Projet Jeu

Space Raclure

La Providence (2019-2020)

BTS SN2

Brard Adrien / Lhermitte Nathan / Boury Aurélien / Jouen Matthias

SOMMAIRE

Présentation du jeu

Nous avons eu pour but de créer un jeu dans l'un de notre projet cette année.

Notre jeu est un jeu vidéo d'obstacle, il est codé en Java ; un langage que nous voulions approfondir/découvrir pour élargir nos connaissances.

Notre choix du Java :

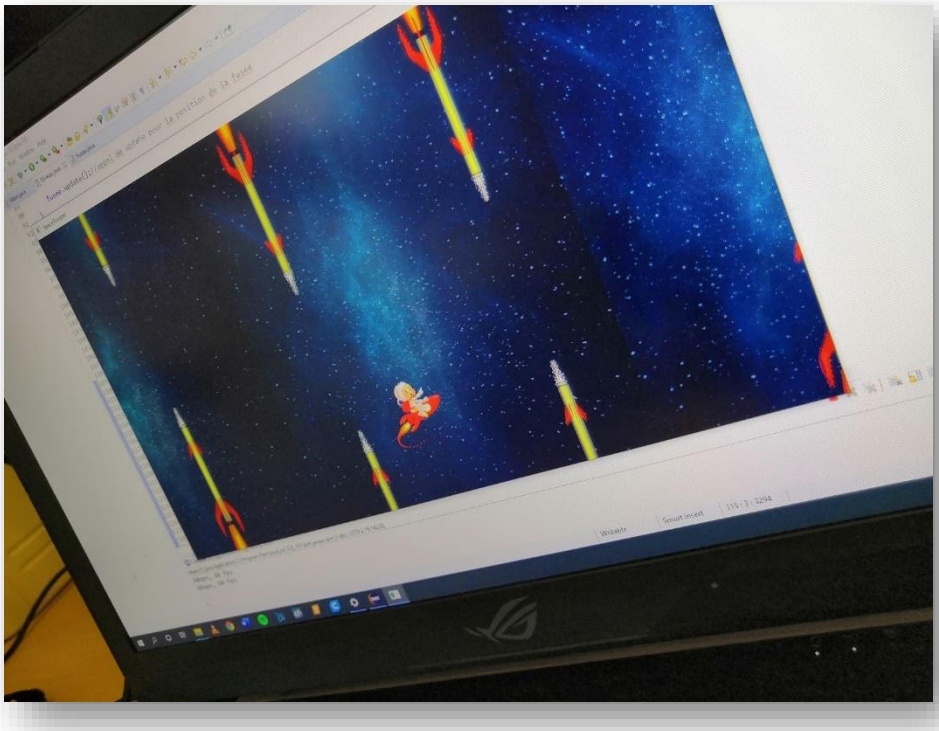
Nous avions besoin d'un langage adapté pour créer un jeu assez simple et en peu de temps. Nous avions donc le choix entre le C++ et le Java.
Nous avons choisi le Java pour plusieurs raisons :

- C'est un langage assez simple à comprendre et qui s'approche de ce qu'on connait déjà
- C'est un langage populaire c'est à dire que nous avons pu nous documenter un peu partout sur internet pour résoudre certains soucis.
- L'utilisation des bibliothèques facilite la tâche notamment pour des fonctions spécifiques au jeu.

Nous avions pour objectif d'améliorer le jeu en lui ajoutant des bonus ou malus.

Le but du jeu est de faire avancer un vaisseau dans l'espace tout en évitant des missiles présents en haut et en bas de l'écran.

Les règles du jeu sont très simples : lorsque le vaisseau touche un obstacle, la partie se termine



Les objets techniques utilisés

Nous avons utilisé plusieurs bibliothèques pour ce projet :

- **LWJGL**
- **OpenGL**
- **GLFW**

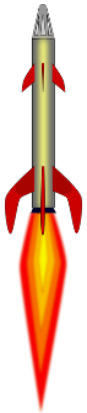
La première bibliothèque que nous allons vous présenter est **LWJGL** acronyme de Lightweight Java Game Library. LWJGL est une bibliothèque open source multiplateforme gérant l'image (avec **OpenGL**), l'audio (avec OpenAL) et le calcul parallèle (avec OpenCL). Elle est conçue pour réaliser des frameworks de jeu. Dans les avantages que nous propose LWJGL, nous avons notamment utiliser OpenGL et GLFW.

OpenGL (*Open Graphics Library*) est un ensemble normalisé de fonctions de calcul d'images 2D ou 3D. Cette interface de programmation est disponible sur de nombreuses plateformes.
OpenGL permet à un programme de déclarer la géométrie d'objets sous forme de points, de vecteurs, de polygones, de bitmaps et de textures. OpenGL effectue ensuite des calculs de projection en vue de déterminer l'image à l'écran, en tenant compte de la distance, de l'orientation, des ombres, de la transparence et du cadrage.

GLFW : c'est une bibliothèque offrant la possibilité de créer notamment des fenêtres applicatives avec OpenGL ou alors de recevoir des informations sur les touches appuyées par exemple.

