# RAPPORT DE PROJET

## Programmation Orientée Objet Avancée - C++


### Résumé

Asteroid Destruction, un petit jeu programmé en C++



Simon Naveau;Loïc Travaillé

naveau.e1600174@etud.univ-ubs.fr – travaille.e1804502@etud.univ-ubs.fr

# Table des matières

# Présentation globale du jeu

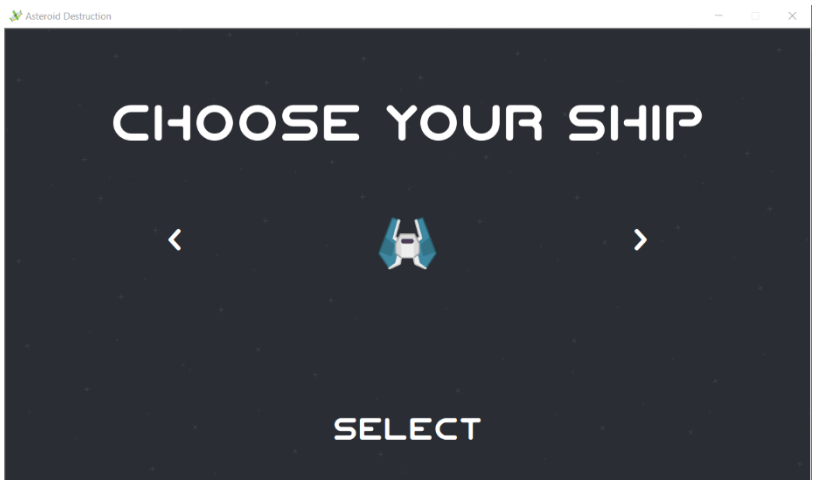
## Description :

Nous avons développé un jeu dans lequel il faut piloter un vaisseau spatial et survivre aux vagues d'astéroïdes un certain temps. Le jeu est composé de 10 niveaux, avec une difficulté croissante au fil des niveaux. En effet, le nombre d'astéroïdes augmente, et le temps nécessaire pour finir le niveau est de plus en plus long. Les astéroïdes apparaissent aléatoirement sur l'écran du jeu. Il est possible de détruire ces astéroïdes en tirant dessus ou en ayant une collision avec eux. Attention cependant, chaque collision engendre une perte de points de vie importante du vaisseau.

Les astéroïdes ont 3 tailles : grand, moyen, petit. Ils apparaissent sous leur plus grande forme, et si le joueur tire dessus une fois, le gros astéroïde se divise en 3 moyens, et s'il tire à nouveau sur un moyen, il se divise en 3 petits.

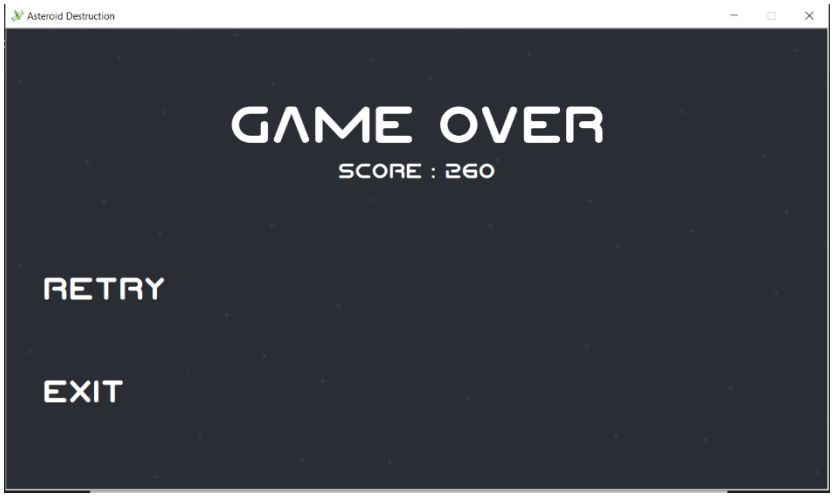
Le joueur commence la partie avec 1000 points de vie. Si un gros astéroïde le touche, il en perd 300, 150 pour un moyen et 100 pour un petit. Cependant, à la destruction d'un gros astéroïde, il peut aléatoirement y avoir un bonus qui apparaît, redonnant alors 100 points de vie au joueur.

