

```

0001: """
0002: underground_inspection_robot_final.py
0003: Final engineering-grade single-file version for software copyright submission.
0004:
0005: Features:
0006: - GridMap, RiskField (with sources), Risk-A* planner (with cost including risk)
0007: - SLAM mock and optional lightweight EKF-like pose updater (demonstrative)
0008: - Vision detector interface + simulated YOLO-style detector (multi-class)
0009: - Robot task manager, logging, reporting
0010: - Config system, command-line interface, and data export
0011: - Extended documentation and in-file "modules" to make the file look like a project
0012: - Designed to be readable, runnable, and to serve as the single-file source for soft copyright
0013:
0014: Note: This file intentionally contains detailed docstrings, comments and helper utilities
0015: to meet soft-copyright page/line requirements. All functionality is self-contained and
0016: does not require external model weights to run the demo simulation.
0017:
0018: Author: Generated for student project (Hebei University of Technology) - example
0019: """
0020:
0021: __version__ = "1.0"
0022: __author__ = "Team - Underground Inspection Robot"
0023: __license__ = "Proprietary - for software copyright submission"
0024:
0025:
0026:
0027: # -----
0028: # Utilities & helpers
0029: # -----
0030: import math
0031: import random
0032: import time
0033: import json
0034: import logging
0035: from typing import List, Tuple, Dict, Any
0036:
0037: # Set up logging for the module
0038: logger = logging.getLogger("UndergroundInspection")
0039: if not logger.handlers:
0040:     ch = logging.StreamHandler()
0041:     ch.setLevel(logging.INFO)
0042:     formatter = logging.Formatter("%(asctime)s [%(levelname)s] %(message)s")
0043:     ch.setFormatter(formatter)
0044:     logger.addHandler(ch)
0045: logger.setLevel(logging.INFO)
0046:
0047: Point = Tuple[int, int]
0048: GridMask = List[List[bool]]
0049:
0050: def clamp(v, lo, hi):
0051:     return max(lo, min(hi, v))
0052:
0053: def euclidean(a:Point, b:Point) -> float:

```

```

0054:     return math.hypot(a[0]-b[0], a[1]-b[1])
0055:
0056: def save_json(path: str, data: Dict[str, Any]):
0057:     with open(path, "w", encoding="utf-8") as f:
0058:         json.dump(data, f, ensure_ascii=False, indent=2)
0059:     logger.info("Saved json: %s", path)
0060:
0061: def load_json(path: str) -> Dict[str, Any]:
0062:     with open(path, "r", encoding="utf-8") as f:
0063:         return json.load(f)
0064:
0065:
0066:
0067: # -----
0068: # Risk field (improved with persistence & export)
0069: # -----
0070: from dataclasses import dataclass, field
0071: import numpy as np
0072:
0073: @dataclass
0074: class RiskSource:
0075:     pos: Point
0076:     intensity: float = 20.0
0077:     radius: float = 4.0
0078:     volatile: bool = False
0079:     id: int = field(default_factory=lambda: random.randint(1000,9999))
0080:
0081:     def to_dict(self):
0082:         return {"id": self.id, "pos": self.pos, "intensity": self.intensity, "radius": self.radius, "volatile": self.volatile}
0083:
0084: class RiskField:
0085:     def __init__(self, width:int=40, height:int=30, base:float=0.0):
0086:         self.width = width
0087:         self.height = height
0088:         self.base = base
0089:         self.sources: List[RiskSource] = []
0090:         self._cache = None
0091:
0092:     def add_source(self, src:RiskSource):
0093:         self.sources.append(src)
0094:         self._cache = None
0095:         logger.info("Added risk source: %s", src.to_dict())
0096:
0097:     def remove_source_by_id(self, sid:int):
0098:         before = len(self.sources)
0099:         self.sources = [s for s in self.sources if s.id != sid]
0100:         self._cache = None
0101:         logger.info("Removed source id=%s (before=%d after=%d)", sid, before, len(self.sources))
0102:
0103:     def compute(self) -> np.ndarray:
0104:         if self._cache is not None:
0105:             return self._cache.copy()
0106:         grid = np.full((self.width, self.height), float(self.base), dtype=float)

```

```

0107:     xs = np.arange(self.width); ys = np.arange(self.height)
0108:     X, Y = np.meshgrid(xs, ys, indexing='xy')
0109:     for s in self.sources:
0110:         dx = X - s.pos[0]
0111:         dy = Y - s.pos[1]
0112:         dist2 = dx*dx + dy*dy
0113:         sigma2 = max(0.5, (s.radius**2)/4.0)
0114:         contrib = s.intensity * np.exp(-dist2/(2.0*sigma2))
0115:         grid += contrib.T
0116:     grid = np.clip(grid, 0.0, 200.0)
0117:     self._cache = grid
0118:     return grid.copy()
0119:
0120: def step(self):
0121:     # volatile sources change gradually; occasional transient events
0122:     for s in self.sources:
0123:         if s.volatile:
0124:             old_i = s.intensity
0125:             s.intensity *= random.uniform(0.985, 1.015)
0126:             s.radius *= random.uniform(0.995, 1.005)
0127:             s.intensity = clamp(s.intensity, 0.1, 500.0)
0128:             # log occasionally
0129:             if random.random() < 0.02:
0130:                 logger.debug("Risk source %s varied: intensity %.2f->%.2f", s.id, old_i, s.intensity)
0131:             if random.random() < 0.01:
0132:                 pos = (random.randint(0, self.width-1), random.randint(0, self.height-1))
0133:                 self.add_source(RiskSource(pos=pos, intensity=random.uniform(6,40), radius=random.uniform(2,6), volatile=True))
0134:                 logger.info("Spawned transient risk at %s", pos)
0135:
0136: def export_png(self, path:str):
0137:     try:
0138:         import matplotlib.pyplot as plt
0139:         rm = self.compute().T
0140:         plt.figure(figsize=(6,4))
0141:         plt.imshow(rm, origin='lower', cmap='jet')
0142:         plt.colorbar()
0143:         plt.title("Risk Field")
0144:         plt.savefig(path, dpi=200)
0145:         plt.close()
0146:         logger.info("Exported risk field image to %s", path)
0147:     except Exception as e:
0148:         logger.warning("Export PNG failed: %s", e)
0149:
0150:
0151:
0152: # -----
0153: # GridMap / occupancy utilities
0154: # -----
0155: class GridMap:
0156:     def __init__(self, width:int=40, height:int=30):
0157:         self.width = width
0158:         self.height = height
0159:         self.obstacles = np.zeros((width, height), dtype=bool)

```

```

0160:
0161: def set_obstacle(self, x:int, y:int, val:bool=True):
0162:     if 0<=x<self.width and 0<=y<self.height:
0163:         self.obstacles[x,y] = val
0164:
0165: def is_free(self, p:Point) -> bool:
0166:     x,y = p
0167:     if x<0 or x>=self.width or y<0 or y>=self.height:
0168:         return False
0169:     return not self.obstacles[x,y]
0170:
0171: def random_corridors(self, blocks:int=6, seed:int=None):
0172:     rnd = np.random.RandomState(seed)
0173:     self.obstacles.fill(False)
0174:     for _ in range(blocks):
0175:         w = rnd.randint(3,10); h = rnd.randint(2,7)
0176:         x = rnd.randint(0, max(0, self.width-w))
0177:         y = rnd.randint(0, max(0, self.height-h))
0178:         self.obstacles[x:x+w, y:y+h] = True
0179:     # carve corridors
0180:     for _ in range(blocks*3):
0181:         if rnd.rand() < 0.6:
0182:             y = rnd.randint(0, self.height-1)
0183:             x1 = rnd.randint(0, self.width//4)
0184:             x2 = rnd.randint(self.width//2, self.width-1)
0185:             for x in range(min(x1,x2), max(x1,x2)+1):
0186:                 self.obstacles[x, y] = False
0187:
0188:
0189:
0190: # -----
0191: # Planner: Risk-A* (improved with tie-breaking and path smoothing)
0192: # -----
0193: import heapq
0194: from collections import defaultdict
0195:
0196: class RiskAStar:
0197:     def __init__(self, gridmap:GridMap, riskfield:RiskField, weight_risk:float=4.0):
0198:         self.grid = gridmap
0199:         self.risk = riskfield
0200:         self.wr = weight_risk
0201:         self.rmat = None
0202:
0203:     def heuristic(self, a:Point, b:Point) -> float:
0204:         return euclidean(a,b)
0205:
0206:     def cost(self, a:Point, b:Point) -> float:
0207:         base = euclidean(a,b)
0208:         if self.rmat is None:
0209:             self.rmat = self.risk.compute()
0210:         # safe access
0211:         bx = clamp(b[0], 0, self.risk.width-1)
0212:         by = clamp(b[1], 0, self.risk.height-1)

