A Basic Projectile Motion Game Using Projectile Physics

Alexander van Lomwel

2022

Contents

1 Introduction

This short paper serves as an overview of a basic projectile motion game I have created to expand my python knowledge. The game is coded using basic equations in physics which describe the motion of a friction-less object under the action of gravity. I used the `VPython` package to create a basic visual for the game.
The goal of the game is simple. A projectile is fired at a speed and angle, chosen by the user, towards a target. To win the game, the target must topple over. If the projectile misses the target or fails to topple over the target, the user loses.

1.1 Theory

Before we explore the code, we must define the equations used in this project. In 2 dimensional space, the position of the projectile $\mathbf{r}$ can be expressed using the following equations.

$$r_x = x_0 + v_0 t \cos\theta \tag{1}$$

$$r_y = y_0 + v_0 t \sin\theta - \frac{gt^2}{2} \tag{2}$$

x_0, y_0 are the initial x, y positions of the projectile. v_0 is the initial velocity and θ is the angle at which the projectile is fired at. The gravitational acceleration g only affects motion in the y direction as it has no contribution to the x direction, i.e. perpendicular. We can additionally define the momentum of a projectile of mass m.

$$\mathbf{p} = m\frac{d\mathbf{r}}{dt} \tag{3}$$

And in its respective directions:

$$p_x = mv_0 \cos\theta \tag{4}$$

$$p_y = mv_0 \sin\theta - mgt \tag{5}$$

If the projectile makes contact with the target, i.e. r_y coincides with the height of the target, the user won't necessarily win. Assuming the target cannot slide on the ground and that the centre of mass is in the geometric centre of the target (constant density), we can define some criteria for the target to topple over for the user to win. We first consider the torque which restores the target to its upright position.

$$\boldsymbol{\tau}_{restoring} = \mathbf{F}_{grav} \times \mathbf{d}_r \tag{6}$$

If we define the width of the target as w, the centre of mass will be at $w/2$ from our assumptions. Hence, $\mathbf{d}_r = w/2$. Therefore, we obtain (where $\phi = \pi/2$ as the force and position vectors are normal to each other):

$$|\boldsymbol{\tau}_{restoring}| = -mg\frac{w}{2}\sin\phi = -mg\frac{w}{2} \tag{7}$$

We now consider the applied torque by the collision.

$$\boldsymbol{\tau}_{applied} = \mathbf{F}_{applied} \times \mathbf{d}_a \tag{8}$$

In this case, $\mathbf{d}_a$ is a vector from the rotation pivot to the point of impact. For $\mathbf{F}_{applied}$ we can consider the impulse.

$$\mathbf{I} = \mathbf{F}_{applied}\Delta t = \Delta\mathbf{p} \tag{9}$$

$$\mathbf{F}_{applied} = \frac{\Delta \mathbf{p}}{\Delta t} \tag{10}$$

We can make a simplifying assumption that **all** of the momentum is transferred to the target. Thus,

$$\mathbf{F}_{applied} = \frac{\mathbf{p}_{proj}}{\Delta t} \tag{11}$$

And we finally arrive at our toppling criteria.

$$|\boldsymbol{\tau}_{applied}| > |\boldsymbol{\tau}_{restoring}| \tag{12}$$

1.2 Assumptions

I have already mentioned a few assumptions in the equations, but here we will formally define everything I will assume.

- No energy is lost during collisions, i.e. elastic.
- The projectile is brought completely to rest so all the momentum is transferred to the target.
- There is a finite contact time between the projectile and the target which is a fixed parameter $\Delta t_{collision}$ for **all** collisions.
- The projectile remains intact after the collision, even when speeds are very high.
- The surface at which the target sits on has infinite friction meaning that rotation pivot stays fixed and does not slide.

2 The Code

The code will follow a basic structure as follows.

- User inputs v_0 and θ.
- Launch projectile.
- Does it hit the target? If **NO**, user loses and tries again. If **YES**, calculate the momentum, impulse, applied force and torque.
- Will the target topple? If **NO**, user loses and tries again. If **YES**, the user wins the game.

For this code, we need to import the following libraries. VPython is not automatically included in Python and must be installed from the Anaconda Prompt.

```
import numpy as np
from vpython import sphere, color, rate, canvas, vector, curve, label,
box, cross, mag, random, arrow
```

And to setup our environment/scene:

```
scene = canvas(width=640, height=480, center=vector(8,5,0),range=8)
ground = curve(pos=[(0,0,0),(16,0,0)],color=color.green)
```

