

TP de programmation concurrente

Billard revisité

Enoncé

Le but de ce TP est de constituer un jeu de billard un peu spécial, car les boules ne sont pas « tirées » par un joueur ici. Le plateau de jeu contient au départ 9 boules rouges et une boule bleue. Les boules rouges ont une vitesse et une direction qui sont déterminées aléatoirement et la boule bleue est à l'arrêt. L'accélération de la boule bleue peut être contrôlée par l'utilisateur afin de percuter les autres boules.

Tant que les boules rouges sont en mouvement, elles peuvent percuter les autres (y compris la bleue) et les font dévier de leur trajectoire selon les règles de la physique.

Mais lorsqu'une boule rouge est arrêtée et qu'elle est percutée, cette dernière se « désagrège » sur place et finit par disparaître. On appelle cela une « explosion » dans cet énoncé.

A n'importe quel moment, le joueur doit pouvoir relancer la partie (il n'est pas obligé d'attendre que toutes les boules rouges aient disparu). Le joueur peut aussi décider de sortir du jeu à n'importe quel moment.

Spécification détaillée

Afin d'entraîner la programmation concurrente, les tâches suivantes devront être définies :

- Un thread par boule
- Un thread pour la gestion de l'affichage
- Un thread pour la gestion du clavier

Gestion des boules

- La base de temps pour chaque déplacement est de 7 ms.
- A chaque début de partie, les boules doivent être disposées de façon équidistante en diagonale sur tout le billard. A l'exception de la boule du joueur (bleue) qui est arrêtée au départ, les autres boules démarrent avec un vecteur vitesse aléatoire. Les vecteurs vitesse sont représentés en 2 dimensions (x,y) où x et y $\in [-3 ; 3]$ pixel/7 ms. Si la vitesse d'une boule passe au-dessous de 0.05 pixel/7 ms, **la boule s'arrête**. Notez que le module de la vitesse peut être estimé ainsi : $|\vec{V}| \cong |V_x| + |V_y|$.
- La vitesse de la boule du joueur doit être déterminée par son accélération qui est de 0.1 pixel/7 ms² en direction de la touche flèche appuyée (voir « gestion du clavier »). Rappel : $V(t) = a \cdot t$. Donc en partant d'une vitesse nulle, $V(7 \text{ ms}) = 0.1$, $V(14 \text{ ms}) = 0.2$, etc...
- Sauf si elle est accélérée, chaque boule décélère par frottement de 0.002 pixels/7 ms².
- La collision d'une boule avec un bord inverse la composante du vecteur vitesse concerné. Ainsi une collision avec un bord gauche ou droit inverse la composante 'x' du vecteur vitesse alors qu'une collision avec un bord haut ou bas inverse la composante 'y' de ce même vecteur.

- La collision avec une autre boule modifie les vecteurs vitesse des 2 boules en collision selon les lois de la physique. La fonction fournie `calc_collision_speeds()` calcule les vitesses résultantes en cas de collision. Vous pouvez néanmoins vous baser sur l'article en référence ci-dessous¹ si vous voulez les calculer vous-même (**bonus +0.5 point**). Le son « chock » doit être produit lors d'une collision entre boules (voir la primitive SDL à utiliser dans l'annexe).
- A l'exception de la boule du joueur, lorsqu'une boule qui est à l'arrêt est percutée, celle-ci « explose » puis disparaît. L'explosion est matérialisée par l'affichage d'une succession d'image particulières (voir gestion des explosions ci-dessous).

Gestion des explosions

Lorsqu'une explosion intervient, le son « broken » doit être produit, puis toutes les 200 ms, il s'agit d'afficher les matrices « explode0 » à « explode4 » et terminer par effacer la boule avec la texture « black_ball ». Celle-ci disparaît jusqu'à ce que le joueur relance la partie. **Toutes les collisions ayant lieu avec une boule en train d'exploser sont ignorées.**

Gestion du clavier

L'utilisation du clavier est résumé dans la table suivante.