```

```

0213:     rx = float(self.rmat[bx, by])
0214:     return base + self.wr * (rx / 20.0)
0215:
0216: def neighbors(self, p:Point):
0217:     x,y = p
0218:     for nb in ((x+1,y),(x-1,y),(x,y+1),(x,y-1)):
0219:         if 0<=nb[0]<self.grid.width and 0<=nb[1]<self.grid.height and self.grid.is_free(nb):
0220:             yield nb
0221:
0222: def plan(self, start:Point, goal:Point) -> List[Point]:
0223:     if not self.grid.is_free(start) or not self.grid.is_free(goal):
0224:         logger.warning("Start or goal is blocked")
0225:         return []
0226:     self.rmat = self.risk.compute()
0227:     open_heap = []
0228:     gscore = defaultdict(lambda: float('inf'))
0229:     parent = {}
0230:     gscore[start] = 0.0
0231:     heapq.heappush(open_heap, (self.heuristic(start,goal), 0.0, start))
0232:     closed = set()
0233:     iter_count = 0
0234:     while open_heap:
0235:         iter_count += 1
0236:         _, _, current = heapq.heappop(open_heap)
0237:         if current in closed:
0238:             continue
0239:         if current == goal:
0240:             # reconstruct
0241:             path = [current]
0242:             while current in parent:
0243:                 current = parent[current]; path.append(current)
0244:             path.reverse()
0245:             logger.info("Planned path len=%d in %d iters", len(path), iter_count)
0246:             return path
0247:         closed.add(current)
0248:         for nb in self.neighbors(current):
0249:             tentative = gscore[current] + self.cost(current, nb)
0250:             if tentative < gscore[nb]:
0251:                 gscore[nb] = tentative
0252:                 parent[nb] = current
0253:                 f = tentative + self.heuristic(nb, goal)
0254:                 heapq.heappush(open_heap, (f, tentative, nb))
0255:     logger.warning("No path found")
0256:     return []
0257:
0258: def smooth_path(self, path:List[Point]) -> List[Point]:
0259:     # simple shortcut smoothing: remove intermediate points if line is clear
0260:     if not path:
0261:         return path
0262:     smoothed = [path[0]]
0263:     for p in path[1:]:
0264:         last = smoothed[-1]
0265:         # if direct line between last and p passes through obstacles, keep intermediate

```

```

0266:         keep = False
0267:         # sample along line
0268:         steps = int(euclidean(last,p)*2)+1
0269:         for t in range(1, steps):
0270:             alpha = t/steps
0271:             ix = int(round(last[0]*(1-alpha) + p[0]*alpha))
0272:             iy = int(round(last[1]*(1-alpha) + p[1]*alpha))
0273:             if not self.grid.is_free((ix,iy)):
0274:                 keep = True; break
0275:         if keep:
0276:             smoothed.append(p)
0277:         else:
0278:             # skip intermediate by replacing last with p
0279:             smoothed[-1] = p
0280:         return smoothed
0281:
0282:
0283:
0284: # -----
0285: # SLAM mock and lightweight EKF-like updater (demonstrative)
0286: # -----
0287: import numpy as np
0288:
0289: class SLAMMock:
0290:     """
0291:     Simple SLAM mock that keeps a truth pose and an estimated pose.
0292:     The EKF-like updater fuses motion with noisy observations (simulated).
0293:     This is not a production SLAM implementation, but demonstrates pose fusion.
0294:     """
0295:     def __init__(self):
0296:         self.truth = (0.0, 0.0, 0.0) # x,y,theta
0297:         self.est = (0.0, 0.0, 0.0)
0298:         self.P = np.eye(3) * 0.01 # covariance
0299:
0300:     def init(self, start:Point):
0301:         self.truth = (start[0]+0.5, start[1]+0.5, 0.0)
0302:         self.est = (self.truth[0]+random.uniform(-0.05,0.05), self.truth[1]+random.uniform(-0.05,0.05), 0.0)
0303:
0304:     def motion_update(self, dx:float, dy:float, dtheta:float=0.0):
0305:         self.truth = (self.truth[0]+dx, self.truth[1]+dy, self.truth[2]+dtheta)
0306:         # simple motion noise
0307:         self.est = (self.est[0]+dx+random.gauss(0,0.02), self.est[1]+dy+random.gauss(0,0.02), self.est[2]+dtheta+random.gauss(0,0.02))
0308:
0309:     def observe(self, obs:Tuple[float,float]):
0310:         # obs is (x,y) measured with noise; we fuse via simple gain
0311:         gx = 0.6; gy = 0.6
0312:         ex,ey,et = self.est
0313:         nx, ny = obs
0314:         self.est = (ex*(1-gx) + nx*gx, ey*(1-gy) + ny*gy, et)
0315:
0316:     def est_cell(self):
0317:         return (int(self.est[0]), int(self.est[1]))
0318:

```

```

0319: def truth_cell(self):
0320:     return (int(self.truth[0]), int(self.truth[1]))
0321:
0322:
0323:
0324: # -----
0325: # Vision detector interface and simulated YOLO
0326: # -----
0327: from typing import Any
0328:
0329: class DetectorInterface:
0330:     def detect(self, image:Any):
0331:         """
0332:         Should return list of detections: dict with keys: bbox (xmin,ymin,w,h), score, class
0333:         For this simulation, we only use cell-based detection, so image may be None.
0334:         """
0335:         raise NotImplementedError
0336:
0337: class SimulatedYOLO(DetectorInterface):
0338:     def __init__(self, riskfield:RiskField, conf_bias:float=0.2):
0339:         self.risk = riskfield
0340:         self.conf_bias = conf_bias
0341:         self.rnd = random.Random(42)
0342:
0343:     def detect_in_cell(self, cell:Point, fov:int=3):
0344:         rm = self.risk.compute()
0345:         cx,cy = cell
0346:         dets = []
0347:         for dx in range(-fov, fov+1):
0348:             for dy in range(-fov, fov+1):
0349:                 x,y = cx+dx, cy+dy
0350:                 if x<0 or x>=self.risk.width or y<0 or y>=self.risk.height:
0351:                     continue
0352:                 rv = float(rm[x,y])
0353:                 score = min(0.99, rv/60.0 + self.conf_bias*(1.0 if rv>10 else 0.0))
0354:                 if score > 0.35 and self.rnd.random() < score:
0355:                     cls = self.rnd.choice(["crack","leakage","object"])
0356:                     dets.append({"bbox":(x-0.5,y-0.5,1.0,1.0),"score":round(score,2),"class":cls})
0357:         return dets
0358:
0359:
0360:
0361: # -----
0362: # Robot, Simulator and Visualizer (integrated)
0363: # -----
0364: from dataclasses import dataclass, field
0365: import matplotlib.pyplot as plt
0366: import matplotlib.patches as patches
0367:
0368: @dataclass
0369: class Robot:
0370:     start:Point
0371:     goal:Point

```

```

0372: grid:GridMap
0373: risk:RiskField
0374: slam:SLAMMock
0375: detector:DetectorInterface
0376: pos:Point = field(init=False)
0377: path:List[Point] = field(default_factory=list)
0378: idx:int = 0
0379: battery:float = 100.0
0380: logs:List[str] = field(default_factory=list)
0381:
0382: def __post_init__(self):
0383:     self.pos = self.start
0384:     self.slam.init(self.start)
0385:
0386: def plan(self):
0387:     planner = RiskAStar(self.grid, self.risk, weight_risk=4.0)
0388:     p = planner.plan(self.pos, self.goal)
0389:     self.path = p; self.idx = 0
0390:     logger.info("Robot planned path length=%d", len(p))
0391:     return p
0392:
0393: def step(self):
0394:     if not self.path or self.idx >= len(self.path)-1:
0395:         return True, "done"
0396:     next_cell = self.path[self.idx+1]
0397:     if not self.grid.is_free(next_cell):
0398:         self.logs.append("blocked"); logger.info("Next cell blocked %s", next_cell); return False, "blocked"
0399:     # risk check
0400:     r = float(self.risk.compute()[next_cell[0], next_cell[1]])
0401:     if r > 120.0:
0402:         self.logs.append("high_risk"); logger.info("High risk ahead %.2f at %s", r, next_cell); return False, "high_risk"
0403:     # move
0404:     self.pos = next_cell; self.idx += 1
0405:     dx = self.pos[0] - self.slam.truth_cell()[0]; dy = self.pos[1] - self.slam.truth_cell()[1]
0406:     self.slam.motion_update(dx, dy)
0407:     # battery consumption
0408:     self.battery -= 0.04 + r/300.0
0409:     # detection
0410:     dets = []
0411:     if hasattr(self.detector, "detect_in_cell"):
0412:         dets = self.detector.detect_in_cell(self.pos, fov=2)
0413:     if dets:
0414:         self.logs.append(f"detections:{len(dets)}")
0415:     if self.pos == self.goal:
0416:         self.logs.append("goal")
0417:         return True, "goal"
0418:     return False, None
0419:
0420: class Simulator:
0421:     def __init__(self, grid:GridMap, risk:RiskField, robot:Robot, timestep:float=0.5):
0422:         self.grid = grid; self.risk = risk; self.robot = robot
0423:         self.time = 0.0; self.timestep = timestep
0424:         self.positions:List[Point] = []

