Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

---

Programación Gráfica con Python y JavaScript

Álvaro Alarcón Granero

Héctor Arranz Jiménez

Índice

1. Introducción teórica

El movimiento de las partículas ha sido siempre objeto de estudio en la ingeniería. En este trabajo se ha realizado un modelo para estudiar el comportamiento de las partículas, llamado *modelo de Vicsek*. Para introducir al lector a este ámbito, se realizará un breve esbozo de dicho modelo.

Hace 20 años el físico húngaro Tamás Vicsek diseñó un modelo para predecir el comportamiento de partículas en las que se las impone un objetivo. Cuando la cantidad de partículas es elevada, pueden cruzarse y chocarse entre ellas; o si las hubiese, con las paredes. Para evitar tal choque, se definirán 3 fuerzas que determinarán el comportamiento de cada partícula. Así, la fuerza que ejerce la partícula j sobre la partícula i se define como:

$$\vec{f}_{ij} = \vec{f}_{repulsion} + \vec{f}_{alineamiento} + \vec{f}_{friccion}$$

Estando definidas cada una de ellas como:

$$\vec{f}_{repulsion} = Ae^{(R_{ij}-d_{ij})/B}\vec{n}_{ij}$$
$$\vec{f}_{alineamiento} = k\mu(R_{ij} - d_{ij})\vec{n}_{ij}$$
$$\vec{f}_{friccion} = \kappa\mu(R_{ij} - d_{ij})\vec{t}_{ij}$$

En la que la funcion μ vale 0 cuando es negativa y 1 cuando es positiva; esto quiere decir que las funciones de alineamiento y fricción sólo se consideran cuando las partículas se están tocando. Para establecer este fenómeno, definimos el radio de influencia de cada partícula, así cuando el centro de otra partícula esté dentro de este radio, se estarán chocando, y por consiguiente aparecerán las fuerzas de fricción y alineamiento.

Las constantes A, B, k y κ se usan para introducir los órdenes de magnitud en las fuerzas.

Los vectores normales y tangenciales están definidos de la siguiente forma:

$$\vec{n}_{ij} = (r_i - r_j)/d_{ij}$$
$$\vec{t}_{ij} = (-n_{ij}^2, n_{ij}^2)$$

Para el caso de las paredes, se utilizarán las mismas fuerzas, pero tan solo sustituyendo la distancia entre partículas por la distancia a la pared.

Como es esperable, se podría profundizar mucho más en la explicación de este modelo y de desarrollar su completo funcionamiento, pero esto abarcaría un trabajo demasiado denso.

2. Código

Ahora se va a proceder a explicar como hemos implementado el modelo, y las conexiones entre *Python* y *JavaScript*.

Lo primero que hemos hecho es realizar el programa principal, una vez que nos cercioramos de su correcto funcionamiento, pasamos a recibir los datos desde un servidor en localhost. Para ellos, utilizamos la carpeta tornado facilitada en la plataforma *Moodle*. Para conectar ambos programa introdujimos nuestro programa principal en este nuevo programa.

```
    def on_message(self, message):

        json_string = u'%s' %(message,)

        mensajedeJS = json.loads(json_string)

        msg_particulas = mensajedeJS['particulas']
        msg_tiempo_foco = mensajedeJS['tiempo_foco']



        n = int(msg_particulas)
        t_in = int(msg_tiempo_foco)


###AQUI EMPIEZA EL PROGRAMA PRINCIPAL###

        def vec_aleat(k):
            if k == 1:

                vec = [np.random.randint(0, 50), np.random.randint(0, 50)]
                # print(vec)

            elif k == 2:

                vec = [np.random.randint(-10, 10), np.random.randint(-10, 10)]
```

También se puede ver como n y t_{in} son las variables de entrada a nuestro programa principal.

Por otro lado, para enviar los datos que genera el archivo principal al html, hemos creado un archivo con la extensión *.json* de la siguiente forma

```
file = open('data.json', 'w')
file.write('[\n')
for f in range(stop):
    line = '[{"x": ' + str(pos_foco[0]) + ', "y": ' + str(pos_foco[1]) + "},"
    for g in range(n):
        line = line + '{"x":' + str(matrizposicionx[f, g]) + ', "y": ' + str(matrizposiciony[f, g]) + "},"

    line = line[:-1]
    line = line + "],"
    if f == (stop - 1):
        line = line[:-1]

    file.write(line + '\n')

file.write(']\n')
file.close()
```

Una vez tenemos el archivo *.json* empleamos la librería D3 javascript, con la que se pueden leer y manipular datos fácilmente.

