Electrostatic Projectile Game Prospectus

Introduction

This prototype project, *Electrostatic Projectile Game*, is a game made using Python and intended to be played by physics and astronomy students. The goal of this game is for the users to deduce the location and charge of various point charges that are hidden across an XY-plane. In this game, the user will launch charged projectiles and follow their trajectories in order to see where these hidden charges were initially placed. The locations of these point charges are chosen by me and are designed to be not too easy or overly difficult for the players to guess the locations of.

Game Physics

As this game is targeted for physics and astronomy students, a very simple understanding of physics is required in order to play the game. The only knowledge a player must have before starting this game is that like charges repel, opposites attract, and the strength (force) of a charge on a projectile is dependent on the distance from that charge. Within the making of the game, the use of two major formulas is used in behind-the-scenes computation. The first, most recognizable formula is the formula for electrostatic force: $\frac{kQ}{r^2}$ This formula is directly proportional to charge and follows the inverse square law of distance from the charge. It should also be noted that the total force exerted on a charged projectile is the vector sum of all the individual forces if multiple charges are present. This formula denotes that the greater the magnitude of charge on a hidden point charge is going to result in a greater force being exerted on a charged projectile. Reversely, the greater the distance from the point charge, the less force being exerted on that projectile. This electrostatic force formula was used in the internal calculations of the movement of the projectiles. It should be noted that those with an elementary background in physics will know that an arbitrary charged projectile is always assumed to be of positive charge, unless stated otherwise. The second major formula used is that which gives the potential energy of the previously given force law: $\varphi(r) = kQln\left(\frac{r_0}{r}\right)$. This holds very similar properties to the force law; however, the distance is not related to the potential by an inverse square law. It is also different in that when multiple charges are present, the total potential is the scalar sum of the potentials due to each charge. This formula was used in the calculations and showing of the equipotential in the function reveal_potential().

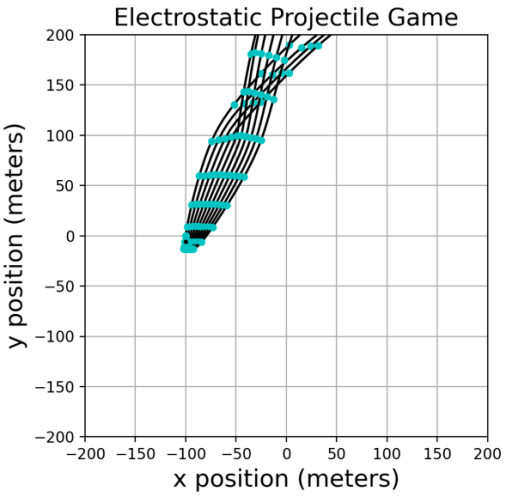
Implementation

In describing the implementation of the projectile game, I will first describe the simplest functions which serve a necessary but simple purpose. The clear function is used to create the initial plot where the game is played, the titles, and the XY-limits. These are all created using simple functions that can be self-explanatory by just looking at them, such as: ax.title(), ax.set_ylabel(), ax.set_xlim(), or ax.grid(). The next function is the play function. This function's

purpose is to allow the developer of the game to tweak initial values in order the make the game look more appealing or understandable to the player. The function allows the changing of the starting velocity, angle, and y-position of the projectile which is a lot faster to do using the play function than just making small adjustments and running and calling functions every time. The next function is the solve_it function. This is where the user will be inputting their answers into the game and seeing if they were correct or not. This function uses simple if and else statements and stores input integers by the user in order to decide what lines to print in response (whether the user input the correct answer). If the user gets all 4 questions correct, the function reveal_potential() is called. This function serves to show the sign and equipotential of the hidden charges after the user has made their guesses. This function makes use of several numpy function. The first is numpy linspace which created an array of evenly spaced values within the specified interval (-200 to 200 because that is where the game window is set). The numpy meshgrid function then takes the previously made linspace arrays and created a matrix of those values. Then, a blank 2-dimensional array is created using numpy zeros function which has the parameters of the lengths of both linspaces which make up the matrix. Then the XY domain is manually looped over, using 2 for loops for x and y and the calculation of the potential formula is called for each point in the empty matrix and meshgrid. Then, the contour function is used to draw the equipotential by calling the variables in the for loops. The last, most essential function is the plot_trajectory function. This does exactly what its name entails, plotting the trajectory of the projectile when influenced by the charges. This function defines a function named derivatives which is in terms of t and s. Within this function, the force in the x and y directions are defined, and a size 4 array is created. This array stores the velocity in the x, the acceleration (force) in the x, the velocity in the y, and the acceleration in the y. Outside of this, the initial velocity in the x and y are defined based on simple trigonometric identities in terms of the angle. The start and stop times are created and the tt linspace which goes from the stop and start times is made. Another variable vx0vals is defined using numpy arange which has a similar function to linspace. Then a for loop is created where for initial x-velocity in vx0vals, the derivative of the derivatives function is calculated using solve_ivp in order to calculate the projectile trajectory and is then plotted. Array slicing is also used to plot evenly timed dots along the trajectory which can help the player to see where the projectile is moving fast or slow, speeding up or slowing down.

