Projet Technologique « Battlefield Mario » : Fiche n°3

1 Gestion du temps

Pour intégrer le concept d'événements dans le jeu, vous allez mettre en place des outils de contrôle basés sur le temps. Ils s'avèrent très utiles pour générer de façon irrégulière des ennemis ou réaliser des transitions d'état d'objets animés indépendamment des actions du joueur. Vous pouvez par exemple utiliser un temporisateur pour gérer le temps pendant lequel l'oiseau est immobilisé suite à une collision.

1.1 Les temporisateurs

Dans un premier temps vous allez ajouter un nouveau module destiné à créer et utiliser des objets permettant l'exécution d'un code au terme d'un certain délai. La figure 1 présente l'interface de ce module. Implémentez ces fonctions dans le fichier `timer.c` au moyen des méthodes `SDL_AddTimer`, `SDL_RemoveTimer` et `SDL_PushEvent`.

Remarque : Assurez-vous de la présence du flag `SDL_INIT_TIMER` dans la liste des paramètres de l'appel à `SDL_Init`.

```c
#ifndef TIMER_IS_DEF
#define TIMER_IS_DEF

#include "SDL.h"
#include <stdint.h>

typedef SDL_TimerID timer_id_t;

int timer_init (void);
timer_id_t timer_set (Uint32 delay, void *param);
int timer_cancel (timer_id_t timer_id);

#endif
```

FIGURE 1 – Interface du module timer.h

1.2 Un temporisateur par objet

Un temporisateur doit être associé à un objet et chaque type d'objet doit avoir une fonction spécifique définissant l'action réalisée à l'expiration du temporisateur. Rajoutez un champ contenant un identifiant de temporisateur dans la structure d'objet. Toujours dans `object.h`, créez un type `timer_func_t` qui correspond à un pointeur de fonction qui ne renvoie rien et prend un pointeur de `dynamic_object_t` en paramètre. Ajoutez un objet de ce type dans la

structure `object_type_t` afin que chaque type d'objet ait une fonction timer définie.

Dans le module `animation`, définissez et implémentez les fonctions suivantes :

— `void animation_timer_add (dynamic_object_t *obj, Uint32 delay)` qui ajoute un temporisateur associé à l'objet `obj`;

— `void animation_timer_cancel(dynamic_object_t *obj)` qui supprime le temporisateur associé à l'objet `obj`;

— `void animation_timer_expired (void *arg1, void *arg2)` qui appelle la fonction d'animation de temporisateur expiré associée à l'objet dont l'adresse est contenue dans le pointeur `arg1`. Celle-ci sera appelé dans le traitant d'événements de la boucle principale du programme.

1.3 Comme un oiseau sans ailes

Vous allez faire en sorte que le contact de l'oiseau avec la limite supérieure de la fenêtre d'affichage rende impossible tout contrôle de l'utilisateur (déplacement ou tir de missile). Cet état doit perdurer quelques secondes seulement. Pour ce faire, vous implémenterez dans le module `bird` la fonction

```
void animation_bird_timer_expired (dynamic_object_t *obj)
```

qui sera appelée à l'expiration du temporisateur associé à l'oiseau.

Ajoutez un nouvel état d'objet `OBJECT_STATE_DEAD` dans `object.h` et affectez le à l'oiseau pendant la durée de son immobilisation.

1.4 Le canal Alpha

Pour souligner l'état d'immobilisation, vous allez modifier l'animation de l'oiseau de façon à ce que celui-ci *clignote* à l'écran (affichant une faiblesse temporaire). Pour y parvenir, vous avez la possibilité de jouer sur la transparence de l'objet, c'est-à-dire de modifier la valeur du canal alpha. La fonction suivante affecte cette valeur sur une texture donnée :

```
int SDL_SetTextureAlphaMod(SDL_Texture* texture, Uint8 alpha)
```

Utilisez cette méthode pour tester son effet et obtenir le comportement souhaité.

2 Les obstacles

2.1 Ennemis

Créez le module `bad.h`, `bad.c` définissant les fonctions des objets d'oiseaux ennemis. Utilisez le fichier `bad_bird.png` pour le sprite. Ces oiseaux ont une trajectoire sinusoïdale en sens inverse de celle du personnage principal.

2.2 Générateur d'ennemis

Utilisez les temporisateurs pour créer un module `generator` qui ajoute des oiseaux ennemis arrivant vers notre oiseau et ce à différentes hauteurs.

2.3 Détection des collisions

Pour l'heure l'oiseau se déplace librement en passant au travers de ses ennemis. L'objectif de cette partie est de modifier ce comportement en gérant les situations de collision entre notre oiseau et les autres objets solides.

Afin de simplifier notre problématique, vous allez considérer que deux objets donnés rentrent en collision si les bords des rectangles englobant les objets partagent les mêmes coordonnées.

Définissez un module `collision.h`, `collision.c`.