Análisis de Complejidad en Simulación de Partículas

1. Introducción

Este documento analiza un sistema de simulación de partículas que incluye:

- Interacciones gravitatorias
- Detección de colisiones
- Optimización espacial con Quadtree
- Visualización con Pygame

Se enfoca en la complejidad computacional de cada método y las optimizaciones implementadas.

2. Análisis por Clases

2.1. EventHandler

Gestiona eventos de entrada del usuario.

```
class EventHandler:
    @staticmethod
    def procesar_eventos(particula_manager):
        # Implementacion
```

Complejidades:

- `procesar_eventos`: $O(n)$ donde n = número de partículas
  - Búsqueda lineal para detección de clics
  - Otros eventos: $O(1)$

2.2. ParticulaManager

Gestiona la lógica física y almacenamiento de partículas.

```
class ParticulaManager:
    def inicializar_particulas(self):
    def construir_quadtree(self):
    def actualizar_fisica(self):
```

Complejidades:

- `inicializar_particulas`: $O(n)$
- `construir_quadtree`: $O(n \log n)$ promedio, $O(n^2)$ peor caso
- `actualizar_fisica`: $O(n^2)$
  - Doble bucle para fuerzas gravitatorias
  - Manejo de colisiones: $O(n)$

2.3. QuadtreeNode

Implementa la estructura de datos para partición espacial.

```
class QuadtreeNode:
    def insertar(self, particula):
    def buscar_vecinos(self, particula):
    def subdividir(self):
```

Complejidades:

- `insertar`: $O(\log n)$ promedio, $O(n)$ peor caso
- `buscar_vecinos`: $O(\log n)$ promedio
- `subdividir`: $O(1)$

2.4. Renderer

Maneja la visualización gráfica.

```
class Renderer:
    def dibujar_particulas(self):
    def dibujar_linea_minima(self):
```

Complejidades:

- `dibujar_particulas`: $O(n)$
- `dibujar_linea_minima`: $O(1)$

2.5. Simulador

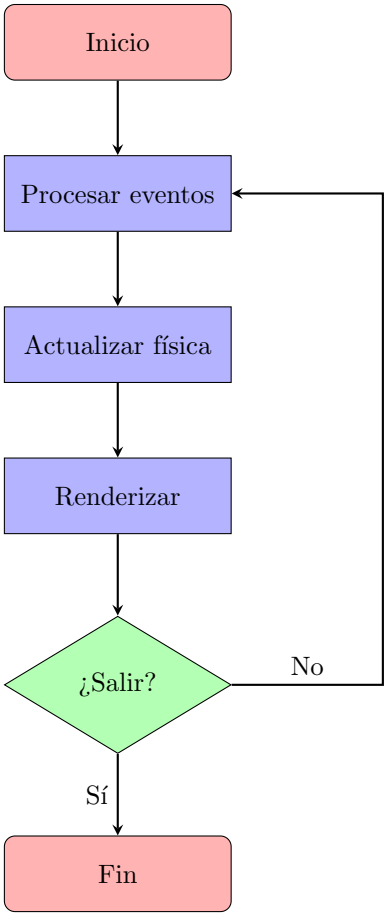
Clase principal que coordina la simulación.

```
class Simulador:
    def ejecutar(self):
```

Complejidad por frame:

- Procesamiento de eventos: $O(n)$
- Actualización física: $O(n^2)$
- Renderizado: $O(n)$
- Total por frame: $O(n^2)$

3. Diagrama de Flujo



4. Conclusiones y Recomendaciones

- **Cuarto de fuerza gravitatoria**: $O(n^2)$ es el principal cuello de botella
- **Quadtree subutilizado**: Actualmente solo se construye pero no se usa para optimizar cálculos físicos
- **Optimizaciones sugeridas**:
  1. Usar Quadtree para reducir complejidad de fuerza gravitatoria a $O(n \log n)$
  2. Implementar Barnes-Hut para aproximación de fuerzas
  3. Paralelizar cálculos con NumPy
  4. Usar estructuras de datos espaciales para colisiones
- **Renderizado**: Adecuado para n pequeños, considerar Level of Detail (LOD) para grandes cantidades

5. Apéndice: Complejidades Totales

Método	Complejidad Promedio	Peor Caso
EventHandler.procesar_eventos	$O(n)$	$O(n)$
ParticulaManager.actualizar_fisica	$O(n^2)$	$O(n^2)$
QuadtreeNode.insertar	$O(\log n)$	$O(n)$
Renderer.dibujar_particulas	$O(n)$	$O(n)$