Nonlinear Analysis: Modelling and Control, Vol. vv, No. nn, YYYY

Simulación de una escena en 2D

Camilo Alcocer, Harold Alfaro, Esteban Montero

IES "INFOTEP", Ciénaga Magdalena

Received: 03/03/2025 /

Abstract

Este informe presenta el desarrollo de un algoritmo en Python usando "Jupyter Notebook" para la simulación de una escena 2D, en la que un punto se desplaza sobre un rectángulo de fondo.

Keywords: Python, Jupyter Notebook, Simulación, 2D

1 Introduction

En este informe se presenta el algoritmo que permite dibujar una escena en 2D sobre un rectangulo que contiene en su interior un punto, definiendo escenas que permiten dibujar, cambiar y actualizar el rectangulo, mediante el uso de Python.

2 Objetivos de la actividad

Dentro de los objetivos de la actividad está el familiarizarnos con las herramientas y sus interfaces, de este mismo modo entender el simulado de un objeto y poder manipular sus parámetros.

3 Actividad

Este código define primero una clase "Escena" representando así el espacio 2D (Bidimensional) donde se dibujará el rectángulo y el punto dentro del mismo.

```python
class Escena:
    def __init__(self, ancho, alto, color_fondo, punto, color_punto):
        """Inicializa la escena con un rectángulo y un punto superpuesto."""
        self.ancho = ancho
        self.alto = alto
        self.color_fondo = color_fondo
        self.punto = np.array(punto, dtype=float)
        self.color_punto = color_punto
```

Figure 1: Inicialización de la clase "Escena"

A continuación, se establecen las funciones de nuestro algoritmo, como: "dibujar escena", "desplazar punto", "cambiar escena", "simular" y "actualizar".

```python
def cambiar_escena(self, ancho, alto, color_fondo, punto, color_punto):
    """Permite cambiar los parámetros de la escena."""
    self.ancho = ancho
    self.alto = alto
    self.color_fondo = color_fondo
    self.punto = np.array(punto, dtype=float)
    self.color_punto = color_punto

def dibujar_escena(self):
    """Dibuja la escena con el rectángulo y el punto."""
    fig, ax = plt.subplots()
    ax.set_xlim(0, self.ancho)
    ax.set_ylim(0, self.alto)
    ax.set_aspect('equal')
    ax.set_facecolor(self.color_fondo)
```

```python
def desplazar_punto(self, x, direccion):
    """Desplaza el punto en una dirección dada por un valor x."""
    movimientos = {
        'derecha': np.array([x, 0]),
        'izquierda': np.array([-x, 0]),
        'arriba': np.array([0, x]),
        'abajo': np.array([0, -x])
    }

    if direccion in movimientos:
        self.punto += movimientos[direccion]
    else:
        raise ValueError("Dirección no válida. Use: 'derecha', 'izquierda', 'arriba' o 'abajo'.")
```

```python
def simular(self, velocidad, direccion, pasos=10):
    """Genera una animación del desplazamiento del punto."""
    fig, ax = plt.subplots()
    ax.set_xlim(0, self.ancho)
    ax.set_ylim(0, self.alto)
    ax.set_aspect('equal')
    ax.set_facecolor(self.color_fondo)

    rect = plt.Rectangle((0, 0), self.ancho, self.alto, color=self.color_fondo)
    ax.add_patch(rect)
    punto_plot, = ax.plot([], [], 'o', color=self.color_punto, markersize=10)

    movimientos = {
        'derecha': np.array([velocidad, 0]),
        'izquierda': np.array([-velocidad, 0]),
        'arriba': np.array([0, velocidad]),
        'abajo': np.array([0, -velocidad])
    }

    if direccion not in movimientos:
        raise ValueError("Dirección no válida. Use: 'derecha', 'izquierda', 'arriba' o 'abajo'.")
```

```python
def actualizar(frame):
    self.punto += movimientos[direccion]
    punto_plot.set_data([self.punto[0]], [self.punto[1]])
    return punto_plot,

ani = animation.FuncAnimation(fig, actualizar, frames=pasos, interval=100, blit=True)
plt.show()
```

Figure 2: Definición de funciones

4 Resultados

Por último se definen los parámetros que imprimen el resultados de nuestras gráficas, ejmeplo:

```
# Uso
escena = Escena(ancho=25, alto=10, color_fondo='yellow', punto=[3, 3], color_punto='red')
escena.dibujar_escena()
escena = Escena(ancho=25, alto=10, color_fondo='blue', punto=[5, 5], color_punto='black')
escena.dibujar_escena()
escena.simular(velocidad=0.5, direccion='derecha', pasos=20)
```

Figure 3: Imprimir resultados

4.1 Gráficas

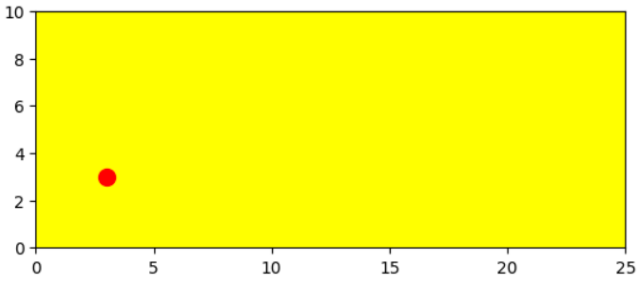

Figure 4: Primera gráfica con coordenadas [3,3]

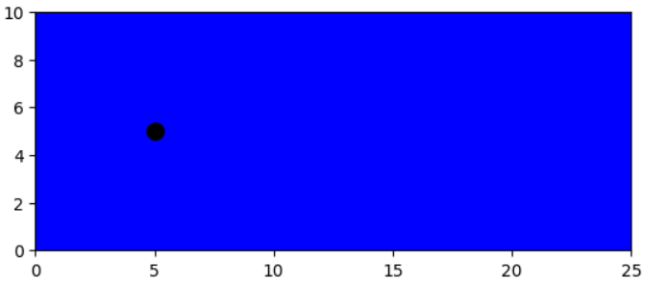

Figure 5: Segunda gráfica con coordenadas [5,5]

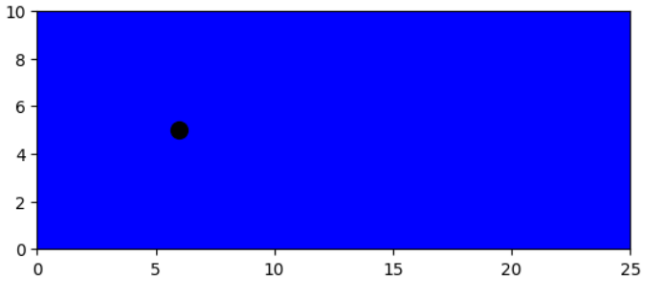

Figure 6: Tercera gráfica con coordenadas [5,5] ligeramente movida hacía la derecha

5 Conclusión

Finalmente, la actividad nos permitió poder familiarizarnos más con las interfaces de desarrollo, a su vez permite la retroalimentación de temas como programación orientada a objetos y Python. La simulación del punto que se desplaza en distintas direcciones según las medidas que le otorguemos permite tener conocimientos base de la materia.