Space Invaders

Projet réalisé par Ismaël Costel et Steven Ea en EISE4

Corrigé par : Madame Cécile Braunstein

Introduction

Dans le cadre de notre projet en C++, nous avions dû développer un programme qui rentre dans le thème imposé : **Le monde d'après**.

Malgré le thème imposé, le sujet est plutôt libre car le thème est très large et il dépend de comment chaque humain l'imagine. Bien sûr, en plus du thème imposé, nous sommes aussi soumis à des contraintes qui concerne le code comme par exemple l'utilisation d'un certain nombre de classes, ou encore d'au moins trois niveaux de hiérarchie.

En ce qui concerne le choix de notre sujet, nous avons tout d'abord décidé de concevoir un jeu. Pour nous, le monde d'après nous évoque un peu l'apocalypse/la fin du monde, on pense qu'il y a surement d'autre être vivant doté d'intelligence dans la galaxie. C'est donc pour ça, que l'espace, qui n'a pas encore pu être complétement exploré par l'être humain peut être une source de trésor inépuisable vu l'immensité de l'univers. Bientôt, la terre ne pourra plus nous fournir les ressources nécessaires à notre survie. On pense donc que pour nous en sortir, il va donc falloir trouver de nouvelles planètes habitables, mais aussi nous défendre contre des aliens qui viendraient de ces planètes. C'est pourquoi on a décidé de partir sur une bataille spatiale, et quoi de mieux que le bon vieux jeu rétro:le space invaders. Dans ce jeu, des rangées d'ennemis, des aliens, essayent de nous envahir, tandis que nous, équipé d'un vaisseau qui peut tirer des lasers et des missiles, on doit défendre la planète en tirant sur eux pour les faire disparaitre.

Description du jeu développé

L'application est un Space invaders classique. Des vagues d'aliens apparaissent et avance. On contrôle un vaisseau et notre but est tirer sur tous les aliens jusqu'à faire disparaitre toute la vague. Si on arrive à faire disparaitre la vague alors, une autre vague apparait avec une ligne d'alien en plus. Lorsqu'un alien touche le vaisseau ou le bas de l'écran, c'est game over.

Pour contrôler le vaisseau, on utilise les flèches directionnelles du clavier pour se déplacer, et on tire avec la barre espace et x. La barre espace permet de tirer des lasers, tandis que x permet de tirer des missiles.

Pour développer ce Space invaders, on a utilisé la librairie graphique SFML qui est très simple a utilisé. Elle nous a permit d'afficher le jeu dans une fenêtre graphique à la place du terminal et d'ajouter des images pour les textures dans le jeu.

Description des classes

On a donc créé plusieurs classes pour le jeu :

Personnage : Cette classe permet d'ajouter une texture pour la classe **Vaisseau**, **Arme** et **Alien** car il faut leur affecter une texture lorsque on les crée pour pouvoir les afficher dans l'interface graphique.

Vaisseau : Elle hérite donc de la classe **Personnage**. La classe **Vaisseau** possède trois attributs, un ***booléen alive*** qui permet de savoir si le vaisseau est toujours en vie, et de deux armes. C'est cette classe qui possède les méthodes pour faire déplacer le vaisseau et sélectionner l'arme avec laquelle tirer.

Arme : Elle hérite de la classe **Personnage**. Elle possède un attribut ***int dispo*** qui permet de savoir si l'arme est prête pour tirer, et elle a une méthode pour tirer.

Laser et **Missile :** Les deux classes hérite de la classe **Arme**. On les a créés pour pouvoir affecter deux textures différentes (**Laser** et **missiles**) lorsque l'on tire.

Alien : Elle hérite de la classe **Personnage**. Elle possède un ***booléen alive*** qui est utiliser pour savoir si l'alien est encore en vie grâce à la méthode ***isAlive()***. Les méthodes ***updatestatus()*** et ***updategamestatus()*** permet de détecter si il y a une collision entre l'alien et une arme, ou entre l'alien et le vaisseau. S'il y a une collision, on change le booléen de l'alien ou du vaisseau en fonction de la collision.