## Aperçus

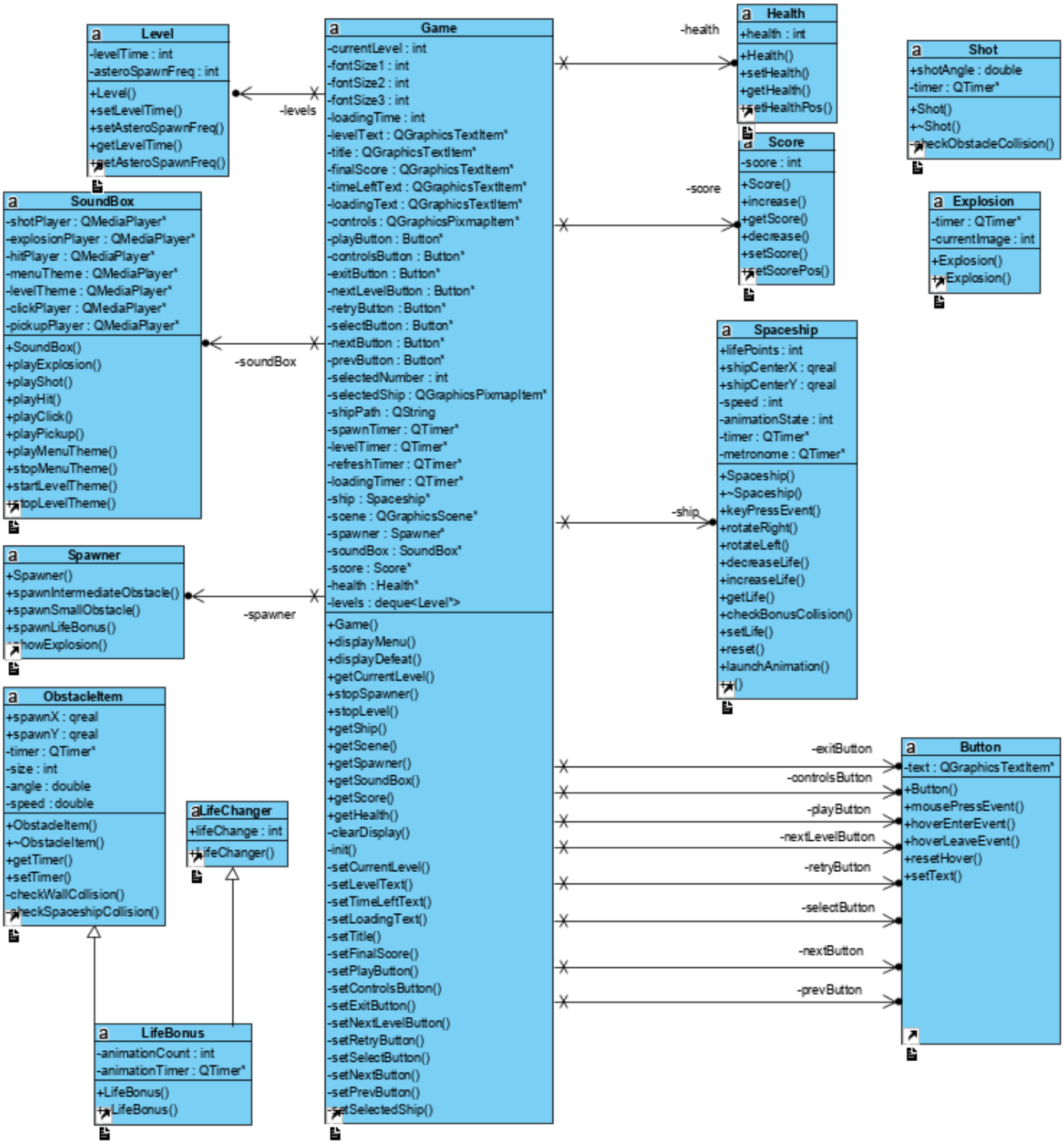

Asteroid Destruction
LEVEL 2
HEALTH: 850
SCORE: 210
TIME : 0:15


Asteroid Destruction
LEVEL COMPLETE
NEXT LEVEL


Asteroid Destruction
GAME OVER
SCORE : 260
RETRY
EXIT

## Diagramme de classes

Voici le diagramme de classes final du projet




*"LifeBonus"* est un *"ObstacleItem"* tout en étant aussi un *"LifeChanger"*.

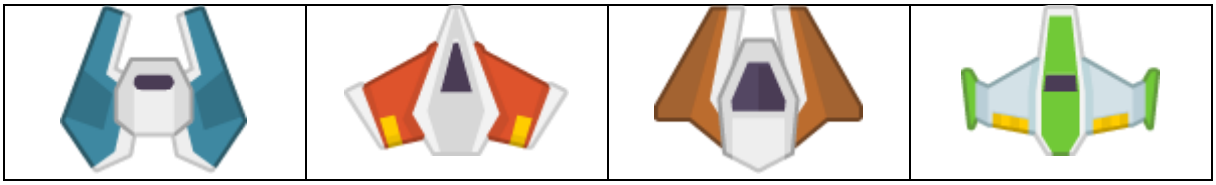
## Description des fonctionnalités

Lors du lancement du jeu, le joueur arrive sur un menu et il dispose de 3 boutons. Le premier permet de lancer une partie, le deuxième permet de voir les commandes qui lui seront utiles pour jouer et le troisième permet de quitter le jeu.

