

# **RAPPORT**

## **PROJET LUNARLANDRY**

### **TEAM PLS**

INF1

---

Pablo Stoeri  
Landry Reynard  
Samy Francelet

19 juin 2020

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Contexte . . . . .	2
1.2	Objectif du document . . . . .	2
1.3	Spécifications - Problématique . . . . .	2
1.4	Méthodologie . . . . .	3
<b>2</b>	<b>Méthodes</b>	<b>4</b>
2.1	Méthodes . . . . .	4
<b>3</b>	<b>Résultats</b>	<b>5</b>
3.1	Choix d'implémentation et variations effectuées . . . . .	5
3.2	LunarPhysics . . . . .	5
3.2.1	PhysicalObject . . . . .	5
3.2.2	PhysicsSimulator . . . . .	5
3.2.3	Simulatable . . . . .	5
3.2.4	Particles . . . . .	6
3.2.5	Ground . . . . .	6
3.2.6	Constants . . . . .	6
3.2.7	Collisionable . . . . .	7
3.3	LunarMain . . . . .	7
3.3.1	PolygonWorking . . . . .	7
3.3.2	Gegner . . . . .	7
3.3.3	LandZone . . . . .	7
3.3.4	SpaceShip . . . . .	7
3.3.5	LunarLander_Main . . . . .	8
<b>4</b>	<b>Bilan</b>	<b>9</b>
4.1	Problèmes rencontrés et solutions apportées . . . . .	9
4.2	Description des fonctionnalités . . . . .	9
4.3	Améliorations possibles . . . . .	10
4.4	Conclusion . . . . .	10
4.5	Signatures et date . . . . .	10
<b>A</b>	<b>Package Principal</b>	<b>12</b>
<b>B</b>	<b>Package Physique</b>	<b>25</b>

# 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 filière 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 située 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. Pour cela nous avons utilisé la librairie GDX2D créée par Pierre-André Mudry [3][2].

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

Pour ce projet, nous allons suivre globalement l'ordre qui nous a été donné sur le site d'informatique, mais étant donné que nous sommes trois dans un groupe et qu'il est important de ne pas faire des tâches à double, cet ordre est un peu perturbé.

De même, pour ce travail de groupe, nous avons choisis d'utiliser GitHub pour que tous les trois membres du groupe puissent coder en même temps sur des fonctionnalités différentes et mettre ensuite tout le travail en commun. L'avantage de Git et GitHub est de pouvoir créer une nouvelle fonctionnalité dans une « branche », sans perturber le travail principal de ses collègues.

Il est important pour ce genre de projet de bien s'attribuer des tâches, surtout une fois que la base du programme est faite. Et c'est encore plus important quand il y a des différences de niveau au sein du groupe pour que le travail d'équipe se fasse proprement.

## 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 :

$$\vec{F} = M \cdot \vec{a}$$

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

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

Où  $g = -9.81 \text{ ms}^{-2}$  sur la terre et  $g = -1.62 \text{ ms}^{-2}$  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 :

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

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, faciliter 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 créés dans PhysicalObject.
2. Les collisions entre les différents corps.

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 `contains` 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.

#### 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.

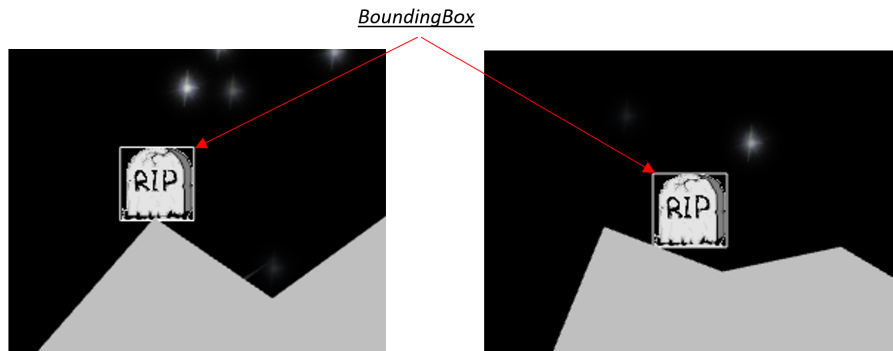


FIGURE 3.1 – Collision entre un point du polygone et le rectangle et inversement

### 3.2.4 Particules

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. La couleur de la représentation graphique est sélectionnée aléatoirement entre deux couleurs.

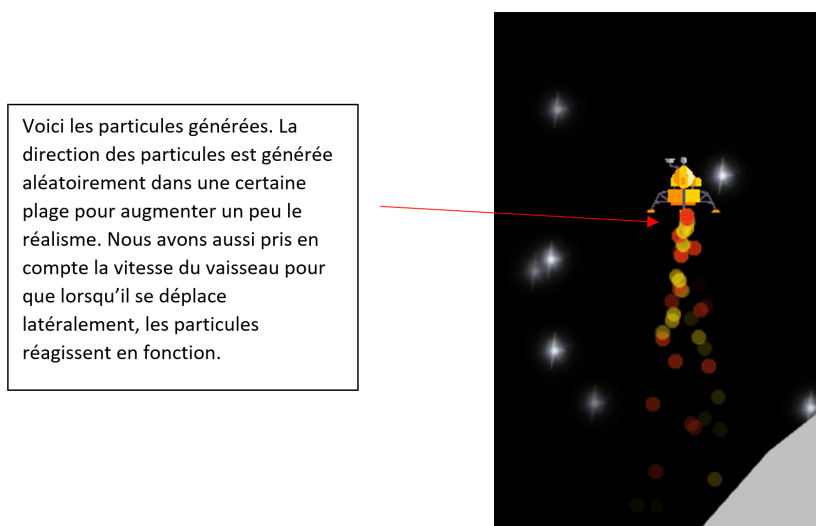


FIGURE 3.2 – Vaisseau VSS\_Couchepin avec ses particules du réacteur

### 3.2.5 Ground

La classe Ground génère le sol de la lune. Il est représenté par un polygone. Lorsque l'on crée 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.

