VEILLE DE SPÉCIALITÉE

CLYDE BARRULL - CPI A2 INFO

Comment fonctionne l'aléatoire dans les jeux Pokémon de 3ème génération et comment la manipuler ?

Table des matières

En plus de ce document, vous trouverez trois annexes jointes :

- Un programme python permettant de simuler le GLC de Pokémon Rubis et Saphir et Emeraude (Utilisez la fonction GLCRS ou GLCE avec comme paramètres les valeurs d'intervalle des termes que vous souhaitez obtenir.)

- Un Excel contentant les 11000 premières valeurs du GLC de Pokémon Rubis, Saphir et Emeraude, une page pour la distribution mâle-femelle, et une page pour déduire la nature d'un Pokémon via son PID.

INTRODUCTION

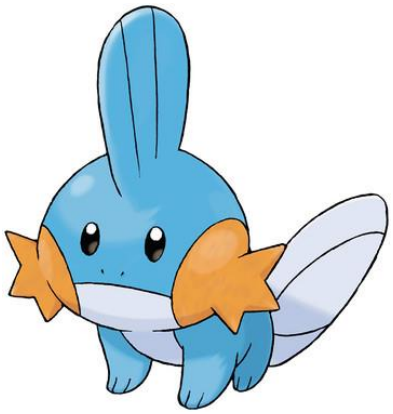
L'aléatoire est un sujet extrêmement complexe, reproduire le hasard de manière mathématique semble impossible car cela va à l'encontre même de la définition du hasard, quelque chose qu'on ne peut pas reproduire ou prédire. Malgré cela, les jeux incorporent des notions de hasard depuis des centaines d'années, un jeu de dé par exemple est un exercice d'aléatoire avec six résultats possibles.

Depuis une trentaine d'année, notamment avec l'arrivée de Diablo et ses récompenses aléatoires lorsque l'on défait un monstre, les jeux vidéo incorporent eux aussi des parts d'aléatoires au sein de leur conception. On parle alors du concept de « RNG », c'est-à-dire Random Number Generator ou « Générateur de nombres aléatoires ». Pour faire plus clair il est question de générer à chaque tirage un nombre qui semblerait aléatoire pour émuler cette sensation de hasard, cependant ceux-ci ne sont pas vraiment objet à l'aléatoire car dans un ordinateur, le hasard n'existe pas ! On parle alors de « Pseudorandom number generator » pour un processus qui vise à simuler l'aléatoire.

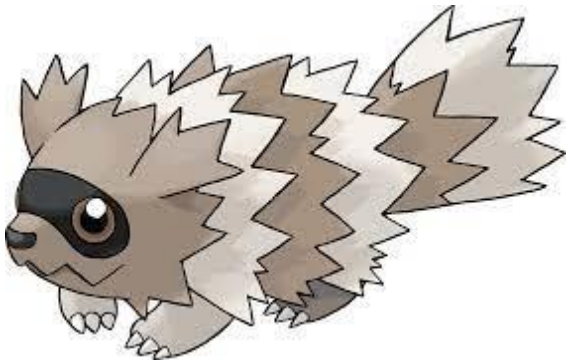
Pour prendre un exemple particulièrement ancien, le premier Final Fantasy (série désormais culte de Square Enix) faisait appel à une liste de 256 valeurs stockées dans la mémoire morte (ROM) de la console avec un index pointant sur une valeur, lorsque le joueur se déplaçait les mouvements qu'il entreprenait faisait changer l'index et l'on se retrouvait avec une certaine valeur, qui provoquait ou non l'affrontement contre un monstre.

La faille ici est bien évidemment qu'un joueur qui redémarre sa console (pour remettre à zéro le contenu de la mémoire morte) et fait une certaine séquence de pas pourra manipuler l'aléatoire sans difficulté. Si celui-ci répète la séquence qu'il avait fait précédemment, il lancera à coup sur un affrontement contre le même type de monstre qu'auparavant.

Dans l'optique d'aller plus loin dans l'étude des nombres pseudos aléatoires et de leur application dans les jeux vidéo, nous traiterons dans ce document le fonctionnement de l'aléatoire des jeux Pokémon de 3ème génération, c'est-à-dire Pokémon Rubis, Saphir et Emeraude (nous exclurons les versions Rouge Feu et Vert Feuille car largement différente sur le point de vue de la génération d'aléatoire.)



A gauche, un Pokémon appelé Gobou et à droite une autre espèce appelée Zigzaton, ils nous serviront fréquemment d'exemples dans ce document pour expliquer divers concepts.



Dans un premier temps, nous allons étudier le fonctionnement d'un générateur congruentiel linéaire, nous détaillerons ensuite comment ce système et ces valeurs pseudos aléatoires impacte le jeu. On expliquera par la suite plusieurs méthodes de manipulation d'aléatoire sur les versions Saphir et Emeraude pour ensuite les appliquer dans une partie finale, qui sera le détail des expériences que j'aurais réalisé pour cette veille, sur plusieurs supports.

PARTIE I : COMMENT FONCTIONNE UN GÉNÉRATEUR CONGRUENTIEL LINÉAIRE

Origine du générateur congruentiel linéaire

Premièrement, nous devons expliquer comment ce processus est venu au monde. Introduit par Derrick Lehmer en 1968, c'est un algorithme permettant la génération de nombre pseudos aléatoires. La date peut sembler frappante, car à cette époque les ordinateurs comme on les connait n'existait pas encore et ce concept ne fut donc pas utilisé pleinement avant bien plus tard, quand des développeurs qui se retrouvait dans le besoin de générer une grande quantité de nombre paraissant aléatoire eurent besoin d'utiliser cet algorithme, plus par soucis de vitesse d'exécution que réellement par soucis de produire quelque chose dit « d'aléatoire ». Du à ce manque de rigueur, les premiers générateurs basé sur l'algorithme de Lehmer étaient pour la plupart complètement erronés. Aujourd'hui, c'est l'algorithme de génération de nombres pseudos

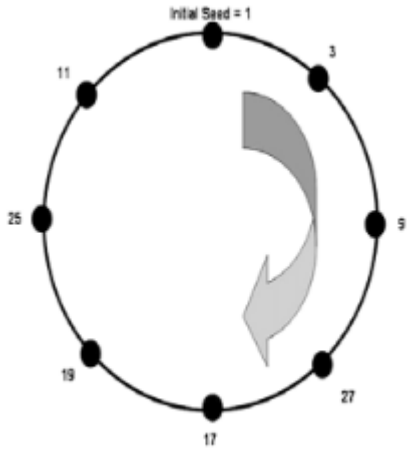
aléatoires le plus utilisé. (Ci-contre une photo de Derrick Lehmer prise entre 1942 et 1947.)