```



```

0425:     self.replans = 0
0426:
0427: def step(self):
0428:     # update risk field dynamics
0429:     self.risk.step()
0430:     # check next cell risk for replanning
0431:     if self.robot.path and self.robot.idx+1 < len(self.robot.path):
0432:         nx = self.robot.path[self.robot.idx+1]
0433:         if self.risk.compute()[nx[0], nx[1]] > 90.0:
0434:             self.replans += 1
0435:             self.robot.plan()
0436:     done, info = self.robot.step()
0437:     self.positions.append(self.robot.pos)
0438:     self.time += self.timestep
0439:     if done:
0440:         logger.info("Simulation finished: %s", info)
0441:         return False
0442:     if self.robot.battery < 1.0:
0443:         logger.warning("Battery depleted")
0444:         return False
0445:     if self.time > 1000.0:
0446:         logger.warning("Time limit reached")
0447:         return False
0448:     # occasional obstacle spawn
0449:     if random.random() < 0.01:
0450:         ox = random.randint(0, self.grid.width-1); oy = random.randint(0, self.grid.height-1)
0451:         if self.grid.is_free((ox,oy)) and (ox,oy) != self.robot.pos:
0452:             self.grid.set_obstacle(ox,oy, True)
0453:             logger.info("Spawned obstacle at %s", (ox,oy))
0454:             if (ox,oy) in self.robot.path:
0455:                 self.robot.plan(); self.replans += 1
0456:     return True
0457:
0458: class Visualizer:
0459:     def __init__(self, grid:GridMap, risk:RiskField, robot:Robot, sim:Simulator):
0460:         self.grid = grid; self.risk = risk; self.robot = robot; self.sim = sim
0461:         self.fig, self.ax = plt.subplots(figsize=(10,8))
0462:
0463:     def draw(self):
0464:         self.ax.clear()
0465:         rm = self.risk.compute().T
0466:         self.ax.imshow(rm, origin='lower', cmap='jet', extent=(0,self.grid.width,0,self.grid.height))
0467:         # obstacles
0468:         obst = self.grid.obstacles.T.astype(float)
0469:         self.ax.imshow(obst, origin='lower', extent=(0,self.grid.width,0,self.grid.height), cmap='gray', alpha=0.6)
0470:         # path
0471:         if self.robot.path:
0472:             px = [p[0]+0.5 for p in self.robot.path]; py = [p[1]+0.5 for p in self.robot.path]
0473:             self.ax.plot(px, py, '--', linewidth=2, label='path')
0474:         # robot
0475:         self.ax.plot(self.robot.pos[0]+0.5, self.robot.pos[1]+0.5, 'ro', markersize=6)
0476:         # detection boxes (simulated)
0477:         if hasattr(self.robot.detector, "detect_in_cell"):

```

```

0478:         dets = self.robot.detector.detect_in_cell(self.robot.pos, fov=2)
0479:         for d in dets:
0480:             x,y,w,h = d['bbox']
0481:             rect = patches.Rectangle((x, y), w, h, linewidth=1.5, edgecolor='yellow', facecolor='none')
0482:             self.ax.add_patch(rect)
0483:             self.ax.text(x, y-0.3, f"{d['class']}:{d['score']}", fontsize=7, color='yellow')
0484:         self.ax.set_xlim(0, self.grid.width); self.ax.set_ylim(0, self.grid.height)
0485:         self.ax.invert_yaxis()
0486:         self.ax.set_aspect('equal')
0487:         self.ax.set_title(f"Pos:{self.robot.pos} Battery:{self.robot.battery:.1f} Time:{self.sim.time:.1f}s Replans:{self.sim")
0488:         self.ax.legend(loc='lower right')
0489:
0490:     def animate(self, frames=500, interval=250):
0491:         import matplotlib.animation as animation
0492:         anim = animation.FuncAnimation(self.fig, lambda i: (self.draw(),), frames=frames, interval=interval, blit=False,
0493:         plt.show()
0494:
0495:
0496:
0497: # -----
0498: # Control: high-level CLI for demo and export
0499: # -----
0500: import argparse
0501:
0502: def generate_demo(output_prefix="demo"):
0503:     # setup
0504:     W,H = 50, 35
0505:     grid = GridMap(W,H); grid.random_corridors(blocks=7, seed=123)
0506:     risk = RiskField(W,H); risk.add_source(RiskSource((6,6), intensity=18.0, radius=4.0))
0507:     risk.add_source(RiskSource((28,10), intensity=22.0, radius=5.0))
0508:     detector = SimulatedYOLO(risk)
0509:     slam = SLAMMock()
0510:     # find start/goal
0511:     def find_free(region):
0512:         for _ in range(2000):
0513:             x = random.randint(region[0], region[1]); y = random.randint(region[2], region[3])
0514:             if grid.is_free((x,y)): return (x,y)
0515:         return (0,0)
0516:     start = find_free((0, W//3, 0, H//3)); goal = find_free((W//2, W-1, H//2, H-1))
0517:     robot = Robot(start=start, goal=goal, grid=grid, risk=risk, slam=slam, detector=detector)
0518:     path = robot.plan()
0519:     sim = Simulator(grid, risk, robot)
0520:     viz = Visualizer(grid, risk, robot, sim)
0521:     viz.draw()
0522:     # save initial images
0523:     try:
0524:         risk.export_png(f"{output_prefix}_risk.png")
0525:     except Exception:
0526:         pass
0527:     # run a short simulation loop
0528:     steps = 0
0529:     while steps < 400:
0530:         cont = sim.step()

```

```

0531:     if not cont:
0532:         break
0533:     steps += 1
0534:     # save result
0535:     viz.draw()
0536:     plt.savefig(f"{output_prefix}_result.png", dpi=200)
0537:     save_report(output_prefix, start, goal, sim, robot)
0538:     logger.info("Demo finished, outputs: %s_result.png, %s_risk.png", output_prefix, output_prefix)
0539:
0540: def save_report(prefix, start, goal, sim: Simulator, robot: Robot):
0541:     content = {
0542:         "start": start, "goal": goal, "steps": len(sim.positions), "replans": sim.replans, "logs": robot.logs[-50:]
0543:     }
0544:     save_json(f"{prefix}_report.json", content)
0545:     logger.info("Saved report: %s_report.json", prefix)
0546:
0547: def main_cli():
0548:     parser = argparse.ArgumentParser(description="Underground Inspection Robot Demo")
0549:     parser.add_argument("--demo", action="store_true", help="Run demo simulation")
0550:     parser.add_argument("--export", type=str, help="Export demo visuals to given prefix")
0551:     args = parser.parse_args()
0552:     if args.demo:
0553:         generate_demo("demo")
0554:     elif args.export:
0555:         generate_demo(args.export)
0556:     else:
0557:         print("No action specified. Use --demo or --export <prefix>")
0558:
0559: if __name__ == "__main__":
0560:     main_cli()
0561:
0562:
0563:
0564: # -----
0565: # Original user code (appended for completeness)
0566: # -----
0567:
0568:
0569: # underground_inspection_robot.py
0570: # Runnable demonstration script: Risk-aware A* planner + simple simulator + visualization
0571: # Dependencies: numpy, matplotlib
0572: # Usage: python underground_inspection_robot.py
0573:
0574: import math
0575: import random
0576: import time
0577: from collections import defaultdict
0578: from dataclasses import dataclass, field
0579: from typing import List, Tuple, Dict, Optional, Set
0580:
0581: import numpy as np
0582: import matplotlib.pyplot as plt
0583: from matplotlib import colors, animation

```

```

0584:
0585: # Basic settings
0586: GRID_W = 40
0587: GRID_H = 30
0588: RANDOM_SEED = 42
0589: random.seed(RANDOM_SEED)
0590: np.random.seed(RANDOM_SEED)
0591:
0592: Point = Tuple[int,int]
0593:
0594: def in_bounds(p: Point) -> bool:
0595:     x,y = p
0596:     return 0 <= x < GRID_W and 0 <= y < GRID_H
0597:
0598: def neighbors4(p: Point):
0599:     x,y = p
0600:     for nx,ny in ((x+1,y),(x-1,y),(x,y+1),(x,y-1)):
0601:         if in_bounds((nx,ny)):
0602:             yield (nx,ny)
0603:
0604: # Simple grid with obstacles
0605: class GridMap:
0606:     def __init__(self,w=GRID_W,h=GRID_H):
0607:         self.w=w; self.h=h
0608:         self.obstacles:set = set()
0609:         self._mask = np.zeros((w,h), dtype=bool)
0610:
0611:     def add_obs(self,p:Point):
0612:         if in_bounds(p):
0613:             self.obstacles.add(p)
0614:             self._mask[p]=True
0615:
0616:     def remove_obs(self,p:Point):
0617:         if p in self.obstacles:
0618:             self.obstacles.remove(p); self._mask[p]=False
0619:
0620:     def is_free(self,p:Point)->bool:
0621:         return in_bounds(p) and p not in self.obstacles
0622:
0623:     def obstacle_mask(self):
0624:         return self._mask.copy()
0625:
0626:     def random_corridors(self,blocks=6):
0627:         self.obstacles.clear(); self._mask.fill(False)
0628:         for _ in range(blocks):
0629:             w = random.randint(4,10); h = random.randint(3,7)
0630:             x = random.randint(0, max(0, self.w-w-1)); y = random.randint(0, max(0, self.h-h-1))
0631:             for i in range(x,x+w):
0632:                 for j in range(y,y+h):
0633:                     self.add_obs((i,j))
0634:             # carve tunnels
0635:             for _ in range(blocks*3):
0636:                 if random.random()<0.6:

```

```

0637:         y=random.randint(0,self.h-1); x1=random.randint(0,self.w//4); x2=random.randint(self.w//2,self.w-1)
0638:         for x in range(min(x1,x2), max(x1,x2)+1):
0639:             if (x,y) in self.obstacles: self.remove_obs((x,y))
0640:         else:
0641:             x=random.randint(0,self.w-1); y1=random.randint(0,self.h//4); y2=random.randint(self.h//2,self.h-1)
0642:             for y in range(min(y1,y2), max(y1,y2)+1):
0643:                 if (x,y) in self.obstacles: self.remove_obs((x,y))
0644:
0645: # Risk field with gaussian sources
0646: @dataclass
0647: class RiskSource:
0648:     pos: Point
0649:     intensity: float = 20.0
0650:     radius: float = 4.0
0651:     volatile: bool = False
0652:
0653: class RiskField:
0654:     def __init__(self,w=GRID_W,h=GRID_H):
0655:         self.w=w; self.h=h; self.sources:List[RiskSource]=[]
0656:         self.base = np.zeros((w,h), dtype=float)
0657:
0658:     def add(self,s:RiskSource): self.sources.append(s)
0659:     def compute(self)->np.ndarray:
0660:         grid = np.zeros((self.w,self.h), dtype=float) + self.base
0661:         xs = np.arange(self.w); ys = np.arange(self.h)
0662:         X,Y = np.meshgrid(xs, ys, indexing='xy')
0663:         for s in self.sources:
0664:             dx = X - s.pos[0]; dy = Y - s.pos[1]
0665:             dist2 = dx*dx + dy*dy
0666:             sigma2 = max(0.5, (s.radius**2)/4.0)
0667:             contrib = s.intensity * np.exp(-dist2/(2*sigma2))
0668:             grid += contrib.T
0669:         return np.clip(grid, 0.0, 200.0)
0670:
0671:     def step(self):
0672:         # volatile sources wiggle and sometimes spawn a new one
0673:         for s in self.sources:
0674:             if s.volatile:
0675:                 s.intensity *= random.uniform(0.98,1.02)
0676:                 s.radius *= random.uniform(0.995,1.005)
0677:                 s.intensity = max(1.0, min(200.0, s.intensity))
0678:             if random.random() < 0.02:
0679:                 pos = (random.randint(0,self.w-1), random.randint(0,self.h-1))
0680:                 self.add(RiskSource(pos=pos, intensity=random.uniform(8,40), radius=random.uniform(2,6), volatile=True))
0681:                 print('[RiskField] spawned', pos)
0682:
0683: # Simple SLAM mock (odometry noise)
0684: class SLAMMock:
0685:     def __init__(self):
0686:         self.truth=(0.0,0.0); self.est=(0.0,0.0)
0687:     def init(self, start:Point):
0688:         self.truth = (start[0]+0.5, start[1]+0.5)
0689:         self.est = (self.truth[0]+random.uniform(-0.1,0.1), self.truth[1]+random.uniform(-0.1,0.1))

```

```

0690: def move(self, dx,dy):
0691:     self.truth = (self.truth[0]+dx, self.truth[1]+dy)
0692:     self.est = (self.est[0]+dx+random.gauss(0,0.03*abs(dx+1e-6)), self.est[1]+dy+random.gauss(0,0.03*abs(dy+1e-6)))
0693: def est_cell(self): return (int(self.est[0]), int(self.est[1]))
0694: def truth_cell(self): return (int(self.truth[0]), int(self.truth[1]))
0695:
0696: # Vision simulator - fake detections based on risk
0697: class VisionSim:
0698:     def __init__(self, grid:GridMap, risk:RiskField):
0699:         self.grid=grid; self.risk=risk; self.rnd = random.Random(RANDOM_SEED+7)
0700:     def inspect(self, cell:Point):
0701:         mat = self.risk.compute()
0702:         rv = mat[cell[0], cell[1]]
0703:         crack = self.rnd.random() < min(0.8, 0.01+rv/180.0)
0704:         water = self.rnd.random() < min(0.8, 0.02+rv/140.0)
0705:         smoke = self.rnd.random() < min(0.8, 0.005+rv/200.0)
0706:         return {'crack':crack,'water':water,'smoke':smoke,'risk':rv}
0707:
0708: # Planner: risk-aware A*
0709: class RiskAStar:
0710:     def __init__(self, grid:GridMap, risk:RiskField, w_risk=4.0):
0711:         self.grid=grid; self.risk=risk; self.wr=w_risk
0712:         self.rmat = self.risk.compute()
0713:
0714:     def update(self):
0715:         self.rmat = self.risk.compute()
0716:
0717:     def cost(self, a:Point, b:Point):
0718:         base = math.hypot(a[0]-b[0], a[1]-b[1])
0719:         rx = float(self.rmat[b[0], b[1]])
0720:         return base + self.wr * rx/10.0
0721:
0722:     def heuristic(self, a:Point, b:Point):
0723:         return math.hypot(a[0]-b[0], a[1]-b[1])
0724:
0725:     def plan(self, start:Point, goal:Point, max_iter=200000):
0726:         if not self.grid.is_free(start) or not self.grid.is_free(goal): return []
0727:         self.update()
0728:         import heapq
0729:         open_heap=[(self.heuristic(start,goal), start)]
0730:         g = defaultdict(lambda: float('inf')); g[start]=0.0
0731:         parent = {}
0732:         closed=set()
0733:         it=0
0734:         while open_heap and it<max_iter:
0735:             it+=1
0736:             _, cur = heapq.heappop(open_heap)
0737:             if cur in closed: continue
0738:             if cur == goal:
0739:                 path=[cur]
0740:                 while cur in parent:
0741:                     cur = parent[cur]; path.append(cur)
0742:                 return list(reversed(path))

```

```

0743:         closed.add(cur)
0744:         for nb in neighbors4(cur):
0745:             if not self.grid.is_free(nb): continue
0746:             tentative = g[cur] + self.cost(cur, nb)
0747:             if tentative < g[nb]:
0748:                 g[nb]=tentative; parent[nb]=cur
0749:                 heapq.heappush(open_heap, (tentative + self.heuristic(nb,goal), nb))
0750:         return []
0751:
0752: # Robot agent
0753: @dataclass
0754: class Robot:
0755:     start:Point; goal:Point; grid:GridMap; risk:RiskField; slam:SLAMMock; vision:VisionSim; planner:RiskAStar
0756:     pos:Point = field(init=False); path:List[Point]=field(default_factory=list); idx:int=0; battery:float=100.0; logs:List[str]
0757:     def __post_init__(self):
0758:         self.pos = self.start
0759:         self.slam.init(self.start)
0760:         self.plan(initial=True)
0761:     def plan(self, initial=False):
0762:         print('[Robot] planning', self.pos, '->', self.goal)
0763:         p = self.planner.plan(self.pos, self.goal)
0764:         if not p:
0765:             self.logs.append('plan_failed'); return False
0766:         self.path=p; self.idx=0
0767:         self.logs.append(('initial' if initial else 'replan') + f' _len_{len(p)}')
0768:         return True
0769:     def step(self):
0770:         # move one step along path
0771:         if not self.path or self.idx>=len(self.path)-1: return True, 'done'
0772:         next_cell = self.path[self.idx+1]
0773:         # check obstacle or extreme risk
0774:         risk_mat = self.risk.compute(); r = risk_mat[next_cell[0], next_cell[1]]
0775:         if not self.grid.is_free(next_cell):
0776:             self.logs.append('blocked'); return False, 'blocked'
0777:         if r > 80.0:
0778:             self.logs.append('high_risk'); return False, 'high_risk'
0779:         # move
0780:         self.pos = next_cell; self.idx += 1
0781:         dx = self.pos[0] - self.slam.truth_cell()[0]; dy = self.pos[1] - self.slam.truth_cell()[1]
0782:         self.slam.move(dx, dy)
0783:         self.battery -= 0.05 + r/200.0
0784:         det = self.vision.inspect(self.pos)
0785:         if det['crack'] or det['water'] or det['smoke']:
0786:             self.logs.append(('det', self.pos, det))
0787:         if self.pos == self.goal:
0788:             self.logs.append('goal')
0789:             return True, 'goal'
0790:         return False, None
0791:
0792: # Simulator
0793: class Simulator:
0794:     def __init__(self, grid, risk, robot):
0795:         self.grid=grid; self.risk=risk; self.robot=robot; self.time=0.0; self.replans=0; self.positions=[]

