Research Project

Title	Research Project
Tags	Personal Project
Created	@August 16, 2022 11:10 AM
Progress	
Deadline	
Reviews	
Priority	
Archived	☐

Introduction

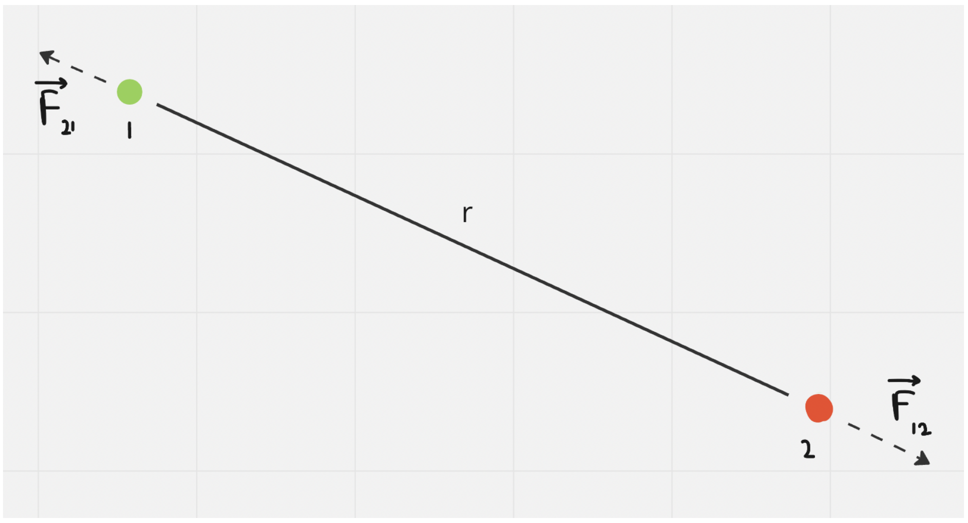
Science always begs the question why, how and when to everything. The entire purpose of science is to question any and everything and find out answers to answer those questions. But sometimes certain unrealistic questions arise where we cannot use calculations to predict the outcome or find an answer as it is simply too tiring or there are simply too many factors to consider every second. But, what if we let a computer do the job?

We can create a fake world of ourselves, where we can state and control the circumstances we want and the exact situation we need to find the answer of without calculating all of that ourselves or witnessing the event in real life. This requires the integration of real world physics with programming.

Theory of the Simulation

We start by creating a red dot.

We give this red dot some properties, for example mass. Now we create another green dot with a different value of mass. Now we create a space and randomly place these two dots on this space. Hence, we create a force between these two objects. According to Newtons law of Gravitation: $\vec{F} = \frac{Gm_1m_2}{r^2}\hat{n}$. Here, the value of the force on the two objects is given by the ratio of the product of the Universal Gravitational Constant ($= 6.67 \times 10^{-11} Nm^2/kg^2$)and the masses of both the objects to the square or the distance between the two objects.



We also know, from Newton's second law of motion, that $\vec{F} = m\vec{a}$. Now to find the acceleration that one of the dot experiences can be derived like this:

$$\text{equating,}$$
$$m_1\vec{a} = \frac{Gm_1m_2}{r^2}\hat{n}$$
$$\vec{a} = \frac{Gm_2}{r^2}\hat{n}$$

This will be the acceleration experienced by the one of the dots.

Further, we can add factors like electric charge, electrostatic forces and magnetic forces, electric fields etc.

Let us look at the electric part of this simulation.

Suppose we give the two dots another property of charge. Giving both the dots some unit charge will result in an electrostatic force between them which is given by $\vec{F} = \frac{kq_1q_2}{r^2}\hat{n}$.