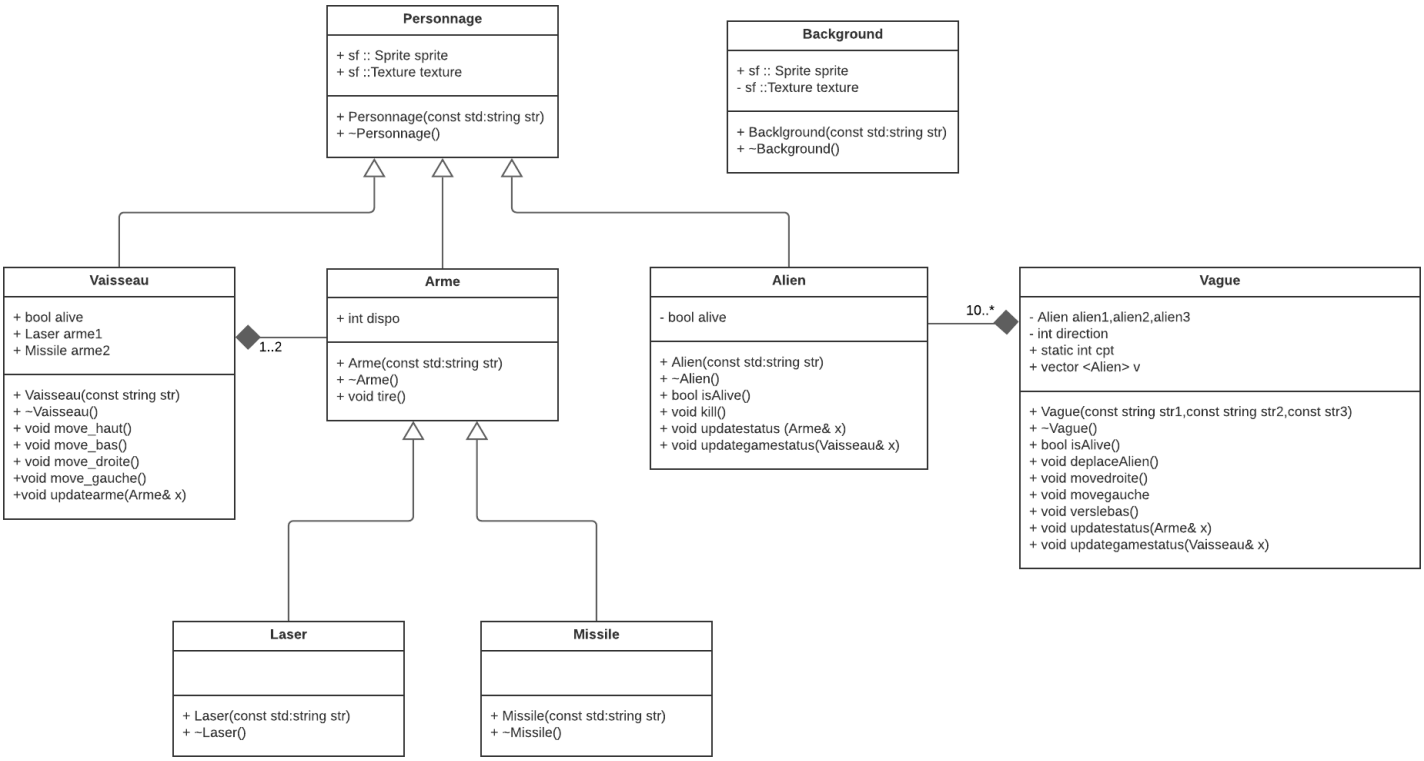
Vague : La classe **Vague** est composé de plusieurs Aliens et c'est la classe la plus complexe. C'est cette classe qui va nous permettre de créer des vagues d'alien. Chaque vague sera constitué de ligne de 10 aliens stocker dans un **conteneur STL <vector>**. Dans notre jeu, la première vague possède une ligne de 10 aliens, et chaque fois que la vague est détruite, on rajoute une autre ligne d'alien.

