Mini-Projet 2015-16

Simulateur de Voyage Spatial

L'objectif de ce mini-projet est de réaliser un simulateur de voyage spatial. Plus exactement, il s'agit de réaliser l'affichage d'un ciel étoilé observé par les passagers d'un vaisseau spatial se déplaçant à une vitesse proche de celle de la lumière.

Figure 1: https://www.youtube.com/watch?v=iJLbAntU118

Le ciel est représenté sous forme d'un écran noir et les étoiles sous forme de pixels blancs. A chaque étape de la simulation, le vaisseau étant considéré au centre de l'affichage, les étoiles se déplacent vers l'extérieur de l'écran suivant leur position et leur éloignement. Par exemple, une étoile de coordonnée (10, 10) peu éloigné dans quadrant haut-gauche se déplacera par pas de (-2, -2). Une étoile très éloignée (100, 20) dans le cadrant haut-droit de coordonnée se déplacera par pas de (-5, 8). A chaque étape de la simulation, le vecteur de déplacement est additionné aux coordonnées. Répété plusieurs fois par seconde, on obtiendra un effet de ciel étoilé défilant.

Fonction de tirage aléatoire

Pour éviter que notre animation soit toujours la même, nous allons tirer de manière aléatoire la position initiale des étoiles et leur vecteur de déplacement. Pour cela, il nous faut une fonction de tirage pseudo-aléatoire dont l'algorithme est donné ci-dessous (nommé SimpleRNG) :

```
int z = 0xCAFEBABE, w = 0xB00FBEA6 ;
int random(int n) {
      z = 0x9069 * (z & 0xFFFF) + (z >> 16) ;
      w = 0x4650 * (w & 0xFFFF) + (w >> 16) ;
      return ((z <<  16) | w) % n ;
}
```

Note : on prendra toujours *n* comme étant une puissance de 2.

Selon une graine stockée dans les variables z et w, une séquence de nombre est générée en modifiant à chaque appel les valeurs de z et w : la dispersion des nombres produits peut faire penser à une séquence aléatoire mais dépend de la graine, c'est-à-dire des valeurs initiales de z et w.

A faire : implanter la fonction `random` en assembleur en considérant que *n* est toujours une puissance de 2.

Pour tester le programme, avec la graine ci-dessus et *n* = 256, on doit obtenir la séquence suivante :

239, 13, 48, 124, 90, 183, 22, 99, 19, 148

Simulation de déplacement des étoiles

Pour traiter chacune des étoiles, il faut représenter ses coordonnées courante (*x*, *y*) et son vecteur de déplacement (*dx*, *dy*) avec *x*, *y*, *dx* et *dy* étant des entiers. L'idée est de découper la simulation en pas réalisés les uns après les autres (en boucle). A chaque pas, *x* est augmenté de la quantité *dx* et *y* est augmenté de la quantité *dy*.

Pour initialiser nos points, nous avons d'abord besoin de définir les dimensions de notre écran :

```
.equ  LARGEUR, 128
.equ  HAUTEUR, 128
```

Bien que vous puissiez par la suite utiliser des écrans plus grands, les dimensions données ci-dessus vous permettront facilement d'afficher vos étoiles dans l'afficheur de mémoire d'Eclipse.

Afin d'initialiser les étoiles, on va faire un tirage aléatoire de leur vecteur de déplacement :

$dx \in [-8, 7]$,

$dy \in [-8, 7]$

Le vecteur de déplacement (0, 0) est interdit. Puis leur position initiale grâce à un facteur temps *t* tiré de manière aléatoire $t \in [0, 7]$. *t* permet de déterminer la position initiale avec la formule suivante :

$x = t * dx + \text{LARGEUR} / 2$

$y = t * dy + \text{HAUTEUR} / 2$

Pour stocker ces informations, nous allons utiliser une structure pour chacun de ces éléments d'une étoile. Si on considère que cette structure est stockée à l'adresse *a*, on obtient l'organisation mémoire suivante :

- adresse *a* + 0, 4 octets pour *x*,
- adresse *a* + 4, 4 octets pour *y*,
- adresse *a* + 8, 4 octets pour *dx*,
- adresse *a* + 12, 4 octets pour *dy*.

