Applied Physics - Practical SFML 2

To be completed at the latest by 1st November

**Purpose:** To model the motion of a projectile in 2-dimensions, using SFML.

**To Do:**

1. Open a new project named “Practical2” and create a circle on the ground. You could use

your version of “Practical 1” as a starting point. Use a new variable float pixelsToMeters and set it to a value for example 20. Multiply gravity by this variable. Change the size of the circle to a larger size.

2. When the space-key is hit by the user, make the circle move along the x-y plane as a projectile would when acting under gravity. The equations for the motion of a projectile are given below. **(30% Marks)**

3. Each time the R-key is hit the circle should be returned to its original position and the projectile can be fired again. **(10% Marks)**

4. Allow the user to increase/decrease the angle of projection by hitting the Y-key/U-key and to increase/decrease the velocity by hitting the V-key/B-Key. Output these values.

**(20% Marks)**

5. When the circle hits the ground the first time output the maximum height reached, the total time in the air and the distance moved along the x-direction. **(20% Marks)**

6. Amend the program so that the circle bounces up again and follows a new projectile path each time it hits the ground. **(20% Marks)**

**Some theory from physics:**

Since the acceleration is constant, the only equations from physics needed here are:

v = u + a\*t and s = s0 + u\*t + 0.5\*a\*t2.

We will apply these as vector equations using the SFML Vector class.

At each frame update, we will simply move the circle to the next position using the below 2 lines in **bold**

The following is an update method to calculate the position from one frame to the next:

The values: m, g, and the starting values and will be variables in your

// accel is simply gravity = (0, 9.81)

//incTime is the time elapsed between frames.

**m\_position = m\_position + m\_velocity\* incTime + 0.5f\*accel\*( incTime)2**

**m\_velocity = m\_velocity + accel\* incTime**

On a diagram:

(pos.x, pos.y, 0) at time t

(pos.x, pos.y, 0), at time t + Δt

Initial speed,U

Angle α

To allow the circle to bounce after it hits the ground apply the coefficient of restitution, e, in the y-direction only so that immediately after hitting the ground, the next x-value for the position is calculated as before but the y-velocity changes direction (from negative to positive) and is multiplied by e, which will in turn make the position of the bullet bounce back up off the ground.

So just after hitting the ground, use

m\_velocity.y = – e\*m\_velocity.y

to calculate the next y-value for the velocity and leave everything else as before.

Note: e is the coefficient of restitution and takes values from 0 to 1, (0 = no bounce, and 1 means perfect elasticity so bounces to same height again)

Try different values for the coefficient e until it looks realistic.