Now, the total force experienced by any of the dots is given by:

$$\vec{F}_t = \vec{F}_g + \vec{F}_e$$
$$\vec{F}_t = \frac{Gm_1m_2}{r^2}\hat{n} + \frac{kq_1q_2}{r^2}\hat{n} = \frac{1}{r^2}\Big[Gm_1m_2 + kq_1q_2\Big]\hat{n}$$

And then using $\vec{F} = m\vec{a}$, $\vec{a} = \frac{\vec{F}_t}{m}$. So this way, we add more factors to reach a point which can make a simulation very realistic.

Implementation

Now, to implement this using python, I'm going to use pygame to create a blank space.

```
window_size = 300
pygame.init()
window = pygame.display.set_mode((window_size, window_size))
```

We create a function which initialises all the properties of a single atom using a dictionary.

```
def atom(x, y, c, m):
    return {"x": x, "y": y, "v_x": 0, "v_y": 0, "color": c, "m": m}
```

Here v_x and v_y refer to the horizontal and vertical component of the velocity of the atom.

Now we create a function to randomly place and create these “atoms”.

```
#random position generator
def randomxy():
    return round(random.random()*window_size + 1)

atoms=[]

def create(number, color, mass):
    group = []
    for i in range(number):
        group.append(atom(randomxy(), randomxy(), color, mass))
        #adds all the atom's properties to list for use
        atoms.append((group[i]))
    #returns the properties of all the atoms of that kind in a list for use
    return group
```

The `create()` function returns a list of all the properties of each atom of that kind. It also adds that list to another list which is meant to store the properties of all the atoms in the space.

Now we can start to place dots randomly across the space. For simplicity’s sake, we have kept the value of g variable for 2 different particles. For example, a red and interacts with a red dot with the g value of 0.1 whereas a red dot interacts with a green dot with a g value of -0.1.

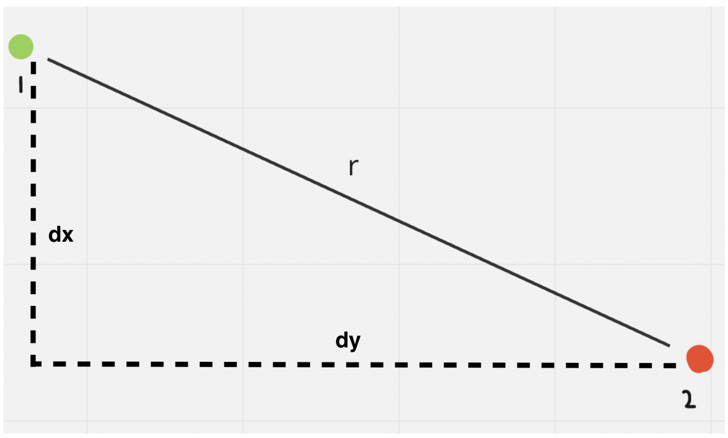
To make the dots experience some force and initiate some motion, we create the following function.

```
def force(atoms1, atoms2, g):
    for i in range(len(atoms1)):
        f_x = 0 #horizontal component of force
        f_y = 0 #vertical component of force
        for j in range(len(atoms2)):
            a = atoms1[i]
            b = atoms2[j]
            dx = a["x"] - b["x"]
            dy = a["y"] - b["y"]
            r = (dx*dx + dy*dy)**0.5 #distance between the particle using pythagoras
            if( r > 0 and r < 80):
                F = (g*a["m"]*b["m"])/(r**2)
                f_x += F*dx
                f_y += F*dy
        a["v_x"] += f_x/a["m"] #a["a_x"] (horizontal component of acceleration)
        a["v_y"] += f_y/a["m"] #a["a_y"] (vertical component of acceleration)
        a["x"] += a["v_x"]
        a["y"] += a["v_y"]
        #to prevent any particles from leaving the map
        if(a["x"] <= 0 or a["x"] >= window_size):
            a["v_x"] *=-1
        if(a["y"] <= 0 or a["y"] >= window_size):
            a["v_y"] *=-1
```

Here,

```
a = atoms1[i]
b = atoms2[j]
dx = a["x"] - b["x"]
dy = a["y"] - b["y"]
r = (dx*dx + dy*dy)**0.5
```

This part of the code calculates the distance between the two particles at any given moment using the pythagoras theorem.



$$r = \sqrt{(dx)^2 + (dy)^2}$$

It then checks if particle is too far (r >100 pixels) or at the same position as the taken particle (r = 0). If it is, then the value of force on the two particles becomes 0.

