupe nous a permis d'être efficaces puisque nous nous répartissions nos tâches au fur et à mesure de l'avancement. Nous avons pu mettre toutes nos idées en place en discutant dessus et avons pris le temps de nous organiser tout en tenant compte des difficultés et des points forts de chacun. Nous n'avions que cinq jours pour réaliser ce projet, donc il a fallu commencer dès le premier jour sans nous précipiter et bâcler le projet. Ainsi, nous avons pu achever et finir le projet Fil rouge dans les délais.

Sources

Définition du chaos :

https://www.youtube.com/watch?v=m923FsfhNYE

Opérations liées aux bits :

https://www.tutorialspoint.com/cprogramming/c_bitwise_operators.htm

Exemple de table de vérité :

https://fr.wikipedia.org/wiki/Table_de_v%C3%A9rit%C3%A9

Modèles qui nous ont inspiré :

https://en.wikipedia.org/wiki/Xorshift

Mots clés en C :

https://koor.fr/C/Tutorial/Qualificateurs.wp

Explication de fonction pseudo aléatoire

http://www.numdam.org/issue/MSM_1962__153__3_0.pdf

Test Nist:

https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software