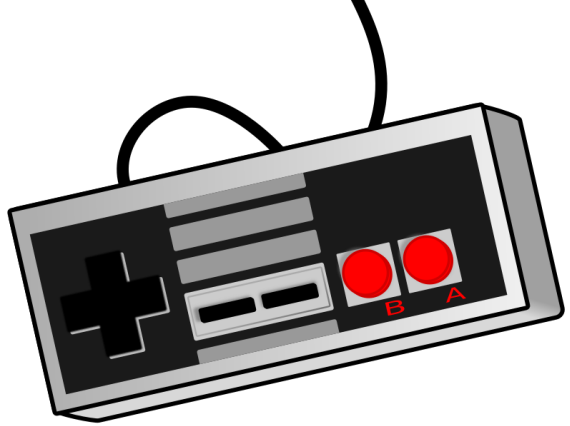


Lab_8

Greenfoot Simulation (Part 1)

420-141-VA - GAME PROGRAMMING 1 - VANIER COLLEGE

Outline

- Setting up the Simulation Scenario
- Instantiate a Cannon ball
- Game Physics (Euler Integration)
- Calculating the duration of time steps
- Making the Cannon ball move

Step 1: Setting up the Simulation Scenario

Greenfoot Simulation (Part 1)

- Download the **Lab_8.zip** file from Omnivox, which contains the Scenario
- Unzip the contents to somewhere on your USB key or hard disk.
- Open the scenario in that location with **Greenfoot**
- You should see the standard Greenfoot interface, with an empty world