Bases du fonctionnement d'un Générateur Congruentiel Linéaire

Pour comprendre le GCL, il faut que l'on considère son fonctionnement comme celle d'une suite. L'algorithme stipule que les nombres pseudos aléatoires forment une suite dont chaque terme du précédent tel que :

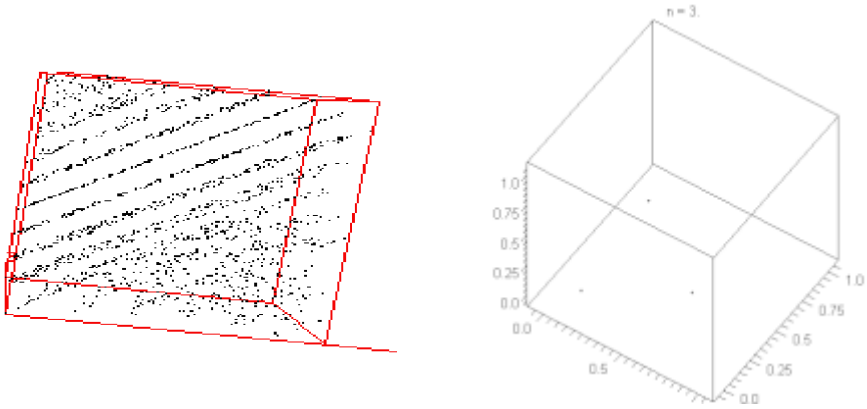
$$X_{n+1} = (aX_n + c)\ mod\ N$$

a étant le multiplicateur, c le pas (mis à 0 dans le cas d'un GLC classique) et N le module et a, c, N sont des constantes que l'on a sélectionnés. X_0 est la valeur que l'on appelle « seed initiale » et est la première valeur sortie de notre GCL. Pour utiliser le GLC efficacement, les paramètres en entrée doivent être choisis avec prudence, car l'utilisation de modulos limite le nombre de résultats possibles, et il est possible de se retrouver à court de nombre et que le GCL se répète (il finit une période après avoir visité chaque valeur et commence une nouvelle période avec ces mêmes valeurs.). Pour éviter ce problème, il est judicieux de choisir comme module un nombre assez grand pour obtenir une période qui est assez large et qui ne s'épuisera pas au bout de quelques requêtes. Il est aussi conseillé que le modulo et le multiplicateur soient des nombres premiers pour éviter que le modulo se retrouve divisible par le multiplicateur.



Ici, un schéma représentant une phase, la seed initiale est égale à un et les valeurs sont décidées les unes à la suite des autres de manière linéaire, si la seed initiale change c'est toutes les valeurs de la phase qui changent.

Le GLC possède des avantages et des désavantages notables. L'avantage le plus important réside dans le fait que les GLC sont rapides et demandent peu de mémoire, ce qui en fait un atout important quand on considère la mémoire des consoles sur lesquels fonctionnent les jeux que nous étudions dans ce document. La faiblesse la plus notable se traduit en un théorème (Birthday Theorem) qui stipule que la génération aléatoire de nombre sur 32 bits commencera à se dupliquer après $\sqrt{N} \approx 2^{16}$. D'autres problèmes comme l'apparition d'hyperplans provoquent une corrélation entre les résultats au bout d'un nombre significatif de résultats.



Ici, des cubes dans lequel les valeurs sont distribuées, au bout d'un nombre conséquent d'itérations, on voit des valeurs se corréler en formant des lignes puis former une distribution quasiment symétrique qu'une phase à une autre.

Générateur Congruentiel Linéaire spécifique aux jeux Pokémons de 3ème génération

Maintenant que nous avons vu mathématiquement comment fonctionne le GLC, il est temps de parler plus précisément du GLC utilisé dans les jeux Pokémon de 3ème génération.

La seed utilisée pour le GLC des jeux est une valeur décimale entre 0 et 0xFFFFFFFF, si le GLC à déjà été tiré (c'est-à-dire que le tirage au lancement du jeu à été effectué), la seed utilisée pour le tirage suivant est composée des 32 bits de poids faible (En outre les 4 hexadécimaux plus à droite) du résultat précédent déterminé par le GLC.

La formule du GLC pour les jeux Pokémon de génération 3 est la suivante :

Résultat = 0x41C64E6D * Résultat précédent (ou seed) + 0x00006073

Dans ces jeux, le GLC ne renvoie que les 16 premiers bits de la seed calculée au jeu pour les expériences aléatoires. Voici un exemple :

0x41C64E6D * 0x000005A0 + 0x00006073 = 0x171FB798593

La nouvelle seed est égale aux 32 derniers bits : 0xFB798593

La valeur renvoyée au jeu est égale aux 16 premiers bits de la nouvelle seed : 0xFB79

0xFB79 = 64377, le jeu reçoit cette valeur et l'attribue à une variable 32 bits destinée aux expériences aléatoires

Le modulo N change selon les cas (par exemple pour vérifier si un combat doit se lancer contre un Pokémon sauvage, le programme applique un modulo 2880 sur notre résultat et le compare avec une autre valeur, plus de détails la dessus plus tard.).

Rappel : 0xFFFFFFF, 0x41C64E6D sont des valeurs hexadécimales écrites en notation préfixée ! c'est un système de notation qui utilises 16 symboles, 0 à 9 et A à F. On lit les chiffres de l'entier de droite à gauche et sa valeur vaut la somme des chiffres affectés de poids correspondant au aux puissances successives du nombre 16, le 0x au début est à ignorer, il s'agit d'une notation pour identifier sans ambiguïté que notre nombre est noté en hexadécimal.

Les lettres A à F représentent les valeurs décimales 10 à 15.

Exemples : 0x00006073 = (3 x 16^0 + 7 x 16^1 + 6 x 16^3) = 3 + 112 + 24576 = 24691

0x000000FE = (14 x 16^0 + 15 x 16^1) = 14 + 240 = 254

Notre capacité à déterminer les résultats du GLC dépendent donc de la seed, si nous connaissons la seed, nous pouvons déterminer toutes les sorties suivantes du GLC. Dans la troisième partie nous irons plus en détails sur le fonctionnement de la seed initiale et nous expliquerons comment deux failles de conception (présentes respectivement dans Rubis/Saphir et Émeraude) nous permettent de documenter et de manipuler les comportements aléatoires des jeux.

Dans le cas de notre exemple les jeux Pokémons de 3ème génération utilisent un Générateur Linéaire Congruentiel sur 32-bit. Un tirage est fait à chaque frame et on sait que le jeu tourne à 60 frames par secondes, on a donc 60 tirages par secondes. Lorsque l'on est en combat, contre un Pokémon sauvage ou un dresseur rival, le jeu fait deux tirages par frame, on arrive donc à un total de 120 tirages par secondes en situation de combat. Cela peut sembler abstrait pour l'instant, mais une fois que nous aurons étudiés l'impact des valeurs données par le GLC, connaitre la routine de tirage du jeu nous permettra de déterminer des évènements redondants liées à une frame en particulière (restez patients tout cela arrive 😊.).

PARTIE II : COMMENT L'ALEATOIRE IMPACTE LE JEU

Dans cette partie je vais expliquer comment le GLC influence le jeu en lui procurant des valeurs pseudos-aléatoires qui sont, comme nous le verrons dans la troisième partie, prédictibles grâce à une faille dans le code du jeu.

L'évènement qui m'a convaincu de prendre ce sujet comme sujet de ma veille était une compétition dont le but était de finir Pokémon Emeraude le plus vite possible, un joueur appelé « Thetyrant 14 » expliquait qu'il connaissait par cœur la frame du jeu sur laquelle le GLC sortait une valeur qui l'arrangerait et qui lui conférerait un spécimen parfait de Gobou (l'un des 3 Pokémons que le joueur peut choisir au début de son aventure.), sous les yeux ébahis des commentateurs et

spectateurs, il réussit sa manipulation et obtint un spécimen parfait, lui donnant un avantage sur ses concurrents.