### 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

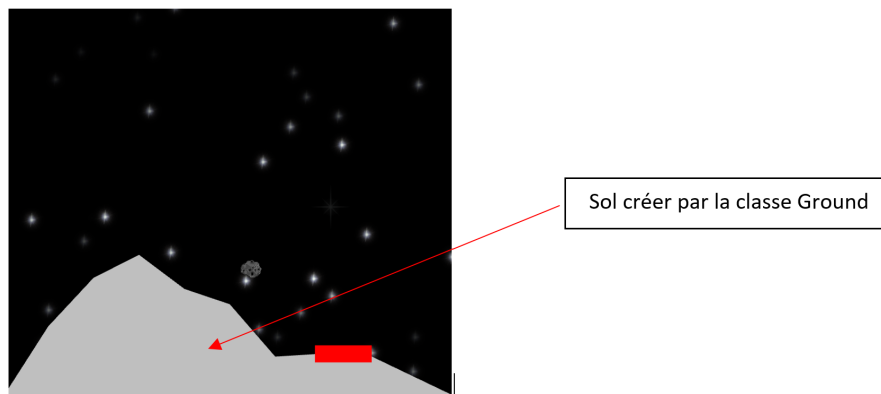


FIGURE 3.3 – Image d’illustration du polygone

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

Cette classe qui hérite de la classe PhysicalObject et qui implémente la classe DrawableObject génère les météorites. On crée dans cette classe des météores avec des vitesses aléatoire et qui ne sont pas soumises à la gravité de la lune pour faire un visuel plus intéressant. (Gegner signifie ennemi en allemand)

### 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.



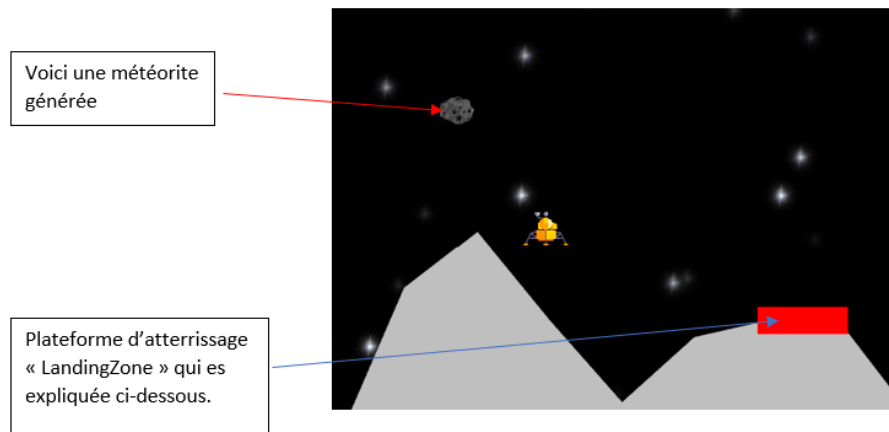


FIGURE 3.4 – Image avec une météorite, le vaisseau et la plateforme d’atterrissage

### 3.3.5 LunarLander\_Main

LunarLander\_Main est la classe où se trouve l’exécution de toute les tâches du jeu. Les éléments importants à comprendre dans cette classe sont :

1. La méthode OnInit : On appelle cette méthode pour lancer la toute première image du jeu.
2. La méthode OnGraphicRender : Cette méthode appelle tous les éléments qui doivent être affichés tout au long du jeux. On voit bien dans cette méthode que nous appelons tous les éléments mais que ceux-ci ne sont pas forcément « activé ». Par exemple le laser s’affichera seulement lorsqu’on fait un clic avec la souris mais il est déjà appelé dans cette méthode.
3. Les méthode OnKeyUp et OnKeyDown permettent la commande avec les touches.
4. La méthode Replay : Le but est de pouvoir rejouer. Pour se faire nous enlevons tous les éléments de la fenêtre et les remettons directement en fonction du nombre de partie jouées (le nombre de météorites change).

Dans cette classe nous avons énormément d’éléments qui concernent les options ajoutées par après.

## 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 rencontrés lors du codage de notre Lunar Lander :

1. Générateur de particules : Nous avons tout d'abord utilisé le système de particules de la librairie 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 librairie 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 implémentée dans la classe Particules.
2. La musique et les sons : Notre musique ou certains sons sont beaucoup trop fort par rapport à d'autres. Nous n'avons pas réussi à faire fonctionner les méthodes concernant le volume, par exemple `setVolume` ou `modifyPlayingVolume` 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 remplit tout le cahier des charges, c'est-à-dire que le vaisseau apparaît 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. Les sons ont été enregistrés via les enregistreurs de nos ordinateurs respectifs. Un son a également été téléchargé depuis le site de la Nasa[1].
- 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 possibles

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-même, il aurait été possible de mettre une hiérarchie avec des packages plus compréhensible. De même, notre main contient beaucoup de code, ce qui le rend moins lisible. Avec un peu plus de temps, il aurait été intéressant de créer des classes pour chaque option ajoutée.

### 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

Londres, le 19 juin 2020, 14h10 UTC+1

Pablo Stoeri

Landry Reynard

 Samy Francelet



## Bibliographie

- [1] Nasa Neil Armstrong. One small step for a man, one giant leap for mankind.
- [2] Pierre-André Mudry. A gaming library for the inf1 course.
- [3] Pierre-André Mudry. Inf1 website.

