No-Slip BILLIARDS WITH GRAVITY

A PREPRINT

Jurgen Xhafaj
Department of Mathematics
Tarleton State University
Stephenville, TX 76402
jurgen.xhafaj@go.tarleton.edu

Dr. Christopher Cox
Department of Mathematics
Tarleton State University
Stephenville, TX 76402
clcox@tarleton.edu

May 13, 2019

ABSTRACT

We consider a relatively new type of billiards that has attracted interest in the recent years in the mathematics community: no-slip billiards. Our project is focused on a slim vertical strip where a theoretical point-mass object bounces from one pole to the other. The goal of this project is to visually prove that the simulation will result in a bounded section of this space, even after a large number of bounces.

Keywords Billiards · No-slip · Gravity · Simulation

1 Introduction

Firstly, it is best to define what a no-slip billiards is, and how it differs from another (more well-known) type; the specular billiards. While specular billiards only consider the linear movement of a ball, we are adding a third component in no-slip billiards: the angular momentum. For our specific project, we are involving gravity as a factor in the object's movement. Below, we illustrate a sample no-slip collision.

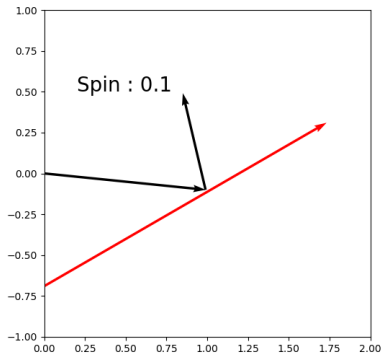

Figure 1: No-Slip Collision

2 Implementation Process

To make the mathematical simulation possible, we used Jupyter Notebook and collaboration was facilitated through CoCalc. The programming language of choice was Python 3, due to its simplicity and flexibility. Below, we explain the details of our simulation.

2.1 Python Packages

We used the following Python packages when coding:

- NumPy - provides a comprehensive set of tools for mathematical calculations; mainly linear algebra operations
- MatPlotLib - packed with options to plot points, lines, and shapes of all types and colors
- WinSound - produces a sound when algorithm finishes running (especially useful in long iterations)

2.1.1 Storing Information

We set up our code so that it can be easily edited and re-purposed for other experiments. Plotting a continuous stream of points (via PyPlot, an interface for MatPlotLib) gives us a clear visual clue for our algorithm's performance. This algorithm was adapted from our initial rectangular billiards implementation, therefore it has a tall rectangle, where its sides AB and CD are the "poles" (we keep this notation throughout the paper).

The movement vector of our point includes three main components: the spin, the x-velocity, and the y-velocity. For additional purposes, such as plotting the phase portraits of our simulation, we added another matrix that collects information as the point bounces on poles. This matrix contains the point's movement components, as well as an additional two parameters for the phase portrait. The information gathered from this matrix was often used to evaluate of the validity of the object's trajectory.

2.1.2 Bouncing Mechanism

The bouncing mechanism owes much to Dr. Cox's previous work with no-slip billiards.«REFERENCE» To involve the angular momentum, we use a parameter called gamma (γ), which is an indicator of the mass distribution of our object. We used uniform mass-distribution, with $\gamma = \frac{1}{\sqrt{2}}$. The following T_{ns} matrix is what makes no-slip collisions possible:

$$T_{ns} = \begin{bmatrix} \frac{\gamma^2-1}{1+\gamma^2} & \frac{-2\cdot\gamma}{1+\gamma^2} & 0 \\ \frac{-2\cdot\gamma}{1+\gamma^2} & \frac{1-\gamma^2}{1+\gamma^2} & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

To implement the rotation of our particle upon bouncing, we used the following two matrices, called R_1 and R_2 :

$$R_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad ; \quad R_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

This is how the incoming vector is transformed into an outgoing vector: $v_{out} = R_1 \times T_{ns} \times R_2 \times v_{in}$