Influence de l'aléatoire en jeu

Concrètement, quels sont les effets notables de l'aléatoire sur les jeux Pokémon ? La réponse est extrêmement vaste, quasiment chaque aspect de Pokémon est régi au moins en partie par l'aléatoire. Pour ce document nous nous concentrerons sur la manière dont les rencontres avec un Pokémon sauvage fonctionnent, sur la génération de l'identifiant dresseur et comment les Pokémons sont générés grâce à des valeurs aléatoires données par le GLC. Même s'il existe d'autres expériences aléatoires (mécaniques en combat, génération d'un œuf de Pokémon, les machines du casino de Lavandia, l'utilisation de certains objets en combat.), Les traiter tous amèneraient à une répétition et une redondance entre les techniques et formules utilisées, nous allons donc les ignorer.

Avant même de rencontrer un Pokémon, il faut s'aventurer dans des hautes herbes, si un Pokémon vous saute dessus et lance un combat contre vous, c'est dû à résultat aléatoire. L'espèce et le niveau du Pokémon en question dépendront de valeurs demandées au GLC ainsi qu'à l'environnement dans lequel le joueur se trouve (pour prendre un exemple, si l'on s'aventure sur la route 104, nous pourrons croiser des Zigzatons, Chenipottes, Goélises et Nirondelles. Ce « pool » est défini par les développeurs du jeu et sert à établir une progression dans la puissance des créatures que le joueur rencontrera au cours de son aventure.). Ce n'est pas pour autant que rencontrer deux Zigzatons revient à reproduire la même expérience, car chaque Pokémon est généré de manière unique grâce aux valeurs du GLC (Oui, encore lui !).

Un dresseur dans des hautes herbes

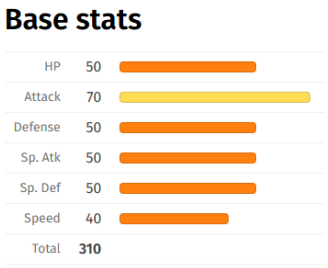
Grâce à l'aléatoire, chaque créature est unique !

Deux Pokémons de la même espèce n'auront donc pas forcément le même sexe, les mêmes faiblesses, forces, nature, talent et parfois cela peut-même s'étendre jusqu'à l'apparence du Pokémon qui change.

Chaque Pokémon possède plusieurs caractéristiques qui lui sont propres, certaines sont communes à son espèce comme les attaques qu'il peut apprendre (deux Carapuces différents apprendront l'attaque Charge au même niveau) ou le/les type(s) qu'il possède ainsi que des statistiques « de base », c'est-à-dire des statistiques minimales pas encore affectés par les facteurs aléatoires et uniques à chaque individu d'une espèce, ces statistiques sont composés :

- Du nombre de point de vies dont dispose le Pokémon.
- Sa statistique d'attaque physique (qui impactent les attaques s'apparentant à des coups de poings, de queues, de pattes etc.).
- Sa statistique de défense physique (la résistance aux attaques physiques donc).
- Sa statistique d'attaque spécial (qui impactent les attaques s'apparentant à de la magie ou de la manipulation d'éléments).
- Sa statistique de défense spéciale (la résistance aux attaques spéciales).
- Sa vitesse, lors d'un affrontement le Pokémon ayant la vitesse la plus haute attaque en premier.

Exemple : les stats de bases de l'espèce de Pokémon Gobou au niveau maximal (Niveau 100)



Maintenant que l'on a vu ce qui rend une espèce ce qu'elle est, il est temps de se pencher sur les statistiques individuelles, les IV (pour « Individual Values » ou Valeurs Individuelles), les natures et les talents :

- Il existe une IV pour chaque statistique qui vient s'additionner à la valeur de base de la statistique et qui dépend de la génération aléatoire de chaque individu. Chaque IV peut prendre les valeurs 0 à 31.
- Chaque Pokémon possède une nature depuis la troisième génération, il en existe un total de 25 et chacune d'entre elle est composé d'un bonus et d'un malus de 10% sur des statistiques différentes (omis le nombre de points de vies) qui viendront s'appliquer au calcul des statistiques du Pokémon. Par exemple, un Pokémon avec la nature « Brave » obtiendra une augmentation de 10% sur son attaque physique totale au détriment d'un perte de 10% sur sa vitesse totale.
- Les Pokémon possèdent tous un talent, une sorte de compétence passive qui les définit, chaque Pokémon peut posséder un seul talent ou deux talents différents.
- Un octet dans sa data qui détermine la distribution mâle-femelle de l'espèce

(Deux spécimens de Gobou différent, celui de gauche possèdent de meilleures IV en défense spéciale et en vitesse, tandis que l'autre possède de meilleurs IV en attaque et en défense.)

Dimension stratégique et combats

Les IV, la nature et le talent d'un Pokémon sont définis grâce aux valeurs sorties du GLC lorsque la créature est générée, plus de détails là-dessus dans la 3ème partie.

On comprend donc l'enjeu de l'aléatoire dans la création du meilleur spécimen possible. Un Pokémon dont toutes les IV sont à 31 est considéré comme un spécimen « parfait ».

De plus, chaque Pokémon est spécialisé pour quelque chose, si l'on souhaite un attaquant rapide mais fragile pour pouvoir défaire ses opposants avant d'encaisser des coups, il sera judicieux qu'il possède une nature qui l'avantage en attaque ou en vitesse au détriment de sa défense, un talent adéquat pour combler ses faiblesses peut aussi aider.

En situation de combat, le GLC travaille deux fois, à deux valeurs par frame. L'aléatoire est une composante importante des affrontements entre Pokémons, étant donné que les dégâts des attaques infligées ou subies par les Pokémon témoignent d'une variabilité dans les résultats, et des aspects aléatoires comme la possibilité d'infliger un coup critique ou de manquer son coup viennent rajouter une couche de travail au GLC que nous traiterons plus tard.

(Interface de combat des versions Rubis, Saphir et Emeraude présentant un affrontement entre un Gobou et un Zigzaton)

PARTIE III : COMPRENDRE COMMENT LES TERMES DU GLC SONT LUES ET COMMENT MANIPULER LA PSEUDO ALÉATOIRE

Maintenant que vous avez bien assimilées ce qu'implique l'aléatoire au niveau des mécaniques du jeu, nous pouvons repartir sur quelque chose de plus technique et expliquer comment les valeurs du GLC sont utilisés par le code pour déterminer des évènements aléatoires, en jeu et dans la génération de Pokémons.

Génération de l'ID Dresseur

Commençons par le commencement. Au lancement du jeu le joueur choisis s'il préfère incarner une fille ou un garçon, rentre son nom et l'aventure peut commencer ! Cependant, sans que le joueur ne le sache la première expérience aléatoire s'est déjà faite ! Le jeu demande à la création d'un personnage que le GLC fasse trois itérations. Les deux derniers tirages définiront l'identifiant dresseur du joueur, composé d'une partie visible en jeu sur la carte de dresseur du joueur (troisième terme), et d'une partie cachée au joueur (deuxième terme). Les deux valeurs étant générées l'une après l'autre, identifier quel terme de la suite du GLC correspond à notre ID publique nous permet de trouver l'ID secrète.

Ici, vous pouvez voir ma carte de dresseur sur le jeu Pokémon Emeraude
avec la partie visible de mon ID dresseur en haut à droite ! elle est notée 860F en hexadécimal.

