Projet Technologique « Battlefield Mario » : Fiche n°2

1 Des animations à la chaîne

Désormais notre oiseau vole fièrement dans un paysage arboré qui défile. Pour progresser dans la création d'un jeu de plateforme, nous allons intégrer d'autres élements animés tels des missiles, des explosions, du texte, d'autres oiseaux.

Nous pourrions continuer à ajouter ces élements dans `main.c` mais le code deviendrait à terme moins lisible et beaucoup moins maintenable.

1.1 Un module pour les animer tous

Définissez un module `animation.h`, `animation.c` qui gérera l'ensemble des animations du jeu. Vous pouvez ainsi écrire la fonction

```
void animation_init (void)
```

appelée dans le `main.c` qui se chargera d'initialiser l'ensemble des objets apparaissant au départ du jeu. Pour le moment, seul l'oiseau est concerné.

Ajoutez une fonction, également appelée dans le `main.c`,

```
void animation_one_step (int left, int right, int up, int down)
```

qui exécutera les routines d'animation des objets animés.

Une dernière fonction

```
void animation_render_objects (void)
```

exécutera les appels à `graphics_render_objects()` sur tous les objets animés présents en jeu.

Déplacez et modifiez le code nécessaire dans ces fonctions et intégrez leurs appels dans les modules `main.c` et `graphics.c`.

1.2 Liste

Étant donné que des objets, en quantité variable, seront créés au cours du jeu, il nous faut prévoir une structure de donnée adaptée à leur gestion. Quels seraient les avantages et inconvénients à utiliser un tableau d'objets ? Pour quelle raison est-il préférable de choisir une liste d'objets ?

Nous proposons d'utiliser une liste qui sera initialisée, manipulée et libérée dans le module `animation.c`. Le fichier `list.h` propose une implémentation de liste doublement chaînée. Cette implémentation est issue des sources du noyau Linux. Vous pouvez trouver des explications sur son utilisation sur ce site : http://a.michelizza.free.fr/pmwiki.php?n=TutoOS.Linkedlist#linux. La figure 1 illustre un exemple de liste d'objets.

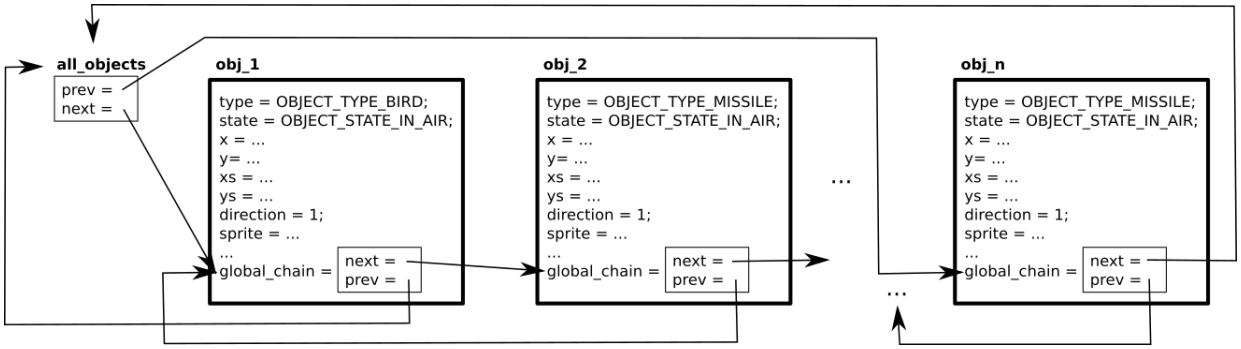

FIGURE 1 – Liste d'objets

Créez une liste d'objets intitulée `all_objects` et implémentez les outils de manipulation suivants :

1. Une macro `for_all_objects(var)` qui permet de parcourir la liste des objets en définissant une variable temporaire de nom `var`.
2. Une fonction d'ajout :
   ```
   void animation_mobile_object_add (dynamic_object_t *obj)
   ```
3. Une fonction de suppression d'un objet particulier :
   ```
   void animation_mobile_object_del (dynamic_object_t *obj)
   ```

Au sein de la routine d'initialisation des animations, ajoutez l'objet `bird` à la liste `all_objects`.

Ajoutez au module la fonction

```
void animation_clean (void)
```

qui retirera tous les élements présents dans la liste et les supprimera (sauf l'oiseau). Modifiez la fonction `animation_one_step` pour appeler les routines d'animation de tous les objets de la liste. De même, faites en sorte d'appliquer les textures de tous les objets de la liste dans la méthode `animation_render_objects`.

À ce point, un seul élément est présent dans la liste : notre oiseau. Vérifier que tout fonctionne comme avant.

2 Ajout d'une nouvelle animation

2.1 Un système de défense aérien

En vue des menaces futures (et comme tout individu de la population aviaire usuelle), notre oiseau a besoin de projeter des missiles. Faites en sorte de générer un objet de type `OBJECT_TYPE_MISSILE` à chaque appui sur la touche *Espace*. Ce missile partira de la position de l'oiseau et aura une trajectoire rectiligne et une vitesse fixe. Pour ce faire vous devrez définir un module `missile.h`, `missile.c`. La figure 2 vous suggère une interface.

```
1 #ifndef MISSILE_IS_DEF
2 #define MISSILE_IS_DEF
3
4 void animation_missile_add (dynamic_object_t *obj, int y_offset);
5
6 int animation_missile_onestep (dynamic_object_t *obj);
7
8 #endif
```

FIGURE 2 – Interface du module missile.h

Vous pouvez utiliser le fichier `shot.png` comme texture pour le missile.

2.2 Les missiles se cachent pour mourir

Un missile sorti de la carte n'ayant plus de raison d'exister, il nous faut le détruire. A cette fin, nous pouvons utiliser le code de retour de la fonction d'animation des missiles pour transmettre cette information. Ainsi si la fonction renvoie 0, le missile peut continuer d'exister. Dans le cas contraire, celui-ci doit être supprimé de la liste des objets et la mémoire allouée à sa création doit être libérée.

Vérifier à l'aide d'appels aux fonctions de débogage que les missiles créés sont bien détruits.

2.3 Quand notre missile fait Boum

Ajoutez une animation d'explosion à la disparition des missiles. Pour ce faire, vous créerez le module `explosion.h`, `explosion.c`.

2.4 Vers un réalisme poussé

Afin de rendre l'événènement d'explosion plus réaliste, faites en sorte d'appliquer un déplacement horizontal à l'explosion dans le même sens et à la même vitesse que les arbres du premier plan.