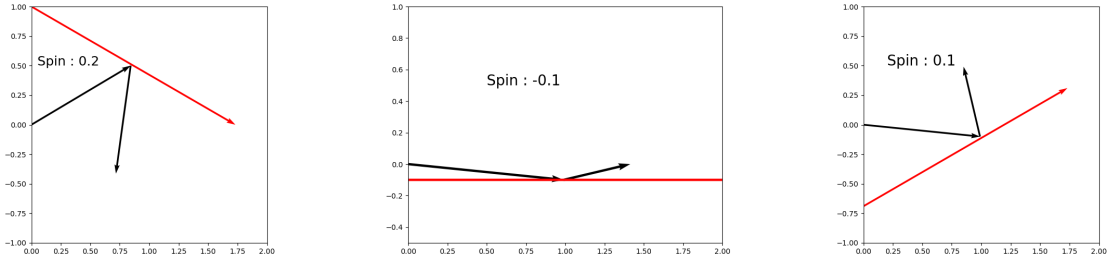

Figure 2: Setting up no-slip collisions for 3 line scenarios

3 Issues faced during the experimentation

3.1 Picking a visualization package

Our original choice of package was $PyGame$, a Python package that specializes in making games and animations. Such a package has proven to be very effective with rectangular specular billiards, as it shows the moving particle in motion. It also offers the option for the particle to leave a trace. However, PyGame was not enough once we moved to other shapes of billiards (circular, triangular, etc.). The first issues were noticed when working with specular triangular billiards. The incoming and outgoing angles of our particle were simply not the same. After some troubleshooting, we found the issue to be the rounding that PyGame does when projecting the movement data onto the screen. Due to this serious flaw on our initial package pick, we had to switch to another package, the popular MatPlotLib. While the latter does not support real time animation by default, its advantage is the powerful computing capabilities while maintaining a very high level of accuracy.

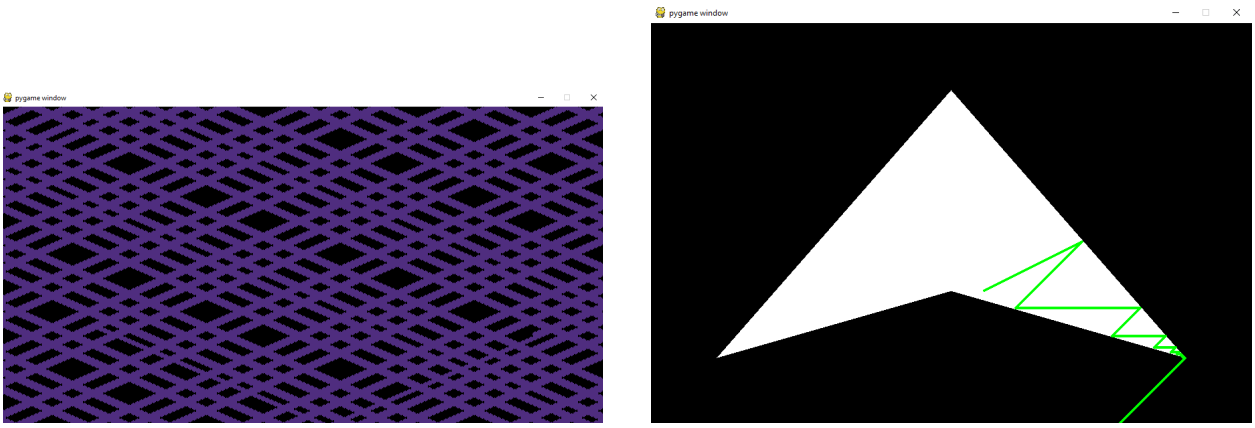

Figure 3: PyGame for rectangular (specular) and arrow (specular) billiards

3.2 Implementing gravity

The main goal of this project was to introduce gravity to no-slip billiards, and we started by adding a constant drop to our moving particle on every step of the way. Upon further analysis, we noticed that the particle was gradually dropping down. The underlying issue was that our particle's y-displacement was overall negative, which indicated an imbalance in how the energy was distributed. Adding an energy exchange mechanism on our poles helped solve the issue. We decided that y-velocity and spin would exchange energy upon the particle hitting the pole, and our results were significantly better.

