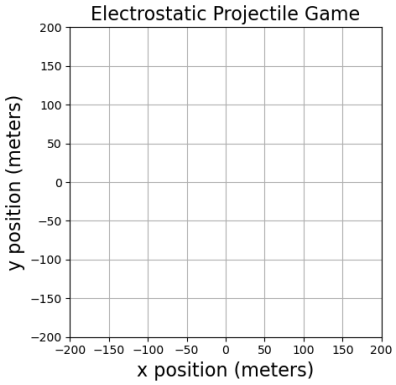
Prospectus for Electrostatic Projectile Game

INTRODUCTION:

This Python-operated game is intended for physics and astronomy students. It employs simple principles of electrostatics in order to give students a primitive understanding of the nature of particle motion within force fields. Some of the concepts have been simplified for better gameplay, but this game would be useful at an introductory level in order to pique interest and teach beginner concepts.
The Electrostatic Projectile Game is downloadable as a .py file, and is played within the Spyder Console through commands and prompts, as well as a Game Window to see all the visuals.
The empty Game window is shown below:



(1)

The goal of the game is to find out where 5 hidden stationary charges are on the xy-playing field. These charges vary in magnitude and sign (+ or -) and will attract or repel a particle that the player will send moving through the field. To launch this particle and see its trajectory, the player will input a starting velocity, angle of launch, and starting y-level. The starting x-level remains at -100 for the whole game.

GAME PHYSICS:

Two formulas are used in the Electrostatic Projectile Game in order to generate the field in which particle motion will occur.
Mainly, the following formula will be used to describe the electrostatic force:

$$\vec{F}(r) = \frac{kQ}{r}\hat{r}_{12}$$

(2)

While normal electrostatic force falls off at a rate of $1/r^2$, the small scale of the game allows us to simplify the motion to $1/r$.

Given the equation, Q represents the magnitude and sign of the hidden charges, r and $\hat{r}$ regard the distance between the launched particle and the hidden charge, and k is a constant. The resulting force varies depending on the strength of the charges and the distance away from them.
In simple terms, a negative charge attracts the projectile and a positive charge repels the projectile.

Another formula is used for a contour plot of the potential field lines that will be displayed after the player wins. This formula,

$$\varphi(r) = kQln(r_0/r) \quad (3)$$

provides the potential at every distance r away from the charges, and when fields overlap, the potential is summed.

IMPLEMENTATION:

Here I will describe the purpose and method of every function within the game code:
The lines in which they occupy are listed in the brackets.

[19-28]: The magnitude and location of all 5 charges are defined.
[39-62]: The functions clear() and create_game_window() spawn the Game Window shown in (1).

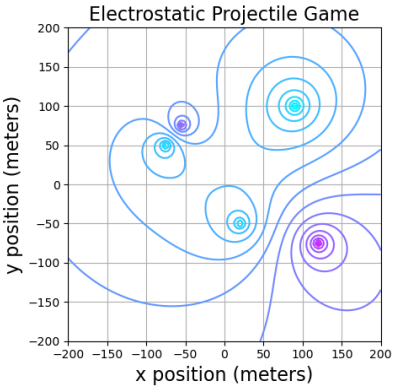
- These use matplotlib and fig and ax objects to open a plot in a new window. Parameters are added to the axes, such as a title, x and y labels, and a grid.

[67-84]: play():

- This function prompts the player to enter an initial speed (between 0 and 100), an initial angle of launch (in degrees), and the initial y-position (between -200 and 200).
- Prompts are made using the input() function, which asks the user a question and takes their response as a float value, which is then saved in the variables v0, angle, and y0.

[90-120]: reveal_ potential()

- This function is called at the end of the game to reveal the location of the charges.
- It looks like this:



- The negative field is represented by blue and the positive by purple.
- The reveal_potential() function works by using a numpy meshgrid, which uses 2 pre-made arrays for x and y to return 2d grid coordinates. The x and y arrays are manually looped, along with the hidden charges, in the function calculate_potential(). This function is analogous to (3) and returns the potential. Then, matplotlib's contour plot is used to map the potential along different levels of the magnitude of the potential.

[138-169]: plot_trajectory()

- This function takes the given initial velocity, angle, and y-value to plot the motion of a particle in the force field using Newton's 2nd law.
- An array of time is created for the motion.
- derivatives(t,s) within the function sets up the values for scipy's solve_ivp() function. An array in derivatives(t,s) relays each of its elements into solve_ivp(), which takes the derivative of each element. In this case, derivatives(t,s) and solve_ivp() take the derivative and 2nd derivative of the electrostatic force (assuming mass is 1), returning the velocity and position of the particle.
- Only the x and y coordinates of the particle's position are needed for the matplotlib plot, and some array slicing is used to show the black markers.

[173-199]: solve_it()

- This function is called to see if the player's guesses are correct.
- input() functions are used to retrieve the player's answers and if else statements are used to compare their answer to the correct value. At the end an if else statement with the 'and' logic statement sees if the player was correct on all 4 questions, if so, they win the game.

GAME PLAY:

The gameplay is as follows.