## **A Package Principal**

```

1  package ch.hevs.gdx2d.lunar.main;
2
3  import java.util.ArrayList;
4  import java.util.Random;
5
6  import com.badlogic.gdx.Input;
7  import com.badlogic.gdx.graphics.Color;
8  import com.badlogic.gdx.math.Rectangle;
9  import com.badlogic.gdx.math.Vector2;
10
11 import ch.hevs.gdx2d.lib.GdxGraphics;
12 import ch.hevs.gdx2d.lunar.physics.Constants;
13 import ch.hevs.gdx2d.lunar.physics.Ground;
14 import ch.hevs.gdx2d.lunar.physics.Particles;
15 import ch.hevs.gdx2d.lunar.physics.PhysicsSimulator;
16 import ch.hevs.gdx2d.components.audio.MusicPlayer;
17 import ch.hevs.gdx2d.components.audio.SoundSample;
18 import ch.hevs.gdx2d.desktop.PortableApplication;
19
20 public class LunarLander_Main extends PortableApplication {
21
22     // Game components
23     PhysicsSimulator physics;
24     Spaceship ssLandry;
25     Ground sol;
26     LandZone lz;
27     ArrayList<Gegner> meteors;
28     private int gameNb;
29
30     // music
31     MusicPlayer music;
32     SoundSample noFuel;
33     SoundSample bruitExplosion;
34     SoundSample winSound;
35     SoundSample pew;
36     private boolean doSoundFuel = true;
37     private boolean doExplosion = true;
38     private boolean doWinSound = true;
39
40     // Shooting related
41     private ArrayList<Particles> laserExplo;
42     boolean mouseActive = false;
43     Vector2 positionClick;
44     static int waitLaser;
45
46     // Stars particles
47     private ArrayList<Particles> stars;
48     static final Random rand = new Random();
49     static int waitStar;
50
51     public LunarLander_Main() {
52         super(Constants.WIN_WIDTH, Constants.WIN_HEIGHT);
53     }
54
55     @Override
56     public void onInit() {
57         setTitle("LunarLandry (Team PLS)");
58         gameNb = 1;
59         waitStar = 0;
60         waitLaser = 0;
61         playMusic();
62         ssLandry = new Spaceship(new Vector2(100, 700), gameNb);
63         sol = new Ground();
64         lz = new LandZone(sol.getPolyPoint(Constants.FLAT_ZONE));
65         physics = new PhysicsSimulator(Constants.WIN_WIDTH, Constants.WIN_HEIGHT);
66         physics.changePlayground(sol.getPolygon(), lz);
67         physics.addSimulatableObject(ssLandry);
68         stars = new ArrayList<Particles>();
69         laserExplo = new ArrayList<Particles>();
70         meteors = new ArrayList<Gegner>();
71         meteors.add(new Gegner(new Vector2(400, 700)));
72         physics.addSimulatableObject(meteors.get(0));
73     }

```

```

74
75 @Override
76 public void onGraphicRender(GdxGraphics g) {
77     // Clears the screen
78     g.clear();
79
80     // Simulate every object
81     physics.simulate_step();
82
83     // Draw basic layout
84     // g.drawFPS();
85     // g.drawSchoolLogo();
86
87     // Draw the stars on the background
88     drawStars(g, 200);
89
90     // Spaceship
91     ssLandry.draw(g);
92
93     // Meteors
94     if (meteors.size() != 0) {
95         for (int i = 0; i < meteors.size(); i++) {
96             meteors.get(i).draw(g);
97         }
98     }
99     drawBoundingBoxes(g);
100
101     if (ssLandry.isFinished() && ssLandry.isKaputt()) {
102         g.drawStringCentered(660, "Appuiez sur 'R' pour recommencer");
103     }
104     if (ssLandry.isFinished() && ssLandry.isLanded()) {
105         g.drawStringCentered(660, "Appuiez sur 'R' pour continuer");
106     }
107     playSound();
108     g.drawFilledPolygon(sol.getPolygon(), Color.LIGHT_GRAY);
109     drawLandZone(g);
110     // g.drawLine(0, Constants.GROUND_ALTITUDE, Constants.WIN_WIDTH,
111     // Constants.GROUND_ALTITUDE, Color.WHITE);
112     drawLaser(g);
113     drawLaserExplo(g, 80);
114
115 }
116
117 void drawBoundingBoxes(GdxGraphics arg0) {
118     if (Constants.DRAW_BOUNDINGBOXES) { // Hitboxes
119         Rectangle box = ssLandry.getBoundingBox();
120         arg0.drawRectangle(box.getX() + box.getWidth() / 2, box.getY() +
121             box.getHeight() / 2, box.getWidth(),
122             box.getHeight(), 0);
123         if (meteors.size() != 0) {
124             for (int i = 0; i < meteors.size(); i++) {
125                 box = meteors.get(i).getBoundingBox();
126                 arg0.drawRectangle(box.getX() + box.getWidth() / 2, box.getY() +
127                     box.getHeight() / 2,
128                     box.getWidth(), box.getHeight(), 0);
129             }
130         }
131     }
132
133 void drawLaser(GdxGraphics arg0) {
134     if ((mouseActive || waitLaser > 0) && !ssLandry.isFinished()) {
135         arg0.drawLine(positionClick.x, positionClick.y, ssLandry.position.x,
136             ssLandry.position.y, Color.RED);
137         waitLaser--;
138     }
139 }
140
141 void drawLaserExplo(GdxGraphics arg0, int age) {
142     // Laser logik
143     if (mouseActive && !ssLandry.isFinished() && waitLaser < 0) {
144         pew = new SoundSample("data/sons/BruitLaser_low.mp3");
145         pew.play();

```

```

144     pew.setVolume(0.1f);
145     mouseActive = false;
146     waitLaser = 30;
147     if (meteors.size() != 0) {
148         for (int i = 0; i < meteors.size(); i++) {
149             if (meteors.get(i).getBoundingBox().contains(positionClick)) {
150                 physics.removeObjectFromSim(meteors.get(i));
151             }
152         }
153     }
154
155     Vector2 vec;
156     for (int i = 0; i < 100; i++) {
157         vec = new Vector2(1, 1).setToRandomDirection();
158         laserExplo.add(new Particles(new Vector2(positionClick.x,
159             positionClick.y), vec.scl(0.2f),
160             rand.nextInt(age),
161             rand.nextBoolean() ? "data/images/fire_particle.png" :
162             "data/images/reactor_particle.png"));
163     }
164     // Explosion laser animation
165     if (laserExplo.size() != 0) {
166         for (int j = 0; j < laserExplo.size(); j++) {
167             Particles p = laserExplo.get(j);
168             p.update();
169             p.draw(arg0);
170             if (p.shouldBeDestroyed()) {
171                 laserExplo.remove(p);
172             }
173         }
174     }
175
176     void drawLandZone(GdxGraphics arg0) {
177         Color color = Color.RED;
178         if (ssLandry.isLanded()) {
179             color = Color.GREEN;
180         }
181         arg0.drawFilledRectangle(lz.landBox.getX() + Constants.Z_WIDTH / 2,
182             lz.landBox.getY() + Constants.Z_HEIGHT / 2,
183             Constants.Z_WIDTH, Constants.Z_HEIGHT, 0, color);
184     }
185
186     void drawStars(GdxGraphics arg0, int age) {
187         waitStar++;
188
189         // Adds a star every n frames
190         if (waitStar == 5) {
191             final String img = "data/images/star.png";
192             final String img2 = "data/images/star2.png";
193             final String img3 = "data/images/star4big.png";
194             String imgRand;
195
196             int value = (int) (Math.random() * 20);
197             switch (value) {
198                 case 3:
199                     imgRand = img2;
200                     age = age / 3;
201                     break;
202                 case 6:
203                     imgRand = img3;
204                     age = age / 3;
205                     break;
206                 default:
207                     imgRand = img;
208                     break;
209             }
210             stars.add(new Particles(new Vector2(rand.nextFloat() *
211                 Constants.WIN_WIDTH,