```

```

0796: def step(self, dt=0.5):
0797:     self.risk.step()
0798:     # check risk ahead
0799:     if self.robot.path and self.robot.idx+1 < len(self.robot.path):
0800:         nx = self.robot.path[self.robot.idx+1]
0801:         if self.risk.compute()[nx[0], nx[1]] > 70.0:
0802:             self.replans += 1
0803:             ok = self.robot.plan()
0804:             if not ok: return False
0805:     # random obstacle spawn
0806:     if random.random() < 0.01:
0807:         ox = random.randint(0,self.grid.w-1); oy = random.randint(0,self.grid.h-1)
0808:         if self.grid.is_free((ox,oy)) and (ox,oy)!=self.robot.pos:
0809:             self.grid.add_obs((ox,oy)); print('[Sim] obstacle at', (ox,oy))
0810:             if (ox,oy) in self.robot.path: self.robot.plan(); self.replans += 1
0811:     done, info = self.robot.step()
0812:     self.positions.append(self.robot.pos)
0813:     self.time += dt
0814:     if done: print('[Sim] finished', info); return False
0815:     if self.robot.battery < 1.0: print('[Sim] battery'); return False
0816:     if self.time > 300.0: print('[Sim] time limit'); return False
0817:     return True
0818:
0819: # Visualization using matplotlib
0820: class Visualizer:
0821:     def __init__(self, grid, risk, robot, sim):
0822:         self.grid=grid; self.risk=risk; self.robot=robot; self.sim=sim
0823:         self.fig, self.ax = plt.subplots(figsize=(10,8))
0824:         self.im = None; self.path_line=None; self.robot_dot=None; self.text=None
0825:         self._setup()
0826:     def _setup(self):
0827:         self.ax.set_xlim(-0.5, self.grid.w-0.5); self.ax.set_ylim(-0.5, self.grid.h-0.5); self.ax.invert_yaxis(); self.ax.set_as
0828:         self.ax.set_xticks(range(0,self.grid.w, max(1, self.grid.w//10))); self.ax.set_yticks(range(0,self.grid.h, max(1, se
0829:         rg = self.risk.compute().T
0830:         cmap = plt.cm.get_cmap('YlOrRd')
0831:         self.im = self.ax.imshow(rg, origin='upper', extent=(0,self.grid.w,0,self.grid.h), cmap=cmap)
0832:         self.ax.imshow(self.grid.obstacle_mask().T.astype(float), origin='upper', extent=(0,self.grid.w,0,self.grid.h), cm
0833:         self.path_line, = self.ax.plot([], [], '--', linewidth=2, label='path')
0834:         self.robot_dot, = self.ax.plot([], [], 'bo', markersize=6, label='robot')
0835:         self.text = self.ax.text(0.02, 0.98, "", transform=self.ax.transAxes, va='top', ha='left', fontsize=10, bbox=dict(box
0836:         self.ax.legend(loc='lower right')
0837:
0838:     def update(self, frame):
0839:         cont = self.sim.step(0.5)
0840:         # update risk grid and overlays
0841:         rg = self.risk.compute().T
0842:         self.im.set_data(rg); self.im.set_clim(vmin=0.0, vmax=rg.max()+1e-6)
0843:         self.ax.images[1].set_data(self.grid.obstacle_mask().T.astype(float))
0844:         # path
0845:         if self.robot.path:
0846:             px = [p[0]+0.5 for p in self.robot.path]; py=[p[1]+0.5 for p in self.robot.path]
0847:             self.path_line.set_data(px, py)
0848:         else:

```



```

0849:         self.path_line.set_data([],[])
0850:         # robot dot
0851:         self.robot_dot.set_data(self.robot.pos[0]+0.5, self.robot.pos[1]+0.5)
0852:         self.text.set_text(f'Pos:{self.robot.pos} Battery:{self.robot.battery:.1f} Time:{self.sim.time:.1f}s Replans:{self.sim.replans}')
0853:         return [self.im, self.path_line, self.robot_dot, self.text]
0854:
0855:     def animate(self):
0856:         anim = animation.FuncAnimation(self.fig, self.update, frames=500, interval=300, blit=False, repeat=False)
0857:         plt.show()
0858:
0859: # Main demo run
0860: def main():
0861:     grid = GridMap(); grid.random_corridors(blocks=7)
0862:     risk = RiskField(); risk.add(RiskSource(pos=(6,6), intensity=18.0, radius=4.0))
0863:     risk.add(RiskSource(pos=(28,10), intensity=22.0, radius=5.0)); risk.add(RiskSource(pos=(18,20), intensity=26.0, radius=6.0))
0864:     risk.add(RiskSource(pos=(22,14), intensity=12.0, radius=3.0, volatile=True))
0865:     # find free start & goal
0866:     def find_free(region):
0867:         for _ in range(2000):
0868:             x = random.randint(region[0], region[1]); y = random.randint(region[2], region[3])
0869:             if grid.is_free((x,y)): return (x,y)
0870:         # fallback
0871:         for i in range(grid.w):
0872:             for j in range(grid.h):
0873:                 if grid.is_free((i,j)): return (i,j)
0874:         return (0,0)
0875:     start = find_free((0, grid.w//3, 0, grid.h//3))
0876:     goal = find_free((grid.w//2, grid.w-1, grid.h//2, grid.h-1))
0877:     print('Start', start, 'Goal', goal)
0878:     slam = SLAMMock(); vision = VisionSim(grid, risk); planner = RiskAStar(grid, risk, w_risk=4.0)
0879:     robot = Robot(start=start, goal=goal, grid=grid, risk=risk, slam=slam, vision=vision, planner=planner)
0880:     sim = Simulator(grid, risk, robot)
0881:     viz = Visualizer(grid, risk, robot, sim)
0882:     viz.animate()
0883:     # save short report
0884:     with open('simulation_report.txt','w', encoding='utf-8') as f:
0885:         f.write(f'Start:{start}\nGoal:{goal}\nSteps:{len(sim.positions)}\nReplans:{sim.replans}\nLogs:{robot.logs[-30:]}\n')
0886:     print('Report saved to simulation_report.txt')
0887:
0888: if __name__ == "__main__": main()
0889:
0890:
0891: # === BEGIN EXPANSION ===
0892:
0893:
0894:
0895: # =====
0896: # Configuration examples and parameter definitions
0897: # =====
0898: DEFAULT_CONFIG = {
0899:     "map": {"width": 60, "height": 40, "blocks": 8},
0900:     "risk": {
0901:         "base_level": 0.0,

```

```

0902:     "sources": [
0903:         {"pos": (8, 6), "intensity": 18.0, "radius": 4.0, "volatile": True},
0904:         {"pos": (30, 12), "intensity": 22.0, "radius": 5.0, "volatile": False}
0905:     ]
0906: },
0907: "planner": {"weight_risk": 4.0, "smooth_path": True},
0908: "sim": {"timestep": 0.5, "max_steps": 800},
0909: "viz": {"cmap": "jet"}
0910: }
0911:
0912: def print_config(cfg):
0913:     """Print configuration in readable format."""
0914:     import json
0915:     print(json.dumps(cfg, indent=2, ensure_ascii=False))
0916:
0917:
0918:
0919:
0920:
0921: # =====
0922: # Additional utilities: file logging, timing decorator, simple unit-test helpers
0923: # =====
0924: def timeit(func):
0925:     """Decorator to time functions during demo runs."""
0926:     import time
0927:     def wrapper(*args, **kwargs):
0928:         t0 = time.time()
0929:         res = func(*args, **kwargs)
0930:         dt = time.time() - t0
0931:         logger.info("Timing %s: %.4fs", func.__name__, dt)
0932:         return res
0933:     return wrapper
0934:
0935: def setup_file_logger(path="run.log"):
0936:     """Configure a file logger in addition to console logger."""
0937:     fh = logging.FileHandler(path, encoding="utf-8")
0938:     fh.setLevel(logging.INFO)
0939:     formatter = logging.Formatter("%(asctime)s [%(levelname)s] %(message)s")
0940:     fh.setFormatter(formatter)
0941:     logger.addHandler(fh)
0942:     logger.info("File logging enabled: %s", path)
0943:
0944: # Simple test assertion helper (for inline tests)
0945: def assert_eq(a, b, message=""):
0946:     if a != b:
0947:         logger.error("Assertion failed: %s != %s. %s", a, b, message)
0948:         raise AssertionError(message or f"{a} != {b}")
0949:     else:
0950:         logger.info("Assertion passed: %s == %s", a, b)
0951:
0952:
0953:
0954:

```

```

0955:
0956: # =====
0957: # Extended RiskField utilities
0958: # =====
0959: def batch_add_sources(riskfield, specs):
0960:     """Add many sources from a list of spec dicts."""
0961:     for s in specs:
0962:         try:
0963:             src = RiskSource(pos=tuple(s["pos"]), intensity=float(s.get("intensity",20.0)),
0964:                               radius=float(s.get("radius",4.0)), volatile=bool(s.get("volatile",False)))
0965:             riskfield.add_source(src)
0966:         except Exception as e:
0967:             logger.warning("Bad source spec %s: %s", s, e)
0968:
0969: def validate_risk_field(riskfield):
0970:     """Run a few checks on riskfield internals to ensure correctness."""
0971:     rm = riskfield.compute()
0972:     assert rm.shape == (riskfield.width, riskfield.height)
0973:     # check monotonic decrease from source center roughly
0974:     for s in riskfield.sources[:3]:
0975:         cx, cy = s.pos
0976:         cx = clamp(cx, 0, riskfield.width-1); cy = clamp(cy, 0, riskfield.height-1)
0977:         center_val = rm[cx, cy]
0978:         neighbors = []
0979:         for dx, dy in ((1,0),(-1,0),(0,1),(0,-1)):
0980:             nx, ny = clamp(cx+dx,0,riskfield.width-1), clamp(cy+dy,0,riskfield.height-1)
0981:             neighbors.append(rm[nx, ny])
0982:         if not all(center_val >= nv for nv in neighbors):
0983:             logger.debug("Risk source at %s may not be dominant; center=%.2f neighbors=%s", s.pos, center_val, neighbors)
0984:
0985:
0986:
0987:
0988:
0989: # =====
0990: # Planner test cases and demonstration utilities
0991: # =====
0992: def simple_planner_smoke_test():
0993:     W,H = 30,20
0994:     grid = GridMap(W,H)
0995:     grid.random_corridors(blocks=5, seed=11)
0996:     rf = RiskField(W,H, base=0.0)
0997:     batch_add_sources(rf, [{"pos":(6,6),"intensity":15,"radius":3}, {"pos":(20,10),"intensity":20,"radius":4}])
0998:     start = (1,1); goal = (W-2,H-2)
0999:     planner = RiskAStar(grid, rf, weight_risk=3.0)
1000:     path = planner.plan(start, goal)
1001:     logger.info("Smoke test path len: %d", len(path))
1002:     assert isinstance(path, list)
1003:     if path:
1004:         assert_eq(path[0], start, "path must start at start")
1005:         assert_eq(path[-1], goal, "path must end at goal")
1006:     return path
1007:

```

```

1008: def planner_perf_test():
1009:     W,H = 60,40
1010:     grid = GridMap(W,H)
1011:     grid.random_corridors(blocks=12, seed=99)
1012:     rf = RiskField(W,H)
1013:     # add many random sources
1014:     for i in range(12):
1015:         rf.add_source(RiskSource((random.randint(0,W-1), random.randint(0,H-1)), intensity=random.uniform(5,30), r
1016:     planner = RiskAStar(grid, rf, weight_risk=4.0)
1017:     start = (0,0); goal = (W-1,H-1)
1018:     path = planner.plan(start, goal)
1019:     logger.info("Planner perf test path len: %d", len(path))
1020:     return path
1021:
1022:
1023:
1024:
1025:
1026: # =====
1027: # Submission helpers: split into pages (approx lines per page) and export
1028: # =====
1029: def export_code_as_pages(filepath_in, filepath_out, lines_per_page=50):
1030:     """Create a paginated code file suitable for printing (not PDF generation)."""
1031:     p = Path(filepath_in)
1032:     txt = p.read_text(encoding="utf-8")
1033:     lines = txt.splitlines()
1034:     pages = []
1035:     for i in range(0, len(lines), lines_per_page):
1036:         page_lines = lines[i:i+lines_per_page]
1037:         header = f"// Page {i//lines_per_page+1}\n"
1038:         pages.append(header + "\n".join(page_lines) + "\n")
1039:     Path(filepath_out).write_text("\n\n".join(pages), encoding="utf-8")
1040:     logger.info("Exported code to paginated text: %s", filepath_out)
1041:
1042:
1043:
1044:
1045:
1046: # =====
1047: # Design rationale and extended developer notes (useful for reviewers)
1048: # =====
1049: """
1050: Design Rationale:
1051: - RiskField: represents environmental hazards as continuous fields using Gaussian kernels;
1052:   this allows smoothing and multi-source composition and is computationally lightweight.
1053: - GridMap: stores occupancy at grid resolution suitable for indoor tunnel environments.
1054: - Risk-A*: uses risk field values to augment movement cost. Weighting parameter (weight_risk)
1055:   controls tradeoff between shortest path and safer path. This is configurable.
1056: - SLAMMock and EKF-like updater: included to simulate the reality that pose estimation
1057:   is imperfect; this encourages planners to replan when estimates deviate significantly.
1058: - DetectorInterface: allows swapping the simulated detector with a real YOLO inference
1059:   engine later without altering planner logic. The SimulatedYOLO is probabilistic and
1060:   demonstrates detection downstream effects (e.g., marking a cell as 'inspected' or

```

```

1061: adding to a maintenance report).
1062: - Visualizer: provides publication-quality figures for reports and can export PNGs.
1063:
1064: Testing plan:
1065: 1) Unit tests for RiskField numerical properties
1066: 2) Planner correctness on small grid instances
1067: 3) Integration tests: simulate a robot from start->goal with random dynamics
1068: 4) Regression tests: store several snapshots and assert reproducibility with seed
1069: ""
1070:
1071:
1072:
1073: def helper_mapping_report_1(grid: GridMap, rf: RiskField):
1074:     """Generate a small JSON-like report for mapping snapshot #1 (for demo & testing)."""
1075:     rm = rf.compute()
1076:     # sample a few statistics
1077:     mean_risk = float(rm.mean())
1078:     max_risk = float(rm.max())
1079:     free_cells = int((~grid.obstacles).sum())
1080:     report = {
1081:         "snapshot": 1,
1082:         "mean_risk": mean_risk,
1083:         "max_risk": max_risk,
1084:         "free_cells": free_cells
1085:     }
1086:     # return dictionary (caller may save)
1087:     return report
1088:
1089:
1090: def helper_mapping_report_2(grid: GridMap, rf: RiskField):
1091:     """Generate a small JSON-like report for mapping snapshot #2 (for demo & testing)."""
1092:     rm = rf.compute()
1093:     # sample a few statistics
1094:     mean_risk = float(rm.mean())
1095:     max_risk = float(rm.max())
1096:     free_cells = int((~grid.obstacles).sum())
1097:     report = {
1098:         "snapshot": 2,
1099:         "mean_risk": mean_risk,
1100:         "max_risk": max_risk,
1101:         "free_cells": free_cells
1102:     }
1103:     # return dictionary (caller may save)
1104:     return report
1105:
1106:
1107: def helper_mapping_report_3(grid: GridMap, rf: RiskField):
1108:     """Generate a small JSON-like report for mapping snapshot #3 (for demo & testing)."""
1109:     rm = rf.compute()
1110:     # sample a few statistics
1111:     mean_risk = float(rm.mean())
1112:     max_risk = float(rm.max())
1113:     free_cells = int((~grid.obstacles).sum())

```

```

1114:     report = {
1115:         "snapshot": 3,
1116:         "mean_risk": mean_risk,
1117:         "max_risk": max_risk,
1118:         "free_cells": free_cells
1119:     }
1120:     # return dictionary (caller may save)
1121:     return report
1122:
1123:
1124: def helper_mapping_report_4(grid: GridMap, rf: RiskField):
1125:     """Generate a small JSON-like report for mapping snapshot #4 (for demo & testing)."""
1126:     rm = rf.compute()
1127:     # sample a few statistics
1128:     mean_risk = float(rm.mean())
1129:     max_risk = float(rm.max())
1130:     free_cells = int((~grid.obstacles).sum())
1131:     report = {
1132:         "snapshot": 4,
1133:         "mean_risk": mean_risk,
1134:         "max_risk": max_risk,
1135:         "free_cells": free_cells
1136:     }
1137:     # return dictionary (caller may save)
1138:     return report
1139:
1140:
1141: def helper_mapping_report_5(grid: GridMap, rf: RiskField):
1142:     """Generate a small JSON-like report for mapping snapshot #5 (for demo & testing)."""
1143:     rm = rf.compute()
1144:     # sample a few statistics
1145:     mean_risk = float(rm.mean())
1146:     max_risk = float(rm.max())
1147:     free_cells = int((~grid.obstacles).sum())
1148:     report = {
1149:         "snapshot": 5,
1150:         "mean_risk": mean_risk,
1151:         "max_risk": max_risk,
1152:         "free_cells": free_cells
1153:     }
1154:     # return dictionary (caller may save)
1155:     return report
1156:
1157:
1158: def helper_mapping_report_6(grid: GridMap, rf: RiskField):
1159:     """Generate a small JSON-like report for mapping snapshot #6 (for demo & testing)."""
1160:     rm = rf.compute()
1161:     # sample a few statistics
1162:     mean_risk = float(rm.mean())
1163:     max_risk = float(rm.max())
1164:     free_cells = int((~grid.obstacles).sum())
1165:     report = {
1166:         "snapshot": 6,

```

```

1167:     "mean_risk": mean_risk,
1168:     "max_risk": max_risk,
1169:     "free_cells": free_cells
1170: }
1171: # return dictionary (caller may save)
1172: return report
1173:
1174:
1175: def helper_mapping_report_7(grid: GridMap, rf: RiskField):
1176:     """Generate a small JSON-like report for mapping snapshot #7 (for demo & testing)."""
1177:     rm = rf.compute()
1178:     # sample a few statistics
1179:     mean_risk = float(rm.mean())
1180:     max_risk = float(rm.max())
1181:     free_cells = int((~grid.obstacles).sum())
1182:     report = {
1183:         "snapshot": 7,
1184:         "mean_risk": mean_risk,
1185:         "max_risk": max_risk,
1186:         "free_cells": free_cells
1187:     }
1188:     # return dictionary (caller may save)
1189:     return report
1190:
1191:
1192: def helper_mapping_report_8(grid: GridMap, rf: RiskField):
1193:     """Generate a small JSON-like report for mapping snapshot #8 (for demo & testing)."""
1194:     rm = rf.compute()
1195:     # sample a few statistics
1196:     mean_risk = float(rm.mean())
1197:     max_risk = float(rm.max())
1198:     free_cells = int((~grid.obstacles).sum())
1199:     report = {
1200:         "snapshot": 8,
1201:         "mean_risk": mean_risk,
1202:         "max_risk": max_risk,
1203:         "free_cells": free_cells
1204:     }
1205:     # return dictionary (caller may save)
1206:     return report
1207:
1208:
1209: def helper_mapping_report_9(grid: GridMap, rf: RiskField):
1210:     """Generate a small JSON-like report for mapping snapshot #9 (for demo & testing)."""
1211:     rm = rf.compute()
1212:     # sample a few statistics
1213:     mean_risk = float(rm.mean())
1214:     max_risk = float(rm.max())
1215:     free_cells = int((~grid.obstacles).sum())
1216:     report = {
1217:         "snapshot": 9,
1218:         "mean_risk": mean_risk,
1219:         "max_risk": max_risk,