Nous avons aussi utilisé Photoshop qui n'est pas directement acteur du code, mais qui nous a servi à faire les textures comme le fond, les obstacles et le personnage / joueur.

Photoshop qu'est-ce que c'est ? C'est un logiciel professionnel de graphisme qui offre des possibilités quasiment infinies. Ce logiciel est utilisé par un grand nombre de professionnels : architectes, dessinateurs, illustrateurs, cartographes... Il permet de retravailler la photo, de réaliser un photomontage etc.… dans le but de professionnaliser la création graphique.



Les problèmes rencontrés

L'un de nos problèmes physiques rencontré était d'effectuer une rotation du vaisseau, c'est-à-dire que lorsqu'il est en phase montante, il fallait le faire piquer vers le haut, et lors des phases descendantes qu'il pique vers le bas.

La solution apportée :
Après des recherches nous avons découvert une méthode de Java qui s'appelle Math.toRadians(). Elle permet de convertir un angle degré en radians. Nous avons donc créé une nouvelle fonction qui fait passer un angle en paramètre (70) et qui sera appliqué aux positions de la fusée en temps réel. C'est-à-dire que la fusée pivotera selon le fait qu'elle monte où qu'elle descend.



Avant de commencer les collisions nous avions besoin que les tailles du vaisseau ou bien des obstacles soient stockés. Malheureusement cela aurait été idéal de simplement définir une taille et d'avoir tous les points nécessaires. Ce n'est pas le cas bien évidemment. Nous devions trouver une solution pour que nos objets intégrés soient bien « détourés ».

Pour résoudre ce souci, nous avons découvert le principe de « sommets » (vertices en anglais). Il a fallu que nous stockions chaque « coté » de nos images. Car si elles avaient simplement une taille elles ne seraient pas représentées en 2D. Il fallait donc créer un tableau contenant chaque sommet.

```
float[] vertices = new float[]
{
    0.0f, 0.0f, 0.1f, // Coin en bas à gauche
    0.0f, height, 0.1f, // Coin en haut à gauche
    width, height, 0.1f, // Coin en haut à droite
    width, 0.0f, 0.1f // Coin en bas à droite
};
```

En sachant que height et width sont la hauteur et la largeur de base d'un obstacle.
C'est notamment grâce à ce tableau que nous pourrons effectuer les collisions.
Il reste encore un souci malheureusement à cette technique. Nous ne savons que définir une forme carrée ou rectangle, c'est-à-dire que le vaisseau ou les obstacles sont définis par des formes rectangulaires. A cause de ça, la collision ne sera plus précise à 100% mais elle reste presque parfaite malgré tout.

Un autre souci a été d'appliquer les collisions au jeu.
Après avoir réussi à intégrer les obstacles aléatoirement dans le niveau et avoir réussi à faire monter ou descendre le personnage, il fallait ajouter le fait que le vaisseau tombe et que la partie se termine lorsqu'il touche un obstacle.

La solution apportée :
Pour ça, nous allons récupérer les coordonnées précises de chaque côté de la fusée et également des obstacles en créant des nouvelles fonctions dans les fichiers fusee.java et Obstacle.java qui vont récupérer les coordonnées et qui sont faites en continue pendant la partie (du style GetX). Ensuite on a tout mis dans des conditions pour vérifier à quel moment les coordonnées du vaisseau deviennent inférieurs aux coordonnées des obstacles. Si c'est le cas, alors ça voudra dire qu'il y a collision. Dans ce cas, on fait perdre le joueur.

Le code de récupération des coordonnées et de vérification de collisions est disponible ci-dessous :

