HAUTE ÉCOLE D'INGÉNIEUR DU VALAIS

RAPPORT

PROJET LUNARLANDRY

TEAM PLS

INF1

Pablo Stoeri
Landry ReynardSamy Francelet[2cm] 19 juin 2020

Table des matières

1 Introduction

1.1 Contexte

Après avoir suivis le cours d'informatique n°1 pendant 10 mois environ, nous avons accumulé beaucoup de matière sur la programmation en Java en passant par des sujets très variés. Il est donc maintenant temps de réaliser un projet sur lequel nous allons appliquer et surtout réunir un bon nombre des éléments appris. Pour réaliser ce projet, nous avons formé un groupe de trois personnes de la filières Systèmes Industriels et dans notre cas, de la classe bilingue. Voici les membres du groupe : Pablo Stoeri, Landry Reynard et Samy Francelet. Le jeu implémenté est un Lunar Lander, celui-ci nous a été proposé à la suite de l'interruption des cours en présentiel causée par la crise sanitaire du covid19 et correspond à notre niveau de programmation atteint en cette fin d'année scolaire 2019-2020.

1.2 Objectif du document

Ce document a pour but d'accompagner les personnes qui vont faire fonctionner, contrôler, ou encore modifier notre programme, en leur donnant les informations suivantes :

— Explications générales

— Spécificités propres à notre code

— Problèmes et solutions que nous avons connu durant le codage

— Améliorations possibles

L'objectif est donc de rendre notre code compréhensible et accessible à n'importe qui, pour autant qu'il ait les compétences nécessaires en Java.

1.3 Spécifications - Problématique

Le but de ce jeu est de faire atterrir un vaisseau spatial sur une plateforme situé sur la lune. Il faut donc comme en réalité éviter les obstacles et atterrir en douceur. Dans le cas contraire le vaisseau ne supportera pas l'impact et sera détruit. La problématique est donc la suivante : Coder un Lunar Lander avec ces différentes fonctionnalités.

> Voici le cahier des charges qui nous a été transmis :Au lancement du jeu, le vaisseau doit se trouver dans le ciel. Pour déplacer le vaisseau, il y a trois commandes/moteur à disposition : gauche, droite, vertical vers le haut. Chaque moteur doit donner une impulsion au vaisseau qui accélère dans la direction voulue, tout en respectant les lois physiques de la gravité et de la densité de l'atmosphère qui ne sont évidemment pas les même que sur terre. La quantité de carburant disponible dans le vaisseau est limitée. Le carburant est consommé de manière proportionnelle avec le nombre de moteurs activés. C'est-à-dire que chaque moteur a sa propre consommation de carburant et que si

l'on active deux moteurs en même temps ils vont chacun consommer du carburant. Une fois que la réserve de carburant est vide, plus aucune action ne va pouvoir être effectuée sur le vaisseau et celui-ci va poursuivre sa trajectoire. Le vaisseau spatial peut se poser seulement à un endroit précis, prévu à cet effet et il doit se poser en douceur pour éviter l'explosion. S'il touche le relief à un endroit où il n'y est pas autorisé, il va aussi exploser. Dans le cahier des charges de bases, il n'y a pas de menu de jeu afin de ne pas perdre de temps sur un élément qui n'est pas très intéressant au niveau de la programmation.

1.4 Méthodologie

2 Méthodes

2.1 Méthodes

Pour ce projet, nous sommes partis de grandeurs et de données physique comme la gravité, la masse et l'accélération. Cela nous a permis de construire une base solide de physique pour ensuite venir ajouter les différents éléments de programmation derrière. De manière générale, nous pensons qu'il est important de se concentrer sur les faits réels de la vie avant de commencer à coder des interfaces graphiques ou autres. Voici la physique que nous avons utilisé : **?**

— $\overrightarrow{F} = M \cdot \overrightarrow{a}$

Qui est transformé pour l'accélération en chute libre par :

$$\overrightarrow{F}_{grav} = M \cdot \overrightarrow{g}$$

Où g = -9.81 ms-2 sur la terre et g = -1.62 ms-2 et M est la masse de l'objet soumis à la gravité, dans notre cas le vaisseau spatial. Ensuite, nous avons aussi utilisé le coefficient de frottement en rapport avec la densité de l'air :

$$\overrightarrow{F}_{friction} = -k \cdot \overrightarrow{v}$$

Où k est le coefficient de friction de l'atmosphère,en l'occurrence k = 0 sur la lune vu qu'il n'y a pas d'air. De ce fait, tant que le vaisseau est en l'air, seul la force de gravité a un impact sur le vaisseau lorsque les moteurs sont éteints.