```



```

212         new Vector2(0.1f, 0), age, imgRand));
213
214         waitStar = 0;
215     }
216
217     // Draw the stars
218     if (stars.size() != 0) {
219         for (int i = 0; i < stars.size(); i++) {
220             Particles p = stars.get(i);
221             p.update();
222             p.draw(arg0);
223             if (p.shouldBeDestroyed()) {
224                 stars.remove(p);
225             }
226         }
227     }
228 }
229
230 public int getNbGame() {
231     return gameNb;
232 }
233
234 void playMusic() {
235     if (rand.nextInt(100) <= 10) {
236         music = new MusicPlayer("data/sons/zambla.mp3");
237     } else {
238         music = new MusicPlayer("data/sons/sound1_low.mp3");
239     }
240     music.loop();
241 }
242
243 void playSound() {
244     if (ssLandry.isDry() && doSoundFuel) {
245         final String dry1 = "data/sons/NoFuel.mp3";
246         final String dry2 = "data/sons/Ecolo.mp3";
247         final String dry3 = "data/sons/Sub.mp3";
248         String dry;
249         int value = rand.nextInt(4);
250         switch (value) {
251             case 2:
252                 dry = dry2;
253                 break;
254             case 3:
255                 dry = dry3;
256                 break;
257             default:
258                 dry = dry1;
259                 break;
260         }
261         noFuel = new SoundSample(dry);
262         noFuel.play();
263         doSoundFuel = false;
264     }
265     if (ssLandry.isKaputt() && doExplosion) {
266         gameNb = 1;
267         final String kaputt1 = "data/sons/bruitExplo.mp3";
268         final String kaputt2 = "data/sons/doucement.mp3";
269         String kaputt;
270         int value = rand.nextInt(3);
271         switch (value) {
272             case 2:
273                 kaputt = kaputt2;
274                 break;
275             default:
276                 kaputt = kaputt1;
277                 break;
278         }
279         bruitExplosion = new SoundSample(kaputt);
280         bruitExplosion.play();
281         doExplosion = false;
282     }
283     if (ssLandry.isLanded() && doWinSound) {
284         if (gameNb == 11) {

```

```

285         winSound = new SoundSample("data/sons/OneSmallStep.mp3");
286     } else if (gameNb < 11) {
287         winSound = new SoundSample(rand.nextBoolean() ?
            "data/sons/Sympa.mp3" : "data/sons/bof_low.mp3");
288     } else {
289         winSound = new SoundSample("data/sons/VSS.mp3");
290     }
291     winSound.play();
292     winSound.mofidyPlayingVolument(0.1f, 1);
293     doWinSound = false;
294     gameNb++;
295 }
296 }
297
298 @Override
299 public void onClick(int x, int y, int button) {
300     super.onClick(x, y, button);
301     mouseActive = true;
302     if (waitLaser <= 0) {
303         positionClick = new Vector2(x, y);
304     }
305 }
306
307 @Override
308 public void onRelease(int x, int y, int button) {
309     super.onRelease(x, y, button);
310     if (waitLaser <= 0) {
311         positionClick.x = x;
312         positionClick.y = y;
313     }
314     mouseActive = false;
315 }
316
317 @Override
318 public void onKeyUp(int keycode) {
319     switch (keycode) {
320         case Input.Keys.UP:
321             ssLandry.thrustUp = false;
322             break;
323         case Input.Keys.LEFT:
324             ssLandry.thrustLeft = false;
325             break;
326         case Input.Keys.RIGHT:
327             ssLandry.thrustRight = false;
328             break;
329         case Input.Keys.W:
330             ssLandry.thrustUp = false;
331             break;
332         case Input.Keys.A:
333             ssLandry.thrustLeft = false;
334             break;
335         case Input.Keys.D:
336             ssLandry.thrustRight = false;
337             break;
338         default:
339             break;
340     }
341 }
342
343 @Override
344 public void onKeyDown(int keycode) {
345     switch (keycode) {
346         case Input.Keys.UP:
347             ssLandry.thrustUp = true;
348             break;
349         case Input.Keys.LEFT:
350             ssLandry.thrustLeft = true;
351             break;
352         case Input.Keys.RIGHT:
353             ssLandry.thrustRight = true;
354             break;
355         case Input.Keys.W:
356             ssLandry.thrustUp = true;