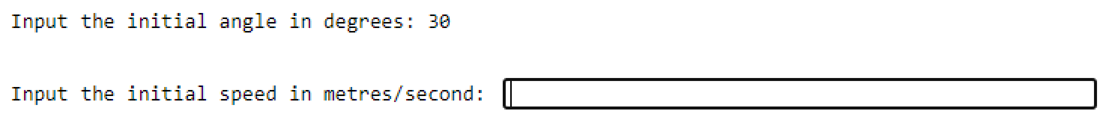
Now we must assign some values to the constants in the above equations. The gravitational acceleration g will be assumed to be the standard 9.8m/s^2. The ball will be fired from an x value of 0m (x_0), whereas the y_0 value will be a randomly generated number from 0m to 0.5m. The animation will keep running provided the y coordinate of the projectile is greater or equal to zero (i.e. until it touches the ground) or until the projectile makes contact with the target. The projectile has a mass of 0.1kg and the target has a mass of 100kg. The width of the target is 0.5m and hence we can introduce a criteria for which the x value corresponds with x_{box}:

$$x_{box} - 0.251 \leq x \leq x_{box} + 0.251$$

where we introduce a 0.001m leniency because the projectile itself has dimensions. x_{box} is the x position of the target. The height of the target is 2m, therefore we further require that $y < 2$ in order for contact. The above can be summarised in the code below.

```
while yt >= 0 and xt <= (xbox - 0.25) or 1.9 <= yt <= 2.1 and (xbox - 0.25)
<= xt <= (xbox + 0.25):
    t = t + dt
    rate(2500)
    xt = x0 + v0*t*np.cos(theta)
    yt = y0 + v0*t*np.sin(theta) - (0.5*g*t*t)
    movingball.pos = vector(xt,yt,0)
    momentumyplane = ballmass*v0*np.sin(theta) - ballmass*g*t
    vector1.pos = movingball.pos
    vector1.axis = vector(momentumxplane, momentumyplane, 0)
    label.pos = movingball.pos

if yt<2 and (xbox - 0.251) <= xt <= (xbox+0.251):
    print("You have hit the target!")
    #Printing data about the flight of the object.
    print()
    print("The time of flight of the ball was: {0:0.2f}".format(t, t),
    "seconds (2 DP).")
    print()
    print("The vertical displacement was: {0:0.2f}".format(yt - yplatform,
    yt - yplatform), "metres (2 DP).")
    print()
    print("The horizontal displacement was: {0:0.2f}".format(xt, xt),
    "metres (2 DP)")
else:
    while yt >= 0:
        t = t + dt
        rate(2500)
        xt = x0 + v0*t*np.cos(theta)
        yt = y0 + v0*t*np.sin(theta) - (0.5*g*t*t)
        movingball.pos = vector(xt,yt,0)
        momentumyplane = ballmass*v0*np.sin(theta) - ballmass*g*t
        vector1.pos = movingball.pos
        vector1.axis = vector(momentumxplane, momentumyplane, 0)
```

The first section has the requirements for the loop to keep "looping". The second section determines whether the projectile has made contact with the target and the last section is when the projectile missed the target. If the user is successful in hitting the target, they will receive the message: `You have hit the target!`. If the user fails: `You have missed the target!`. The user is also able to enter their desired initial speed and angle, as shown below:



```
Input the initial angle in degrees: 30

Input the initial speed in metres/second:
```

Figure 1: The user is able to choose their initial speed v_0 in units m/s and their initial angle θ in units degrees.

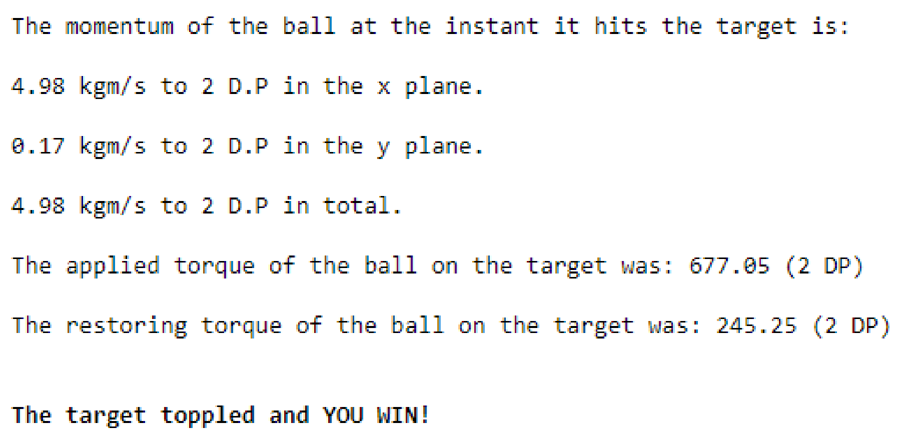
If the user enters a speed/angle combination that does not hit the target, the user fails and can re-enter different values. If the target has been hit, the code now determines whether the target has been toppled.