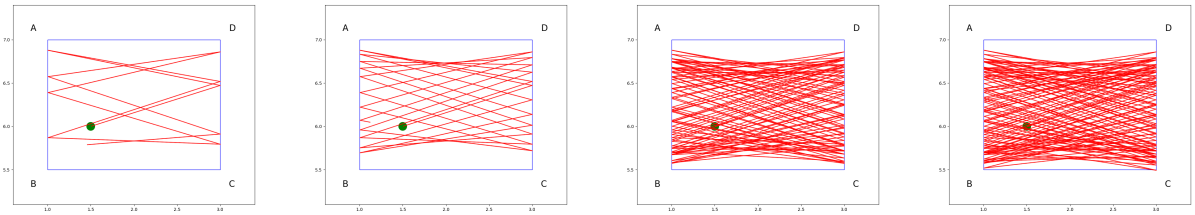

Figure 4: Gravity wins without energy exchange at poles

3.3 Adjusting gravity impact

This particularly delicate issue was one that followed us throughout the whole project. Since the beginning, scaling the image and deciding a proper value for gravity took a lot of trial-and-error. As we moved on with the project, we

implemented code that made the drawing proportions and gravity to be related to each other. Another remedy to help our case was changing the dimensions of our elongated rectangle (poles), to better visualise the impact of gravity. During our final stage of the project, we made the code run by collisions, instead of blits plotted, and that was a huge step ahead when detecting gravity impact. During one particular try, a "large" (0.0004) value of gravity meant the code would run for over 2 minutes to go from 2^{nd} to 3^{rd} collision, as compared to only 5 seconds with a gravity value of 0.0003.

* Note that our implementation by blits did not need a time variable since one blit was essentially a unit of time.