```

```

357         break;
358     case Input.Keys.A:
359         ssLandry.thrustLeft = true;
360         break;
361     case Input.Keys.D:
362         ssLandry.thrustRight = true;
363         break;
364     case Input.Keys.R:
365         if (ssLandry.isFinished()) {
366             replay();
367         }
368     default:
369         break;
370 }
371 }
372
373 public void replay() {
374     if (ssLandry.isLanded()) {
375         winSound.stop();
376     }
377
378     physics.removeAllObjectsfromSim();
379     ssLandry = new Spaceship(new Vector2(100, 700), gameNb);
380     sol = new Ground();
381     lz = new LandZone(sol.getPolyPoint(Constants.FLAT_ZONE));
382     physics.changePlayground(sol.getPolygon(), lz);
383     physics.addSimulatableObject(ssLandry);
384
385     meteors.clear();
386     for (int i = 0; i < gameNb; i++) {
387         meteors.add(new Gegner(new Vector2(rand.nextInt(300) + 400,
388             rand.nextInt(300) + 500)));
389         physics.addSimulatableObject(meteors.get(i));
390     }
391
392     doSoundFuel = true;
393     doExplosion = true;
394     doWinSound = true;
395 }
396
397 public static void main(String[] args) {
398     new LunarLander_Main();
399 }
400

```

```

1  package ch.hevs.gdx2d.lunar.main;
2
3  import java.util.Random;
4
5  import com.badlogic.gdx.graphics.Texture;
6  import com.badlogic.gdx.math.Vector2;
7
8  import ch.hevs.gdx2d.lib.GdxGraphics;
9  import ch.hevs.gdx2d.lib.interfaces.DrawableObject;
10 import ch.hevs.gdx2d.lunar.physics.Constants;
11 import ch.hevs.gdx2d.lunar.physics.PhysicalObject;
12
13 public class Gegner extends PhysicalObject implements DrawableObject{
14
15     private boolean destroyed;
16
17     private static final Random rand = new Random();
18
19     private Texture meteor;
20
21     public Gegner(Vector2 p) {
22         super(p, new Vector2(rand.nextFloat() * rand.nextInt(50) *
23             (rand.nextBoolean() ? 1 : rand.nextBoolean() ? 1 : -0.05f),
24                 rand.nextFloat() * rand.nextInt(10) * (-1)), Constants.GEGNER_MASS,
25                 40, 40);
26         meteor = new Texture("data/images/meteor.png");
27         destroyed = false;
28     }
29
30     @Override
31     public void step() {
32         this.force.y = -Constants.GRAVITY * this.mass;
33     }
34
35     @Override
36     public void draw(GdxGraphics arg0) {
37         if (!destroyed) {
38             arg0.draw(meteor, position.x - 25, position.y - 30, 50, 50);
39         }
40     }
41
42     @Override
43     public void removedFromSim() {
44         destroyed = true;
45     }
46
47     @Override
48     public boolean notifyCollision(int energy) {
49         return (energy >= Constants.DESTRUCTION_ENERGY);
50     }
51 }

```

```
1  package ch.hevs.gdx2d.lunar.main;
2
3  import com.badlogic.gdx.math.Rectangle;
4  import com.badlogic.gdx.math.Vector2;
5
6  import ch.hevs.gdx2d.lunar.physics.Constants;
7
8  public class LandZone {
9
10     public Rectangle landBox;
11
12     public LandZone(Vector2 position) {
13         landBox = new Rectangle(position.x, position.y, Constants.Z_WIDTH,
14             Constants.Z_HEIGHT);
15         landBox.setCenter(position.x + (800/Constants.SCALE/2), position.y);
16     }
17 }
```

```
1  package ch.hevs.gdx2d.lunar.main;
2
3  import com.badlogic.gdx.math.Intersector;
4  import com.badlogic.gdx.math.Vector2;
5  import com.badlogic.gdx.utils.Array;
6
7  import ch.hevs.gdx2d.components.graphics.Polygon;
8
9  public class PolygonWorking extends Polygon {
10     public PolygonWorking(Vector2[] points) {
11         super(points);
12     }
13
14     @Override
15     public boolean contains(Vector2 p) {
16         Vector2[] v = Polygon.float2vec2(this.getVertices());
17         Array<Vector2> a = Array.with(v);
18         return Intersector.isPointInPolygon(a, p);
19     }
20 }
```

```

1  package ch.hevs.gdx2d.lunar.main;
2
3  import java.util.ArrayList;
4  import java.util.Random;
5
6  import com.badlogic.gdx.graphics.Color;
7  import com.badlogic.gdx.graphics.Texture;
8  import com.badlogic.gdx.graphics.g2d.BitmapFont;
9  import com.badlogic.gdx.math.Vector2;
10
11 import ch.hevs.gdx2d.lib.GdxGraphics;
12 import ch.hevs.gdx2d.lib.interfaces.DrawableObject;
13 import ch.hevs.gdx2d.lunar.physics.Constants;
14 import ch.hevs.gdx2d.lunar.physics.Particles;
15 import ch.hevs.gdx2d.lunar.physics.PhysicalObject;
16
17 public class Spaceship extends PhysicalObject implements DrawableObject {
18
19     private int fuel;
20     private int gameNb;
21
22     public boolean thrustUp;
23     public boolean thrustLeft;
24     public boolean thrustRight;
25
26     private boolean landed;
27     private boolean kaputt;
28     private boolean firstExplo;
29
30     // Particles stuff
31     private ArrayList<Particles> reactor;
32     private ArrayList<Particles> explosion;
33
34     static final Random rand = new Random();
35
36     // Textures
37     private Texture spaceship;
38     private Texture ded;
39
40     public Spaceship(Vector2 p, int gameNb) {
41         super(p, new Vector2(0, 0), Constants.BASE_MASS, 50, 50);
42
43         fuel = (int) Constants.MAX_FUEL;
44         this.gameNb = gameNb;
45
46         thrustUp = false;
47         thrustLeft = false;
48         thrustRight = false;
49
50         kaputt = false;
51         landed = false;
52         firstExplo = true;
53
54         reactor = new ArrayList<Particles>();
55         explosion = new ArrayList<Particles>();
56
57         spaceship = new Texture("data/images/ssLandry.png");
58         ded = new Texture("data/images/Rip.png");
59     }
60
61     @Override
62     public void draw(GdxGraphics arg0) {
63         // arg0.drawFilledRectangle(position.x, position.y+8, 10, 16, 0, Color.BLUE);
64         if (kaputt) {
65             Vector2 vec;
66             if (firstExplo) {
67                 for (int i = 0; i < 500; i++) {
68                     vec = new Vector2(1, 1).setToRandomDirection();
69                     explosion.add(new Particles(new Vector2(position.x, position.y),
70                         vec.scl(rand.nextFloat() * 2),
71                         rand.nextInt(80),
72                         rand.nextBoolean() ? "data/images/fire_particle.png" :
73                             "data/images/reactor_particle.png"));