Game Play

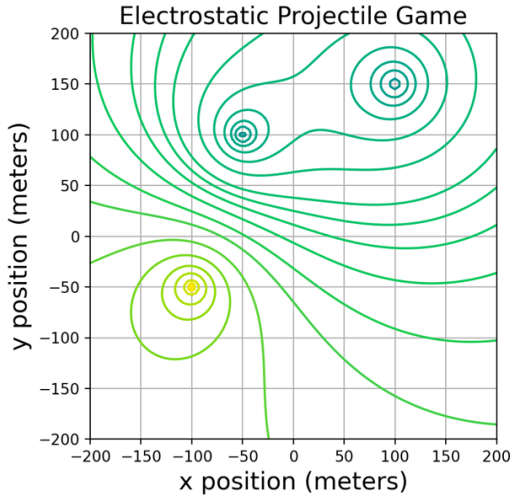
The player will start off seeing an array of trajectories with even time intervals as shown in the example below.



They will then deduce where the hidden charges are located based off these sample trajectories and make their guesses using solve_it(). This will prompt questions asking what sign of the hidden charge is in which quadrant, if any at all like shown below.

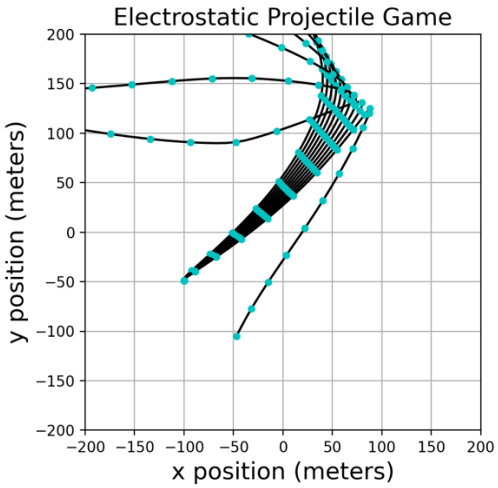
```
(4)Both?
2
Correct!

In quadrant 2, is there a: (1)Positive charge, (2)Negative, (3)Neither, or
(4)Both?
1
Correct!

In quadrant 3, is there: (1)Positive charge, (2)Negative, (3)Neither, or
(4)Both?
Note: it may prove useful to look at the origin of the trajectories
1
Correct!

In quadrant 3, is there: (1)Positive charge, (2)Negative, (3)Neither, or
(4)Both?
3
Correct!
Congratulations! You got all questions correct and won the game!!
The hidden charge pontentials have been revealed!
```

At the end if all questions are answered correctly, a map of the hidden charges and their equipotential will be revealed on top of the trajectories. The following is an example of the potentials without the trajectory overlapping.

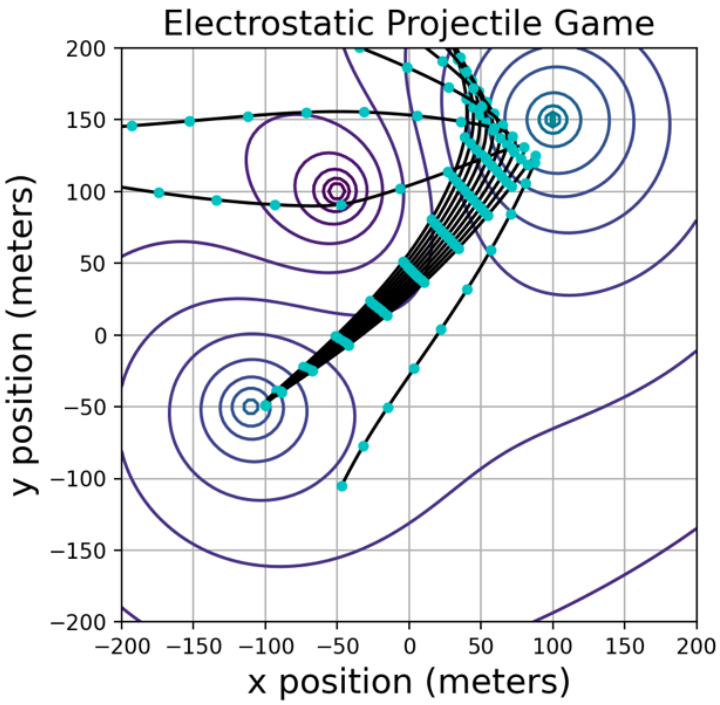


Game Solution

The game can be solved through a bit of pattern recognition and logic. The things a player must take into account in determining the location of the hidden charges is position on the plane, speed, curve, and influence of other charges. Consider the example below:



Firstly, we can see that the projectiles start in the third quadrant and seem to all be pushed away into the first quadrant. From this, we can see that there must be a close positive hidden charge in the third quadrant. Next, we see that in the first quadrant the projectiles split up going all over to the left. Since all these bend to the left, we can either assume that there is a positive charge in the first quadrant or a very strong negative charge in the second quadrant. However, if the latter was true, there would be a much greater influence in the beginning, leading to parabolic-looking trajectories. Since this is not the case, we can now also assume that there is a positive hidden charge in quadrant one where they split up. Lastly, if we follow the middle 2 trajectories in the second quadrant, we can see that there is a curve in the direction of the middle, which leads us to believe there is also a not very strong negative hidden charge in the second quadrant. It is important to note that there *could* be a weak hidden charge in the fourth quadrant, however, following the bottom trajectory we find no influence when it is nearest to the fourth quadrant. Therefore, we can assume there is no hidden charge there. Revealing the solution, we find the equipotential to look like this:



From this, we can clearly see that our deductions were correct. There is a positive close charge in the third quadrant, a positive strong charge in the first quadrant, and a negative charge in the second quadrant.