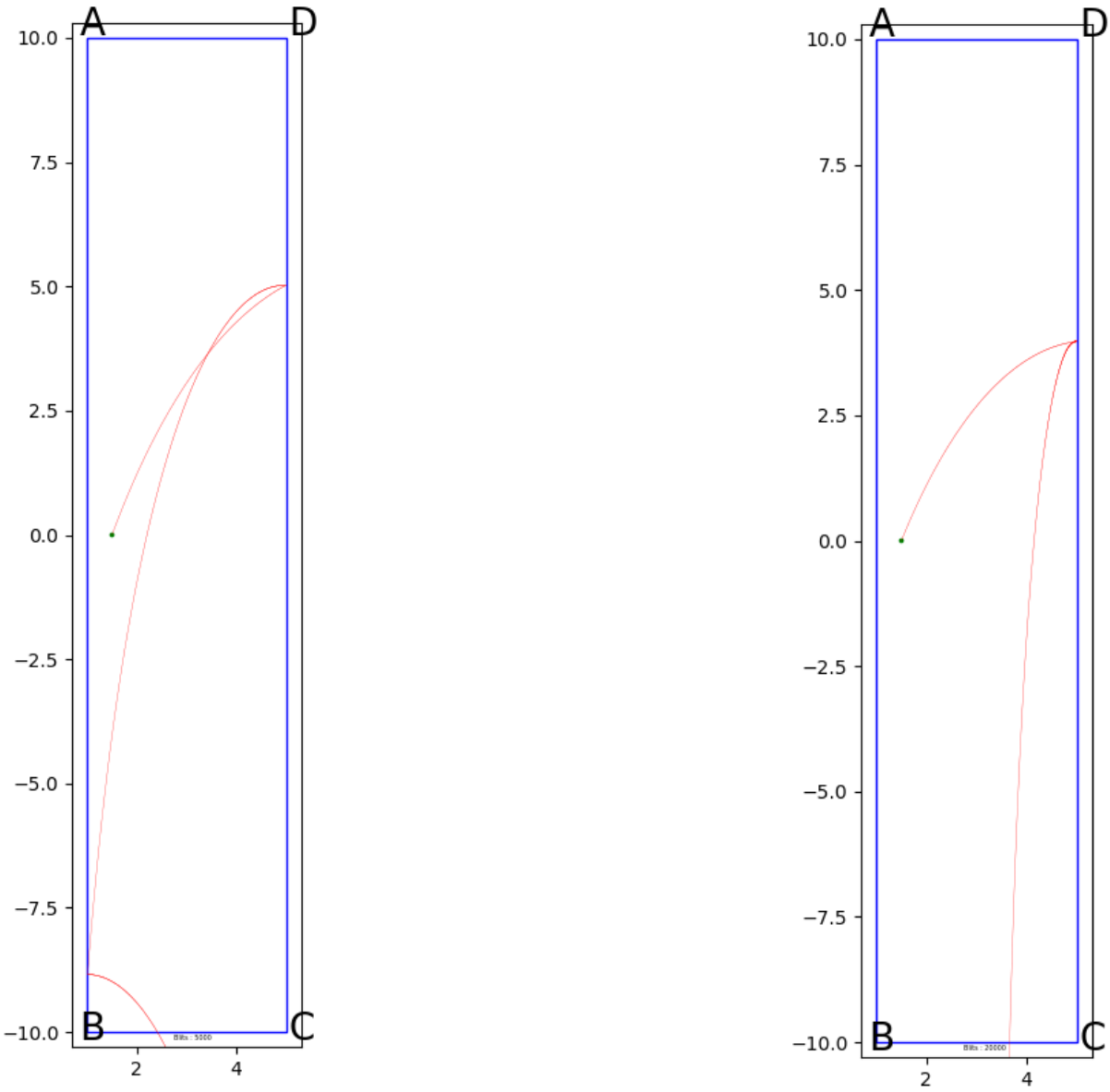

Figure 5: Gravity parameter 0.0003 vs 0.0004

3.4 Adjusting particle speed

Due to the nature of our experiment, the poles had to be rather close to each other for us to be able to observe the particle's behavior after a considerable number of collisions. For that reason, we scaled down our vector by an initial

factor of 0.01. This helped the particle display its behavior more accurately and considerably faster than moving to the right by one whole unit at a time. While this implementation worked throughout almost all of our project, we noticed in our last week that our movement vector was slightly smaller than what it should have been. Upon changing the scaling coefficient to 0.02, we obtained the results that we were looking for in this project.

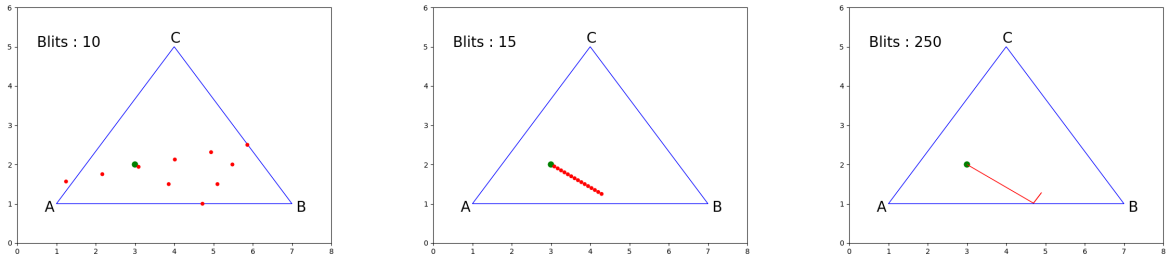

Figure 6: Scaling down the movement vector

3.5 Labels

This feature was implemented into our project to be able to easily distinguish the poles, as well as annotating the number of blits, and later on the number of collisions too. After the gravity was successfully implemented, we adjusted the parameters of our display and made each label's size and position dependent on the overall rectangle size, to obtain a better appearance of our final image. Below we present a "timeline" of our annotation evolution.

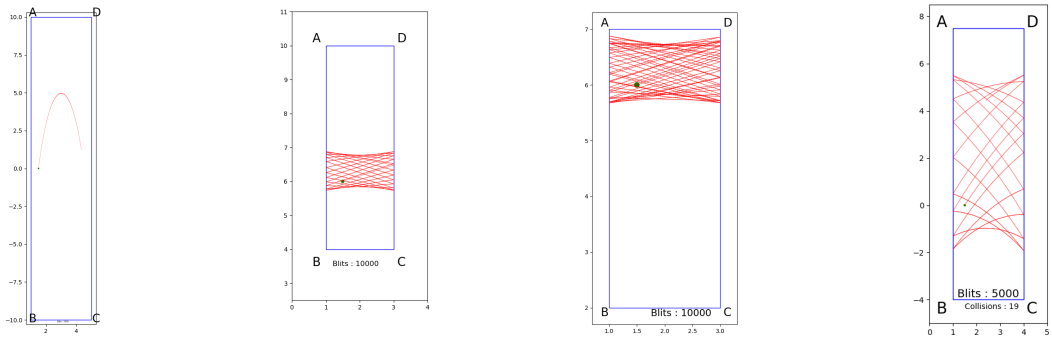

Figure 7: The evolution of graphics in our project

The entirety of our project can be found at:

`https://github.com/jurgenxhafaj/Billiards`

4 Conclusions

After numerous simulations, we managed to show that in our implementation of gravity billiards, the path of our point-mass object would be bounded. Below we show some screenshots of our finished product. We added labels for the number of collisions and blits to help the reader better understand the progress flow.
We also noticed that the upper and lower bound are not the same curvy shape (arc), and this is a direct result of gravity. When we changed gravity values, we got different arcs.

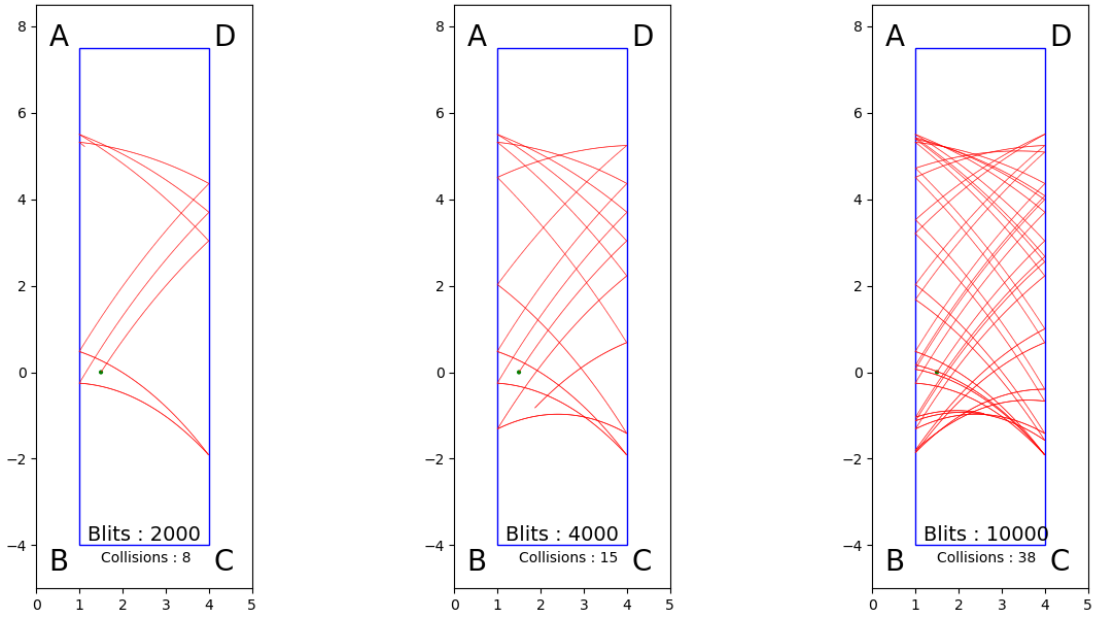

Figure 8: Up to 10000 blits

