Informe:

Mi práctica se basa en un juego sencillo en el que controlas una roomba con wasd, tu objetivo es eliminar todas las pelusas de la sala. Para ello puedes tanto pasar por encima de ellas o dispararles balas con click derecho. La posición de spawn de las pelusas es aleatoria, y tu posición de spawn también. El juego termina cuando has eliminado todas las pelusas.

He separado la entrega en distintos archivos separados por las clases roomba, bullet, la función de pelusas, la configuración de valores iniciales y el main.

El enfoque distribuido utilizado es el uso de concurrent.futures.ThreadPoolExecutor para la ejecución de hilos paralelos, asegurando que el spawn de la Roomba y las pelusas ocurra de forma concurrente.

Las tareas que se ejecutan concurrentemente son:

Spawn de la Roomba en una posición aleatoria.

Spawn de las pelusas en posiciones aleatorias.

La forma en la que se hace uso de la concurrencia es la siguiente:

En main()

Se crea un pool de hilos.

Executor.submit ejecuta la función spawn_roomba/spawn_pelusas en hilos separados.

As_completed() procesa las tareas a medida que se completan.

Future.result() obtiene el resultado de la tarea.

Si la tarea se completa, el resultado se asigna a roomba_pos o pelusas

Si la tarea no se completa, se intenta obtener el resultado de la tarea y si no lo obtiene salta una excpeción.

Una vez todas las tareas se han completado, se muestra por pantalla.

Algunas posibles extensiones futuras son:

Hacer la detección de colisiones concurrente en un hilo separado para mejorar el rendimiento del juego.

Manejar los inputs de teclado y ratón del jugador de forma concurrente en un hilo separado.