¡Vamos a ordenarlo! Aquí tienes un **checklist** directo contra los objetivos del instructivo, marcando lo que YA tienes y lo que FALTA hacer.

Checklist por objetivos

1) Representar el problema (estado, sucesores, meta)

- ✅ **Estado & sucesores (N-puzzle):** environments/sliding_graph.py (grafo perezoso, mueve 0 en ↑↓←→, costo=1).
- ✅ **Estado objetivo canónico:** (1..N^2−1, 0) — ya usas ese criterio en tus módulos.
- ☐ **Chequeo de solubilidad** (paridad/inversiones) antes de lanzar búsqueda para los experimentos masivos.
  Acción: agregar función es_soluble(estado, n).

2) Definir funciones sucesores (restricciones)

- ✅ Generación de vecinos válida (limita bordes, intercambia con 0).
- ✅ Costo uniforme (=1).

3) Definir tres heurísticas admisibles

- ✅ **Hamming** (h_hamming_factory).
- ✅ **Manhattan** (h_manhattan_factory).
- ✅ **Manhattan + Linear Conflict** (h_manhattan_linear_conflict_factory).
- ✅ (Bonus) **Gaschnig** (h_gaschnig_factory).

4) Implementar Greedy y A* con las tres heurísticas

- ✅ **Greedy** en algorithms/informed.py.
- ✅ **A*** en algorithms/informed.py.
- ☐ **Selector por CLI** para elegir algoritmo y heurística al vuelo.
  Acción: en main_agente_informado_sliding_v2.py, agregar flags --algo {greedy,a*} y --h {hamming,manhattan,linear_conflict} y cablearlos.

5) Generar 1000 estados iniciales aleatorios (válidos) y resolver con Greedy/A*

- ☐ **Generador de estados aleatorios** que garantice solubilidad.
  Acción: función generar_estado(n) + filtro es_soluble.

- [ ] **Runner de experimentos** para iterar 1000 casos por combinación (algoritmo × heurística).
  Acción: script experiments/benchmark.py.

6) Promedio del número de pasos

- [ ] Calcular y reportar **promedio de pasos** por (algoritmo, heurística).
  Acción: acumular len(camino)-1 y promediar.

7) Tiempo promedio

- [ ] Medir y reportar **tiempo promedio** (ms) por (algoritmo, heurística).
  Acción: time.perf_counter() alrededor de cada llamada.

8) Completitud (¿siempre encuentran solución?)

- [ ] Registrar **tasa de éxito**: (#resueltos / 1000) por combinación.
  Acción: contar fallos/éxitos y reportar.

9) Seleccionar la mejor heurística con evidencia

- [ ] Concluir **cuál es mejor** según: menor pasos, menor tiempo, y tasa de éxito (empates → criterio claro).
  Acción: tabla + breve análisis.

10) Integración con Pygame (jugar y ejecutar agente en cualquier momento)

- [ ] **Hook "Resolver ahora"** (tecla, p.ej. R): computa camino con A*/heurística seleccionada y **reproduce animación**.
- [ ] **Hook "Sugerir siguiente movimiento"** (tecla, p.ej. N): calcula un paso del plan y lo aplica.
- [ ] **UI feedback**: mostrando g, h, f, pasos restantes, y/o tiempo.
- [ ] **Bloqueo no-bloqueante** o **cálculo en lote** del plan**:** que no congele el loop (p. ej. precomputar la solución y luego animar frame a frame).

---

4) Implementar Greedy y A* con las tres heurísticas

☐ **Selector por CLI:**

TERMINADO ✅

python main_agente_informado_sliding_v2Practica.py --algo a* --h manhattan

python main_agente_informado_sliding_v2Practica.py --algo greedy --h hamming

python main_agente_informado_sliding_v2Practica.py --algo a* --h linear_conflict

5) Generar 1000 estados iniciales aleatorios (válidos) y resolver con Greedy/A*

- ☐ **Generador de estados aleatorios** que garantice solubilidad.
  Acción: función generar_estado(n) + filtro es_soluble.

  TERMINADO ✅ en algorithms/npuzzle_utils.py
- ☐ **Runner de experimentos** para iterar 1000 casos por combinación (algoritmo × heurística).
  Acción: script AGENTES/practica_sliding_benchmark.py.
- TERMINADO ✅

PASOS 6- 9 TERMINADOS.

python practica_sliding_benchmark.py --n 3 --count 1000 --shuffle 100 --out resultsbenchmark.csv