A l'aide d'un programme python que j'ai réalisé pour cette veille et joint avec ce document, nous pouvons trouver notre ID secrète en repérant notre ID publique et en prenant la valeur précédente. On les écrit ensuite l'une à la suite de l'autre (secrète puis publique) et l'on obtient l'ID dresseur complète.

```
{1: ('0x000005a0', '0x0000', 0),
 2: ('0x171fb798593', '0xfb79', 64377),
 3: ('0x409ca7f1bc230a0a', '0xbc23', 48163),
 4: ('0x3056a24915b6b2b5', '0x15b6', 5558),
 5: ('0x5943555163b9d84', '0x163b', 5691),
 6: ('0x5b65be9f978a9a7', '0xf978', 63864),
```

Voici un exemple : Si la valeur de notre ID dresseur publique était 48163, on pourrait déduire que l'ID secrète est 64377. L'ID dresseur complète serait donc 6437748163.

Il est également important de noter que cette manipulation ne peut pas être faite sur la version Emeraude ! Sur cette version la partie visible et secrète sont générés à des moments bien distincts (à savoir le moment ou le joueur peut contrôler le personnage et le moment ou le jeu se démarre pour la première fois respectivement), pas l'un après l'autre comme dans Rubis et Saphir !

Rencontre des Pokémons sauvages

Intéressons-nous ensuite à la manière dont est faite en interne la mécanique de rencontre des Pokémons. Quand un joueur rentre dans des hautes herbes, trois tirages sont réalisés, voici un exemple (pour cet exemple la valeur de X_1 est arbitraire) :

(Rappel : Résultat = 0x41C64E6D * Résultat précédent (ou seed) + 0x00006073)

$X_1 =$ **0x000005A0**, cette première valeur n'est pas utilisée

X_2 = 0x41C64E6D * 0x000005A0 + 0x00006073 = **0x171FB798593**, non utilisé aussi

(Rappel : uniquement les 32 derniers bits du résultat sont utilisés comme seed du suivant et un chiffre hexadécimal est égal à 4 bits)

X_3 = 0x41C64E6D * 0xFB798593 + 0x00006073 = 0x**409CA7F1BC230A0A**, valeur utile

(Rappel : uniquement les 16 premiers bits de la nouvelle seed sont utilisés pour les expériences aléatoires)

La valeur que le programme utilise pour déterminer si un Pokémon va nous sauter dessus est donc BC23 (48163 en décimal), il calcule donc l'inégalité : $48163\ mod(2880) < 320$. Si l'inégalité est vérifiée vraie, un combat contre un Pokémon sauvage se lance ce qui n'est pas le cas dans notre exemple, notre résultat étant 2083 < 320, l'inégalité est fausse.

Dans le cas ou notre joueur était déjà présent dans l'herbe et se déplaçait simplement dans la broussaille, c'est le deuxième tirage (X_2) qui est utilisé pour déterminer ou non une rencontre avec un Pokémon sauvage.

Génération d'un Pokémon unique

Nous allons désormais traiter la manière dont le jeu génère un Pokémon. Les jeux de 3ème génération le font de deux manières différentes, la première méthode (méthode 1) est appliquée sur les Pokémons fixes, dont l'apparition est garantie. Par exemple, au début du jeu, quand il est demandé au joueur de choisir son premier compagnon pour son aventure parmi les trois choix qui lui sont proposés, ce sont des Pokémons fixes, si vous choisissez Gobou, vous serez garantis d'avoir Gobou. La deuxième méthode (méthode 2) est appliquée sur les Pokémon sauvages qui sortent des hautes herbes dont le fonctionnement a été expliqué précédemment. Voyons en détail comment ces méthodes fonctionnent en interne dans le jeu :

Pour la méthode 1, le jeu génère 5 valeurs à la suite sur une frame à l'aide du GLC. La première valeur est inutilisée, tandis que les deux suivantes vont composer le PID (Ou Pokémon ID), si cela vous dit quelque chose, c'est tout simplement car c'est le même principe que pour l'ID de dresseur ! Le PID est la valeur qui détermine le sexe, son talent et s'il est chromatique. La nature est définie différemment selon la méthode utilisée (1 ou 2).

Aparté : Un Pokémon chromatique est un Pokémon dont la couleur est différente des autres spécimens de son espèce, le jeu détermine si un Pokémon est chromatique si l'ID du dresseur et le PID du Pokémon sont identiques ! Ce phénomène est donc extrêmement rare dans le cas d'une aventure normale. Si l'on possède un Pokémon chromatique, on peut déduire son PID pour trouver l'ID secrète de son dresseur ! C'est la seule manière de déduire l'ID secrète d'un joueur sur la version Emeraude. (À gauche un Léviathor normal et à droite un Léviathor chromatique)

Sexe et talent du Pokémon généré :

Reprenons notre étude du PID, il est composé des 16 premiers bits de deux valeurs pseudo aléatoires qui se suivent donc 8 valeurs hexadécimales. Pour pouvoir lire les valeurs du PID nous devons le décomposer en 32 bits (un hexadécimal correspond à 4 bits).

Prenons comme exemple le PID 47A699F9, en binaire celui-ci devient :

01000111 10100110 10011001 1111100**1**

Les valeurs en rouge servent à déterminer le sexe du Pokémon et la valeur surlignée en vert à choisir son talent. Pour le talent, comme mentionné précédemment chaque Pokémon en a un ou deux différents qui prennent donc les valeurs 0 et 1.

Pokémon	Talent 0	Talent 1
Zigzaton	Ramassage	Glouton
Gobou	Torrent	Torrent

Voici un exemple concret, le dernier bit du PID étant à 1, si le Pokémon concerné était un Zigzaton il aurait le talent glouton, si c'était un Gobou le résultat n'aurait pas d'importance.

Pour le sexe du Pokémon, il faut utiliser l'octet de poids faible du PID ainsi qu'une table qui nous indique les distributions mâles/femelles d'une espèce.

Pour donner un exemple prenons le PID que nous avions détaillés précédemment, l'octet de poids faible est donc 11111001, soit en décimal 249. Nous allons nommer cette valeur Y.

La distribution est décidée par un octet dans la data d'une espèce :

- Si celle-ci est à 0 (00000000), les Pokémons de l'espèce sont tous des mâles
- Si celle-ci est à 254 (11111110), les Pokémons de l'espèce sont tous femelles.
- Si celle-ci est à 255 (11111111), le sexe du Pokémon est inconnu (Pokémons qui ne peuvent pas se reproduire).
- Les valeurs entre 1 et 253 (inclus) équivalents à un ratio mâle-femelle, Si la valeur Y d'un Pokémon est au-dessus ou égal à cette valeur le spécimen sera un mâle, si elle est plus petite le spécimen sera une femelle.

Pour des raisons de simplicité, les développeurs du jeu ont limité à 4 différentes valeurs l'octet qui gère la distribution mâle/femelle (en plus des valeurs 0,254 et 255 qui sont des valeurs particulières), les valeurs en questions sont :

- 31, Si Y < 31 le spécimen est une femelle (probabilité de 12.5% d'être une femelle)
- 64, Si Y < 64 le spécimen est une femelle (25%)
- 127, Si Y < 127 le spécimen est une femelle (50%)
- 191, Si Y < 191 le spécimen est une femelle (75%)

> (Si l'on réfléchit un peu, on se rend compte que dans le cas d'une distribution 50/50, sur un grand nombre spécimens nous aurons plus de mâles que de femelles, car la condition pour un mâle inclut aussi l'égalité.)