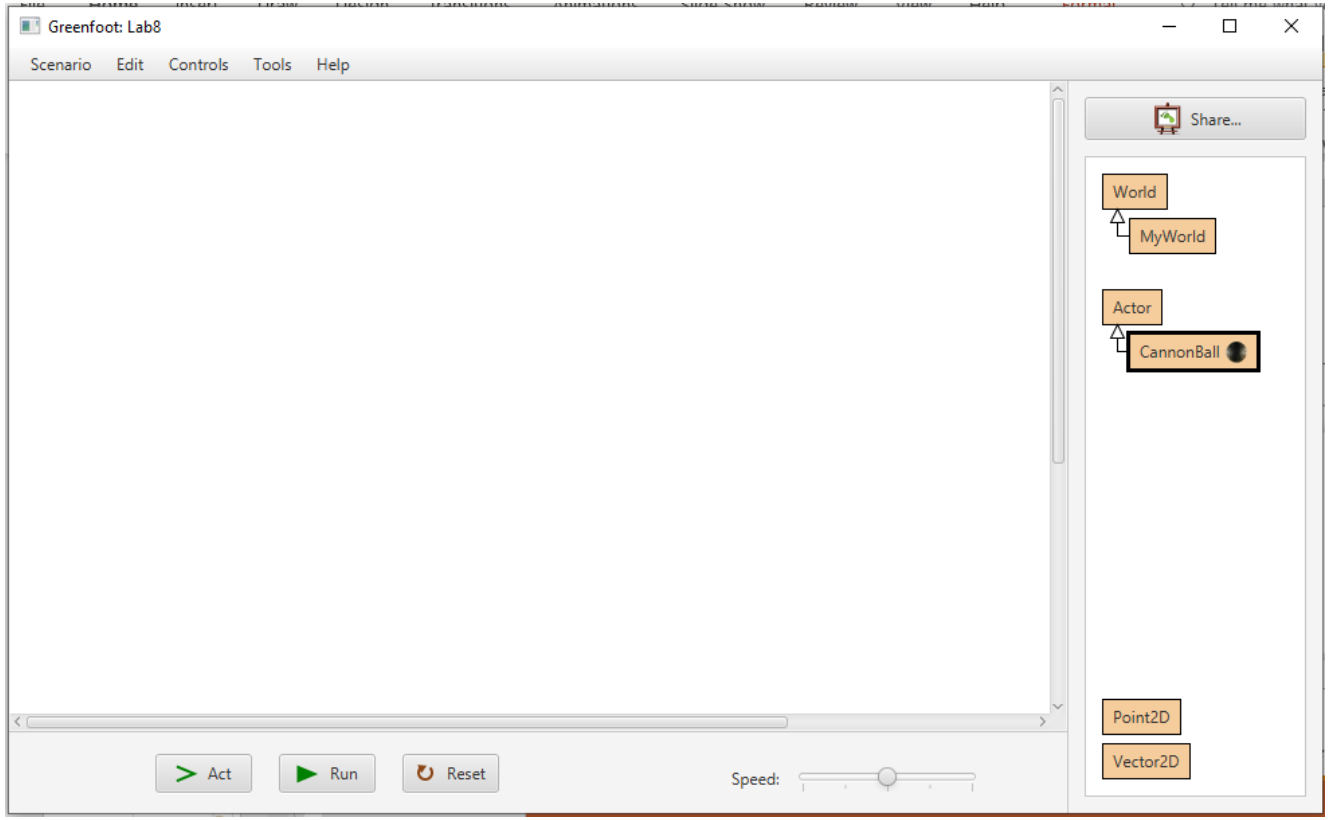


Figure 1

Setting up the Scenario

The project contains:

- Images


cannon.png


cannonball.png


smoke.png


target.png


targetDestroyed.png

- Sound effects and music
- Vector2D and Point2D class
- Empty CannonBall actor class

Step 2: Instantiate a Cannon ball

Setup Scenario

- Instantiate a Cannon ball (or a few) towards the top of the world
- Save the World
- Next, we will apply gravity to the balls