Notre structure aura une taille de 16 octets. Si on considère qu'on dispose de N[1] étoiles, le tableau des étoiles pourra être alloué par :

```
            .equ  N, …
etoiles :   .fill N * 16, 1, 0
```

On pourra s'aider des constantes suivantes pour accéder aux différents éléments de la structure :

```
.equ  X,          0
.equ  Y,          4
.equ  DX,         8
.equ  DY,         12
.equ  TAILLE,     16
```

1 Pour déboguer plus facilement, on pourra commencer avec un N petit, par exemple 4.

A faire

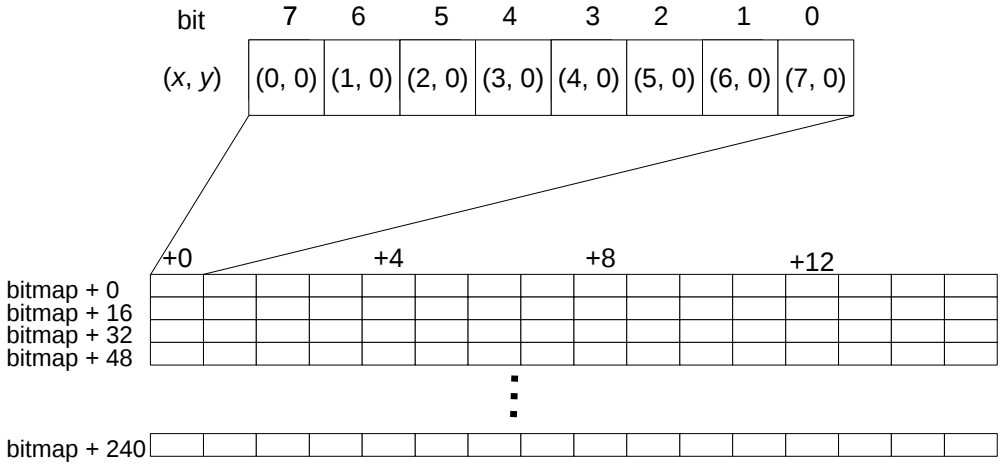
1. la fonction `init_etoile` qui prend en paramètre dans R_0 l'adresse d'une étoile à initialiser,
2. la fonction `deplacer_etoile` qui prend en paramètre dans R_0 l'adresse d'une étoile pour laquelle il faut faire un pas de déplacement (si l'étoile sort de l'écran, il faudra la ré-initialiser en notant que l'étoile part toujours du centre à ce moment là).
3. Écrire un programme principal qui initialise les N étoiles et procède à 100 pas de simulation.

Affichage des étoiles

Pour réaliser l'affichage des étoiles, nous allons utiliser une bitmap, « carte de bit », de profondeur 1, c'est-à-dire n'acceptant que 2 couleurs : 0 pour le noir, 1 pour le blanc. Ainsi, chaque pixel sera représenté par un bit donnant sa couleur. Comme un octet compte 8 bits donc 8 pixels, il faudra (LARGEUR * HAUTEUR) / 8 octets pour représenter la bitmap :

```
bitmap :    .fill (LARGEUR * HAUTEUR) / 8, 1, 0
```

La ligne précédente initialise notre bitmap avec un écran noir (tous les bits sont à 0). Pour manipuler les pixels, il faut convenir d'une organisation en ligne et en colonne : les pixels sont rangés lignes par ligne de LARGEUR / 8 octets. L'octet à l'adresse bitmap représentera les pixels (0, 0) à (7, 0) de notre écran. L'octet à l'adresse bitmap + 1 les octets (8, 0) à (15, 0) de notre écran et ainsi de suite. Ensuite, l'octet à l'adresse bitmap + LARGEUR / 8 représentera les pixels (0, 1) à (7,1) de notre écran et ainsi de suite pour chacune des lignes. Comme dans le schéma ci-dessous :



Si on veut maintenant représenter l'évolution des étoiles dans la bitmap et ainsi pouvoir visualiser l'animation, il faut à chaque pas :

1. Effacer l'étoile à l'ancienne position – mettre à 0 le bit correspondant au pixel (*x*, *y*),
2. Incrémenter *x* par *dx*, *y* par *dy*,
3. Afficher l'étoile à la nouvelle position – mettre à 1 le bit correspondant au pixel (*x*, *y*).

Et il faut répéter cette procédure pour chacune des étoiles composant l'image.

A faire

1. Réaliser la fonction `allumer` qui prend en paramètre *x* dans R_0 et *y* dans R_1 et met à 1 le bit correspondant de la bitmap.
2. Réaliser la fonction `eteindre` qui prend en paramètre *x* dans R_0 et *y* dans R_1 et met à 0 le bit correspondant de la bitmap.
3. Modifier le programme principal précédent pour afficher la simulation des étoiles dans la bitmap.

Note – pour voir l'affichage, il faut demander à Eclipse d'afficher la mémoire correspondant à l'adresse de bitmap avec le format :

1. Sélectionner le `Memory Browser` (éventuellement à partir du menu `Window` > `Show View` > `Memory Browser`)
2. Entrer l'adresse de la variable `bitmap`
3. bouton droit > `Cell Size` > `1 Byte`
4. bouton droit > `Column` > `16`
5. bouton droit > `Radix` > `Binary`

Il suffira de mettre un point d'arrêt en début de boucle principale pour voir à chaque arrêt la bitmap se transformer.