Voici un exemple avec deux Pokémons dont le PID est identique mais dont la distribution au sein de l'espèce est différente :

Nom de l'espèce	PID	Octet de poids faible	Octets de poid faible en décimal	Valeur de distribution mâle femelle	Pourcentage	Sexe du spécimen ?
Zigzaton	4AAFED37	37	55	127	12,5%	femelle
Gobou	4AAFED37	37	55	31	50%	mâle

Comme vous pouvez le voir, deux Pokémon n'auront pas forcément le même sexe même en ayant un PID identique.

Nature du Pokémon généré :

Déterminons ensuite la nature du Pokémon à l'aide de son PID, ceci n'est possible que sur des générations de Pokémon faites avec la méthode 1, nous verrons le fonctionnement en méthode 2 plus tard.

La méthode pour cela est relativement simple, il faut :

- Convertir le PID en décimal
- Prendre les deux derniers nombres de la valeur décimale
- Si celle-ci est supérieure à 24, soustraire 25, recommencer jusqu'à avoir une valeur qui se situe entre 0 et 24.
- Les natures sont au nombre de 25 (comme expliqué dans la partie 2) et sont stockées en mémoire dans un tableau, la valeur finale est l'index dans le tableau de la nature du Pokémon.

Quelques exemples réalisés sur Excel avec cette méthode :

Numéro	PID	PID converti en décimal	Valeurs utiles	Valeur définitive	Nature du Pokémon
1	55D77EFA	1440186106	06	6	Docile
2	A4C5AD96	2764418454	54	4	Mauvais
3	2740AB8E	658549646	46	21	Gentil
4	DECE514E	3738063182	82	7	Relax
5	2EF74571	787957105	05	5	Assuré
6	F9B69C38	4189494328	28	3	Rigide

(Si vous souhaitez connaitre la table des natures et faire des générations aléatoires, l'Excel est joint à ce document.)

IVs du Pokémon généré :

Nous en avons désormais fini avec la partie PID (nous la reverrons brièvement avec la notion de « Tentatives PID » quand l'on parlera de la méthode 2), nous allons désormais nous intéresser à la manière dont les IV sont générées aléatoirement, pas de panique, cela n'est pas si différent de ce que nous avons vu avec le PID.

Les IV sont TOUJOURS générées directement après le PID (ce sont les 2 valeurs restantes dans les 5 qui composent la génération d'un Pokémon, comme nous l'avions annoncé au tout début de cette partie). Le programme demande deux tirages au GLC, ces tirages sont ensuite convertis de l'hexadécimal au binaire et se décomposent de cette manière :

Premier tirage : 3056, en binaire cela correspond à

0011 0000 0101 0110

Deuxième tirage : 37E8, en binaire cela correspond à

0011011111101000

Les bits surlignées en rose ne rentrent pas en jeu dans la définition des IV.

Les bits surlignées en vert correspondent à la valeur de l'IV qui se rajoutent à la défense physique de base.

Les bits surlignées en rouge correspondent à la valeur de l'IV qui se rajoute à l'attaque physique de base.

Les bits surlignées en bleu correspondent à la valeur de l'IV qui rajoute des points de vie en plus de ceux de base.

Les bits surlignées en jaune correspondent à la valeur de l'IV qui se rajoute à la défense physique de base.

Les bits surlignées en gris correspondent à la valeur de l'IV qui se rajoute à l'attaque spéciale de base.

Les bits surlignées en bleu clair correspondent à la valeur de l'IV qui se rajoute qui se rajoute à la vitesse de base.

Différences entre la méthode 1 et 2

Maintenant que nous avons fini l'explication en détail de la génération de Pokémons avec la méthode 1, nous pouvons voir comment fonctionne la méthode 2. En vérité, il n'y a pas de différence sur la manière dont le PID et les IV sont générées. La différence principale se trouve dans la manière dont la nature est générée et dans l'ajout du concept de « Tentative PIDs ».

Dans la méthode 1, le jeu demandait au GLC 5 valeurs. La première était inutilisée, la deuxième et la troisième définissait le PID, et le quatrième et cinquième les IVs.

En méthode 2, le jeu génère au mieux 6 valeurs, mais ce chiffre augmente généralement de manière importante. Tout comme pour la méthode 1, la première est inutilisée, c'est sur la deuxième que les choses varient.

En effet, la deuxième valeur est utilisée pour choisir la nature du Pokémon généré, en prenant convertissant la valeur hexadécimale en décimale et en lui appliquant un modulo 25. Voici un exemple (la valeur X correspond la valeur du deuxième tirage) :

X = 3211, converti en décimale cela donne 12817

12817 mod 25 = 17

La nature du Pokémon est donc celle à la nature numéro 17 dans la table des natures (qui pour rappel peut être lue dans l'annexe Excel jointe.), Le Pokémon est donc de nature discrète.

Ensuite, nous utilisons les deux prochaines valeurs pour faire une « Tentative PID », c'est-à-dire créer un PID qui va passer une vérification pour confirmer sa cohérence avec la nature générée juste avant. Pour cela un PID est généré de manière classique, les deux valeurs sont mis bout à bout, la plus récente en poids faible, on applique ensuite à ce PID l'opération mod 25 et si la valeur finale correspond à la valeur de la nature, le PID est validé et est choisi comme celui du Pokémon généré et le programme passe aux tirages des deux prochaines valeurs pour définir les IV. Dans le cas contraire, le programme redemande deux valeurs au GLC et produit une nouvelle Tentative PID, le nombre d'appels au GLC peut donc grimper de manière impressionnante et dépasser les 6 valeurs minimales annoncées.

Si l'on utilise le programme Python pour voir les valeurs suivantes, on peut voir que les deux prochaines valeurs envoyées au programme par le GLC sont 37EF et 2127, la première

tentative PID sera donc 212737EF. Convertissons cela en décimal : 556218351, on applique le mod 25, 556218351 mod 25 = 1. La valeur ne correspond pas à celle de la nature de notre exemple (17), le programme refait alors une tentative PID. Pour votre santé mentale et la mienne, nous allons arrêter l'exemple ici, l'essentiel étant que vous ayez compris les deux fonctionnements distincts des méthodes 1 et 2.

Failles permettant la manipulation

La raison pour laquelle la génération d'aléatoire de cette génération de jeux est si bien documentée, c'est avant tout parce qu'elle présente des failles très facile à exploiter et qui rendent la manipulation de l'aléatoire possible sans se préoccuper avec des paramètres très difficiles à mettre en place.

Il existe deux failles importantes, l'une existe sur les versions Rubis et Saphir et l'autre sur la version Emeraude. Nous allons commencer par celle de Rubis et Saphir, qui est une faille due à des caractéristiques techniques de la cartouche de jeu.

Sur les versions Rubis et Saphir :

L'explication est simple, il est extrêmement compliqué de prédire les valeurs du GLC sans connaitre la première seed, celle générée à la frame 1 (ou 6 sur émulateur), au lancement du jeu. Cependant, la manière dont le jeu génère cette seed est en se basant sur l'Unix Time (ou heure Posix), c'est-à-dire la mesure de temps basée sur le nombre de secondes écoulées depuis le 1er janvier 1970 à 00:00:00. Cette valeur, on la récupère grâce à une horloge RTC présente dans la cartouche du jeu directement, et cette dite horloge opère grâce à une batterie ! Les jeux étant sorties en 2002-2004 selon les régions du monde, une grande partie des batteries présentes dans ses cartouches sont mortes depuis ! Et dans le cas ou le jeu n'arrive pas à récupérer l'heure Posix pour générer la seed de départ, il en utilise une par défaut, à savoir la seed 0x000005A0 (Qui correspond à la date du 1er janvier 2000 à 00:00:00).