3 Résultats

3.1 Choix d'implémentation et variations effectuées

Nous allons dans ce chapitre décrire les classes de notre programme afin d'une part clarifier le code, et d'autre part, facilité la modification et le contrôle.

3.2 LunarPhysics

3.2.1 PhysicalObject

Comme le dit son nom, cette classe s'occupe de créer un objet physique avec les paramètres suivants : la position initiale, la vitesse initiale, la largeur et la hauteur. Sur cet objet vont donc ensuite s'appliquer les forces physiques comme la force de gravité et la poussée des réacteurs. Cette classe implémente les interfaces Simulatable et Collisionable.

3.2.2 PhysicsSimulator

Cette classe représente pour nous la classe la plus difficile à implémenter et à comprendre. Elle contient :

1. La physique qui va agir sur les corps créer dans PhysicalObject
2. Les collisions entre les différents corps

Concernant la physique nous avons choisis de représenter toutes les forces en Vector2, c'est-à-dire un vecteur où l'on donne les points en X et en Y comme paramètre. L'avantage de travailler avec les vecteurs est que toutes les informations concernant la force sont directement accessibles.

Les collisions ont été implémentées de la manière suivante : par exemple pour les collisions entre le paysage et le vaisseau spatial, nous avons créé une BoundingBox autour du vaisseau, c'est-à-dire un rectangle qui évite de devoir faire collisionner la forme compliquée du vaisseau. On a donc maintenant une collision entre un rectangle (vaisseau) et un polygone (paysage). Nous avons donc d'abord demandé si un des coins du rectangle se trouvait à l'intérieur du polygone (avec la méthode countains de Polygon-Working) et nous avons à ce moment là remarqué que si le vaisseau se posait sur un coin du polygone, les points du rectangle ne se trouvaient pas dans le polygone alors que le vaisseau aurait dû exploser. Nous avons donc ajouté une contrainte qui regarde si un point du polygone se trouve dans le rectangle. Les collisions se font correctement de cette manière.

Cette BoundingBox peut être activée et désactivée facilement dans les constantes.

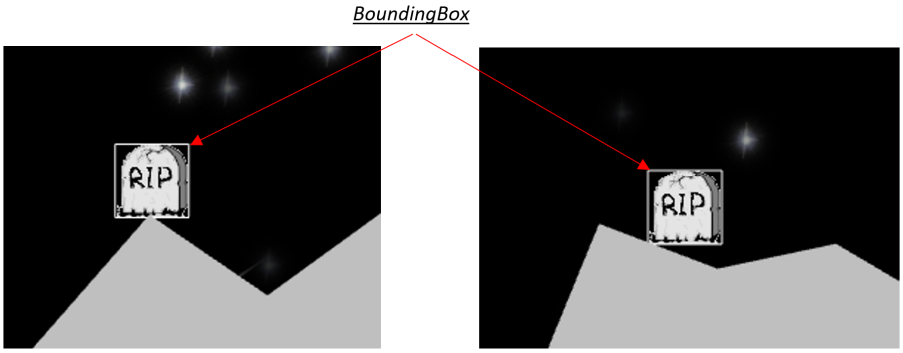

FIGURE 3.1 – Collision entre un point du polygone et le rectangle et inversement

3.2.3 Simulatable

Cette interface permet à chaque objet provenant d'une classe qui hérite de cette interface, d'avoir une méthode step qui va permettre de simuler chaque étape.

3.2.4 Particles

Voici le générateur de particule que nous avons créé à la suite d'un problème avec celui de GDX2D cité dans les problèmes dans ce rapport. Ces particules ont une durée de vie, une direction, une vitesse et une représentation graphique.

3.2.5 Ground

La classe Ground génère le sol de la lune. Il est représenté par un polygone. Lorsque l'on créer un objet Ground, le constructeur de cette classe va déterminer tous les points du polygone en utilisant des vecteurs qui partent tous du point (0,0). Une fois que tous les points sont créés, nous créons un objet. Cet objet va ensuite être affiché et utilisé pour les collisions. PolygonWorking. Ensuite, nous avons la méthode getPolygon qui est appelée dans la classe main pour créer le polygone qui va ensuite s'afficher sur le jeu.