```

```

72         }
73         firstExplo = false;
74     }
75     // Explosion animation
76     arg0.draw(ded, position.x - 25, position.y - 25, 50, 50);
77
78     if (explosion.size() != 0) {
79         for (int i = 0; i < explosion.size(); i++) {
80             Particles p = explosion.get(i);
81             p.update();
82             p.draw(arg0);
83             if (p.shouldBeDestroyed()) {
84                 explosion.remove(p);
85             }
86         }
87     }
88
89     } else {
90         arg0.draw(spaceship, position.x - 25, position.y - 30, 50, 50);
91
92         if (!landed && fuel > 0 && (thrustUp || thrustLeft || thrustRight)) {
93             // Thrust animation
94             reactor.add(new Particles(new Vector2(position.x, position.y - 25),
95                 new Vector2(rand.nextFloat() / 2 * (rand.nextBoolean() ? 1 :
96                     -1), -2).mulAdd(speed, 0.1f),
97                 rand.nextInt(80),
98                 rand.nextBoolean() ? "data/images/fire_particle.png" :
99                     "data/images/reactor_particle.png"));
100         }
101         if (reactor.size() != 0) {
102             for (int i = 0; i < reactor.size(); i++) {
103                 Particles p = reactor.get(i);
104                 p.update();
105                 p.draw(arg0);
106                 if (p.shouldBeDestroyed()) {
107                     reactor.remove(p);
108                 }
109             }
110         }
111
112         drawHUD(arg0);
113     }
114
115 void drawHUD(GdxGraphics arg0) {
116     // Print fond noir
117     arg0.drawFilledRectangle(400, 50, 800, 100, 0, Color.DARK_GRAY);
118     // Print fuel
119     Vector2 POSITION_BAR_FUEL = new Vector2(650, 50);
120     Vector2 POSITION_SPEED = new Vector2(100, 50);
121     Vector2 POSITION_NB_GAME = new Vector2(350, 50);
122
123     arg0.drawRectangle(POSITION_BAR_FUEL.x, POSITION_BAR_FUEL.y, 200, 50, 0);
124     arg0.drawFilledRectangle(POSITION_BAR_FUEL.x - (float) (fuel /
125         Constants.MAX_FUEL) * 100 + 100,
126         POSITION_BAR_FUEL.y, (float) (fuel / Constants.MAX_FUEL) * 200, 50,
127         0, Color.RED);
128     arg0.drawString(POSITION_BAR_FUEL.x - 90, POSITION_BAR_FUEL.y,
129         "Fuel : " + fuel + "/" + (int) Constants.MAX_FUEL);
130
131     // Print speed
132     BitmapFont bfSpeed = new BitmapFont();
133     bfSpeed.setColor(Color.RED);
134     if (speed.len() < Constants.CRASH_SPEED) {
135         bfSpeed.setColor(Color.GREEN);
136     }
137     arg0.drawString(POSITION_SPEED.x, POSITION_SPEED.y, "Speed : " + (int)
138         speed.len() + " m/s", bfSpeed);
139
140     arg0.drawString(POSITION_NB_GAME.x, POSITION_NB_GAME.y, "Apollo " + gameNb);
141 }

```



```

140 @Override
141 public void step() {
142     // Simulate de thrust from the reactors
143     if (!landed) {
144         if (thrustUp && fuel > 0) {
145             force.y = Constants.MAX_THRUST;
146             fuel--;
147         } else {
148             force.y = 0;
149         }
150
151         if (thrustLeft && !thrustRight && fuel > 0) {
152             force.x = -Constants.MAX_THRUST;
153             fuel--;
154         } else if (!thrustLeft && thrustRight && fuel > 0) {
155             force.x = Constants.MAX_THRUST;
156             fuel--;
157         } else {
158             force.x = 0;
159         }
160     }
161 }
162
163 @Override
164 public void removedFromSim() {
165     kaputt = !landed;
166 }
167
168 @Override
169 public boolean notifyCollision(int energy) {
170     landed = energy < Constants.DESTRUCTION_ENERGY;
171     return (!landed);
172 }
173
174 public boolean isLanded() {
175     return landed;
176 }
177
178 public boolean isFinished() {
179     return (kaputt || landed);
180 }
181
182 public boolean isDry() {
183     return (fuel <= 0);
184 }
185
186 public boolean isKaputt() {
187     return kaputt;
188 }
189
190 }
191

```

## **B Package Physique**

```

1  package ch.hevs.gdx2d.lunar.physics;
2
3  import com.badlogic.gdx.math.Rectangle;
4
5  /**
6   * Interface for objects that represent obstacles for the {@link PhysicsSimulator}
7   * @author P.-A. Mudry
8   */
9  public interface Collisionnable {
10     /**
11      * Callback used to notify the object that he was removed form the simulation
12      */
13     void removedFromSim();
14
15     /**
16      * When the {@link PhysicalObject} that implements this
17      * interfaces collides another object, this callback method
18      * is called
19      *
20      * @param energy Energy of the collision
21      * @return true if the object has to be destroyed
22      */
23     public boolean notifyCollision(int energy);
24
25     /**
26      * Gives the bounding box of the object which is used for detecting
27      * collisions.
28      *
29      * @return the bounding box
30      */
31     public Rectangle getBoundingBox();
32 }
33