Voici une photo de ma Nintendo DS Lite avec le jeu Pokémon Saphir inséré dedans, la pile de la cartouche est morte. La console nous affiche donc un message au lancement du jeu « la pile interne est épuisée ». Les batteries utilisées dans ces modèles de cartouche sont des batteries CR1616, qui sont censées durer entre 10 et 20 ans !

Sur la version Emeraude :

La deuxième faille, celle qui concerne Pokémon Emeraude, ne provient pas d'un défaut de conception de la cartouche du jeu mais directement d'un oubli des développeurs dans le code du jeu ! Lorsque le jeu démarre, le programme ne demande pas l'heure Posix à l'horloge RTC comme c'est le cas dans Rubis et Saphir, la valeur de la seed reste donc à 0 (0x00000000).

Connaître la seed de départ nous permet de déterminer l'ensemble des valeurs suivantes calculées par le GLC ! Nous connaissons également la routine du CLG (Pour rappel, 1 tirage par frame, 2 en combat, 5 durant la génération d'un Pokémon etc...), il est donc possible de déterminer la frame précise qui garantira un PID particulier ou une rencontre avec un Pokémon dans des hautes herbes !

Les listes des termes de la suite générée par le GLC pour Rubis/Saphir et Emeraude respectivement peuvent être déterminés avec le programme python réalisé en annexe, en utilisant les fonctions GLCRS (pour Rubis/Saphir) et GLCE.

C'est tout pour la partie théorique, la partie suivante sera donc un détail d'expériences réalisées sur émulateur et console réelle.

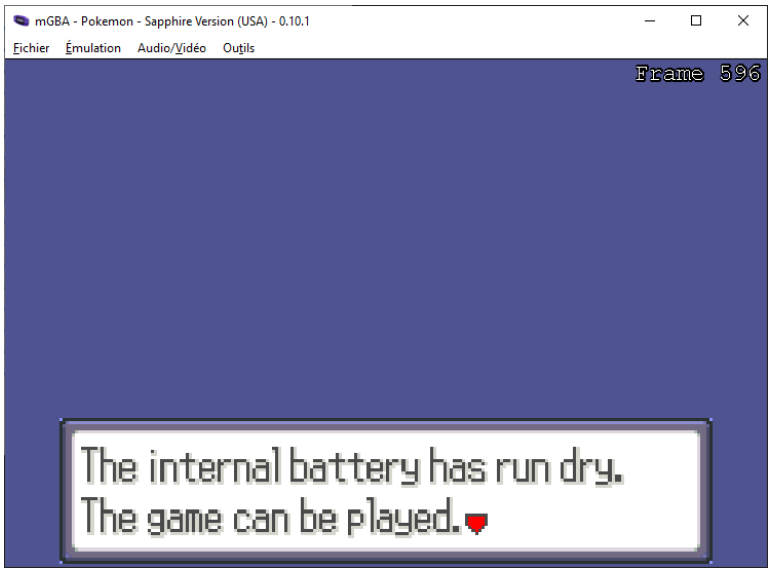
PARTIE IIII : MANIPULATION ET RÉSULTATS DE LA RÉALISATION

Dans cette partie, je vais décomposer, détailler et expliquer une manipulation réalisée sur Pokémon Saphir en intégrant des captures d'écran réalisée par mes soins. Pour cela je vais me baser sur le GLC de Rubis/Saphir avec la seed originelle 5A0 (celle de Rubis/Saphir dans le cas ou la pile de la cartouche est morte) et utiliser le programme Python que j'ai réalisé pour déterminer l'ID secrète de mon personnage, les statistiques du Pokémon de départ et assurer que celui-ci soit chromatique !

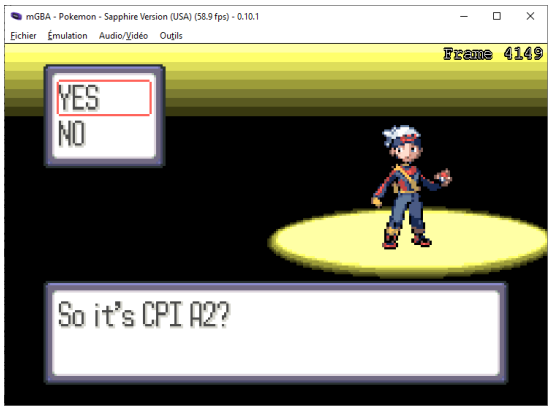
Première partie : Manipulation de l'ID dresseur

Pour cela, je vais exécuter la version américaine du jeu (pour des raisons d'accès légal à la rom) sur l'émulateur mGBA avec un script LUA me permettant de connaitre le nombre de frames écoulées depuis le lancement du jeu ! Ma manipulation s'est déroulée ainsi :

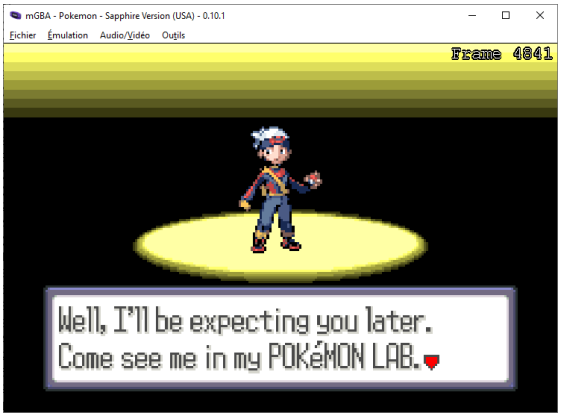
- Je démarre le jeu et m'assure que le message indiquant que la pile est morte s'affiche (cela est nécessaire pour confirmer que la seed initiale soit bien 0x000005A0.).



- Je crée une nouvelle partie et rentre mon nom, j'ai choisi le nom « CPI A2 » pour l'occasion !



- Lorsque j'arrive à la dernière boite de texte de l'introduction, je mets le jeu en pause et je note la frame sur laquelle j'appuie sur A, cela va fermer la boite de dialogue et va provoquer la génération de mon ID dresseur. Sur la capture d'écran vous pouvez voir que j'appuie sur A à la frame 4841, l'input à une frame de retard nous prenons donc la frame précédente (4840) comme référence



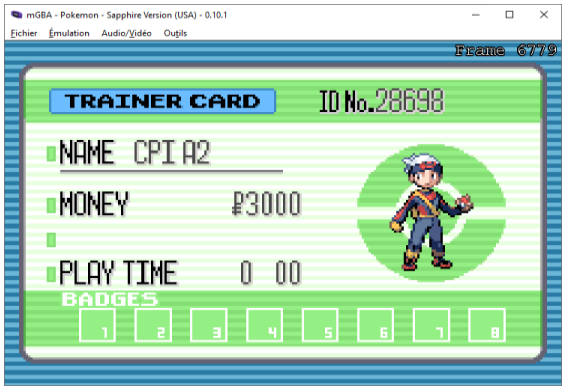
- 75 frames s'écoulent entre le moment ou je ferme la boite de dialogue et la génération de l'ID publique du dresseur, je rajoute donc 75 à ma frame notée dans l'étape d'avant (4840 + 75 = 4915. J'utilise le programme Python pour connaitre la sortie donnée par le