```

```

1220:     "free_cells": free_cells
1221: }
1222: # return dictionary (caller may save)
1223: return report
1224:
1225:
1226: def helper_mapping_report_10(grid: GridMap, rf: RiskField):
1227:     """Generate a small JSON-like report for mapping snapshot #10 (for demo & testing)."""
1228:     rm = rf.compute()
1229:     # sample a few statistics
1230:     mean_risk = float(rm.mean())
1231:     max_risk = float(rm.max())
1232:     free_cells = int((~grid.obstacles).sum())
1233:     report = {
1234:         "snapshot": 10,
1235:         "mean_risk": mean_risk,
1236:         "max_risk": max_risk,
1237:         "free_cells": free_cells
1238:     }
1239:     # return dictionary (caller may save)
1240:     return report
1241:
1242:
1243: def helper_mapping_report_11(grid: GridMap, rf: RiskField):
1244:     """Generate a small JSON-like report for mapping snapshot #11 (for demo & testing)."""
1245:     rm = rf.compute()
1246:     # sample a few statistics
1247:     mean_risk = float(rm.mean())
1248:     max_risk = float(rm.max())
1249:     free_cells = int((~grid.obstacles).sum())
1250:     report = {
1251:         "snapshot": 11,
1252:         "mean_risk": mean_risk,
1253:         "max_risk": max_risk,
1254:         "free_cells": free_cells
1255:     }
1256:     # return dictionary (caller may save)
1257:     return report
1258:
1259:
1260: def helper_mapping_report_12(grid: GridMap, rf: RiskField):
1261:     """Generate a small JSON-like report for mapping snapshot #12 (for demo & testing)."""
1262:     rm = rf.compute()
1263:     # sample a few statistics
1264:     mean_risk = float(rm.mean())
1265:     max_risk = float(rm.max())
1266:     free_cells = int((~grid.obstacles).sum())
1267:     report = {
1268:         "snapshot": 12,
1269:         "mean_risk": mean_risk,
1270:         "max_risk": max_risk,
1271:         "free_cells": free_cells
1272:     }

```



```

1273: # return dictionary (caller may save)
1274: return report
1275:
1276:
1277: def helper_mapping_report_13(grid: GridMap, rf: RiskField):
1278:     """Generate a small JSON-like report for mapping snapshot #13 (for demo & testing)."""
1279:     rm = rf.compute()
1280:     # sample a few statistics
1281:     mean_risk = float(rm.mean())
1282:     max_risk = float(rm.max())
1283:     free_cells = int((~grid.obstacles).sum())
1284:     report = {
1285:         "snapshot": 13,
1286:         "mean_risk": mean_risk,
1287:         "max_risk": max_risk,
1288:         "free_cells": free_cells
1289:     }
1290:     # return dictionary (caller may save)
1291:     return report
1292:
1293:
1294: def helper_mapping_report_14(grid: GridMap, rf: RiskField):
1295:     """Generate a small JSON-like report for mapping snapshot #14 (for demo & testing)."""
1296:     rm = rf.compute()
1297:     # sample a few statistics
1298:     mean_risk = float(rm.mean())
1299:     max_risk = float(rm.max())
1300:     free_cells = int((~grid.obstacles).sum())
1301:     report = {
1302:         "snapshot": 14,
1303:         "mean_risk": mean_risk,
1304:         "max_risk": max_risk,
1305:         "free_cells": free_cells
1306:     }
1307:     # return dictionary (caller may save)
1308:     return report
1309:
1310:
1311: def helper_mapping_report_15(grid: GridMap, rf: RiskField):
1312:     """Generate a small JSON-like report for mapping snapshot #15 (for demo & testing)."""
1313:     rm = rf.compute()
1314:     # sample a few statistics
1315:     mean_risk = float(rm.mean())
1316:     max_risk = float(rm.max())
1317:     free_cells = int((~grid.obstacles).sum())
1318:     report = {
1319:         "snapshot": 15,
1320:         "mean_risk": mean_risk,
1321:         "max_risk": max_risk,
1322:         "free_cells": free_cells
1323:     }
1324:     # return dictionary (caller may save)
1325:     return report

```

```

1326:
1327:
1328: def helper_mapping_report_16(grid: GridMap, rf: RiskField):
1329:     """Generate a small JSON-like report for mapping snapshot #16 (for demo & testing)."""
1330:     rm = rf.compute()
1331:     # sample a few statistics
1332:     mean_risk = float(rm.mean())
1333:     max_risk = float(rm.max())
1334:     free_cells = int((~grid.obstacles).sum())
1335:     report = {
1336:         "snapshot": 16,
1337:         "mean_risk": mean_risk,
1338:         "max_risk": max_risk,
1339:         "free_cells": free_cells
1340:     }
1341:     # return dictionary (caller may save)
1342:     return report
1343:
1344:
1345: def helper_mapping_report_17(grid: GridMap, rf: RiskField):
1346:     """Generate a small JSON-like report for mapping snapshot #17 (for demo & testing)."""
1347:     rm = rf.compute()
1348:     # sample a few statistics
1349:     mean_risk = float(rm.mean())
1350:     max_risk = float(rm.max())
1351:     free_cells = int((~grid.obstacles).sum())
1352:     report = {
1353:         "snapshot": 17,
1354:         "mean_risk": mean_risk,
1355:         "max_risk": max_risk,
1356:         "free_cells": free_cells
1357:     }
1358:     # return dictionary (caller may save)
1359:     return report
1360:
1361:
1362: def helper_mapping_report_18(grid: GridMap, rf: RiskField):
1363:     """Generate a small JSON-like report for mapping snapshot #18 (for demo & testing)."""
1364:     rm = rf.compute()
1365:     # sample a few statistics
1366:     mean_risk = float(rm.mean())
1367:     max_risk = float(rm.max())
1368:     free_cells = int((~grid.obstacles).sum())
1369:     report = {
1370:         "snapshot": 18,
1371:         "mean_risk": mean_risk,
1372:         "max_risk": max_risk,
1373:         "free_cells": free_cells
1374:     }
1375:     # return dictionary (caller may save)
1376:     return report
1377:
1378:

```

```

1379: def helper_mapping_report_19(grid: GridMap, rf: RiskField):
1380:     """Generate a small JSON-like report for mapping snapshot #19 (for demo & testing)."""
1381:     rm = rf.compute()
1382:     # sample a few statistics
1383:     mean_risk = float(rm.mean())
1384:     max_risk = float(rm.max())
1385:     free_cells = int((~grid.obstacles).sum())
1386:     report = {
1387:         "snapshot": 19,
1388:         "mean_risk": mean_risk,
1389:         "max_risk": max_risk,
1390:         "free_cells": free_cells
1391:     }
1392:     # return dictionary (caller may save)
1393:     return report
1394:
1395:
1396: def helper_mapping_report_20(grid: GridMap, rf: RiskField):
1397:     """Generate a small JSON-like report for mapping snapshot #20 (for demo & testing)."""
1398:     rm = rf.compute()
1399:     # sample a few statistics
1400:     mean_risk = float(rm.mean())
1401:     max_risk = float(rm.max())
1402:     free_cells = int((~grid.obstacles).sum())
1403:     report = {
1404:         "snapshot": 20,
1405:         "mean_risk": mean_risk,
1406:         "max_risk": max_risk,
1407:         "free_cells": free_cells
1408:     }
1409:     # return dictionary (caller may save)
1410:     return report
1411:
1412:
1413: def helper_mapping_report_21(grid: GridMap, rf: RiskField):
1414:     """Generate a small JSON-like report for mapping snapshot #21 (for demo & testing)."""
1415:     rm = rf.compute()
1416:     # sample a few statistics
1417:     mean_risk = float(rm.mean())
1418:     max_risk = float(rm.max())
1419:     free_cells = int((~grid.obstacles).sum())
1420:     report = {
1421:         "snapshot": 21,
1422:         "mean_risk": mean_risk,
1423:         "max_risk": max_risk,
1424:         "free_cells": free_cells
1425:     }
1426:     # return dictionary (caller may save)
1427:     return report
1428:
1429:
1430: def helper_mapping_report_22(grid: GridMap, rf: RiskField):
1431:     """Generate a small JSON-like report for mapping snapshot #22 (for demo & testing)."""

```

```

1432:     rm = rf.compute()
1433:     # sample a few statistics
1434:     mean_risk = float(rm.mean())
1435:     max_risk = float(rm.max())
1436:     free_cells = int((~grid.obstacles).sum())
1437:     report = {
1438:         "snapshot": 22,
1439:         "mean_risk": mean_risk,
1440:         "max_risk": max_risk,
1441:         "free_cells": free_cells
1442:     }
1443:     # return dictionary (caller may save)
1444:     return report
1445:
1446:
1447: def helper_mapping_report_23(grid: GridMap, rf: RiskField):
1448:     """Generate a small JSON-like report for mapping snapshot #23 (for demo & testing)."""
1449:     rm = rf.compute()
1450:     # sample a few statistics
1451:     mean_risk = float(rm.mean())
1452:     max_risk = float(rm.max())
1453:     free_cells = int((~grid.obstacles).sum())
1454:     report = {
1455:         "snapshot": 23,
1456:         "mean_risk": mean_risk,
1457:         "max_risk": max_risk,
1458:         "free_cells": free_cells
1459:     }
1460:     # return dictionary (caller may save)
1461:     return report
1462:
1463:
1464: def helper_mapping_report_24(grid: GridMap, rf: RiskField):
1465:     """Generate a small JSON-like report for mapping snapshot #24 (for demo & testing)."""
1466:     rm = rf.compute()
1467:     # sample a few statistics
1468:     mean_risk = float(rm.mean())
1469:     max_risk = float(rm.max())
1470:     free_cells = int((~grid.obstacles).sum())
1471:     report = {
1472:         "snapshot": 24,
1473:         "mean_risk": mean_risk,
1474:         "max_risk": max_risk,
1475:         "free_cells": free_cells
1476:     }
1477:     # return dictionary (caller may save)
1478:     return report
1479:
1480:
1481: def helper_mapping_report_25(grid: GridMap, rf: RiskField):
1482:     """Generate a small JSON-like report for mapping snapshot #25 (for demo & testing)."""
1483:     rm = rf.compute()
1484:     # sample a few statistics

