## **Task 2:**

Part 1:

Successful implementation of power function:

A screenshot of a computer

Description automatically generated with medium confidence

Output:



Part 3:

Successful implementation of classes Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Output:

Chart

Description automatically generated

**Task 3:**

My own function of dot product. It works with (Matrix, Matrix); (Matrix, Vector); (Vector, Vector):

def dot(X, Y):  
 try:  
 if Y.ndim == 1 and X.ndim == 1:  
 c = np.zeros(len(X))  
 if X.ndim == 1:  
 lenght = len(X)  
 else:  
 lenght = len(X[0])  
 for k in range(len(X)):  
 t = 0  
 for i in range(lenght):  
 t += X[k] \* Y[i]  
 c[k] = t  
 elif Y.ndim == 1 or X.ndim == 1:  
 if X.ndim == 1:  
 X, Y = Y, X  
 c = np.zeros(len(X))  
 if X.ndim == 1:  
 lenght = len(X)  
 else:  
 lenght = len(X[0])  
 for k in range(len(X)):  
 t = 0  
 for i in range(lenght):  
 t += X[k][i] \* Y[i]  
 c[k] = t  
 else:  
 c = np.zeros((len(X), len(Y[0])))  
 for k in range(len(X)):  
 for i in range(len(Y[0])):  
 t = 0  
 for j in range(len(Y)):  
 t += X[k][j] \* Y[j][i]  
 c[k][i] = t  
 except:  
 print("Invalid input in dot() function")  
 return  
 Y = c  
 return Y

I have no output example, as I didn’t need it for my calculations, but I can say that no errors were encountered during all those calculations.

Here is my Asteroid class implementation:

class Asteroid(Character):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.pos = np.array([np.random.uniform(-9, 10), np.random.uniform(-9, 10)])  
 self.T = translation\_mat(self.pos[0], self.pos[1])  
 self.color = (np.random.random(), np.random.random(), np.random.random())  
 self.S = skew\_mat(np.random.uniform(-0.3, 0.3), np.random.uniform(-0.3, 0.3))  
 self.speed = np.random.uniform(0.4, 1.5)  
 self.dir\_init = np.array([np.random.uniform(-0.1, 0.1), np.random.uniform(-0.1, 0.1)])  
 self.dir = np.array(self.dir\_init)  
  
 def generate\_geometry(self):  
 theta = 0  
 radius = (np.random.uniform(0.3, 1))  
 while theta <= 360:  
 if theta % 20 == 0:  
 self.geometry.append([self.pos[0] + (radius \* np.cos(np.radians(theta))) + np.random.uniform(0.01, 0.1),  
 self.pos[1] + (radius \* np.sin(np.radians(theta))) + np.random.uniform(0.01,  
 0.1)])  
 theta += 1  
 self.geometry = np.array(self.geometry)  
  
 def draw(self):  
 x\_data = []  
 y\_data = []  
  
 self.C = dot(self.S, self.T)  
 for vec2d in self.geometry:  
 vec3d = vec2d\_to\_vec3d(vec2d)  
  
 vec3d = dot(self.C, vec3d)  
  
 vec2d = vec3d\_to\_vec2d(vec3d)  
  
 x\_data.append(vec2d[0])  
 y\_data.append(vec2d[1])  
  
 plt.plot(x\_data, y\_data, c=self.color)  
  
 def move(self):  
 if self.pos[0] <= -10 or self.pos[0] >= 10:  
 self.dir[0] = -self.dir[0]  
  
 if self.pos[1] <= -10 or self.pos[1] >= 10:  
 self.dir[1] = -self.dir[1]  
  
 vec3d = vec2d\_to\_vec3d(self.dir)  
  
 vec2d = vec3d\_to\_vec2d(vec3d)  
 self.pos[0] += vec2d[0] \* self.speed  
 self.pos[1] += vec2d[1] \* self.speed  
  
 self.T = translation\_mat(self.pos[0], self.pos[1])

I would say, that main result of this work was Rocket class as it was created from scratch and worked properly:

class Rocket(Character):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.S = scale\_mat(0.1, 0.2)  
 self.speed = 1  
 self.pos = np.array(player.get\_pos())  
 self.R = player.get\_R()  
 self.dir = np.array(player.dir)  
  
 def generate\_geometry(self):  
 self.geometry = np.array([  
 [-1, 0],  
 [1, 0],  
 [0, 1],  
 [-1, 0]  
 ])  
  
 def draw(self):  
 x\_data = []  
 y\_data = []  
  
 vec3d = vec2d\_to\_vec3d(self.dir)  
  
 vec3d = dot(self.R, vec3d)  
  
 vec2d = vec3d\_to\_vec2d(vec3d)  
 self.pos[0] += vec2d[0] \* self.speed  
 self.pos[1] += vec2d[1] \* self.speed  
  
 self.T = translation\_mat(self.pos[0], self.pos[1])  
  
 self.C = dot(self.T, self.R)  
 self.C = dot(self.C, self.S)  
  
 for vec2d in self.geometry:  
 vec3d = vec2d\_to\_vec3d(vec2d)  
  
 vec3d = dot(self.C, vec3d)  
  
 vec2d = vec3d\_to\_vec2d(vec3d)  
 x\_data.append(vec2d[0])  
 y\_data.append(vec2d[1])  
  
 plt.plot(x\_data, y\_data, 'b')

How it looks:

Chart, scatter chart

Description automatically generated

Main findings:

I have completed every task except task 3.7. I will refer to that in questions section.

This work reminded me a lot of class usage in Python, but what was much more important – vectors and matrices practice.  
  
From “Fundamentals of Image processing” course I remember working with different matrices to transpose or rotate dots, but with lack of practice in this sphere I forgot a lot.  
This work reminded me much material.

Questions:

For now I have only one question. I can’t implement variable ‘result’ inside ‘Asteroid()’ class.

How it works:

Text

Description automatically generated

How I want to work:

Text

Description automatically generated

Error I get:

Text

Description automatically generated

Frankly speaking, I see no reason for program not to see variable I created.  
So I wanted to ask, what’s the reason? It really bothering me, as it doesn’t allow me to complete task 3.7 (collision implementation).