Here we are using Newtons law of gravitaion to calculate the magnitude of force on the particles.

```
F = (g*a["m"]*b["m"])/(r**2)
f_x += F*dx
f_y += F*dy
```

```
a["v_x"] += f_x/a["m"]
a["v_y"] += f_y/a["m"]
```

Here we use $\vec{a} = \frac{\vec{F}}{m}$ to find out the acceleration (or $\frac{v2-v1}{t2-t1}$) and add that value

```
a["x"] += a["v_x"]
a["y"] += a["v_y"]
```

to the velocity components of the particle. We then add the current velocity to the position components of the particle to show motion.

Then lastly, to prevent any particles from leaving the space, we add this last part to make them bounce back if they hit the walls of the space.

```
if(a["x"] <= 0 or a["x"] >= window_size):
    a["v_x"] *=-1
if(a["y"] <= 0 or a["y"] >= window_size):
    a["v_y"] *=-1
```

Now as you can see there is not a constant value of G here. It is different for the interactions between 2 kinds of atoms, For Example Red-Red would have a g of -0.5 whereas Red-Green would have a g of 0.5. I have done this to simulate the different forces of attraction and repulsion as well between different kinds of particles.

This could have been done be following and expanding our similation to fit another parameter and a the force of charge/electrostatics in our code as we discussed in the Theory section of this paperbut it would make this code a little bit more complex for this research paper. Though the reader is free to try it at home.

Now we can start our simulation and see how it works…

First we create 2 kinds of atoms, namely red and green.

```
green = create(5, "green", 1) # create(number of particles, colour, mass)
red = create(5, "red", 2)
```

Then we create a `draw()` function for pygame to show us the simulation we have created.

```
def draw(surface, x, y, color, size):
    for i in range(0, size):
        pygame.draw.line(surface, color, (x, y-1), (x, y+2), abs(size))
```

Now to start the simulation, we write this block of code at the end.

```
run = True
while run:
    window.fill(0)
    force(red, red, -0.1)
    force(red, green, 0.1)
    force(yellow, green, -0.1)
    for i in range(len(atoms)):
        #draws every atom created using the atoms list
        draw(window,  atoms[i]["x"], atoms[i]["y"], atoms[i]["color"], 3)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
    pygame.display.flip()
pygame.quit()
```

Results

For the results, check out my github repository. It contains various videos and values i have taken. It also has the full code that I had written for this Project.

GitHub Repository:

https://github.com/gavkujo/Particle-simulator

Conclusion

As we can observe that we can simulate real world physics easily with the help of programming if done with research and proper knowledge of the concept along with a powerful enough system to work with depending on the scale and complexity of the simulation

These simulations also help us gain knowledge about impossible and rare events without having to wittness them in real life. For example, we can safely determine/predict the geographical outcomes of detonating a nuclear bomb without having to actually do it in real life.

Further studies in this area and integrating this idea with more complex sciences and machine learning can help humanity solve much larger real life problems and help taking steps to a better world.

Credits

I took help and inspiration from Brainxyz. I took his Python code as a base for my version of his code to create this research paper. I modified the code to fit the more realistic equation Newton's equation gravitation ($\vec{F} = \frac{Gm_1m_2}{r^2}\hat{n}$) as opposed to Brainxyz's equation of gravitation which he had simplified it to $\vec{F} = \frac{G}{r}$.

Brainxyz's Youtube channel:
https://www.youtube.com/channel/UCHnxRUI0vpsikGcl5OWqKGA