**Informe de la Solución**

**Título:**  
*Cálculo Distribuido de Áreas y Estimación de Tiempo de Limpieza con Python*

**Autor:**  
*(Tu nombre o el de tu equipo)*

**Fecha:**  
*(Fecha de entrega o de elaboración)*

**1. Enfoque Distribuido Utilizado**

Para resolver el problema de calcular el área de varias zonas de forma simultánea, se ha utilizado un enfoque de programación concurrente. En concreto, se ha implementado con la clase ThreadPoolExecutor del módulo concurrent.futures de la biblioteca estándar de Python.

1. **Definición de las Zonas**  
   Cada zona se representa como una tupla (largo, ancho), almacenada en un diccionario.
2. **Función para Calcular el Área**  
   Se definió la función calcular\_area(largo, ancho), que retorna el producto de ambos valores.
3. **Distribución del Cálculo**  
   Se crea un ThreadPoolExecutor, donde cada tarea (future) ejecuta la función de cálculo de área para una zona distinta. Esto permite que cada área se calcule de forma concurrente.
4. **Recolección de Resultados**  
   Se utiliza concurrent.futures.as\_completed para ir obteniendo los resultados a medida que cada tarea finaliza. Finalmente, se suman las áreas parciales.

Este enfoque simula la idea de “programación distribuida”, al permitir que cada cálculo se ejecute de forma independiente y en paralelo, optimizando el tiempo total de procesamiento cuando se manejan múltiples zonas.

**2. Elección de Herramientas**

* **Lenguaje de Programación:** Python  
  Python se eligió por su sencillez, legibilidad y la amplia disponibilidad de librerías para tareas de concurrencia y multiprocessing.
* **Módulo concurrent.futures:**  
  Se eligió ThreadPoolExecutor para este ejemplo porque es muy sencillo de utilizar y manejar. Además, se integra fácilmente con otras partes del código sin requerir una gestión compleja de procesos.
* **Alternativa con multiprocessing:**  
  De haberse requerido un mayor rendimiento en sistemas con varios núcleos, se podría haber utilizado multiprocessing.Pool. Sin embargo, para la mayoría de casos de ejemplo, ThreadPoolExecutor es suficiente.

**3. Posibles Mejoras o Extensiones Futuras**

1. **Integración con Sistemas de Control de Robots**
   * El script podría comunicarse directamente con el robot aspirador (Pac-Man) para iniciar la limpieza de cada zona de manera automatizada.
   * Se podrían registrar estadísticas en tiempo real (por ejemplo, porcentaje de limpieza completada).
2. **Manejo de Errores y Validaciones**
   * En el caso de que alguna zona tuviera dimensiones inválidas (valores negativos o no numéricos), se podría implementar un sistema de validación previa para evitar cálculos erróneos.
3. **Entrada Dinámica de Datos**
   * En lugar de tener las dimensiones de las zonas fijas en el código, se podría leer desde un archivo de configuración o incluso desde la línea de comandos.
4. **Extensión a Más Zonas**
   * Si en el futuro se añaden más obstáculos o áreas adicionales, el script podría ampliarse fácilmente con más entradas en el diccionario zonas.
5. **Interfaz Gráfica**
   * Una posible evolución del proyecto sería crear una interfaz visual que muestre la habitación y el robot aspirador en tiempo real, permitiendo una experiencia más interactiva.

**Conclusión:**  
El enfoque de concurrencia con ThreadPoolExecutor facilita el cálculo de áreas de forma distribuida y, en conjunto con la sencilla fórmula de estimación de tiempo de limpieza, brinda una solución efectiva para planificar la limpieza de la habitación del Profesor Polvotrón. Con unas pocas modificaciones, este sistema se puede integrar con robots aspiradores reales, adaptarse a nuevos escenarios y manejar un número mayor de zonas.