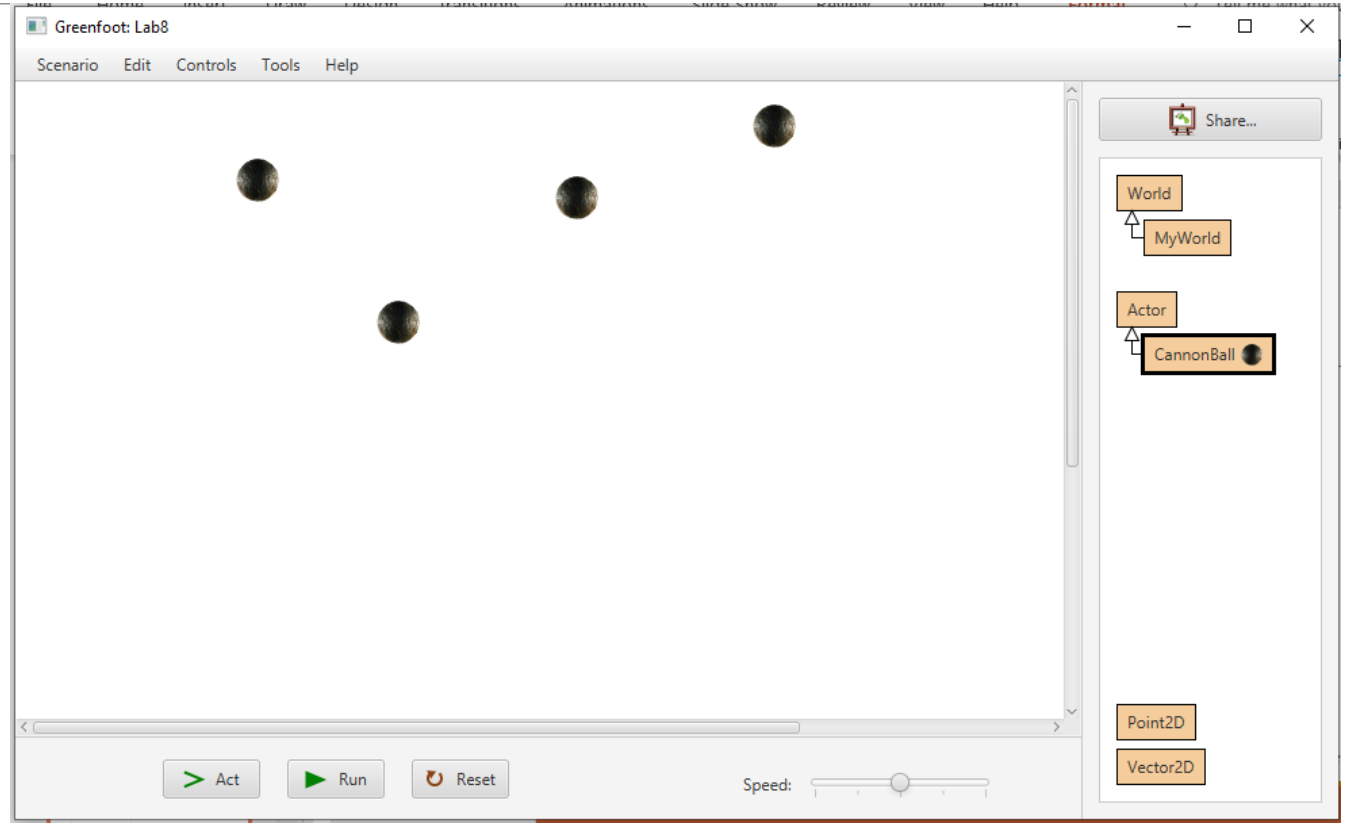


Figure 2

Game Physics (Euler Integration)

Re-compute Acceleration, Velocities and Position once per time step

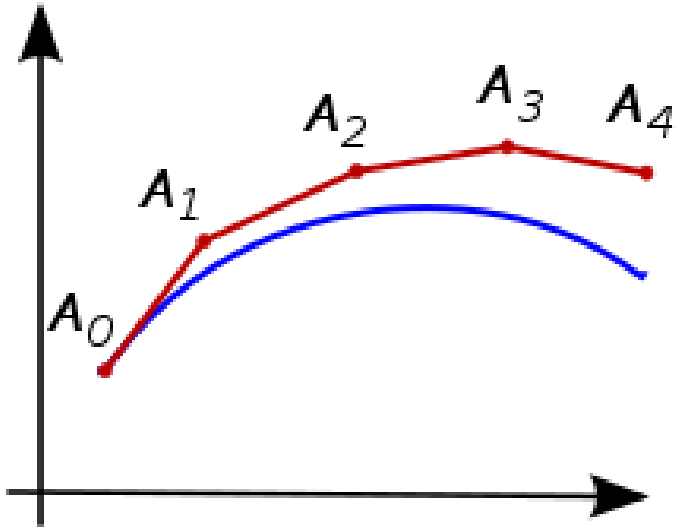
- Acceleration includes Gravity and other forces
- Apply acceleration to velocity
- Apply velocity to positions

Advantages with Euler Integration

- Easy to implement
- Fast to compute

Problems with Euler Integration

- Not perfectly accurate, but good enough for simple motion



Typical Problems

Blue: Correct Positio
Red: Positions from
Euler Integrat

Step 3: Calculating the duration of time steps

To get accurate physics simulation, we must know the amount of time between frames (or time step duration).

This is required in Euler integration for updating the velocity from acceleration and positions from velocity.

We do it in the World, so this calculation is done only once per frame.

The calculation is based on System.getCurrentTimeMillis() which returns the CPU clock time.

```
public class MyWorld extends World
{
    private long lastFrameTimeMS;
    private double timeStepDuration;

    public MyWorld()
    {
        super(1024, 768, 1);
        prepare();

        lastFrameTimeMS = System.currentTimeMillis();
        timeStepDuration = 1.0 / 60; // seems to be the default
    }

    public void act()
    {
        timeStepDuration = (System.currentTimeMillis() - lastFrameTimeMS) / 1000.0;
        lastFrameTimeMS = System.currentTimeMillis();
    }

    public double getTimeStepDuration()
    {
        return timeStepDuration;
    }
```

Figure 3

Step 4: Making the Cannon ball move

Create Fields for Cannon Ball and initialize them in the constructor

```
public class CannonBall extends Actor
{
    private Point2D position;
    private Vector2D velocity;
    private Vector2D acceleration;

    private static final double GRAVITY = 9.8;

    public CannonBall()
    {
        position = null;
        velocity = new Vector2D(0.0, 0.0);
        acceleration = new Vector2D(0.0, GRAVITY);
    }
```

Update Movement in the act() method

```
public void act()
{
    // Initial position
    if (position == null)
    {
        position = new Point2D(getX(), getY());
    }

    // Get time step duration
    MyWorld world = (MyWorld) getWorld();
    double dt = world.getTimeStepDuration();

    // Update velocity
    Vector2D velocityVariation = Vector2D.multiply(acceleration, dt);
    velocity = Vector2D.add(velocity, velocityVariation);

    // Update position
    Vector2D positionVariation = Vector2D.multiply(velocity, dt);
    position.add(positionVariation);

    // Set new actor position
    setLocation((int) position.getX(), (int) position.getY());
}
```

