

## Générateur de nombres pseudo-aléatoires, Remplissage de rectangle, Dessin d'images bitmaps

### Générateur de nombres pseudo-aléatoires: asmSRand et asmRand

Un système informatique comme la GBA est en principe parfaitement **déterministe** : dans les mêmes conditions de départ, même programme et mêmes données, l'exécution sera toujours la même. Mais pour une animation autonome, il peut être utile d'avoir une fonction génératrice de "surprises", c'est à dire une fonction retournant des valeurs arbitraires, toujours différentes. Comment fabriquer du hasard avec l'électronique numérique qui en est a priori totalement dépourvue ? L'idée est d'utiliser une suite entière, définie par récurrence avec une formule qui génère des séquences de valeurs complexes et difficilement prévisibles, qui "ressemblent" à un tirage aléatoire.

Une telle suite doit respecter certaines contraintes :

- Les valeurs qui en sont issues doivent exhiber des propriétés statistiques caractéristiques d'une loi de probabilité uniforme sur un intervalle (et si possible montrer une indépendance des tirages successifs, ce qui est en contradiction avec une définition par récurrence...)
- Les valeurs étant représentées par un nombre fini d'informations binaires, elles sont elles mêmes en nombre fini. Toute suite récurrente entière bornée est nécessairement cyclique. Un bon générateur pseudo-aléatoire doit avoir une période aussi longue que possible avant de se répéter.

L'analyse et la mise au point de telles formules dépasse le cadre du module, nous nous contenterons d'utiliser un des schémas les plus simples : le générateur congruentiel linéaire de la forme  $U_{n+1} = (a.U_n + c) \text{ modulo } m$ .

En particulier le classique générateur de Knuth & Lewis propose la formule :  $U_{n+1} = (1664525.U_n + 1013904223) \text{ modulo } 2^{32}$

Quel est le format binaire naturel pour faire ce calcul ? Cela convient-il au processeur ARM ?

Ecrire 2 sous-programmes en assembleur ARM, sur le modèle des fonctions du C `void srand(unsigned int)` et `int rand(void)` :  
`asmSRand` initialisera la valeur initiale de la séquence

`asmRand` calculera la valeur suivante de la séquence, et retournera à l'appelant seulement les 16 bits de poids forts (qui sont "plus aléatoires")

Quel sera l'intervalle des valeurs pseudo-aléatoires retournées par `asmRand` ?

Comment initialiser la "graine" (`seed` = valeur initiale d'un générateur aléatoire) sachant que la GBA ne dispose pas d'horloge (pas de fonction time) ?

### Ecrire le code assembleur de `asmDrawRect` : remplir un rectangle avec une couleur

On met la valeur `color` dans le rectangle de largeur `w` et de hauteur `h` dont l'adresse du pixel supérieur gauche est donné par `dest`.

Le choix de l'ordre des paramètres est fait pour pouvoir reprendre sans trop de modifications le code de `asmRectcpy` du TD3.

De plus en partant directement de l'adresse du pixel supérieur gauche et non de ses coordonnées `x` et `y` le travail est facilité : pas besoin de calculer l'adresse à partir des coordonnées dans le sous programme assembleur (mais il faudra le faire depuis le C au niveau de l'appel) et on a un sous programme avec seulement 4 paramètres (au delà il faut utiliser la pile). Ces choix ne seraient pas les meilleurs pour développer une librairie sérieuse...

**Prototype C** : `void asmDrawRect(u16 * dest, u16 color, int w, int h);`  
**Entrées** :  
`dest` = adresse de départ du remplissage (r0)  
`color` = valeur de remplissage (demi-mot) (r1)  
`w` = largeur rectangle (r2)  
`h` = hauteur rectangle (r3)

### Ecrire le code assembleur de `asmDrawImage` : dessiner à l'écran une image bitmap importée

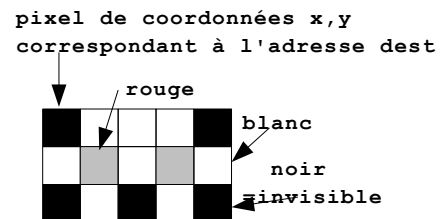
On dispose d'une image graphique importée dans le programme sous la forme d'une structure de type `t_image` initialisée.

Un utilitaire vu au TP3, basé sur allegro, réalise des conversions des formats de fichiers bmp ou pcx vers ces blocs de codes qui peuvent être ensuite inclus dans le projet. Exemple de code et image associée, à dessiner à partir de ces données :

```
typedef struct{
    int w,h;
    u16 img[];
} t_image;

// généré automatiquement à partir
// d'un fichier image invader.bmp

const t_image invader={5,3,{
0x0000,0x7fff,0x7fff,0x7fff,0x0000,
0x7fff,0x001f,0x7fff,0x001f,0x7fff,
0x0000,0x7fff,0x0000,0x7fff,0x0000
}};
```



**Prototype C** : `void asmDrawImage(u16 * dest, u16 * img, int w, int h);`  
**Entrées** :  
`dest` = adresse de départ du remplissage (r0)  
`img` = pointeur sur le tableau de couleurs (r1)  
`w` = largeur rectangle (r2)  
`h` = hauteur rectangle (r3)

Pour afficher le sprite (l'image) de l'invader aux coordonnées `x y`, l'appel depuis le C sera :

```
asmDrawImage((u16 *) (0x06000000+2*x+480*y), invader.img, invader.w, invader.h);
```

Les couleurs sont dans un tableau à une dimension, elles correspondent aux couleurs des pixels successifs du rectangle image dessiné à l'écran selon le balayage habituel de gauche à droite et de haut en bas. Inspirez vous du code de `asmRectcpy` du TD3 pour afficher ces couleurs telles quelles, puis modifiez cette 1ère version pour prendre en compte la transparence : les pixels noirs ne sont pas affichés.

Facultatif : comme pour l'exercice précédent, le choix des paramètres est fait pour faciliter votre développement en assembleur. Un format d'appel plus simple pour l'utilisateur final correspondrait au prototype `void asmDrawImage(const t_image * bmp, int x, int y);`

Comment implémenter la réception de cet appel au niveau de la procédure assembleur ?