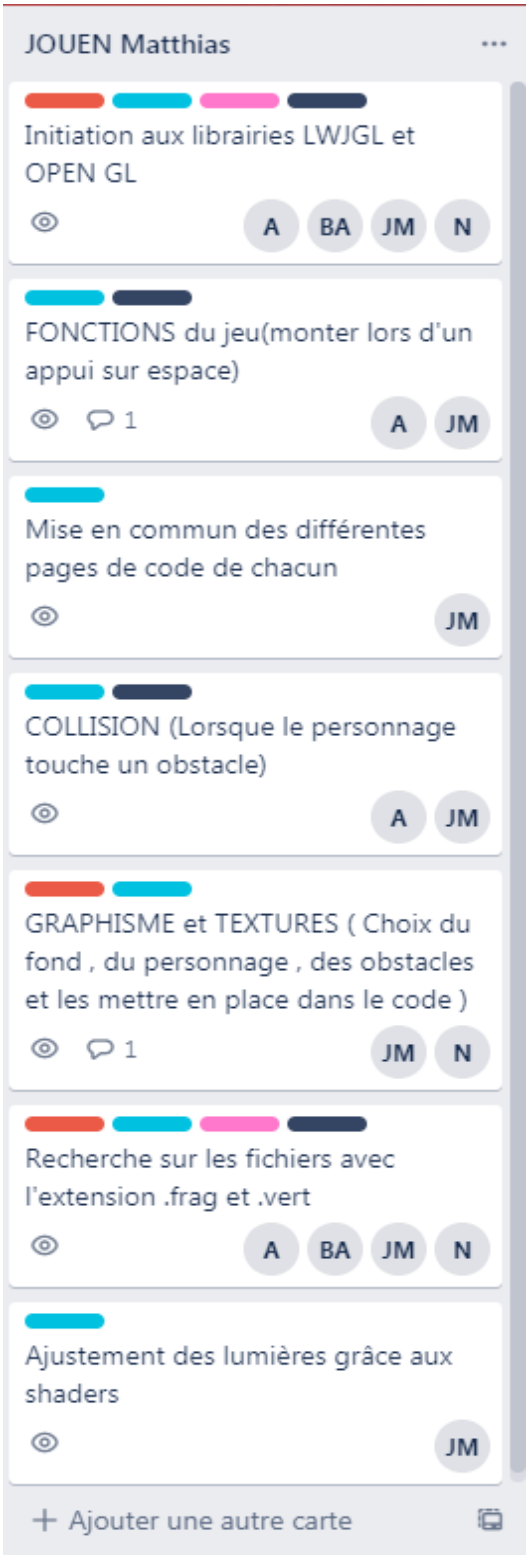
```
private boolean collision()
{
    for (int i = 0; i < 5 * 2; i++)
    {
    float fx = -xScroll * 0.05f; //coordonnees x de la fusee
    float fy = fusee.getY();//coordonnees y de la fusee
    float ox = obstacle[i].getX();//coordonnees x de l'obstacle
    float oy = obstacle[i].getY();//coordonnees y de l'obstacle0

    //on affine la récuperation des coordonnees de la fusee
    float fx0 = fx - fusee.getSize() / 2.0f;//le bas de la fusee
    float fx1 = fx + fusee.getSize() / 2.0f;//le haut de la fusee
    float fy0 = fy - fusee.getSize() / 2.0f;//la gauche de la fusee
    float fy1 = fy + fusee.getSize() / 2.0f;//la droite de la fusee

    //on affine la recup des coordonnees des obstacles
    float ox0 = ox;
    float ox1 = ox + Obstacle.getWidth();
    float oy0 = oy;
    float oy1 = oy + Obstacle.getHeight();

        if (fx1 > ox0 && fx0 < ox1) //on verfie sur l'axe X si la fusee touche les coordonnees de l'obstacle
        {
            if (fy1 > oy0 && fy0 < oy1)////on verfie sur l'axe Y si la fusee touche les coordonnees de l'obstacle
            {
                return true; //si c'est le cas il y a collision
            }
        }
    }
    return false;
}
```

Cahier des charges

Cahier de test :

N°	Action	Description	Résultat (OK/NOK)
1	Lancer le jeu	Le jeu démarre	
2	Le jeu doit être fluide	Jeu en 50 FPS / IMS min	
3	Le joueur doit pouvoir se déplacer	Fonctionnement des touches (ESPACE)	
4	Le joueur peut mourir	Au toucher de l'obstacle, le joueur meurt via les collisions	
5	Réapparition	Reviens au début du jeu	
6	Prendre un bonus	Ex : Passer à travers les obstacles	
7	Fermer le jeu	Il se ferme sans problème	

Répartition des tâches :


JOUEN Matthias
Initiation aux librairies LWJGL et OPEN GL
A BA JM N
FONCTIONS du jeu(monter lors d'un appui sur espace)
1
A JM
Mise en commun des différentes pages de code de chacun
JM
COLLISION (Lorsque le personnage touche un obstacle)
A JM
GRAPHISME et TEXTURES (Choix du fond , du personnage , des obstacles et les mettre en place dans le code)
1
JM N
Recherche sur les fichiers avec l'extension .frag et .vert
A BA JM N
Ajustement des lumières grâce aux shaders
JM
Ajouter une autre carte


BOURY Aurélien
Initiation aux librairies LWJGL et OPEN GL
A BA JM N
Recherche sur les fichiers avec l'extension .frag et .vert
A BA JM N
Fichiers obtacle.java (obstacle.vert et obstacle.frag) , fusee.java (fusee.vert et fusee.frag)
A
Création et Intégration des obstacles et du personnage
A JM
Ajouter une autre carte

Conclusion

Ce projet nous a demandé beaucoup de temps étant donné que nous n'avions jamais développé de jeu, et encore moins en Java. Fort heureusement, il y a énormément d'exemples, de tutos, ou de documentations sur Internet.

Il est clair que cela nous a apporté une autonomie car nous avons dû gérer seuls la répartition des tâches, la réalisation etc…

Malheureusement nous n'avons pas réussi à rajouter des bonus comme voulu au départ par manque de temps et de compétences. Mais nous avons fait de notre mieux.

Ce projet était très instructif malgré le fait qu'il soit très différent de notre formation de base.

Jouen Matthias, Brard Adrien, Boury Aurélien, Lhermitte Nathan.