Figure 4

Making the Cannon ball move

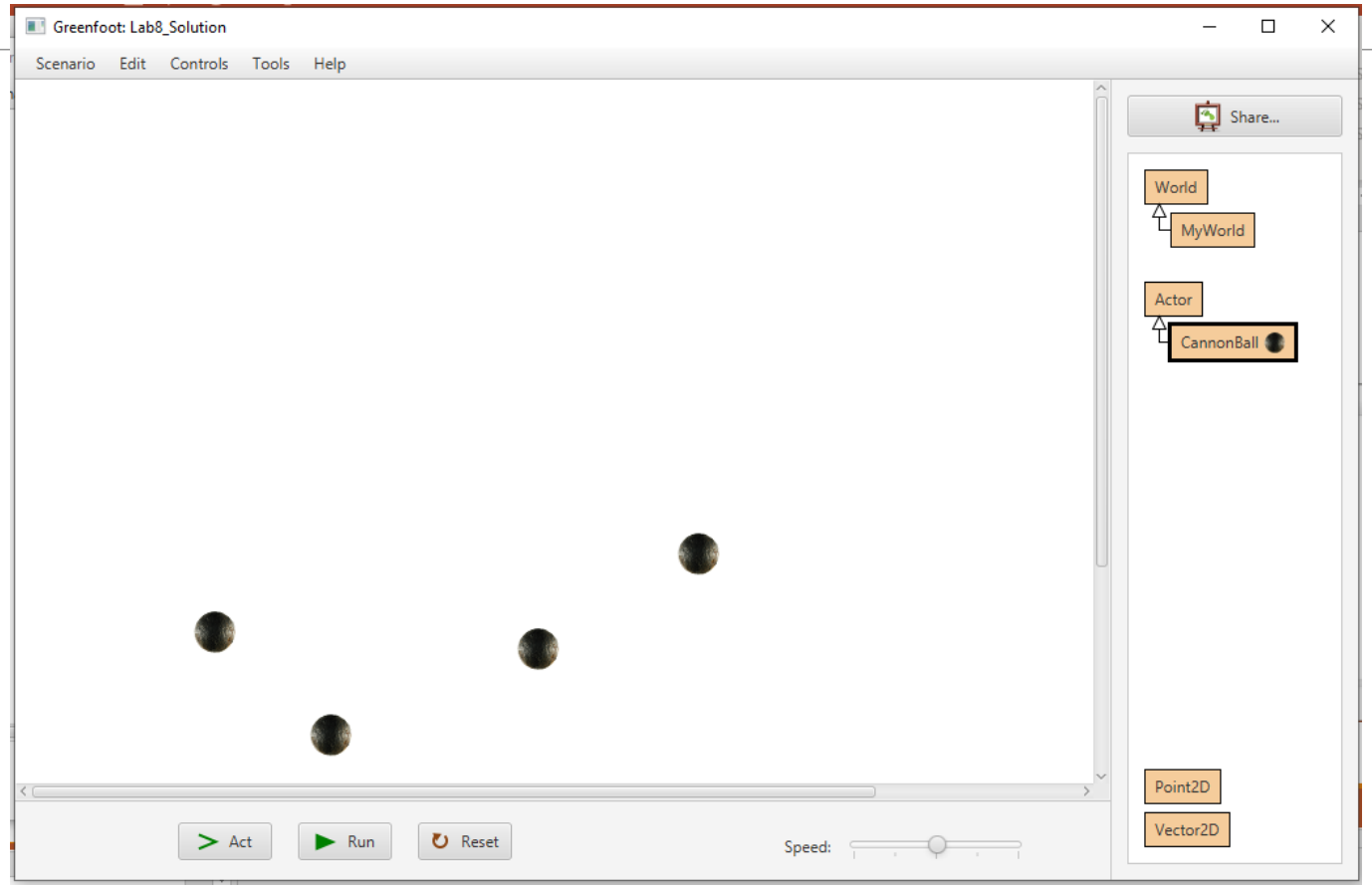


Figure 5

Questions