Billiards Simulation – PHY 494

Authors: Nat Soderberg, Nik O'Brien, Kit Fung
Affiliates: Arizona State University Physics Dept.

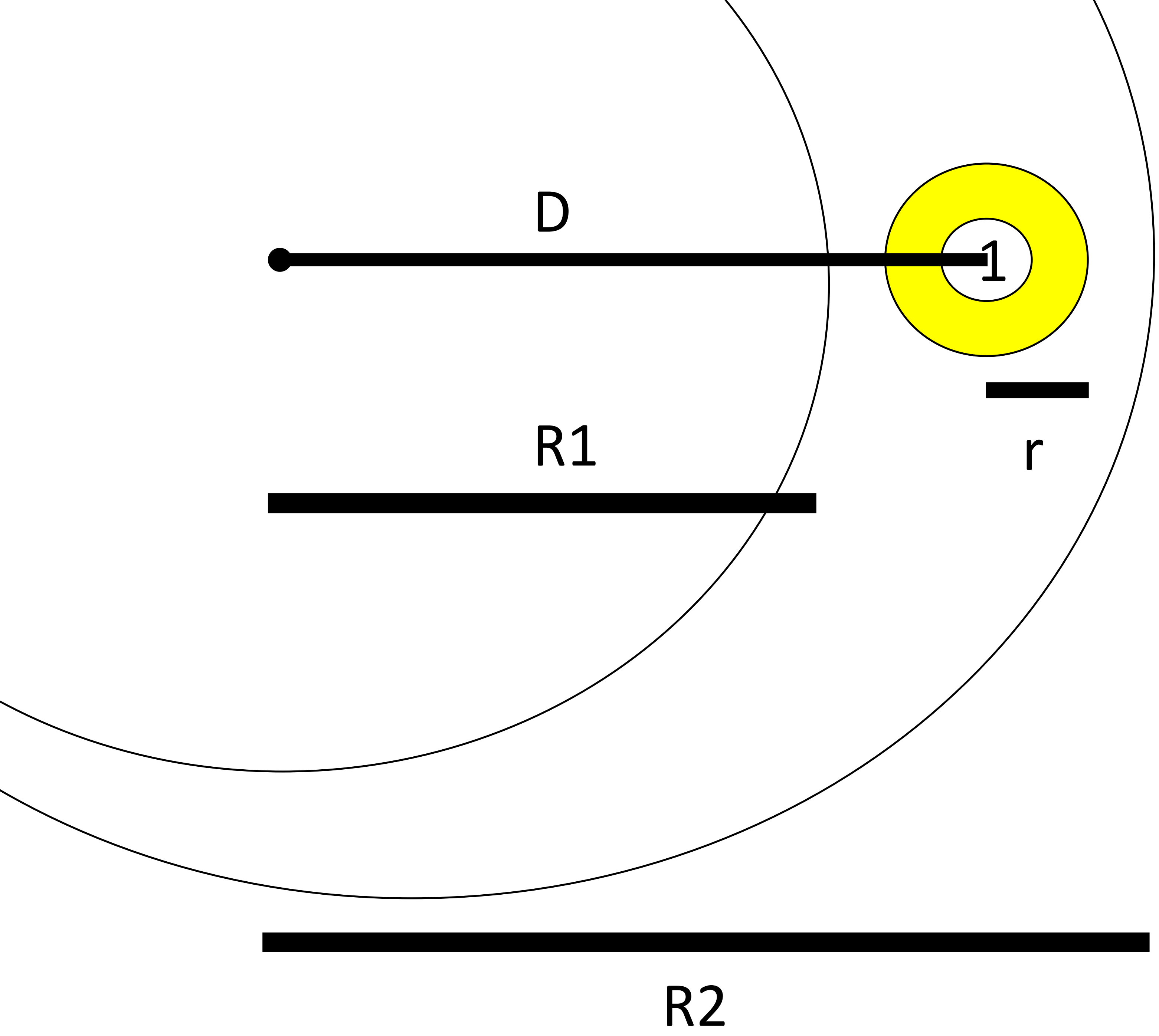
Background

While many people enjoy playing the game of pool, even the best fall prey to human error. Therefore this Python program was created to analyze the physics and geometry of a 2D billiards shot, in order to provide the player with the angle to make their shot most efficient.

In doing so the program and its writers needed to account for momentum transfers through perfectly elastic collisions and make logical simplifications.

Equations to be Solved:

- $m_1 v_{1i} + m_2 v_{2i} = m_1 v_{1f} + m_2 v_{2f}$
- $v_{1x} = v_2 \cos(\theta_2 - \varphi)\cos(\varphi) + v_1 \sin(\theta_1 - \varphi)\cos(\varphi + 0.5\pi)$
- $v_{1y} = v_2 \cos(\theta_2 - \varphi)\sin(\varphi) + v_1 \sin(\theta_1 - \varphi)\sin(\varphi + 0.5\pi)$
- $\mu_{friction} = 0$



Checking for Removal - Visual

```
for ball_i in range balls:
        if D + r < Ri:
                remove(ball_i)
```

So from this visual Ball 1 would be removed for R2, however if we decrease the hole radius size to R1 it would stay on the table

Summary

Shot efficiency in this context can be described as hitting in the most billiard balls in the shortest amount of time. For the performed simulations, the program was able to successfully run and yield the most efficient shot angles given the necessary input parameters.