2.1 Toppling

To determine if the target topples or not, we use equation 12. The code below shows the calculation.

```
#In the x plane.
ballmass = 0.1
momentumxplane = ballmass*v0*np.cos(theta)

#In the y plane.
momentumyplane = ballmass*v0*np.sin(theta) - ballmass*g*t

#Total momentum.
totalmomentum = mag(vector(momentumxplane, momentumyplane, 0))

#Calculating the restoring torque.
targetmass = 100
restoringtorque = cross(vector(0, -9.81*targetmass, 0), vector(0.25, 0, 0))

#Calculating the applied force of the ball on the target.
forceapplied = totalmomentum/0.01
forceappliedvector = 100*vector(momentumxplane, momentumyplane, 0)

#Calculating the applied clockwise torque of the ball on the target.
torqueapplied = cross(forceappliedvector, vector(0, yt, 0))

if mag(torqueapplied) > mag(restoringtorque):
    print("\033[1m")
    print("The target toppled and YOU WIN!")
    print("\033[om")
else:
    print("\033[1m")
    print("You LOSE.")
    print("\033[0m")
```

Let's now consider an example. The user enters $v_0 = 17$ m/s and $\theta = 10$ degrees. The projectile makes contact with the target and torque calculations are made. $|\boldsymbol{\tau}_{applied}|$ was found to be 148.66 Nm and $|\boldsymbol{\tau}_{restored}|$ was found to be 245.25 Nm. In this case $|\boldsymbol{\tau}_{applied}| \ngtr |\boldsymbol{\tau}_{restored}|$ and the target *does not* topple. The user now instead tries $v_0 = 50$ m/s and $\theta = 5$ degrees. We find that $|\boldsymbol{\tau}_{applied}| = 677.05$ Nm and $|\boldsymbol{\tau}_{restoring}| = 245.25$ Nm. We see now that equation 12 is satisfied and the user wins. The user is given the following information:



```
The momentum of the ball at the instant it hits the target is:

4.98 kgm/s to 2 D.P in the x plane.

0.17 kgm/s to 2 D.P in the y plane.

4.98 kgm/s to 2 D.P in total.

The applied torque of the ball on the target was: 677.05 (2 DP)

The restoring torque of the ball on the target was: 245.25 (2 DP)


The target toppled and YOU WIN!
```

Figure 2: The print statements concluding the calculations and the outcome of the game.

The figures below show screenshots of the animation that VPython generates once the user info their chosen speed and angle.

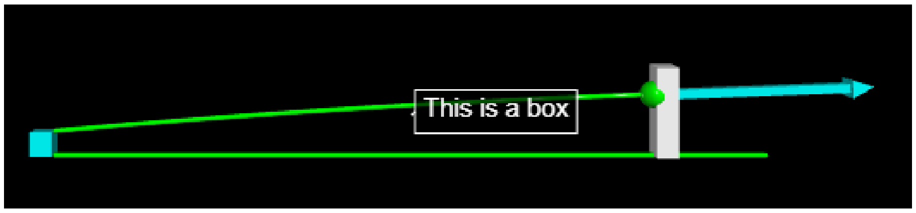


Figure 3: A successful launch.

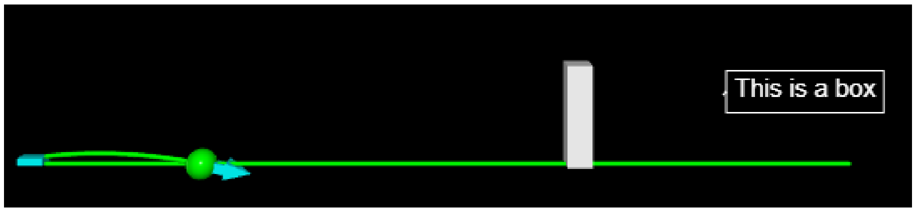


Figure 4: An unsuccessful launch.

3 Limitations

This is a very simple game that demonstrates projectile motion and might look similar to the base code of games like angry birds which also follows the basics of projectile physics. There are many non-realistic assumptions I have made to keep the code as simple as possible. In the realistic scenario, the air resistance would certainly affect the path and speed of the projectile, especially at high launch speeds. The projectile would likely be shattered when fired at high speeds. The target is 1000 times heavier than the projectile and the change of momentum is so great that the force on the projectile would almost certainly shatter it.
There are also no boundaries to the input speed. The user could enter speeds unrealistically high, even greater than the speed of light. If the projectile misses the target, the game will reset when the projectile lands. At extremely high speeds, the user might have to wait multiple hours for the projectile to land and hence would be forced to restart the game.

4 Future Work

If time permits, I would like to implement an animation of the target toppling over when the applied torque is large enough. There should also be a ”penalty” to the user when they select high speeds, for example increased air resistance or more uncertainty in the path curvature. The speed should also be capped at 75 m/s. An interesting addition would be including obstacles between the launch platform and the target. This would mean that only a small range of θ values would reach the target for a given speed and the user would have to trial and error.

5 Concluding Remarks

Overall, this was an enjoyable project and was it interesting to get more comfortable with visual python which is still very new to me. The code itself was relatively simple to write and any errors were debugged almost instantly. This code and theory could be used for many different games that require the launching of projectiles. Please find the link to the code below.

`https://github.com/alexvanlomwel/A-projectile-motion-game`