TD 4 : Algorithmes de jeu vidéo

L'ensemble de ces exercices sont destinés à approfondir votre apprentissage
Merci de ne pas vous servir de ChatGpt ou Copilot

Tout au long du TD6 et du TP6 nous utiliserons la SDL2, Simple DirectMedia Layer.
C'est une librairie cross-plateforme bas niveaux qui permet d'accéder à des fonctionnalités comme la gestion graphique (processeur et GPU), la gestion audio, la gestion des périphériques (souris, clavier, manette).
Elle est utilisable en C (et aussi C++,C#, Python …).

Collisions dans les étoiles

Nous utiliserons le template de code fourni « ship.c ».

Vous retrouvez les includes de la SDL dans les premières lignes du source :
#include <SDL.h>
#include <SDL_image.h>
#include <SDL_ttf.h>

SDL.h
Elle initialise le system SDL, elle va vous permettre de :

- créer une fenêtre dont on spécifiera la taille (640 sur 400)
- créer un canvas, appelé Renderer pour manipuler les pixels de la fenêtre, en affichant une image etc.
- gérer les événements
- définir des primitives type SDL_Surface, SDL_Color etc.

SDL_image.h
Elle gère la chargement des images en prenant en charge de multiples format dont le PNG
Elle permet d'initialiser des primitives de type surface très utilisée pour la suite de notre TD/TP

SDL_ttf.h
Elle gère la partie rendu de texte dans la fenêtre créée via la SDL
La fenêtre en SDL ouvre un rendu en OpenGL. Celui-ci ne permet pas d'afficher des textes à coup de printf.
Le texte demandé par l'utilisateur est rendu en 2D dans une surface qui est ensuite affichée via un RenderCopy qui blitte (copie) la surface dans la fenêtre OpenGl sous la forme d'une texture créée à la volée.

1. Analyser le source

La fonction init créée une fenêtre et son renderer.
Elle colorise les pixels de la fenêtre en noire.
Elle nettoie le renderer (et donc la fenêtre).

La fonction getTextureFromImage définie une surface qui recevra les données du fichier image choisi. Elle transforme ensuite cette surface en texture qui sera utilisable dans le renderer.

La compilation sera réalisée via :

```
gcc -Wall ship.c -o ship $(sdl2-config --cflags --libs) -lSDL2_image -lSDL2_ttf
```

On retrouve dans cette ligne de commande des flags de compilation liés à l'utilisation de la SDL et au lien nécessaire avec les bibliothèques que nous avons ajouté en tête du template.

La structure « astro » sera utilisée pour les asteroids dans le cadre d'un tableau de structure.

SDL_Rect défini une structure composée des 4 éléments : x,y,w,h. Il permet de positionner à l'écran une texture qui sera au préalable initialisée via une surface grâce à la fonction getTextureFromImage.

Nous utiliserons l'équivalent de time.h dans la SDL : SDL_GetPerformanceCounter()
Il nous permettra de mesurer le nombre de FPS que prend chaque tour de boucle principale.

Question 1 : Quelle st le format des images que nous utiliserons dans ce TD/TP (asteroid et ship) ? Pourquoi ?
Question 2 : Comment est géré le déplacement du vaisseau spatial ?

Question 3 : Quelle est la différence entre une surface et une texture ? (RAM / VRAM)

2. Gérer les collisions

Le vaisseau se déplace via les flèches du clavier. Il rentre en collision avec les astéroïdes.
Le nombre de FPS et le nombre de collisions sont affichés en haut à gauche.
Le template utilise la fonction SDL_HasIntersection(SDL_RECT,SDL_RECT).
Elle utilise un algorithme de collision de type AABB comme vu en cours.

Après analyse, remplacer cette fonction par la vôtre.
Elle implémentera le AABB que vous optimiserez pour essayer de battre celui de la SDL.
Pour vous aider :
https://zestedesavoir.com/tutoriels/2835/theorie-des-collisions/collisions-en-2d/formes-simples/

Quel est la complexité de cet algorithme ?
Est-il efficient ? Quel est son principal défaut ?

Collisions taclées

Dans le répertoire balls vous trouverez :

- le source balls.c
- l'image ball.png
- la police de caractère time.ttf

Commande de compilation :
gcc -Wall balls.c -o balls $(sdl2-config --cflags --libs) -lSDL2_image -lSDL2_ttf

1. AABB

Implémenter votre algorithme en AABB dans cet exercice.
Comme pour le précédent exercice on mesurera le nombre de FPS
Après analyse, voyez vous une différence dans la stratégie que vous avez mis en place versus celle de l'exercice précédent ? Expliquez ? Quelle complexité ?

2. Circle collider

Implémenter un algorithme de type Circle Collider à la place du AABB
Mesurer le nombre de FPS
Qu'en concluez-vous ?
Peut on faire mieux ?

3. Diviser pour moins gérer

Implémenter dans votre algorithme précédent la notion de zone comme vu en cours
L'idée :

- diviser l'écran en 4 zones, Nord Ouest, Nord Est, Sud Ouest, Sud Est
- modifier la structure de donnée « Balle » pour ajouter une notion de zone
- adapter votre gestion de collision en tenant compte de cette nouvelle information

Est il possible de faire mieux encore ? (quadtree...)