S'il clique sur le bouton « Jouer », il peut ensuite choisir son vaisseau. 4 vaisseaux sont actuellement disponibles. Il en choisit un, puis clique sur le bouton situé en bas de l'écran. Un compte à rebours « 3 2 1 » apparaît alors, puis la partie se lance. Il se retrouve alors avec son vaisseau placé au milieu de la fenêtre, à l'arrêt. Dès qu'il appuie sur Z (ou flèche du haut) le vaisseau entre dans le mode de déplacement automatique. Il avance, tout seul, tout droit. Il peut tourner vers la gauche avec Q (ou flèche de gauche) et vers la droite avec D (ou flèche de droite). Il peut également effectuer une impulsion vers l'avant avec Z (ou flèche du haut) et tirer des faisceaux lasers en utilisant la barre espace.

Au bout d'un certain temps (affiché en haut à droite), le niveau se termine. Le joueur arrive alors sur un écran lui affichant un message comme quoi il a réussi ce niveau, puis il dispose d'un bouton pour passer au niveau suivant. S'il finit tous les niveaux, il aura un écran similaire mais lui disant qu'il a fini le jeu, lui affichant son score final et lui proposant de rejouer ou de quitter.


Le joueur peut choisir un vaisseau spatial au début de la partie. Il a le choix entre les 4 vaisseaux suivants :



## Enchainement des niveaux

Les 10 niveaux se composent ainsi :

| NIVEAUX | TEMPS DE SURVIE | FRÉQUENCE DE SPAWN |
|---|---|---|
| Niveau 1 | 30 secondes | Toutes les 5 secondes |
| Niveau 2 | 40 secondes | Toutes les 4,8 secondes |
| Niveau 3 | 50 secondes | Toutes les 4,4 secondes |
| Niveau 4 | 1 minute | Toutes les 4,2 secondes |
| Niveau 5 | 1 minute 10 secondes | Toutes les 4 secondes |
| Niveau 6 | 1 minute 20 secondes | Toutes les 3,8 secondes |
| Niveau 7 | 1 minute 30 secondes | Toutes les 3,6 secondes |
| Niveau 8 | 1 minute 40 secondes | Toutes les 3,4 secondes |
| Niveau 9 | 1 minute 50 secondes | Toutes les 3,2 secondes |
| Niveau 10 | 2 minutes | Toutes les 3 secondes |

L'entité de "Game" possède cette liste d'objets "Level" et les enchaine les uns après les autres en mettant à jour à chaque fois les paramètres du "Spawner" et le temps avant le passage au niveau suivant.

## Détection des évènements clavier

```
void Spaceship::keyPressEvent(QKeyEvent *event) {
    if (event->key() == Qt::Key_Left or event->key() == Qt::Key_Q) {
        rotateLeft();

    } else if (event->key() == Qt::Key_Right or event->key() == Qt::Key_D) {
        rotateRight();

    } else if (event->key() == Qt::Key_Space) {

        Shot *shot = new Shot();
        if(rotation()> 150 && rotation() < 210){
            shot->setPos(shipCenterX+(shot->pixmap().height()/2), shipCenterY);
        } else {

            shot->setPos(shipCenterX-(shot->pixmap().height()/2), shipCenterY);
        }
        shot->setZValue(-10);
        scene()->addItem(shot);
        game->getSoundBox()->playShot();

    } else if (event->key() == Qt::Key_Up or event->key() == Qt::Key_Z) {
        speed = 8;
        timer = new QTimer();
        connect(timer, SIGNAL(timeout()), this, SLOT(resetSpeed()));
        timer->start(100);
    }
}
```

Pour détecter les appuis de l'utilisateur sur les touches du clavier l'instance représentant le vaisseau possède une méthode qui agit lors de la détection de n'importe quel événement clavier. Après nous regardons si cette touche correspond à une touche de déplacement ou de tir puis nous appliquons l'action correspondante (créer un laser, effectuer une rotation, …).