Of course this code was written with many simplifications such as having fewer balls on the table, a nice symmetrical setup, and no friction. However in the future this code can be expanded upon to include more realistic features like friction, and multiple cue shots with variable speed and spin to simulate an actual game of pool.

Code and References

Code for this project can be found at:
https://github.com/ASU-CompMethodsPhysics-PHY494/final-billiard-simulation

References:

- http://williamecraver.wix.com/elastic-equations
- https://nickcharlton.net/posts/drawing-animating-shapes-matplotlib.html
- http://formulas.tutorvista.com/physics/elastic-collision-formula.html
- https://en.wikipedia.org/wiki/Elastic_collision

This team would like to acknowledge Ian Kenney for his help in office hours teaching python objects and Dr. Beckstein for his involvement in teaching the PHY494 course and for always being available for questions.

Methods [0]

One of the first steps to starting the program was to find a way to define the balls and table for easy calls into the program functions.

Class Objects:

- Balls – includes Positions and Velocities
- Table – includes Dimensions, Ball Mass & Radii and Hole Radii

Snippet from Table Object Initialization

```
def __init__(self, xdims, ydims, ball_mass, ball_radius, hole_radius):
    self.xdims = xdims
    self.ydims = ydims
    self.x_left = -.5*xdims
    self.x_right = .5*xdims
    self.y_bottom = -.5*ydims
    self.y_top = .5*ydims
    self.balls = []
    self.ball_mass = ball_mass
    self.ball_radius = ball_radius

    self.hole_positions = np.array([[self.x_left, self.y_bottom], [self.x_right, self.y_bottom], \
                          [self.x_left, 0], [self.x_right, 0], \
                          [self.x_left, self.y_top], [self.x_right, self.y_top]])
    self.hole_radius = hole_radius
    self.num_holes = len(self.hole_positions.T[0])
```

Class Benefits:

Using Classes makes the code easier to call, understand and organize.

Without them, a lot of independent variables would be needed.

Methods [1]

Once everything was defined the rest of the program was written. This consisted of various functions that, when combined with the objects, ran a loop for the duration of the simulation.

Path of the Program

Initialize Balls, Table, Break Speed and Shot Angle

These are the steps taken before the loop iterations begin

Create Empty List for Positions at each Step

Choose Simulation Run Time

Color Coding:
Represents Main Path
Represents Secondary Path
Represents Alternative Path

Methods [2]

Check if time is multiple of plot interval time

Take the positions at this time for plot

Move the balls based on current velocities and dt

Check for removal, remove if needed

Check for collisions, run the collision function if needed

Collect change in velocities based on the loop and repeat

Stop the Simulation if

- All balls are removed
- Or if sum of speeds is zero up to 10 decimal places

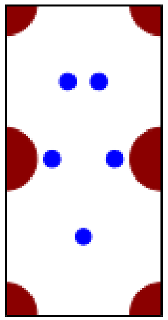
Methods [3]

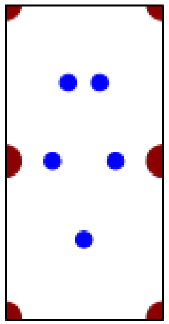
After the loop runs, data for the simulation can be collected

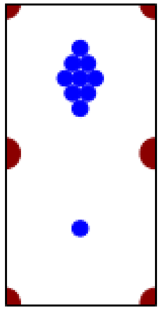
Relevant Data Includes:

- Total Simulation Time (t_steps * dt)
- Number of Balls left on the Table (length of table-balls array)
- Plot of the Balls' Positions taken on every multiple of the time interval (which was previously defined)

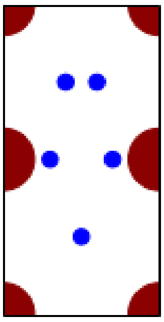
Simulations ran in this project consisted of the following setups:



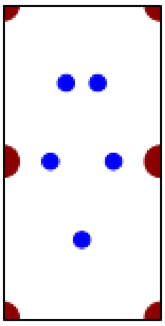




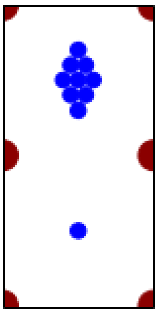
Results



Ball	Ball Size	Pocket Size	Break Force
5	5	20	130
Break Angle (°)	Time (s)	Time Steps	Balls Remaining
0	N/A	25000	5
10	1.74	87	4
20	8.32	416	3
30	109.88	5494	0
40	0.42	21	4
50	0.42	21	4
60	6.56	328	3
70	28.6	1430	1
75	23.04	1152	0
80	8.88	444	1
90	N/A	25000	5



Ball	Ball Size	Pocket Size	Break Force
5	5	10	130
Break Angle (°)	Time (s)	Time Steps	Balls Remaining
0	N/A	25000	5
10	N/A	25000	1
20	N/A	25000	1
30	N/A	25000	1
40	N/A	25000	1
50	407.8	20390	0
60	369.24	18462	0
70	N/A	25000	1
80	N/A	25000	1
90	N/A	25000	5



Ball	Ball Size	Pocket Size	Break Force
10	5	10	130
Break Angle (°)	Time (s)	Time Steps	Balls Remaining
0	N/A	50000	10
10	33.38	1669	0
20	1.12	56	9
30	N/A	50000	1
40	228.78	11439	0
50	359.36	17968	0
60	228.5	11425	0
70	N/A	50000	1
80	244.3	12215	0
90	N/A	50000	10