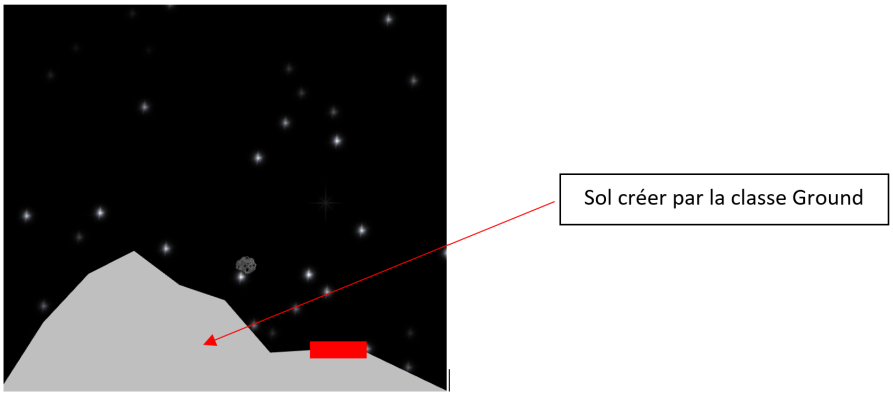

FIGURE 3.2 – Image d'illustration du polygone

3.2.6 Constants

Dans cette classe, nous avons mis toutes les constantes du jeu. Ceci nous permet et permet au prochain utilisateur de modifier simplement des variables dans le jeu sans pour autant devoir comprendre tout ce

qui a été fait dans le code. C'est aussi un récapitulatif de nos paramètres utilisés et cela nous permet de garder un code propre.

3.2.7 Collisionable

Collisionnable est une interface qui s'occupe de voir quand est-ce qu'il y a une collision. C'est aussi dans cette interface qu'il y a les BoundingBox mentionnées ci-dessus. Cette interface nous a été fournie au début du projet.

3.3 LunarMain

3.3.1 PolygonWorking

Cette classe nous a aussi été fournie durant le projet. Elle hérite de la classe Polygon qui sert à créer un polygone. Pour créer une forme avec PolygonWorking, il faut appeler la fonction et donner en paramètre tous les points du polygone. Ces points sont donnés en vecteur.

3.3.2 Gegner

3.3.3 LandZone

Cette classe créer simplement un rectangle qui symbolise la zone d'atterrissage. Ce rectangle va être positionné à la position donnée par le vecteur en paramètre.

3.3.4 SpaceShip

C'est dans cette classe que nous créons le vaisseau. C'est aussi ici que nous faisons l'affichage de différents éléments dans la fenêtre comme la vitesse et le carburant restant. Nous initialisons aussi le fond noir de la fenêtre dans cette classe.

3.3.5 LunarLander$_{M}ain$

4 Bilan

4.1 Problèmes rencontrés et solutions apportées

Nous avons évidemment rencontré des problèmes auxquelles nous avons dû trouver des solutions efficaces. Voici une liste des problèmes principaux rencontrer lors du codage de notre Lunar Lander :

1. Générateur de particules : Nous avons tout d'abord utilisé le système de particules de la libraire GDX2D qui a d'abord très bien fonctionné, mais nous avons rencontré un problème lorsqu'il s'agissait de recommencer la partie après une explosion ou une partie gagnée. La libraire GDX2D n'arrivait pas à recréer les objets ce qui résultait à un crash du programme. Pour contrer ce problème nous avons créé un générateur de particule plus simple avec des particules qui apparaissent à une certaine position avec une vitesse prédéfinie. Cette fonction a été implémenté dans la classe Particules.
2. La musique et les sons : Notre musique ou certains sons sont beaucoup trop fort par rapport à d'autre. Nous n'avons pas réussi à faire fonctionner les méthodes concernant le volume, par exemple setVolume ou mofidyPlayingVolument de la librairie, et la documentation n'est pas très claire pour ces méthodes-ci. (...) Pour modifier le volume de notre piste audio, nous avons finalement utilisé un convertisseur online(https ://www.mp3louder.com/fr/) pour baisser le son de quelques décibels.

4.2 Description des fonctionnalités

Notre code remplis tout le cahier des charges, c'est-à-dire que le vaisseau apparait dans le ciel et il est dirigeable par trois commandes : gauche, droit, et vertical vers le haut. Le vaisseau subit les forces de la gravité et les forces de freinages de la densité de l'atmosphère. Et il est obligatoire d'atterrir en douceur sur la plateforme indiquée en utilisant l'essence à disposition. Dans le cas contraire le vaisseau va exploser.