GLC sur la frame 4915, qui devra coïncider avec mon ID publique (valeur surlignée en jaune).

```
GLCRS(4900,4920)

{1: ('0x000005a0', '0x0000', 0),
 4901: ('0x1c496815444130a4', '0x4441', 17473),
 4902: ('0x11896cae2f6c0e47', '0x2f6c', 12140),
 4903: ('0xc2f2bba2f4616ae', '0x2f46', 12102),
 4904: ('0xc256a7334550c89', '0x3455', 13397),
 4905: ('0xd7221feacfe74c8', '0xacfe', 44286),
 4906: ('0x2c729d744a9d099b', '0x4a9d', 19101),
 4907: ('0x132babc166acb172', '0x66ac', 26284),
 4908: ('0x1a61621385c4a9fd', '0x85c4', 34244),
 4909: ('0x225e8e3f9735d72c', '0x9735', 38709),
 4910: ('0x26d9cd996184662f', '0x6184', 24964),
 4911: ('0x190e2838d3dc3476', '0xd3dc', 54236),
 4912: ('0x366f0689d602aab1', '0xd602', 54786),
 4913: ('0x36fc74f6e50afbd0', '0xe50a', 58634),
 4914: ('0x3ad936997146f803', '0x7146', 28998),
 4915: ('0x1d1ac490701ae3ba', '0x701a', 28698),
 4916: ('0x1ccdaaf648b202a5', '0x48b2', 18610),
 4917: ('0x12ad829f2137c6b4', '0x2137', 8503),
 4918: ('0x888e4c48682d317', '0x8682', 34434),
 4919: ('0x228f69fd3ccf433e', '0x3ccf', 15567),
 4920: ('0xf9fbb01cab0e5d9', '0xcab0', 51888)}
```

- Une fois la cinématique terminée, je regarde ma carte dresseur pour voir mon ID publique, si celle-ci correspond à ce que le programme m'a renvoyé, je peux déterminer l'ID secrète en regardant l'entrée précédente dans la liste et déduire l'ID complète.



Bingo !

```
GLCRS(4900,4920)

{1: ('0x000005a0', '0x0000', 0),
 4901: ('0x1c496815444130a4', '0x4441', 17473),
 4902: ('0x11896cae2f6c0e47', '0x2f6c', 12140),
 4903: ('0xc2f2bba2f4616ae', '0x2f46', 12102),
 4904: ('0xc256a7334550c89', '0x3455', 13397),
 4905: ('0xd7221feacfe74c8', '0xacfe', 44286),
 4906: ('0x2c729d744a9d099b', '0x4a9d', 19101),
 4907: ('0x132babc166acb172', '0x66ac', 26284),
 4908: ('0x1a61621385c4a9fd', '0x85c4', 34244),
 4909: ('0x225e8e3f9735d72c', '0x9735', 38709),
 4910: ('0x26d9cd996184662f', '0x6184', 24964),
 4911: ('0x190e2838d3dc3476', '0xd3dc', 54236),
 4912: ('0x366f0689d602aab1', '0xd602', 54786),
 4913: ('0x36fc74f6e50afbd0', '0xe50a', 58634),
 4914: ('0x3ad936997146f803', '0x7146', 28998),
 4915: ('0x1d1ac490701ae3ba', '0x701a', 28698),
 4916: ('0x1ccdaaf648b202a5', '0x48b2', 18610),
 4917: ('0x12ad829f2137c6b4', '0x2137', 8503),
 4918: ('0x888e4c48682d317', '0x8682', 34434),
 4919: ('0x228f69fd3ccf433e', '0x3ccf', 15567),
 4920: ('0xf9fbb01cab0e5d9', '0xcab0', 51888)}
```

En reprenant le résultat du programme, on peut voir que l'entrée précédente équivaut à 28998 (surlignée en rouge sur la capture d'écran.). L'ID complète de notre dresseur est donc IP publique puis IP secrète mises à la suite : 2869828998 ! Les valeurs en surlignées en vert sont les valeurs suivantes qui vont servir à déterminer les IV du Pokémon.

Deuxième partie : Décomposition du PID/Bits d'IV et prédictions

La première partie de la manipulation est terminée ! La prochaine étape est d'obtenir un Gobou chromatique et de connaître ses statistiques à l'avance, un Pokémon est chromatique lorsque son PID est identique à l'ID de notre dresseur, je peux donc utiliser l'ID complète déduite dans la première partie de la manipulation pour déterminer son sexe, son talent et sa nature ! Voici les résultats :

PID :

ID du Joueur : 28698 28998
PID du Pokémon : 28698 28998 (Chromatique garanti)
PID du Pokémon en hexadécimal : 701A 7146
PID du Pokémon en binaire : 01110000 00011010 01100001 01000110

Sexe du Pokémon :

Distribution mâle/femelle chez les Gobous : 12,5% de femelle (Y < 31 == Femelle)
Y : 8 bits de poids faibles, 01000110 = 70, 70>31 donc le Gobou sera un mâle.

Talent :

Gobou n'as qu'un seul talent possible : Torrent

Nature :

PID du Pokémon en décimal : 1 880 781 126
On prend les 2 derniers chiffres du PID en décimal : 26
Si la valeur dépasse 24, retirer 25 autant de fois qu'il faut : 26 - 25 = 1
Se référer au table des natures : La nature d'index 1 est Solo

IVs :

Première valeur décimale pour les IVs : 18610
Conversion en binaire : 0100 1000 1011 0010
Deuxième valeur décimale pour les IVs : 8503
Conversion en binaire : 0010 0001 0011 0111
IV Défense Physique : 18
IV Attaque physique : 5
IV Points de vie: 18
IV Défense spéciale : 8
IV Attaque spéciale : 9
IV Vitesse : 23

En utilisant ce calculateur, en rentrant la nature, le niveau du Pokémon (Gobou sera niveau 5, c'est le cas pour tout les Pokémon que l'on peut choisir au début du jeu.) et les IV que l'on vient de déduire, le calculateur nous renvoie ceci :

Level 5 Mudkip						
	HP	ATK+	DEF-	SP. ATK	SP. DEF	SPD
0 IV	20	13	9	10	10	9
Provided stats	20	13	9	10	10	10
31 IV	21	14	9	11	11	10

(Mudkip est le nom anglais de Gobou)

La ligne surlignée jaune contient les statistiques de notre Gobou, les autres lignes représentent juste le pire et le meilleur spécimen possible.

Troisième partie : Manipulation du PID du Pokémon de départ

Maintenant que nous avons fini notre prédiction des statistiques et du sexe de notre futur petit compagnon, nous pouvons passer à la deuxième partie de la manipulation, à savoir manipuler le GLC pour générer notre Pokémon sur la même frame que lors de la génération de notre ID secrète. l'ID dresseur devant être identique avec le PID, nous ne sommes pas obligés d'utiliser la même frame pour obtenir ce résultat. Cependant, celle-ci est garantie d'être la première frame commune lorsque l'on démarrera le jeu, on gagne donc un temps considérable en utilisant celle-ci plutôt qu'en attendant une future frame qui pourrait prendre plusieurs minutes voir plusieurs heures à arriver.

