

Intégration des éléments du programme de jeu, optimisations et estimations de performances

Objectifs du TP

A l'issue de ce dernier TP vous devez pouvoir montrer des soucoupes volantes évoluant de manière autonome en plus du vaisseau qui se déplace de manière interactive au dessus du paysage qui défile dans la partie basse de l'écran. Il est souhaitable que vous arriviez à mettre en place la détection de collision du vaisseau avec les soucoupes, ce qui sera indiqué par un effet visuel simple. L'objectif est ensuite d'évaluer les temps d'exécution de sous-programmes assembleur et de voir dans quelle mesure ils peuvent être améliorés. Enfin si il vous reste du temps vous pourrez apporter des améliorations comme une barre de progression, une animation de victoire, une "explosion" du vaisseau...

Copie de projet

Commencez par copier le projet du TP3 dans un nouveau répertoire de travail : copiez simplement l'ensemble du répertoire de départ et donnez lui un nouveau nom. Vous pouvez aussi modifier les noms des fichiers de projet VHAM avec les extensions .vho et .vhw (dans le répertoire). Puis lancez VHAM et ouvrez ce projet (menu -> File -> Open Workspace). Vérifiez que l'ensemble recompile correctement (F10 puis F5).

Importation et animation des soucoupes

Une image de soucoupe prête à être importée dans le projet se trouve dans le fichier **TP3.zip** sur le site, lien au dessus de cet énoncé. (TP3.zip contient également un programme source Allegro **img2tab.c** qui peut convertir vos propres fichiers images en fichiers c d'initialisation).

La méthode pour intégrer l'image de soucoupe est la même que pour le panorama (TP3) : nous partons d'un fichier déjà converti dans **TP3.zip** : **ufo.c**
L'image d'origine (ufo.bmp) mesure 15 pixels de large et 5 pixels de haut.

- Copier **ufo.c** dans le répertoire de projet
- Dans VHAM, ajouter ce fichier source au projet (panneau de gauche) en faisant click droit sur **Source Files** puis **Add file(s)**
- Indiquer au compilateur ce nouveau fichier source en éditant le fichier **makefile** (panneau de gauche) :
Ajouter un fichier objet en plus à la 9ème ligne : **OBJS := main.o armgba.o panorama.o ufo.o**
- La variable **ufo** initialisée dans **ufo.c** est de type **const t_image** , il faudra ajouter dans **projet.h** la déclaration de cette variable pour qu'elle soit visible depuis les autres fichiers sources (mot clé **extern**)
extern const t_image ufo;
- L'image de soucoupe est prête à être affichée en utilisant la procédure **drawImage** (Cf TP3) ou **asmDrawImage** (Cf TD4)

Le code qui suit est une proposition pour réaliser une animation basique de 10 soucoupes... si vous le souhaitez vous pouvez modifier ce code ou réaliser les choses à votre manière (facultatif).

Pour gérer le déplacement aléatoire des soucoupes il faudra utiliser le générateur pseudo-aléatoire vu au TD4.
Intégrez **asmSRand** et **asmRand** au projet : code source assembleur dans **asm.s** et prototypes dans **projet.h**
Testez en initialisant le générateur aléatoire avec une valeur arbitraire puis en affichant des pixels au hasard :

```
for (i=0;i<100;i++) // CODE DE TEST, A ENLEVER APRES CONFIRMATION
    drawPixel(asmRand()%240,asmRand()%160,0x7FFF);
```

Pour "effacer les positions précédentes" des soucoupes on utilisera un affichage de rectangle noir, **asmDrawRect** vu au TD4.
Intégrez **asmDrawRect** au projet : code source assembleur dans **asm.s** et prototypes dans **projet.h** Testez en dessinant un rectangle blanc.

Dans le fichier en-tête **projet.h** on ajoute une constante **NSAUCER** (*nombre de soucoupes*) et un typedef qui regroupe positions et déplacements des soucoupes, ainsi que les prototypes d'initialisation, animation, effacement et affichage de soucoupes :

```
#define NSAUCER 10

typedef struct {
    int x,y,dx,dy; // Position du coin supérieur gauche de la soucoupe (x,y) et vecteur déplacement (dx,dy)
} t_saucer;

void initSaucer(t_saucer *s);
void moveSaucer(t_saucer *s);
void eraseSaucer(t_saucer *s);
void drawSaucer(t_saucer *s);
```