En plus des fonctionnalités de bases, nous avons ajouté des options qui rendent le jeu plus amusant et ludique pour le joueur :

— Des sons pour les différentes actions dans le jeu.

— Des météorites sur lesquelles on peut tirer avec un clic de la souris. Il peut y avoir une collision entre les météorites et le vaisseau donc il est possible qu'il soit même nécessaire de leur tirer dessus pour les détruire.

— Des particules sous le vaisseau lorsqu'un réacteur est activé.

— Des étoiles dynamiques en arrière-plan

— Des niveaux de difficulté où il a de plus en plus de météorites. Le 11ème niveau est notre dernier niveau mais le jeu continue tout de même plus loin

4.3 Améliorations possbibles

Nous aurions bien aimé ajouter les éléments suivants :

— La rotation du vaisseau en prenant en comptes les moments de forces.

— La gestion du volume sonore directement depuis le code et pouvoir même depuis l'interface désactiver la musique ou les sons.

— L'atterrissage automatique du vaisseau sur la plateforme.

— Un relief plus évolué avec des tunnels par exemple selon les niveaux.

— Des bonus que l'on pourrait attraper dans le ciel avec le vaisseau pour avoir des vies ou du carburant par exemple

— Par rapport au code en lui mêm, ...

4.4 Conclusion

Pour nous, ce projet d'informatique de fin de première année a été un succès. Nous avons répondu à tous les éléments du cahier des charges et avons eu un peu de temps pour ajouter encore quelques options pour le plaisir du joueur. La plus grande difficulté pour notre groupe a été de comprendre le concept de base au début du projet. Une fois lancé, le codage s'est fait assez instinctivement avec de plus en plus de plaisir au lancement du programme pour voir ce que l'on a coder prendre forme et évoluer au fil du temps est très motivant. Le système GitHub nous a été très utile durant tout le projet même si celui-ci n'est pas très facile à comprendre. Une fois qu'il est compris, c'est un énorme avantage. Concernant notre groupe, il n'y a pas eu de tensions, même si, sur ce genre de projet, il n'est pas facile de faire en sorte que tout le monde effectue la même quantité de travail. Il est possible que certains membres du groupe aient eu un plus grand impact sur le code que d'autres, mais nous avons tout de même évoluer sur le projet tous ensemble. Pour conclure, nous avons eu du plaisir à faire ce projet car il nous a permis de nous rendre compte de ce que nous étions capable de faire au bout de 10 mois (seulement). C'était intéressant de travailler dans un petit groupe d'amis et d'appliquer nos connaissances communes. En plus de tous les éléments.

4.5 Signatures et date

Sion, le 19 juin 2020
Pablo Stoeri
Landry Reynard
Samy Francelet

A Test Prog

```
0  public class SortApplication {
2    static void displayArray(int[] array) {
       String value = "";
4      for (int i = 0; i < array.length; i++) {
         value = value + array[i] + ",";
6      }
       System.out.println(value);
8    }
10   public static void main(String[] args) {
       //Creation Tableaux
12     int MaxValue = 140000;
       int a[][] = new int[10][];
14     int b[][] = new int[10][];
       int c[][] = new int[10][];
16
18     for (int i = 0; i < a.length; i++) {
         a[i] = ArrayFactory.createRandomArray(MaxValue, 50000);
20     }
       for (int i = 0; i < a.length; i++) {
22       b[i] = ArrayFactory.createInvertedSortedArray(MaxValue);
       }
24     for (int i = 0; i < a.length; i++) {
         c[i] = ArrayFactory.createShuffleArray(MaxValue);
26     }
28
       long startTimeRandom = System.nanoTime();
30     for (int i = 0; i < a.length; i++) {
         SelectionSort.sort(a[i]);
32     }
       long endTimeRandom = System.nanoTime();
34
       long startTimeInverted = System.nanoTime();
36     for (int i = 0; i < a.length; i++) {
         SelectionSort.sort(b[i]);
38     }
       long endTimeInverted = System.nanoTime();
40
```

```
    long startTimeShuffle = System.nanoTime();
42  for (int i = 0; i < a.length; i++) {
      SelectionSort.sort(c[i]);
44  }
    long endTimeSuffle = System.nanoTime();
46


48  System.out.println((endTimeRandom - startTimeRandom) + " [ns]");
    System.out.println((endTimeInverted - startTimeInverted) + " [ns]");
50  System.out.println((endTimeSuffle - startTimeShuffle) + " [ns]");

52  }

54 }
```