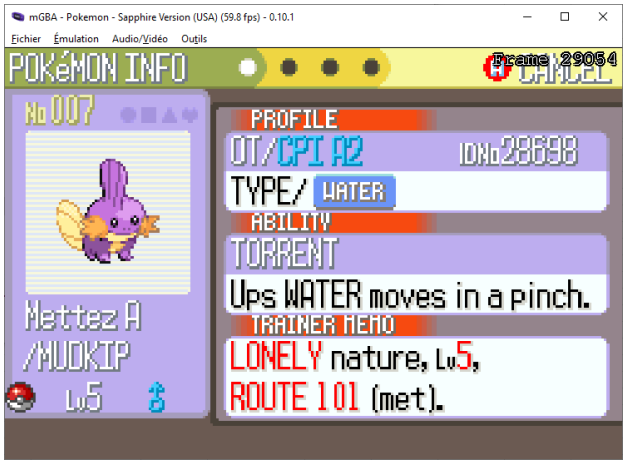
Faisons un petit saut dans le temps. Le début du jeu se déroule normalement jusqu'au moment ou l'on voit le professeur Seko attaqué par un Medhyena, celui-ci nous crie de prendre un Pokémon dans son sac et de le sauver en combattant le Medhyena avec celui-ci. A ce stade nous sommes sur le point de choisir notre premier Pokémon parmi trois options, Poussifeu, Arcko et... Gobou ! C'est ici qu'il va falloir manipuler l'aléatoire de manière à obtenir le Gobou que l'on veut ! Pour cela, il faut d'abord sauvegarder puis redémarrer le jeu pour remettre le GLC à son état inital.

Ensuite, une fois le jeu redémarré, il faut ouvrir le sac du professeur Seko, sélectionner la créature que l'on veut, ici Gobou, et valider notre choix sur la frame précédent 4914 (Lorsqu'un Pokémon est généré le jeu demande trois valeurs au GLC et n'utilise pas la première, nous visons donc 4913 pour que les deux valeurs suivantes soient celles qui composent le PID.). La seed initiale étant la même que lors du premier lancement du jeu, les valeurs aux frames 4914 et 4915 seront identiques que lorsque nous avons généré notre ID de dresseur, garantissant que le PID du Pokémon soit identique et que celui-ci soit chromatique. (Cette technique fonctionne aussi sur les deux autres Pokémon que le joueur peut choisir à la place de Gobou au début du jeu, ainsi que sur le Pokémon légendaire Kyogre ou Groudon selon la version choisie.)

Je me positionne donc sur la boîte de dialogue qui confirme le choix de Gobou, j'attends la frame 4913 et je clique sur A au moment de celle-ci (à 60 frames par secondes ceci est quasiment impossible à faire manuellement, j'utilise donc la fonction d'avancement frame par frame de l'émulateur pour m'aider.).

Et comme prévu, notre dresseur lance la pokéball et un Gobou chromatique en sort ! (Un gobou normal est bleu, un chromatique est violet.) Il ne nous reste qu'à finir le combat pour lui donner un petit nom et vérifier si nos prédictions concernant ses statistiques étaient correctes !

On peut désormais regarder ses infos, et y découvrir qu'il est bien de nature Solo (Lonely en anglais), et que nos prédictions sur ses statistiques étaient correctes ! (Son nom n'est pas du tout un message subliminal destiné aux tuteurs...)



Aller plus loin : Comment les failles ont étés résolues dans les générations suivantes ?

Aujourd'hui, les jeux Pokémon ne disposent plus de failles permettant ce genre de manipulation, et c'est surtout grâce au changement de support que la franchise à subie après la troisième génération. En effet, à partir de la quatrième génération de jeux ceux-ci sortaient sur la Nintendo DS, capable de lire les jeux de GameBoy advance (comme Rubis et Saphir) mais disposant aussi de leur propres cartouches. La série à ensuite évoluée sur la Nintendo 3DS et plus récemment la Nintendo Switch, changeant à chaque fois de format de cartouches.

(De gauche à droite, ma copie de Pokémon Saphir sur GameBoy Advance, Pokémon Noir sur Nintendo DS, Pokémon Lune sur Nintendo 3DS et Pokémon Épée sur Nintendo Switch.)

Comme vous pouvez le voir sur la photo ci-dessus, dès la Nintendo DS les cartouches ont diminuées en taille de manière drastique, ne permettant plus de contenir une pile et une horloge RTC au sein de la cartouche. De manière concrète, les formats de cartouches Nintendo fonctionnent plus comme des cartes SD depuis la Nintendo DS.

L'absence de pile dans les cartouches est palliée par la présence d'une horloge dans les consoles elle-même qui fonctionnent grâce à la batterie de celles-ci ! A partie de la Nintendo DS les consoles Nintendo ont été équipées de batteries Lithium-ion qui durent bien plus longtemps que 10 à 20 ans. De plus, si la batterie meurt c'est la console tout entière qui ne fonctionne plus, impossible donc de se soustraire à l'horloge pour manipuler la seed initiale.

Plan de diffusion

Ce document est destiné à être diffusé sur mon LinkedIn, pour prouver mes compétences en informatique et mathématique ainsi que ma rigueur et ma capacité à rédiger un rapport scientifique à des entreprises intéressés par mon profil. Etant donné la dimension du document, il serait intéressant de le partager sur des communautés en ligne qui se spécialisent dans la manipulation d'aléatoire dans les jeux, comme des Subreddits ou des forums détaillant le speedrun des jeux Pokémon Rubis, Saphir et Emeraude (la pratique du speedrun consiste à finir un jeu le plus vite possible.) mais aussi sur des communautés traitant d'informatique et de mathématique. Le programme et l'Excel que j'ai produit ont également pour but d'être partagés sur mon Github.

Bibliographie

http://web.ist.utl.pt/~ist11038/compute/or/,rand/Pisharath.pdf

https://scholar.archive.org/work/vtc3nu74vfffvi5iiyoa7ljuyi/access/wayback/https://services.door.donau-uni.ac.at/api/object/o:2711/download#page=157

https://link.springer.com/chapter/10.1007/978-3-319-27659-5_27

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=f3ebdb2ceb61a1101b9ec9bc8858f2e6fa1c7fd9

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

https://pycosites.com/pkmn/stat.php

https://tasvideos.org/GameResources/GBA/PokemonGen3/RNG

https://www.smogon.com/ingame/rng/pid_iv_creation

https://www.smogon.com/ingame/rng/emerald_rng_intro

https://www.smogon.com/ingame/rng/rs_nonbredrng

https://www.smogon.com/ingame/rng/emerald_rng_part2

https://www.smogon.com/ingame/rng/emerald_rng_part3

https://www.smogon.com/forums/threads/most-generation-prng-help-information.52180/page-1644

https://bulbapedia.bulbagarden.net/wiki/Pseudorandom_number_generation_in_Pok%C3%A9mon

https://bulbapedia.bulbagarden.net/wiki/Trainer_ID_number

https://bulbapedia.bulbagarden.net/wiki/Pok%C3%A9mon_data_structure_(Generation_III)

https://bulbapedia.bulbagarden.net/wiki/Personality_value

https://bulbapedia.bulbagarden.net/wiki/List_of_Pok%C3%A9mon_by_gender_ratio

https://bulbapedia.bulbagarden.net/wiki/List_of_Pok%C3%A9mon_by_Ability#Generation_III_families

https://www.gamedeveloper.com/programming/how-classic-games-make-smart-use-of-random-number-generation

https://www.nintendo.fr/Assistance/Nintendo-DS-Lite/Informations-produit/Specificites-techniques/Informations-produit-242119.html

https://www.youtube.com/watch?v=5NCfx4Zr0i8