```java
package ch.hevs.gdx2d.lunar.physics;

import java.util.ArrayList;

import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.math.Vector2;

import ch.hevs.gdx2d.lunar.main.LandZone;
import ch.hevs.gdx2d.lunar.main.PolygonWorking;
import ch.hevs.gdx2d.lunar.main.Spaceship;

/**
 * A simple physics simulator for the inf1 project.
 */
public class PhysicsSimulator {

    /**
     * This represents the borders of the simulated area (for collisions)
     */
    double width;
    double height;

    /**
     * Ground & Landing Zone for the spaceship
     */
    PolygonWorking ground;
    LandZone lz;

    private final boolean VERBOSE_PHYSICS = false;

    /**
     * The objects that require physics simulation (objects that move)
     */
    private ArrayList<Simulatable> sim_objects;

    /**
     * @param width  The width of the space for the p simulation
     * @param height The height of the space for the p simulation
     */
    public PhysicsSimulator(double width, double height) {
        sim_objects = new ArrayList<Simulatable>();
        this.width = width;
        this.height = height;
    }

    /**
     * Adds a new object to the simulation framework
     *
     * @param o The object to be added
     */
    public void addSimulatableObject(Simulatable o) {
        sim_objects.add(o);
    }

    /**
     * Remove an object from simulation
     */
    public void removeObjectFromSim(PhysicalObject o) {
        o.removedFromSim();
        sim_objects.remove(o);
    }

    /**
     * Simulates all the objects that ought to be simulated
     *
     * @return
     */
    public void simulate_step() {
        if (sim_objects.size() == 0)
            return;

        for (int i = 0; i < sim_objects.size(); i++) {
            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 + forceGrave = forceSum -> acceleration = GRAVITY -
93                   * (AIR_FRICTION/mass) -> speed(t + DELTA_TIME) = speed(t) +
94                   * acceleration(DELTA_TIME) -> position(t + DELTA_TIME) =
                     position(t) +
95                   * speed(t)*DELTA_TIME
96                   */
97
98                  // acceleration = GRAVITY + ((forceFrix + forceObj)/mass)
99                  p.acceleration = accGravity.mulAdd(forceFrix.add(p.force), 1.0f /
                    (p.mass));
100                 // p.acceleration = accGravity.mulAdd(forceFrix, 1.0f/(p.mass));
101                 // speed = oldSpeed + acceleration*DELA_TIME
102                 p.speed = p.speed.mulAdd(p.acceleration, Constants.DELTA_TIME);
103
104                 if (VERBOSE_PHYSICS) {
105                     System.out.println("Position :" + p.position);
106                     System.out.println("Speed :" + p.speed);
107                     System.out.println("Acceleration :" + p.acceleration);
108                 }
109
110                 /**
111                  * Elastic collisions with borders
112                  */
113 //              // Calculate collision energy Ecin = 1/2 * mv²
114 //              ended = p.notifyCollision((int) (p.mass * p.speed.len() *
    p.speed.len()) / 2);
115                 Rectangle box = p.getBoundingBox();
116                 Vector2[] boxPoints = new Vector2[4];
117                 boxPoints[0] = new Vector2(box.getX(), box.getY());
118                 boxPoints[1] = new Vector2(box.getX() + box.getWidth(), box.getY());
119                 boxPoints[2] = new Vector2(box.getX(), box.getY() + box.getHeight());
120                 boxPoints[3] = new Vector2(box.getX() + box.getWidth(), box.getY() +
                    box.getHeight());
121
122                 // Ground corner into object
123                 for (int j = 0; j < Constants.SCALE; j++) {
124                     if (box.contains(ground.getVertex(j)) || ended) {
125                         ended = true;
126                         break;
127                     }
128                 }
129
130                 // Object corner into ground
131                 for (int j = 0; j < 4; j++) {
132                     if (ground.contains(boxPoints[j]) || ended) {
133                         ended = true;
134                         break;
135                     }
136                 }
137
138                 if (p.position.x >= width || p.position.x <= 0) {
139                     ended = true;
140                 }
141
142                 // LandingZone
```

```
143                     if (box.overlaps(lz.landBox)) {
144                         // Too fast ?
145                         if (p.notifyCollision((int) (p.mass * p.speed.len() *
                            p.speed.len()) / 2)) {
146                             // Destroyed
147                             ended = true;
148                         } else {
149                             ended = true;
150                         }
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)
                                        sim_objects.get(j)).getBoundingBox());
158                             }
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 
```