## Détection des collisions

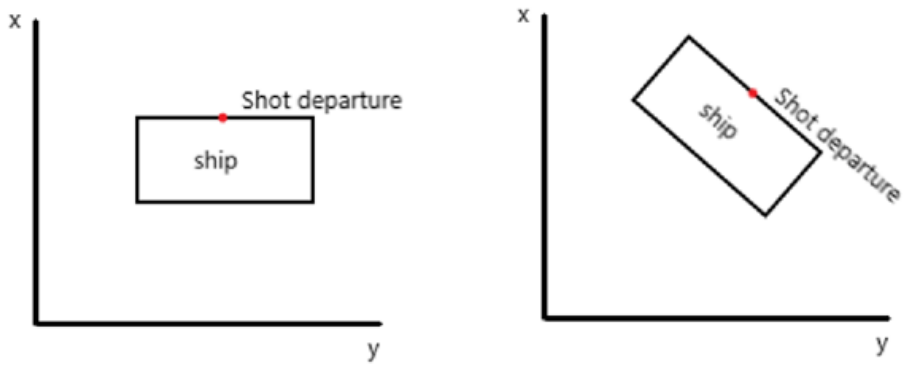
```
QList < QGraphicsItem * > colliding_items = collidingItems();
for (int i = 0, n = colliding_items.size(); i < n; ++i) {
    if (typeid(*(colliding_items[i])) == typeid(LifeBonus)) {     ⚠ expression with
        this->setLife(*this + qgraphicsitem_cast<LifeBonus *>(colliding_items[i]));
        game->getHealth()->setHealth(this->getLife());
        game->getSoundBox()->playPickup();
        scene()->removeItem(colliding_items[i]);
        delete colliding_items[i];
        return;
    }
}
```

Le fonctionnement du jeu est étroitement lié à la détection de collision entre objets d'une scène. Pour effectuer cette détection Qt nous simplifie grandement la vie. En effet nous utilisons sa méthode collidingItems() qui retourne la liste tous les objets en contact avec l'objet sur lequel elle est appliqué. Ainsi nous n'avons plus qu'à boucler sur la liste pour trouver les objets du type que l'on souhaite (Bonus, Asteroid, tir, …) puis effectuer les actions correspondantes.

# Problèmes rencontrés

## Tirs non-centrés

Un problème mathématique c'est rapidement posé lors de la génération des tirs effectués par le vaisseau. En effet si le vaisseau se voyait appliqué une rotation les tirs de celui-ci le devaient aussi. Jusqu'ici aucun problème mais la difficulté se trouva dans l'obtention du point de départ de ces tirs.




En effet comme le montre le schéma ci-dessus, la position de départ des tirs se déplace avec le vaisseau. Après plusieurs heures de réflexion dans les méandres de la géométrie de lycée et d'implémentation de solutions provenant de forums nous avons pensé à déployer une autre stratégie. Comme nos lasers une fois partis suivent très bien leur trajectoire avec la bonne rotation, nous avons choisi de toujours faire partir les tirs du centre du vaisseau avec la bonne rotation. Ainsi en plaçant les lasers sur un plan (z) inférieur on a l'impression qu'ils partent toujours bien de l'avant du vaisseau. Le subterfuge est fonctionnel. Pour cela on garde en variable la position de départ du centre du vaisseau à laquelle on applique les mêmes déplacements que le vaisseau.

```
setPos(x() + dx, y() + dy);
shipCenterX = shipCenterX + dx;
shipCenterY = shipCenterY + dy;
```

## Suppression des pointeurs

Une fois le développement des fonctionnalités de base terminé, nous avons détecté un petit problème d'optimisation. En effet quand nous tirions un grand nombre de laser ou que la partie commençait à s'éterniser, certain de nos tirs commençait à avancer au ralentit. Après quelques recherches nous avons déduit que cela venait d'une saturation de la mémoire. Il fallait donc trouver ce qui n'était pas bien supprimé lors de la destruction des objets. Nous avons repéré une mauvaise suppression des timers de déplacement des tirs. Pour régler ce problème nous avons défini dans chaque classe un destructeur qui arrête proprement les timers et supprime les pointeurs.

```
Shot::~Shot() {
    timer->stop();
    delete timer;
}
```

## Destruction du vaisseau

Un problème très important a été rencontré lorsque le vaisseau n'avait plus de points de vie. Si l'objet du vaisseau était détruit, le jeu se fermait systématiquement. Il était donc impossible de relancer une partie directement depuis le jeu ; et surtout on ne voyait pas son score final. Nous avons donc trouvé une solution moins propre mais efficace, qui consiste juste à masquer le vaisseau lorsque le joueur perd sa partie. Ainsi, on peut afficher un menu lui donnant son score final et lui proposant de rejouer. Tout est alors réinitialisé s'il choisit de rejouer une partie (score, nombre de points de vie, niveau actuel remis à 1, position du vaisseau…).

```
game->stopSpawner();
game->getShip()->clearFocus();
game->getShip()->setVisible(0);
game->getHealth()->setVisible(0);
game->getScore()->setVisible(0);
game->stopLevel();

game->displayDefeat();
delete this;
return;
```