Once the .py file is opened and runs at least once, the player must enter the following into the console:

import lab1a_Kwon as game

import lab1a_utilities as util

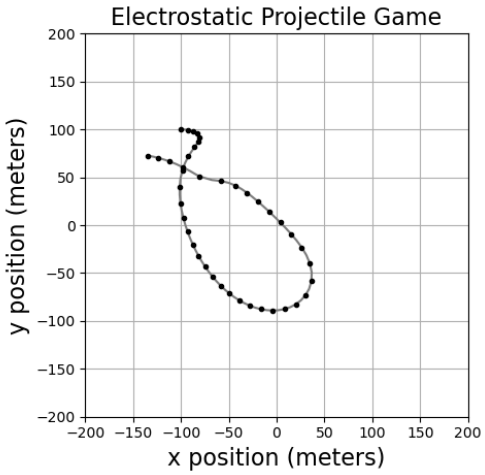
Then, the player should enter **game.create_game_window()** and the figure (1) will appear.

Now the player can enter **game.play()** to begin the game.

The player will be prompted to enter a velocity, initial angle, and y position. To show an example, I'll enter a velocity of 30, angle of 0, and 100 as the y position:

```
In [171]: game.play()
Starting x location is -100
Enter the initial speed between zero and 100.
30
Enter the initial angle in degrees.
0
Enter the initial y position.
100
```

Entering this will generate the following plot on the Game Window.

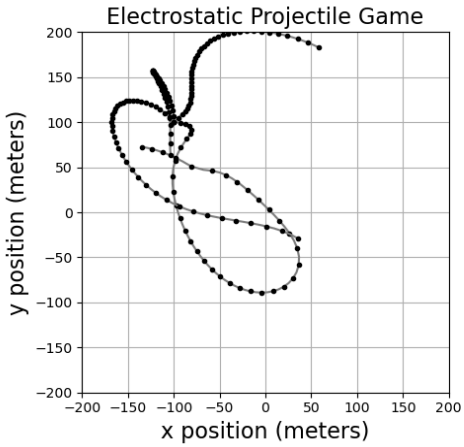


To generate new plots, the player can enter **game.play()** multiple times to get new trajectories.

GAME SOLUTION:

How can the player solve the game?

Here is what the Game Window looks like after several different trajectories starting at a y-position of 100 and a starting velocity of 30:

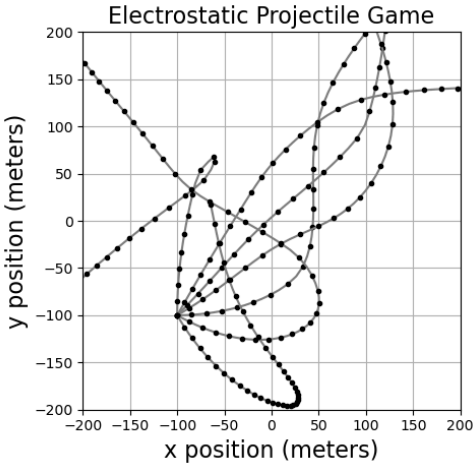


The player could make several inferences from this plot. All of the trajectories seem to dart away from the starting point, indicating that there is at least one positive charge in the 3rd quadrant.

The player can clear the board any time by entering **game.clear()** into the console.

Let's look at another example.

Here is what the window looks like after several different trajectories staring at a y-position of -100 and a starting velocity of 60:



Several inferences can be made. All trajectories travel away from quadrant 4, indicating the presence of a positive charge (repel).

Many trajectories eventually travel to quadrant 4, indicating the presence of a negative charge (attract).

After the player feels they have made enough inferences, they can enter **game.solve_it** to enter their guesses. The following will come up in the console:

Here, the player should type a number between 1 and 4 in integer form to give their answer.

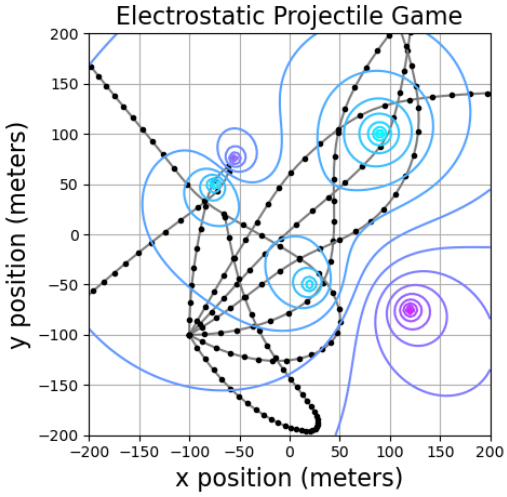
```
In [183]: game.solve_it()
Does quadrant 1 have:
(1) positive charge; (2) negative charge; (3) neither; or (4) both?
2
```

The same question will be asked for each quadrant, and after each guess, the console will return “Correct” or “Incorrect” for the quadrants that were guessed.

```
In [183]: game.solve_it()
Does quadrant 1 have:
(1) positive charge; (2) negative charge; (3) neither; or (4) both?
2
Correct
How about quadrant 2?
```

If the player guesses correctly for all 4 quadrants, the player will be congratulated, if they lose, they can try again later by entering **game.solve_it** at a later time.

Once the player wins the game, the equipotential lines will appear to show the real location of all the charges:



PROPOSAL:

The game’s use of simplified physics principles: Electrostatic force and potential, as well as Newton’s 2nd law, makes this a perfect introduction to both physics and the python window for beginner physics and astronomy students.

I propose to publish my game onto the Physics department website and to have it introduced into the beginning course for Electricity and Magnetism. I’d also like assistance from a web designer to make the design more practical and colorful to better appeal to students, however all game code is completed in its entirety.