Puis ajoutez les définitions correspondantes au projet (soit dans un fichier.c regroupant les fonctions annexes, soit directement dans **main.c** ...)

```
void initSaucer(t_saucer *s){
    s->x=(asmRand()%200)+20;    s->y=(asmRand()%100)+20;
    s->dx=(asmRand()%3)-1;    s->dy=(asmRand()%3)-1;
}
```

```

void moveSaucer(t_saucer *s){
    if ( (asmRand()&31)==0){ // Avec une probabilité de 1/32 ...
        s->dx=(asmRand()%3)-1; // nouvelle direction au hasard
        s->dy=(asmRand()%3)-1;
    }

    s->x+=s->dx; s->y+=s->dy; // Déplacement selon le vecteur dx,dy

    // Gestion des bords : en cas de dépassement on se replace et nouvelle direction au hasard
    if (s->x>225){ s->x=225; s->dx=(asmRand()%3)-1; }
    if (s->x<0) { s->x=0; s->dx=(asmRand()%3)-1; }
    if (s->y>115){ s->y=115; s->dy=(asmRand()%3)-1; }
    if (s->y<8) { s->y=8; s->dy=(asmRand()%3)-1; }
}

void eraseSaucer(t_saucer *s){
    asmDrawRect(pixelAddress(s->x,s->y),0,ufo.w,ufo.h);
}

void drawSaucer(t_saucer *s){
    drawImage(&ufo,s->x,s->y);
}

```

Enfin il reste à déclarer un tableau de structures soucoupes dans le main

```
t_saucer saucer[NSAUCER];
```

et intégrer l'appel à ces procédures depuis le main selon la séquence suivante :

```

MAIN : L'ordre des opérations est important
Initialiser générateur aléatoire
Initialiser mode graphique
Effacer Ecran (utiliser asmDrawRect pour dessiner un rectangle noir de la taille écran)
Afficher panorama en bas de l'écran
Initialiser positions soucoupes ( appels initSaucer )
boucle infinie d'animation :
    Synchro blayage vertical ( appel vSync )
    Effacer positions soucoupes ( appels eraseSaucer )
    Effacer position vaisseau
    Mouvements soucoupes ( appels moveSaucer )
    Interaction vaisseau
    Afficher soucoupes ( appels drawSaucer )
    Appeler afficher nouvelle position vaisseau et récupérer valeur en retour
    si valeur retour non nulle indiquer collision (par exemple dessiner un gros carré rouge)
    Séquence de copies qui réalise le scrolling circulaire
Fin de boucle infinie

```

A chaque étape de gestion des soucoupes il faut prévoir une boucle qui les passe en revue. Exemple pour "Initialiser positions soucoupes" :

```

for (i=0;i<NSAUCER;i++)
    initSaucer(&saucer[i]);

```

Mesure des performances en utilisant les timers, optimisations

Pour cette partie il est nécessaire d'afficher des informations dans la console.

Pour voir ces informations dans le simulateur il faudra faire menu -> Tools -> Logging

Concernant le développement sous LINUX ou OSX (manipulations inutiles sur les machines de l'école) :

Pour le kit sur **LINUX** ou **OSX** : il semble ne pas y avoir d'outil de logging disponible dans le simulateur pour ces 2 plate-formes.

Une version complétée de la mini-librairie vous permettra d'afficher des informations textuelles ou numériques sur l'écran de la GBA :

Chargez http://www.ece.fr/~fercoq/architecture/console_linux_osx.zip

Remplacez `armgba.h` et `armgba.c` dans votre répertoire de projet.

En début de programme indiquez que les message doivent s'afficher à l'écran `traceConsole=2;`

Vous pouvez tester avec `traceStr("Hello screen !");`

Avant chaque message vous pouvez préciser les coordonnées d'affichage avec (par exemple) `textx=170; texty=0;`

Nous allons utiliser les timers de la GBA pour mesurer l'écoulement du temps pendant l'exécution des sous-programmes assembleur. Malheureusement le simulateur utilisé (Visual Boy Advance) ne synchronise pas les compteurs des ses timers simulés à chaque cycle du processeur, on aura donc une résolution temporelle de quelques microsecondes. Le temps sera compté par incréments de 1 tous les 64 cycles du processeur.

Le processeur de la GBA est à 2²⁴ Hz donc la valeur est incrémentée 262144 fois par seconde : ~ 3,81 µs par incrément

Ajoutez à votre projet ces deux procédures en C : `startTimers` qui initialise les timers et `readTimers` qui lit le temps écoulé après l'initialisation.

```

void startTimers(){
    *(volatile u16 *)0x0400010E=0; // Stopper timers 2 et 3 (remet leur compteur à valeur de départ, ici 0)
    *(volatile u16 *)0x0400010A=0;
    *(volatile u16 *)0x0400010E=128|4; // Configurer et lancer les timers 2 et 3
    *(volatile u16 *)0x0400010A=128|1;
}

unsigned int readTimers(){
    // Retourner le temps écoulé sur 32 bits à partir des 2 compteurs 16 bits des timers 2 et 3 en cascade
    return ( *(volatile u16 *)0x0400010C) << 16 | *(volatile u16 *)0x04000108;
}

