2D Shape Collision Simulation

By: Ken Dopp, Tikhon Jelvis, Gregory Nisbet, Jacob Taylor
Approach

Two aspects:

- Physical collision experiment with eraser boxes
defines real world data to corroborate simulation
  - Provides real world data to corroborate simulation

- Interactive computer simulation
  - Simulates collisions with a variety of shapes
**Experiment Setup**

- **Eraser box**: 22.5 grams
- **Weight**: 1.85 meters
Experimental Variables

Independent:
- Initial distance and orientations

Dependent:
- Angles of objects
- Location of objects

Controlled:
- Force applied to object
- Friction of table.
Measurements

1. Distance of first box
2. Distance of second box
3. Angle of first box
4. Angle of second box
Data

1. Effect of initial angle of object 2 on final distance of object 1
   - X-axis: Initial angle of object 2, degrees
   - Y-axis: Final distance of object 1, cm
   - Data points indicate a relationship between the initial angle and final distance.

2. Effect of initial angle of object 2 on final angle of object 1
   - X-axis: Initial angle of object 2, degrees
   - Y-axis: Final angle of object 1, degrees
   - Data points show a trend in the final angles.

3. Effect of initial angle of object 2 on final distance of object 2
   - X-axis: Initial angle of object 2, degrees
   - Y-axis: Final distance of object 2, cm
   - Data points reveal a pattern in the final distances.

4. Effect of initial angle of object 2 on final angle of object 2
   - X-axis: Initial angle of object 2, degrees
   - Y-axis: Final angle of object 2, degrees
   - Data points exhibit a correlation between the initial and final angles.
Simulation
Engine

- Contained in one package (separate from the GUI)
- Contains circles and polygons
- Each shape has a position, velocity, angular velocity, mass, and moment of inertia
  - Moment of inertia: measure of how difficult an object is to turn.
Impulses

- Impulse: a force applied instantaneously, measured in kg m/s.
- Can change a shape's linear and/or angular velocity.
- Rotation depends on moment arm (r).
- $\Delta v = \Delta p/m$, $\Delta \omega = r\Delta p/I$ ($I$ = moment of inertia)
Collisions

- Impulses push the objects apart
- Elastic collisions preserve kinetic energy
  \[ E_k = \frac{1}{2}mv_2^2 + \frac{1}{2}I\omega_2 \]
- Both angular and linear velocity changes
  \[ p = \frac{2(v_1 - v_2 + w_1r_1 - w_2r_2)}{1/m_1 + 1/m_2 + r_{12}/I_1 + r_{22}/I_2} \]
More on Collisions

- Objects have friction and bounciness
- \[ p = \frac{2(v_1 - v_2 + w_1 r_1 - w_2 r_2)}{(1/m_1 + 1/m_2 + r_1^2/I_1 + r_2^2/I_2)} \]
- This formula is only for elastic collisions
- For inelastic collisions, the 2 is replaced by 1 + b_1 b_2 (the bounciness factors, 0 ≤ b ≤ 1)
- Friction: \( \mu = f_1 f_2 \) (the friction factors, 0 ≤ f ≤ 1)
- Friction impulse = \( \mu \times p \) (normal impulse)
Structure

- Modular and extendable
- Model / view separation
  - GUI in different package from model
  - Model: the actual shapes and math
  - GUI: the user interface, uses the model
Synchronization

- Concurrency
  - Animation always requires multiple threads
- Manual synchronization
  - The two main lists needed to be synchronized manually

```java
protected void step(double amount) {
  // collide every body with every other body
  // but don't do reverse collisions
  // e.g. don't do both b.collision(a) and a.collision(b)
  synchronized (bodies) {
    for (int i = 0; i < bodies.size(); ++i) {
      Body b1 = bodies.get(i);
      for (int j = i + 1; j < bodies.size(); ++j) {
        Body b2 = bodies.get(j);
        b1.collision(b2);
      }
    }
  }

  synchronized (bodies) {
    // step all bodies
    for (Body b : bodies) {
      b.step(amount);
      // gravity
      b.addImpulse(new Vector(0, 1 * amount * b.mass() * gravity));
    }
  }

  synchronized (springs) {
    // step all springs
    for (Spring s : springs) {
      s.step(amount);
    }
  }

  synchronized (grabList) {
    if (grabbed != null) {
      Vector mouse = new Vector(mouseX, mouseY);
      Vector diff = mouse.subtract(grabbed.connectedPoint());
      Vector result = diff.multiply(10 * mass);
      grabbed.addImpulse(result);
      grabbed
            .addImpulse(grabbed.getVelocity())
            .multiply(-10 * amount);
    }
  }
```
Documentation

- Javadocs (auto-generated html-based documentation for Java)
- Could help other people working on similar projects