Nous avons mis 3 textures pour les aliens, et à l'aide d'un générateur de chiffre aléatoire, on affecte une texture à chaque alien de la ligne. Ainsi, les lignes d'aliens créé sont différentes les unes des autres. Pour déplacer la vague, on appelle la méthode déplaceAlien() qui utilise la variable ***int direction.*** Elle nous permet de savoir dans quelle direction la vague doit se déplacer, et on appelle la méthode correspondante (**movedroite(), movegauche() ou verslebas()**) pour modifier les coordonnées.

Background : C'est une classe que l'on a créé pour ajouter une texture de fond à notre interface graphique.

Diagramme UML

Voici donc le diagramme UML qui montre la structure de notre système.



Partie Graphique

On utilise la librairie SFML.

La gestion de l'interface graphique du jeu est réalisée dans le fichier ***main.cc***. Le fichier ***Personnage.cc*** et **Background.cc** utilise la librairie pour inclure les images qui serviront de texture dans l'interface graphique.

Tout d'abord, il faut inclure la bibliothèque pour avoir accès aux fonctions.

Puis on crée la fenêtre et on définit la vitesse d'affichage avec les fonctions :

sf::RenderWindow window(sf::VideoMode(800, 800), "SFML works!");

window.setFramerateLimit(60);

Ensuite, on définit une variable ***sf ::Event event*** .La programmation graphique est une programmation évènementielle, c'est-à-dire que dans une boucle infini, nous allons attendre un évènements et faire une action en conséquence.

Dans notre cas, on cherche deux choses :

- Est-ce que la fenêtre a été fermé. Si oui, on ferme donc la fenêtre.
- Quelle touche du clavier a été appuyé. Si c'est une flèche directionnelle, on appelle une méthode pour changer la coordonnée du vaisseau. Si c'est la barre espace ou x alors on appelle une méthode pour générer un tir.

Si un évènement s'est produit, alors on ferme la fenêtre, on met à jour l'interface graphique et on rouvre la fenêtre mise à jour.

Explication du code principale

Après avoir créé une fenêtre pour l'interface graphique, on crée ensuite les classes nécessaires pour le fonctionnement du jeu et on leur affecte une texture.

Background space("space_2.jpeg");

Vaisseau ship("ship_1.png");

*Vague *test= new Vague("alien_1.png","alien_2.png","alien_3.png");*

On utilise ensuite une boucle qui fonctionne lorsque la fenêtre est ouverte qui permet de détecter les évènements.

Une fois qu'un évènement se produit, on sort donc de la boucle et on efface tout ce qu'on a mit dans la fenêtre pour pouvoir la redessiner avec les paramètres mis à jour.

Pour cela on regarde si :

- Des tirs ont été effectué

```
if(!ship.arme2.dispo)
    window.draw(ship.arme2.sprite);

if(!ship.arme1.dispo)
    window.draw(ship.arme1.sprite);
```

- Des aliens sont morts

```
for( size_t j = 0; j<10*test->cpt; j++)
{
    if(test->v[j].isAlive())
        window.draw(test->v[j].sprite);
}
window.draw(ship.sprite);
```

- Le vaisseau est mort

```
if(!ship.alive)
{
    window.draw(stop.sprite);
}
window.display();
```

Lorsque le vaisseau est mort, on a perdu et un game over apparait sur l'écran. L'affichage continue de se faire, mais on ne peut plus interagir avec le jeu.

- Le vaisseau est en vie

```
if(ship.alive)
{
    test->deplaceAlien();
    ship.updatearme(ship.arme1);
    ship.updatearme(ship.arme2);
    test->updatestatus(ship.arme1);
    test->updatestatus(ship.arme2);
    test->updategamestatus(ship);
}
```

Si le vaisseau est encore en vie, alors on va mettre à jour les positions et statuts des aliens et du vaisseau pour préparer l'affichage à la prochaine itération.

Procédure d'installation

Télécharger le projet dans le github (mettre le lien)

Installer la librairie graphique SFML avec la commande *sudo apt-get install libsfml_dev*

Aller ensuite dans le dossier du projet et utiliser la commande *make* dans le terminal pour compiler.

Ensuite il ne vous reste plus qu'à lancer l'exécutable *./main* pour jouer au jeu.

Commande pour jouer :

- Flèche directionnelle : déplacement
- Barre espace et x : tirer