```

Pour évaluer le temps d'exécution d'un appel à un sous-programme il faudra (ré)initialiser les timers avant l'appel et afficher le temps écoulé après.

```
// Exemple Pour Tester le temps d'affichage d'une soucoupe, à placer avant la boucle d'animation

initSaucer(&saucer[0]); // Initialiser une soucoupe
startTimers(); // Démarrage du chronométrage
drawSaucer(&saucer[0]); // Appel de la procédure à chronométrer
traceHex32(readTimers()); // Affichage en console du temps écoulé
while (1); // Boucle infinie pour stopper le programme (le test est terminé)
// il reste à lire le résultat dans menu -> Tools -> Logging
```

Le temps obtenu est en hexadécimal, il faut le convertir en décimal (pour les conversions, calculatrice windows mode scientifique), puis multiplier par 64 pour avoir (approximativement) le nombre de cycles processeur puis diviser par 2^{24} pour avoir des secondes

Mesurez ainsi le temps d'exécution de l'affichage d'une soucoupe, qui utilise la procédure C `drawImage`
Dans `drawSaucer` remplacez l'utilisation de `drawImage` par la procédure assembleur `asmDrawImage` (vue au TD4)
Attention les paramètres ne respectent pas le même format (adapter l'appel)
Après chronométrage : le passage d'une version C à une version assembleur apporte-t-il une amélioration ?
Pouvez-vous améliorer encore la version assembleur en utilisant les adressages post-indexés (voir supplément de cours en ligne) ?

Facultatif : en s'inspirant de l'implémentation de `asmDrawImage` (déplacements d'adresses source et destination, pas de multiplication...) peut-on améliorer les performances de la procédure C `drawImage` ? Dans quelles proportions ?

Mesurez le temps nécessaire à la séquence de copies qui réalise le scrolling circulaire.

Le chapitre 4.2 du site suivant donne des précisions sur le rafraîchissement écran : <http://www.coranac.com/tonc/text/video.htm>

Le sous programme `vSync` attend que le rafraîchissement écran en cours soit terminé. C'est à dire que les opérations de la boucle principale d'animation qui suivent l'appel à `vSync` commencent alors que le balayage est en période dite "vblank" (voir schéma sur le site). Pendant cette période le balayage précédent est terminé mais le prochain balayage n'est pas encore commencé : les opérations en mémoire vidéo ne sont donc pas visibles. C'est donc pendant cette période qu'on efface les anciennes position, qu'on calcule les nouvelles positions et qu'on affiche les nouvelles positions : seul le résultat final (nouvelles positions) sera visible lors du prochain balayage effectif "vdraw". Sans cette précaution, et en l'absence de double-buffering (dessin dans tampon de taille écran et blit comme dans allegro) l'animation montrerait les clignotements dûs à l'effacement.

Essayez d'enlever l'appel à `vSync` pour vérifier que celui-ci est nécessaire.

Compte tenu du temps d'exécution du scrolling circulaire, le rafraîchissement écran est-il encore en période vblank quand la séquence de copies se termine ? Vérifiez en l'affichant en console que la ligne en cours de balayage écran (`REG_VCOUNT`) est encore au dessus de la zone de scrolling quand le scrolling est terminé : `traceHex32(REG_VCOUNT) ;`
Que ce passerait-il si le scrolling circulaire prenait trop de temps ou devait se situer en haut de l'écran ?

Améliorations - Facultatif

- Ajouter une barre de progression en haut de l'écran qui indique le temps passé sans collision. Pour cela prévoyez que les 8 lignes en haut de l'écran ne doivent pas être accédées ni par le vaisseau ni par les soucoupes. Utilisez une variable `progression` qui est incrémentée à chaque itération de la boucle d'animation et qui détermine la longueur de la barre de progression (proportionnellement, par exemple `progression>>3`)
- Quand le vaisseau touche une soucoupe la variable `progression` (et donc la barre de progression) repart à 0
- Quand la barre de progression touche le bord droit de l'écran, lancer une animation de victoire (et relancer une nouvelle partie)
- Améliorer l'effet visuel d'explosion du vaisseau (par exemple utiliser des rectangles de couleurs vives qui grossissent rapidement)
- Prévoir un fond (ciel étoilé, nébuleuse) qui ne soit pas effacé par les déplacements des objets (utiliser la zone tampon de 44 lignes en bas ...)
- Armements, explosions, confédérations interplanétaires et commerce, navigation en 3 dimensions... en assembleur ARM