```

```

1 package ch.hevs.gdx2d.lunar.physics;
2
3 /**
4  * Some useful physics constant that are used by the {@link PhysicsSimulator} class
5  * and others.
6  * @author P.-A. Mudry
7  */
8 public final class Constants {
9     /**
10      * Graphics related constants
11      */
12     public static final int WIN_WIDTH = 800;
13     public static final int WIN_HEIGHT = 800;
14     public static final int FPS = 100;
15
16     /**
17      * Physics environment
18      */
19     public static final float GRAVITY = -1.62f;
20     public static final float DELTA_TIME = 0.1f;
21     public static final float AIR_FRICTION = 0.0f;
22     public static final float DAMPING_FACTOR = 0.9f;
23
24     /**
25      * Maximal impact speed triggering a crash
26      */
27     public static final double CRASH_SPEED = 10;
28
29     /**
30      * Spaceship related
31      */
32     public static final float MAX_THRUST = 1500f;
33     public static final double MAX_FUEL = 300;
34     public static final int BASE_MASS = 300;
35
36     public static final int GEGNER_MASS = 100;
37
38     /**
39      * Game related constants
40      */
41     // Maximal impact energy triggering a object destruction
42     public static final int DESTRUCTION_ENERGY = (int)
43     (BASE_MASS*CRASH_SPEED*CRASH_SPEED/2);
44     public static final int CLOUD_DENSITY = 5;
45     public static final int GROUND_ALTITUDE = 100;
46     public static final boolean DRAW_BOUNDINGBOXES = false;
47
48     /**
49      * Ground parameters
50      */
51     public static final float MAX_INCLINE = 100.0f;
52     public static final int MIN_ALTITUDE = 200;
53     public static final int SCALE = 10;
54     public static final int FLAT_ZONE = 7;
55
56     /**
57      * Landing Zone
58      */
59     public static final int Z_WIDTH = 100;
60     public static final int Z_HEIGHT = 30;
61 }

```

```

1  package ch.hevs.gdx2d.lunar.physics;
2
3  import com.badlogic.gdx.math.Vector2;
4
5  import ch.hevs.gdx2d.lunar.main.PolygonWorking;
6
7  public class Ground {
8
9      private Vector2[] polyPoints = new Vector2[Constants.SCALE];
10     private PolygonWorking groundPoly;
11
12     public Ground() {
13         for (int i = 0; i < polyPoints.length; i++) {
14             if (i == 0) {
15                 polyPoints[i] = new Vector2(0, 100);
16             } else if (i == polyPoints.length - 1) {
17                 polyPoints[i] = new Vector2(800, 100);
18             } else if (i == Constants.FLAT_ZONE + 1) {
19                 polyPoints[i] = new Vector2(800 / polyPoints.length * i,
20                 polyPoints[i - 1].y);
21             } else if (polyPoints[i - 1].y <= Constants.MIN_ALTITUDE) {
22                 polyPoints[i] = new Vector2(800 / polyPoints.length * i,
23                 (float) (polyPoints[i - 1].y + Math.random() *
24                 Constants.MAX_INCLINE * 2));
25             } else {
26                 polyPoints[i] = new Vector2(800 / polyPoints.length * i, (float)
27                 (polyPoints[i - 1].y
28                 + Math.random() * Constants.MAX_INCLINE * 2 -
29                 Constants.MAX_INCLINE));
30             }
31         }
32         groundPoly = new PolygonWorking(polyPoints);
33     }
34
35     public PolygonWorking getPolygon() {
36         return groundPoly;
37     }
38
39     public Vector2 getPolyPoint(int i) {
40         if (i >= polyPoints.length) {
41             i = 0;
42         }
43         return polyPoints[i];
44     }
45 }

```

```
1 package ch.hevs.gdx2d.lunar.physics;
2
3 import com.badlogic.gdx.math.Vector2;
4
5 import ch.hevs.gdx2d.components.bitmaps.BitmapImage;
6 import ch.hevs.gdx2d.lib.GdxGraphics;
7 import ch.hevs.gdx2d.lib.interfaces.DrawableObject;
8
9 public class Particles implements DrawableObject{
10
11     private Vector2 position;
12     private Vector2 speed;
13
14     private float alpha;
15     private BitmapImage img;
16
17     private int lifetime;
18
19     public Particles(Vector2 p, Vector2 s, int lifetime, String imgPath) {
20         this.position = p;
21         this.speed = s;
22         this.lifetime = lifetime;
23
24         this.alpha = 1f;
25         this.img = new BitmapImage(imgPath);
26     }
27
28     public void update() {
29         this.position.add(this.speed);
30         alpha -= 1.0f/lifetime;
31     }
32
33     public void changePosition(float x, float y) {
34         this.position = new Vector2(x,y);
35     }
36
37     public boolean shouldBeDestroyed() {
38         return (alpha <= 0.01f);
39     }
40
41     @Override
42     public void draw(GdxGraphics arg0) {
43         arg0.drawAlphaPicture(this.position, this.alpha, this.img);
44     }
45 }
46
```

```

1  package ch.hevs.gdx2d.lunar.physics;
2
3  import com.badlogic.gdx.math.Rectangle;
4  import com.badlogic.gdx.math.Vector2;
5
6  public abstract class PhysicalObject implements Simulatable, Collisionnable {
7
8      public Vector2 position; // position
9      public Vector2 speed; // speed
10     public Vector2 acceleration; // acceleration
11     public int mass; // mass
12     public Vector2 force; // force applied on the object
13     protected Rectangle boundingBox;
14
15     public PhysicalObject(Vector2 p, Vector2 s, int m, int width, int height) {
16
17         this.position = p;
18         this.speed = s;
19         this.acceleration = new Vector2(0, 0);
20         this.force = new Vector2(0, 0);
21         this.mass = m;
22
23         this.boundingBox = new Rectangle(p.x, p.y, width, height);
24     }
25
26     @Override
27     public void step() {
28         // TODO Auto-generated method stub
29
30     }
31
32     @Override
33     public Rectangle getBoundingBox() {
34         boundingBox.setCenter(position);
35         return boundingBox;
36     }
37 }
38

