Visualizing Swarms

Syed Ibrahim Ali Haider, Meesum QazalBash

Assignment 2 Q2.

Visualizing Swarms

To visualize swarms, I chose to build a fire particle system. The reasoning for choosing to build a fire particle is because of the demo that was given to us in class. When I saw that demo of the fire particle system it ignited my curiously as to how could such an smooth fire be built by individual particles.

So first of all I understood the basics of particles through various online papers and references, after getting an understanding how a particles works and how they collectively work to reach a larger goal. I went on understanding fire, the different layers it had, the varying gradients, the random movements and color shifts.

So in the simulation of the fire particles, particles are used to represent fire, in which each particle is assigned a color, and that color of each particle is known by its position and the position of its neighbors. All of this done with the help of a color gradient that has different shades of our fire.

The problem formulation involved determining the size of the particles, the colors to be used, the color gradient, generation and lifespan of the particles, how the particles move and how the particles interact with each other.

To solve this problem, I used a particle system that is made on a 2D grid. Where each particle is assigned a color based on its position in the grid, the colors of its neighboring particles are used to create a smooth transition between them.

Now to create the the gradient colors of the fire, I have used interpolation between the colors in my list which allows a smooth transitioning. The particles are assigned colors according to the index that they are given and that index gives the color corresponding to our colorlist that we generate using interpolation.

To create the illusion of movement and turbulence, the simulation uses the random function that randomly shifts the position of particles and their colors aswell. This creates the effect of flickering flames and gives the fire a more real life look.

We initialize the main parameters here.

```
def __init__(self,particle_size,width,height,gradient) -> None:
    self.particle_size = particle_size # to be taken as input
    # if color == 'Red':
    #     self.colors = ['black','red','orange','yellow','white'] # to be taken as input
    # else:
    #     self.colors = ['black','blue','turquoise','white'] # to be taken as input

    self.colors = ['black','red','orange','yellow','white'] # to be taken as input

    self.gradient = gradient # variable for transitioning to next color, for smooth transitioning # to be t
    self.colorlist = self.create_colorlist() # getting our colorlist that now contains all gradients accord

    # getting how many pixels will fill so that we create the width and height of our particles accordingly
    self.particles_width = width//self.particle_size
    self.particles_height = height//self.particle_size

    self.particles = self.create_particles() # getting our 2d list of the particles that we want to displa

    pass
```

We then create our color list using interpolation

```
# create a color list, that has the different gradeint colours for our particles
def create_colorlist(self):
    # initialsing our black color, using RGBA Format
    colorlist = [(0,0,0,0)]

    # looping over the colors
    for c, color in enumerate(self.colors[:-1]):
        # using py.Color, that allows RGBA, allowing for opacity that enhances our transitioning
        color1 = py.Color(color) # assiging our first colour
        color2 = py.Color(self.colors[c+1]) # assigning our color next to that color
        for gap in range(self.gradient): # loop till the amount of gradient we want, greater gradeint = s
            new_color = color1.lerp(color2,(gap+0.5)/self.gradient) # interpolating between two colors, c
            colorlist.append(new_color) # adding this to our color list, gradeint being gradually created
    return colorlist
    pass
```

We then create our 2d grid for our particles

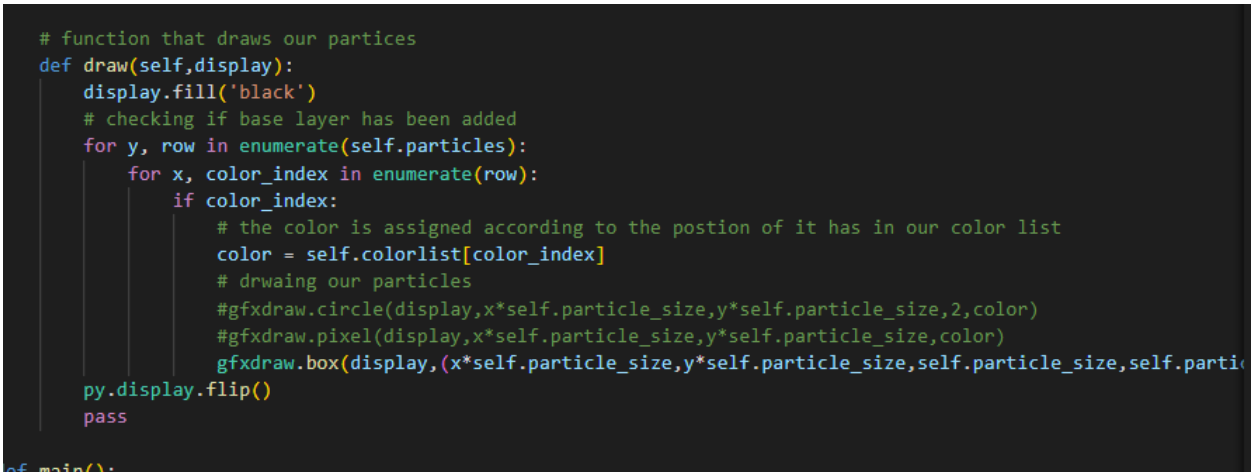
```
# function that creates the base layer of our particles
def create_particles(self):
    # using a 2d-list to store our particles
    particles = []
    # initialsing our array for our particles based on the dimensions for the particles that we have set
    for c in range(self.particles_height):
        row_particles = []
        for i in range(self.particles_width):
            row_particles.append(0)
        particles.append(row_particles)
    # we will add our base layer of to our particles, this will be our brightest color
    # this will be our bottom row
    bottom_row = self.particles_height-1
    # position of brightest color
    brightest_idx = len(self.colorlist)-1
    # looping over last row and adding our brightest color
    for width_idx in range(self.particles_width):
            # takes the position value of brightest color from color list and assigns it to bottom most row
            particles[bottom_row][width_idx] = brightest_idx
    return particles
```

The Turbulance effect, color changing and particle lifespan take place in this section of the code

```
# function that adds up the fire particles and asssigns them their color
def particles_effect(self):
    # looping over columns
    for col in range(self.particles_width):
        # looping over our rows or fire height, except for the frist one as that is black
        for row in range(1, self.particles_height):
            color_index = self.particles[row][col]
            # checking if their is color index already present
            if color_index:
                # generate a random number, which helps us to create turbulance and a varying color graident
                vary_gradient =  randint(0,3)
                # we will be using this formula to shift columns in such a wasy that it does not exceed our
                #   same time give us a turbulance effect
                turbulance = col - vary_gradient + 1
                # asssign the color index values from our color list and mkaing sure that they are not out o
                self.particles[row - 1][ turbulance % self.particles_width] = color_index - vary_gradient %
            else:
                # if they do not have a color index then those particles lifespan ends, by assignng them the
                self.particles[row - 1][col] = 0
    pass

# function that draws our partices
```

Finally our draw function



```
# function that draws our partices
def draw(self,display):
    display.fill('black')
    # checking if base layer has been added
    for y, row in enumerate(self.particles):
        for x, color_index in enumerate(row):
            if color_index:
                # the color is assigned according to the postion of it has in our color list
                color = self.colorlist[color_index]
                # drwaing our particles
                #gfxdraw.circle(display,x*self.particle_size,y*self.particle_size,2,color)
                #gfxdraw.pixel(display,x*self.particle_size,y*self.particle_size,color)
                gfxdraw.box(display,(x*self.particle_size,y*self.particle_size,self.particle_size,self.parti
    py.display.flip()
    pass

def main():
```

LINK: https://youtu.be/DSFjzdiUUUg