En el javascript se accederá a los datos del archivo json a través de la sentencia d3.json('data.json')

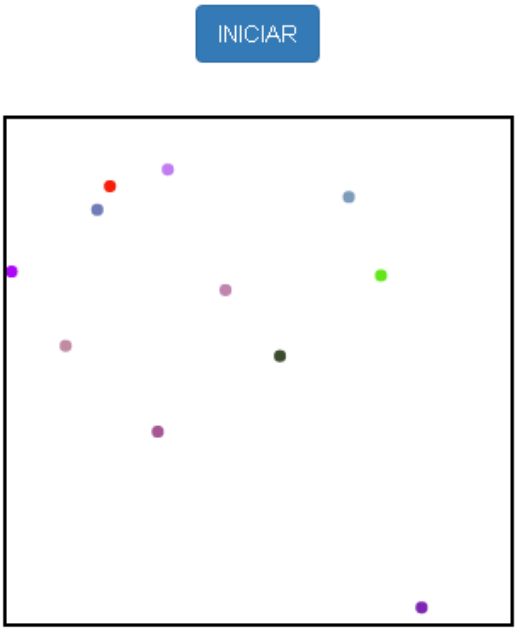
```
function iniciar() {

    $.ajax({
        cache: false,
        url: "data.json",
        dataType: "json",
        success: function(data) {
            d3.json("data.json", function(data) {

                var matriz = [];
                var matriz2 = [];
                var sequence = [];

                data.forEach(function(item){
                    item.forEach(function(element) {
                        sequence.push(element.x);
                        sequence.push(element.y);
                    });
                    matriz2.push(sequence);
                    sequence = [];
                });

                console.log(matriz);
                console.log(matriz2);
```

Como se puede observar, existen tres bloques de código en la lectura del archivo de datos. En el primero almacenamos las posiciones de las partículas en la matriz llamada "matriz2". En el segundo generamos los círculos que utilizaremos para la representación en SVG asignando la posición del centro, radio y color a cada uno de ellos. Por último, generamos el movimiento cambiando la posición del centro de cada partícula a través de un bucle.

La lectura del archivo se implementa en una función en la que se emplea AJAX, con la que somos capaces de leer el archivo json y representar la simulación sin necesidad de refrescar la página web.

```
d3.select("#boton1 button").on("click", function movimiento() {
    setInterval( function () {
        if (i < matriz2.length) {
            for (j = 0; j < particulas.length; j++) {
                particulas[j].setAttribute("cx", matriz2[i][0 + 2*j]);
                particulas[j].setAttribute("cy", 50 - matriz2[i][1 + 2*j]);
            }
        }
        i++;
    },50);
})
```

Por otro lado, el archivo html ha sido diseñado con bootstrap. En la página web se ha realizado un cuadrado en el que se representarán las partículas. El botón iniciar nos facilita poder controlar cuando queremos que se ejecute.



3. Objetivos

Una vez que se han desarrollado los cálculos, se procederá al estudio y análisis. Este proceso consistirá en ir variando los parámetros del programa y ver cómo se pueden optimizar los resultados. Entre los objetivos que hemos podido encontrar son:

1. La velocidad óptima para escapar de un habitación en estado de pánico dista bastante de la máxima que pueden alcanzar las partículas. Cuando se alcanzan velocidades máximas se producen mayor número de choques y ,por consiguiente, el tiempo de evacuación aumenta considerablemente.

2. El parámetro de pánico (definido como la proporción en la que una partícula sigue su instinto o a las masas) tiene un valor entre 0 y 1. Valdrá 0 cuando la partícula solo se fije en su comportamiento y valdrá 1 cuando solo se fije en el comportamiento de las partículas que la rodean. Analizando varios casos hemos podido comprobar que el parámetro óptimo de pánico es 0.4.

3. Imponer que la velocidad de las partículas sea constante optimiza los tiempos de evacuación. Lo que determina que en un estado de pánico se debe mantener la calma para abandonar la sala cuanto antes.