Electrostatic Projectile Game

This is the Electrostatic Projectile Game where it simulates forces of electric charges on a charged projectile. This game would have designated electric charges within a range of [-200,200] in the x and y direction that are hidden. Charges would have an electrostatic force calculated by the law of F(r) = (kQ/r)r12 where is a unit vector from the point charge to the projectile, Q is the charge, k is the coupling constant, and r is the distance from the point charge to the projectile. The potential function that goes with this force law is (r) = kQln(r0/r) where r0 would be set to 1 and k would be set to 1000. These equations would be converted into functions in Python and would be called upon to calculate values as needed in other functions.

To start with, charges must be implemented using a 2D NumPy array with lists with specifications that correspond with the function that calculates the potential. This specification has 3 columns: the first column is the charge value of that specific point charge, the second column is the x-coordinate of that point charge, and the third column being the y-coordinate of the point charge. The number of rows simply corresponds to the number of different charges where in this case, there are 3. The first function to be used is the clear() function that simply creates, labels, and displays the game window. The play() function can be called to allow for the user to input the initial conditions of the projectiles (velocity, angle, and y position). The reveal\_potential() function calculates potential by calling the calculate\_potential function (inputs x and y coordinates as well as the charge, then looped to plot/calculate the potential at all coordinates) to creates a mesh grid and contour of the potential to see a visual representation. Next is the plot\_trajectory() function that uses the values we had designated in play() to calculate the electrostatic force which returns the x and y-components of the force using the initial velocity and initial angle. With a period of time defined, a derivatives() function is created to define what the forces are before we integrate using solve\_ivp. By using the solution, it produces the position that can be plotted and displayed on the game window. Finally, the solve\_it() function is for prompting the user with choices of where the charges could be, whether they were positive or negative, or if there were more than one. Solutions were defined for each quadrant with a Boolean variable used to keep track of whether what the user inputs as the answer is correct or not. If it isn’t correct, the user is prompted to try again and if they were right, it continues to the next quadrant until all four quadrants are done.

The solution of the point charges that I had chosen is a negative charge in quadrant 1, a positive charge in quadrant 2 and 3, and no charges in quadrant 4. Based on trajectory, discovering where the charges are and whether they are positive or negative on if the projectile seems to be attracted to repelled by them and where.