Homework 2.3

Classes

When dealing with any computer program, it is important to group your code effectively. Even if it takes longer to do, careful code structure is always worth it, both because it makes your code easier to read and write and because it makes your code much easier to use in future projects.

In the case of kinematics, clearly we are dealing with a variety of different projectiles. In principle you could have many of these at once, so it is very sensible to make a Projectile class to describe things.

Create a Projectile class which contains all the information unique to a projectile, with sensible ways to access them. Make sure the projectile contains methods that make sense for it to perform – in particular, a Projectile should be able to calculate its own position, velocity, and acceleration, given an external force and a time increment.

When designing classes, make sure to give them data members and methods that make sense physically. It makes sense for a Projectile class to have position and velocity fields, since those are unique to each object. However, a Projectile should not have a time field, since the time is common across all objects, and is really a property of the world itself, not of the particle.

This means that you’ll need another class to represent the world, which contains the Projectiles and directs their motion. This class is often called an engine, a commander, or a world class. This class should take an input of Projectiles and parameters related to the forces involved, and it should run the loop and create the output. This should remove a lot of duplicated code.

The Main() method should be in a separate file, usually referred to as a *driver* (Visual Studio names the file Program.cs and the class the same name as the project). You should have separate functions for each level, and each level should create a world class, put in the appropriate parameters and projectiles, and run it. Thus, the driver should be short and should have no calculations in it, in general – it should just have input parameters (constants and initial values) and basic function calls.

Once you’re done with this, submit your Projectile and world classes to Google Classroom. Then get ready for the next assignment.

class World

{

public World(double time, double c, double g\_accel, List<Projectile> projectiles)

{

Time = time;

C = c;

G\_Accel = g\_accel;

Projectiles = projectiles;

}

public double Time { get; set; }

public double C { get; set; }

public double G\_Accel { get; set; }

public List<Projectile> Projectiles { get; set; }

public void ApplyForcesToProjectiles()

{

foreach (Projectile proj in Projectiles)

{

var f\_gravity = new Vector(0, 0, G\_Accel);

var f\_air = -C \* proj.Velocity.Magnitude \* proj.Velocity.Magnitude \* Vector.Normalize(proj.Velocity);

var f\_net = f\_gravity + f\_air;

proj.ApplyForce(f\_net);

}

}

public void Tick(double dt)

{

Time += dt;

ApplyForcesToProjectiles();

foreach (Projectile proj in Projectiles)

{

proj.Move(dt);

}

}

}

class Projectile

{

public Projectile(Vector pos, Vector velo, Vector accel, double mass)

{

Position = pos;

Velocity = velo;

Acceleration = accel;

Mass = mass;

}

public Projectile(double mass)

{

Position = Velocity = Acceleration = new Vector(0, 0, 0);

Mass = mass;

}

public Vector Position { get; set; }

public Vector Velocity { get; set; }

public Vector Acceleration { get; set; }

public double Mass { get; }

public void Move(double dt)

{

UpdateVelocity(dt);

UpdatePosition(dt);

}

public void ApplyForce(Vector force)

{

Acceleration = force / Mass;

}

private void UpdatePosition(double dt)

{

Position += Velocity \* dt;

}

private void UpdateVelocity(double dt)

{

Velocity += Acceleration \* dt;

}

}