Touche	Action
s	Démarre ou redémarre le jeu comme indiqué dans « gestion des boules ». Le joueur peut décider de relancer une partie en cours.
Escape ou fermeture fenêtre	Sort du jeu (mais interdiction d'utiliser <code>exit()</code> ou <code>pthread_exit()</code> !). Cela peut arriver n'importe quand.
Flèches	Accélère la boule du joueur de 0.1 pixel/7 ms ² dans la (ou les) direction(s) indiquée(s)

Gestion de l'affichage et des sons

Toutes les 20 ms, l'ensemble des boules doit être affiché par le thread qui gère l'affichage. Toutes les primitives SDL qui concernent l'affichage doivent se trouver dans ce thread. La production du son peut être faite dans un autre thread, même si son initialisation est faite dans le thread d'affichage (voir annexe).

¹ [Choc élastique en 2 dimensions](#), Pascal Rebetez

Contraintes de développement

- Utilisez au moins une **variable de condition** dans votre code (au bon endroit bien sûr !)
- **Les sémaphores sont proscrits** pour résoudre ce problème.
- Vous devez permettre la concurrence entre les threads, en évitant tout problème de synchronisation.
- Toutes les attentes doivent être passives.
- Sauf en cas d'erreur, le programme doit se terminer normalement, c'est-à-dire **sans utiliser `exit()` ni `pthread_exit()`, ni `pthread_cancel()`**.
- Testez tous les retours des primitives de l'OS lors de leur initialisation (ou de la destruction d'objets).
- Le code rendu doit **compiler sans erreur et sans warning**.
- Seuls les commentaires utiles à la compréhension du code sont demandés (si vous les jugez nécessaires).

Rendu

Ecrivez un mini-rapport au format PDF qui contient :

- le nom du TP
- vos noms, prénoms et numéro de groupe
- le statut du projet rendu (fonctionnel ou non, bugs résiduels, etc.). Ce statut doit être exhaustif concernant le comportement du programme.
- Un organigramme décrivant le comportement **d'une boule** (pas celui du `main()` !)

Le mini-rapport doit rester court comme son nom l'indique. Ne répétez pas les éléments de cet énoncé, en revanche vous pouvez y faire référence si nécessaire.

Créez une seule archive contenant tous les fichiers du projet ainsi que le mini-rapport et nommez-la **G<numéro_du_groupe>_billard.zip**. Ce fichier doit être rendu sur Cyberlearn.

Annexe : description simplifiée de l'affichage avec la librairie SDL

La SDL (ou SDL2) est une librairie contenant des utilitaires graphiques, de gestion de du son et du clavier principalement, dont la documentation complète se trouve [ici](#).

Son installation peut se faire sur Ubuntu avec la commande suivante :

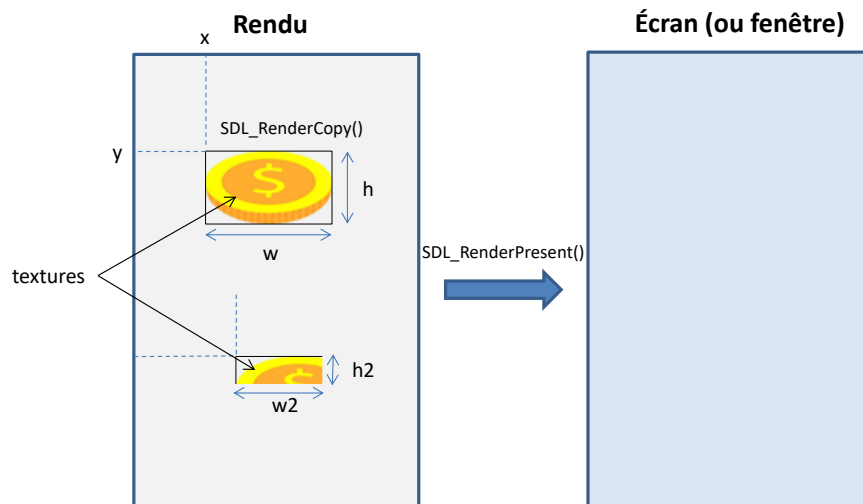
```
sudo apt-get install libghc-sdl2-dev libsdl2-image-dev libsdl2-mixer-dev
```

Note : le warning d'exécution suivant « *libpng warning: iCCP: known incorrect sRGB profile* » provient de la librairie SDL. Il n'est pas gênant pour l'exécution, mais il est compliqué à supprimer : veuillez donc l'ignorer.