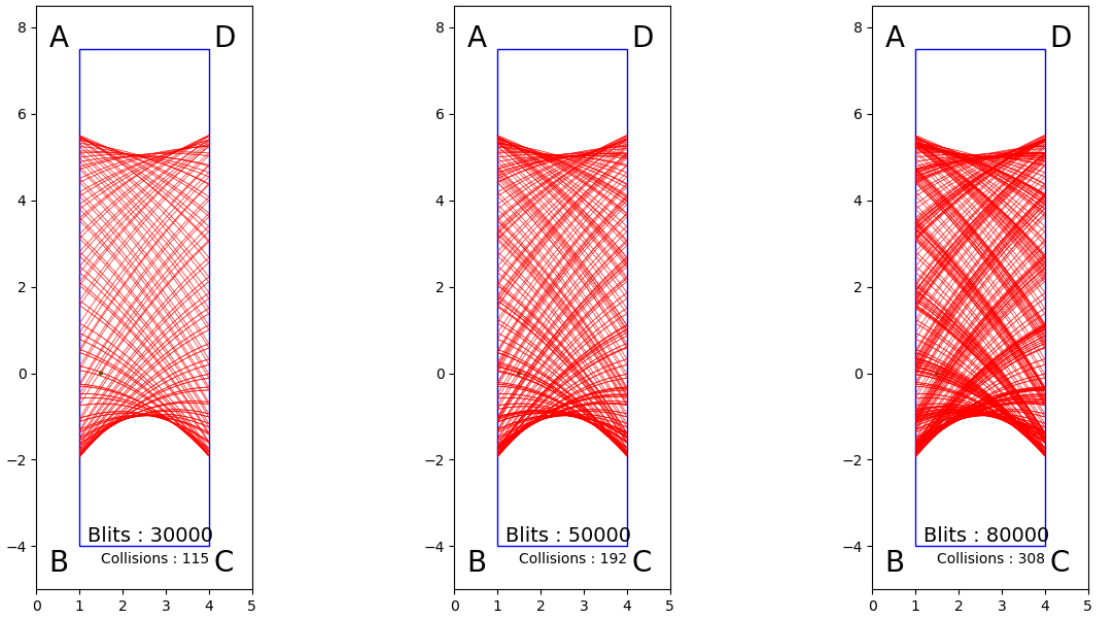

Figure 9: Way to 100k blits

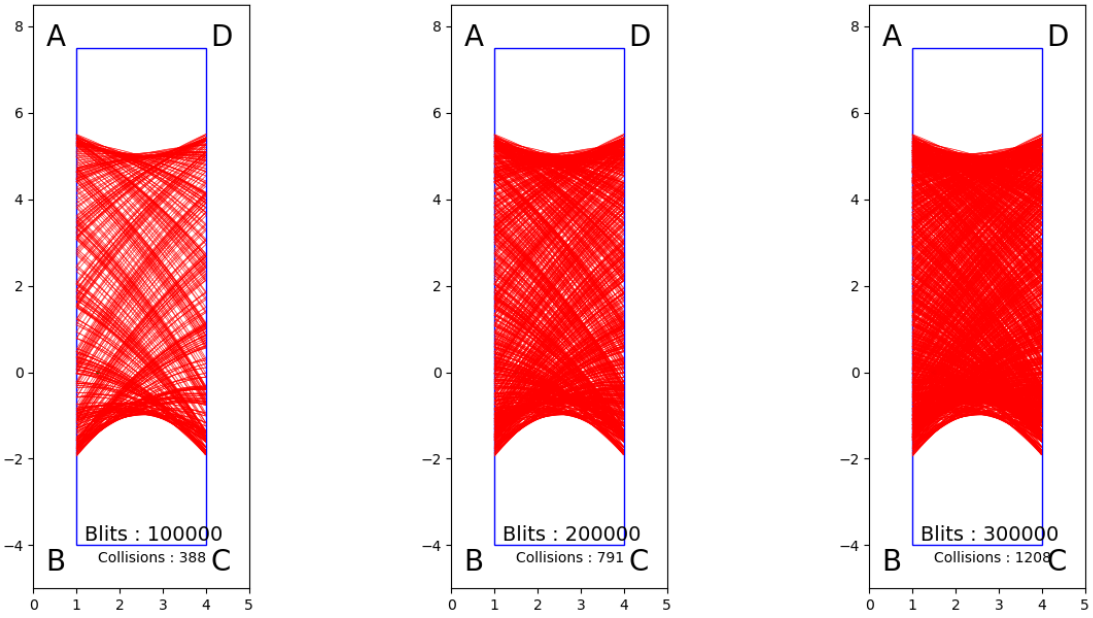

Figure 10: 100k blits and above

5 Future Work

After the successful implementation of gravity no-slip billiards, Dr. Cox will continue the work on this topic during the summer to investigate further and come up with new and interesting results.

References

[1] Christopher Cox and Renato Feres. No-slip billiards in dimension two, in Dynamical systems, ergodic theory, and probability: in memory of Kolya Chernov, 91–110, In *Contemporary Mathematics*, Amer. Math. Soc., Providence, RI, 2017.