```

```

1  package ch.hevs.gdx2d.lunar.physics;
2
3  import java.util.ArrayList;
4
5  import com.badlogic.gdx.math.Rectangle;
6  import com.badlogic.gdx.math.Vector2;
7
8  import ch.hevs.gdx2d.lunar.main.LandZone;
9  import ch.hevs.gdx2d.lunar.main.PolygonWorking;
10 import ch.hevs.gdx2d.lunar.main.Spaceship;
11
12 /**
13  * A simple physics simulator for the infl project.
14  */
15 public class PhysicsSimulator {
16
17     /**
18      * This represents the borders of the simulated area (for collisions)
19      */
20     double width;
21     double height;
22
23     /**
24      * Ground & Landing Zone for the spaceship
25      */
26     PolygonWorking ground;
27     LandZone lz;
28
29     private final boolean VERBOSE_PHYSICS = false;
30
31     /**
32      * The objects that require physics simulation (objects that move)
33      */
34     private ArrayList<Simulatable> sim_objects;
35
36     /**
37      * @param width The width of the space for the p simulation
38      * @param height The height of the space for the p simulation
39      */
40     public PhysicsSimulator(double width, double height) {
41         sim_objects = new ArrayList<Simulatable>();
42         this.width = width;
43         this.height = height;
44     }
45
46     /**
47      * Adds a new object to the simulation framework
48      *
49      * @param o The object to be added
50      */
51     public void addSimulatableObject(Simulatable o) {
52         sim_objects.add(o);
53     }
54
55     /**
56      * Remove an object from simulation
57      */
58     public void removeObjectFromSim(PhysicalObject o) {
59         o.removedFromSim();
60         sim_objects.remove(o);
61     }
62
63     /**
64      * Simulates all the objects that ought to be simulated
65      *
66      * @return
67      */
68     public void simulate_step() {
69         if (sim_objects.size() == 0)
70             return;
71
72         for (int i = 0; i < sim_objects.size(); i++) {
73             boolean ended = false;

```



```

74     Simulatable s = sim_objects.get(i);
75     s.step();
76
77     if (s instanceof PhysicalObject) {
78         PhysicalObject p = (PhysicalObject) s;
79
80         /**
81          * General Physics equations
82          */
83         // 1 - Newton's first law
84         // Vector2 forceSum = oldAcc.scl(p.mass);
85         // 2 - Atmospheric friction => -kv
86         Vector2 forceFrix = new Vector2(-p.speed.x * Constants.AIR_FRICTION,
87             -p.speed.y * Constants.AIR_FRICTION);
88         // 3 - Gravity => mg
89         // Vector2 forceGrav = new Vector2(0, p.mass * Constants.GRAVITY);
90         Vector2 accGravity = new Vector2(0, Constants.GRAVITY);
91         /**
92          * forceFrix + forceGrav = forceSum -> acceleration = GRAVITY -
93          * (AIR_FRICTION/mass) -> speed(t + DELTA_TIME) = speed(t) +
94          * acceleration(DELTA_TIME) -> position(t + DELTA_TIME) =
95          * position(t) +
96          * speed(t)*DELTA_TIME
97          */
98         // acceleration = GRAVITY + ((forceFrix + forceObj)/mass)
99         p.acceleration = accGravity.mulAdd(forceFrix.add(p.force), 1.0f /
100             (p.mass));
101         // p.acceleration = accGravity.mulAdd(forceFrix, 1.0f/(p.mass));
102         // speed = oldSpeed + acceleration*DELA_TIME
103         p.speed = p.speed.mulAdd(p.acceleration, Constants.DELTA_TIME);
104
105         if (VERBOSE_PHYSICS) {
106             System.out.println("Position :" + p.position);
107             System.out.println("Speed :" + p.speed);
108             System.out.println("Acceleration :" + p.acceleration);
109         }
110
111         /**
112          * Elastic collisions with borders
113          */
114         // Calculate collision energy  $E_{cin} = 1/2 * mv^2$ 
115         ended = p.notifyCollision((int) (p.mass * p.speed.len() *
116             p.speed.len()) / 2);
117         Rectangle box = p.getBoundingBox();
118         Vector2[] boxPoints = new Vector2[4];
119         boxPoints[0] = new Vector2(box.getX(), box.getY());
120         boxPoints[1] = new Vector2(box.getX() + box.getWidth(), box.getY());
121         boxPoints[2] = new Vector2(box.getX(), box.getY() + box.getHeight());
122         boxPoints[3] = new Vector2(box.getX() + box.getWidth(), box.getY() +
123             box.getHeight());
124
125         // Ground corner into object
126         for (int j = 0; j < Constants.SCALE; j++) {
127             if (box.contains(ground.getVertex(j)) || ended) {
128                 ended = true;
129                 break;
130             }
131         }
132
133         // Object corner into ground
134         for (int j = 0; j < 4; j++) {
135             if (ground.contains(boxPoints[j]) || ended) {
136                 ended = true;
137                 break;
138             }
139         }
140
141         if (p.position.x >= width || p.position.x <= 0) {
142             ended = true;
143         }
144
145         // LandingZone

```

```

143         if (box.overlaps(lz.landBox)) {
144             // Too fast ?
145             if (p.notifyCollision((int) (p.mass * p.speed.len() *
146                                     p.speed.len()) / 2)) {
147                 // Destroyed
148                 ended = true;
149             } else {
150                 ended = true;
151             }
152         }
153         if (p instanceof Spaceship) {
154             for (int j = 0; j < sim_objects.size(); j++) {
155                 if (j != sim_objects.indexOf(p)) {
156                     ended |= p.getBoundingBox()
157                             .overlaps(((PhysicalObject)
158                                         sim_objects.get(j)).getBoundingBox());
159                 }
160             }
161         }
162         // position = oldPos + oldSpeed*DELTA_TIME
163         p.position = p.position.mulAdd(p.speed, Constants.DELTA_TIME);
164         if (ended) {
165             removeObjectFromSim(p);
166         }
167     }
168 }
169
170 }
171
172 public void removeAllObjectsfromSim() {
173     sim_objects.clear();
174 }
175
176 public void changePlayground(PolygonWorking ground, LandZone lz) {
177     this.ground = ground;
178     this.lz = lz;
179 }
180 }
181

```

```
1 package ch.hevs.gdx2d.lunar.physics;
2
3 /**
4  * Interface for objects that can be simulated via the {@link PhysicsSimulator}.
5  *
6  * @author P.-A. Mudry
7  */
8 public interface Simulatable {
9
10     /**
11      * Notify implementer that a simulation step has been performed
12      */
13     void step();
14 }
15
```