Le but de cette de cette annexe est de vous expliquer le principe d'affichage de la SDL en quelques lignes. **L'initialisation de la librairie n'est pas décrite, car elle est fournie dans le code du projet de base. Faites attention de bien appeler `SDL_Init()` avant toute autre fonction qui utilise la librairie SDL !**

Principe général d'utilisation de la SDL

La SDL permet de manipuler des « textures » : les textures sont des matrices stockées en mémoire qui contiennent une image. Ces textures peuvent être copiées sur un « rendu », qui consiste en une zone mémoire qui sert de tampon, et finalement ce rendu peut être affiché à l'écran.



Les textures pourront être chargées directement depuis des fichiers d'images grâce aux utilitaires fournis dans **utilities.c**.

La fonction ***SDL_RenderCopy()*** copie une texture sur le rendu. Sa taille et sa position sur le rendu sont données par les paramètres de la fonction *src_rect* et *dst_rect* qui sont de type **SDL_Rect**. **SDL_Rect** est une structure qui contient 4 paramètres : x,y,w,h

(x,y) sont les coordonnées du point haut gauche de la texture par rapport au rendu
(w,h) sont les dimensions (largeur, hauteur) de la texture à afficher

Un appel typique prend les paramètres suivants :

```
SDL_RenderCopy(renderer, texture, &src_rect, &dst_rect);
```

Cela permet de copier une partie d'une texture sur une partie du rendu, dont les dimensions (et la position) sont fixées par *src_rect* (côté texture) et *dst_rect* (côté rendu). Si &src_rect=NULL, la texture complète sera copiée (mais pourrait encore être sectionnée par les dimensions spécifiées dans *dst_rect*).

Notez que toutes les textures à afficher doivent faire l'objet d'un appel à *SDL_RenderCopy()*, dans l'ordre dans lequel elles doivent se superposer (fond, roues, puis les pièces de monnaie en commençant par le bas). Ensuite, appelez la fonction suivante qui présente le rendu à l'écran :

```
SDL_RenderPresent(renderer)
```

Création d'un rectangle de couleur unie

```
SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255) et
```

```
SDL_RenderFillRect(renderer, &square_rect)
```

 créent un rectangle blanc sur le rendu (utile pour implémenter le bonus)

Gestion du clavier (et/ou des événements)

Pour la gestion du clavier, un exemple de code vous est fourni dans **display_and_sound.c**. la fonction `SDL_WaitEvent(&event)` attend un événement quelconque (attente passive). Les événements qui nous intéressent sont traités dans l'exemple : appui d'une touche et test de relâchement ou clic sur la fermeture de la fenêtre (voir **display_and_sound.c**).

Production des sons

Pour produire un son il faut :

- Ouvrir le media 'audio'
- Charger le son à jouer
- Jouer le son

Cela peut être fait ainsi :

```
Mix_Chunk *sound;  
Mix_OpenAudio(sampling_frequency, MIX_DEFAULT_FORMAT, nb_channels,  
buffer_size)  
sound = Mix_LoadWAV("sound.wav")  
Mix_PlayChannel(-1, sound, 0)
```

La taille du tampon indique grossièrement la granularité d'un échantillon sonore. Par exemple avec une fréquence d'échantillonnage de 20 kHz et un tampon de 512 échantillons, le son sera découpé et joué toutes les 25.6 ms.

La fonction `Mix_PlayChannel()` peut être appelée dans un autre thread que celui de son initialisation.