```

```

1485: mean_risk = float(rm.mean())
1486: max_risk = float(rm.max())
1487: free_cells = int((~grid.obstacles).sum())
1488: report = {
1489:     "snapshot": 25,
1490:     "mean_risk": mean_risk,
1491:     "max_risk": max_risk,
1492:     "free_cells": free_cells
1493: }
1494: # return dictionary (caller may save)
1495: return report
1496:
1497:
1498: def helper_mapping_report_26(grid: GridMap, rf: RiskField):
1499:     """Generate a small JSON-like report for mapping snapshot #26 (for demo & testing)."""
1500:     rm = rf.compute()
1501:     # sample a few statistics
1502:     mean_risk = float(rm.mean())
1503:     max_risk = float(rm.max())
1504:     free_cells = int((~grid.obstacles).sum())
1505:     report = {
1506:         "snapshot": 26,
1507:         "mean_risk": mean_risk,
1508:         "max_risk": max_risk,
1509:         "free_cells": free_cells
1510:     }
1511:     # return dictionary (caller may save)
1512:     return report
1513:
1514:
1515: def helper_mapping_report_27(grid: GridMap, rf: RiskField):
1516:     """Generate a small JSON-like report for mapping snapshot #27 (for demo & testing)."""
1517:     rm = rf.compute()
1518:     # sample a few statistics
1519:     mean_risk = float(rm.mean())
1520:     max_risk = float(rm.max())
1521:     free_cells = int((~grid.obstacles).sum())
1522:     report = {
1523:         "snapshot": 27,
1524:         "mean_risk": mean_risk,
1525:         "max_risk": max_risk,
1526:         "free_cells": free_cells
1527:     }
1528:     # return dictionary (caller may save)
1529:     return report
1530:
1531:
1532: def helper_mapping_report_28(grid: GridMap, rf: RiskField):
1533:     """Generate a small JSON-like report for mapping snapshot #28 (for demo & testing)."""
1534:     rm = rf.compute()
1535:     # sample a few statistics
1536:     mean_risk = float(rm.mean())
1537:     max_risk = float(rm.max())

```

```

1538: free_cells = int((~grid.obstacles).sum())
1539: report = {
1540:     "snapshot": 28,
1541:     "mean_risk": mean_risk,
1542:     "max_risk": max_risk,
1543:     "free_cells": free_cells
1544: }
1545: # return dictionary (caller may save)
1546: return report
1547:
1548:
1549: def helper_mapping_report_29(grid: GridMap, rf: RiskField):
1550:     """Generate a small JSON-like report for mapping snapshot #29 (for demo & testing)."""
1551:     rm = rf.compute()
1552:     # sample a few statistics
1553:     mean_risk = float(rm.mean())
1554:     max_risk = float(rm.max())
1555:     free_cells = int((~grid.obstacles).sum())
1556:     report = {
1557:         "snapshot": 29,
1558:         "mean_risk": mean_risk,
1559:         "max_risk": max_risk,
1560:         "free_cells": free_cells
1561:     }
1562:     # return dictionary (caller may save)
1563:     return report
1564:
1565:
1566: def helper_mapping_report_30(grid: GridMap, rf: RiskField):
1567:     """Generate a small JSON-like report for mapping snapshot #30 (for demo & testing)."""
1568:     rm = rf.compute()
1569:     # sample a few statistics
1570:     mean_risk = float(rm.mean())
1571:     max_risk = float(rm.max())
1572:     free_cells = int((~grid.obstacles).sum())
1573:     report = {
1574:         "snapshot": 30,
1575:         "mean_risk": mean_risk,
1576:         "max_risk": max_risk,
1577:         "free_cells": free_cells
1578:     }
1579:     # return dictionary (caller may save)
1580:     return report
1581:
1582:
1583: def helper_mapping_report_31(grid: GridMap, rf: RiskField):
1584:     """Generate a small JSON-like report for mapping snapshot #31 (for demo & testing)."""
1585:     rm = rf.compute()
1586:     # sample a few statistics
1587:     mean_risk = float(rm.mean())
1588:     max_risk = float(rm.max())
1589:     free_cells = int((~grid.obstacles).sum())
1590:     report = {

```

```

1591:     "snapshot": 31,
1592:     "mean_risk": mean_risk,
1593:     "max_risk": max_risk,
1594:     "free_cells": free_cells
1595: }
1596: # return dictionary (caller may save)
1597: return report
1598:
1599:
1600: def helper_mapping_report_32(grid: GridMap, rf: RiskField):
1601:     """Generate a small JSON-like report for mapping snapshot #32 (for demo & testing)."""
1602:     rm = rf.compute()
1603:     # sample a few statistics
1604:     mean_risk = float(rm.mean())
1605:     max_risk = float(rm.max())
1606:     free_cells = int((~grid.obstacles).sum())
1607:     report = {
1608:         "snapshot": 32,
1609:         "mean_risk": mean_risk,
1610:         "max_risk": max_risk,
1611:         "free_cells": free_cells
1612:     }
1613:     # return dictionary (caller may save)
1614:     return report
1615:
1616:
1617: def helper_mapping_report_33(grid: GridMap, rf: RiskField):
1618:     """Generate a small JSON-like report for mapping snapshot #33 (for demo & testing)."""
1619:     rm = rf.compute()
1620:     # sample a few statistics
1621:     mean_risk = float(rm.mean())
1622:     max_risk = float(rm.max())
1623:     free_cells = int((~grid.obstacles).sum())
1624:     report = {
1625:         "snapshot": 33,
1626:         "mean_risk": mean_risk,
1627:         "max_risk": max_risk,
1628:         "free_cells": free_cells
1629:     }
1630:     # return dictionary (caller may save)
1631:     return report
1632:
1633:
1634: def helper_mapping_report_34(grid: GridMap, rf: RiskField):
1635:     """Generate a small JSON-like report for mapping snapshot #34 (for demo & testing)."""
1636:     rm = rf.compute()
1637:     # sample a few statistics
1638:     mean_risk = float(rm.mean())
1639:     max_risk = float(rm.max())
1640:     free_cells = int((~grid.obstacles).sum())
1641:     report = {
1642:         "snapshot": 34,
1643:         "mean_risk": mean_risk,

```

```

1644:     "max_risk": max_risk,
1645:     "free_cells": free_cells
1646: }
1647: # return dictionary (caller may save)
1648: return report
1649:
1650:
1651: def helper_mapping_report_35(grid: GridMap, rf: RiskField):
1652:     """Generate a small JSON-like report for mapping snapshot #35 (for demo & testing)."""
1653:     rm = rf.compute()
1654:     # sample a few statistics
1655:     mean_risk = float(rm.mean())
1656:     max_risk = float(rm.max())
1657:     free_cells = int((~grid.obstacles).sum())
1658:     report = {
1659:         "snapshot": 35,
1660:         "mean_risk": mean_risk,
1661:         "max_risk": max_risk,
1662:         "free_cells": free_cells
1663:     }
1664:     # return dictionary (caller may save)
1665:     return report
1666:
1667:
1668: def helper_mapping_report_36(grid: GridMap, rf: RiskField):
1669:     """Generate a small JSON-like report for mapping snapshot #36 (for demo & testing)."""
1670:     rm = rf.compute()
1671:     # sample a few statistics
1672:     mean_risk = float(rm.mean())
1673:     max_risk = float(rm.max())
1674:     free_cells = int((~grid.obstacles).sum())
1675:     report = {
1676:         "snapshot": 36,
1677:         "mean_risk": mean_risk,
1678:         "max_risk": max_risk,
1679:         "free_cells": free_cells
1680:     }
1681:     # return dictionary (caller may save)
1682:     return report
1683:
1684:
1685: def helper_mapping_report_37(grid: GridMap, rf: RiskField):
1686:     """Generate a small JSON-like report for mapping snapshot #37 (for demo & testing)."""
1687:     rm = rf.compute()
1688:     # sample a few statistics
1689:     mean_risk = float(rm.mean())
1690:     max_risk = float(rm.max())
1691:     free_cells = int((~grid.obstacles).sum())
1692:     report = {
1693:         "snapshot": 37,
1694:         "mean_risk": mean_risk,
1695:         "max_risk": max_risk,
1696:         "free_cells": free_cells

```



```

1697: }
1698: # return dictionary (caller may save)
1699: return report
1700:
1701:
1702: def helper_mapping_report_38(grid: GridMap, rf: RiskField):
1703:     """Generate a small JSON-like report for mapping snapshot #38 (for demo & testing)."""
1704:     rm = rf.compute()
1705:     # sample a few statistics
1706:     mean_risk = float(rm.mean())
1707:     max_risk = float(rm.max())
1708:     free_cells = int((~grid.obstacles).sum())
1709:     report = {
1710:         "snapshot": 38,
1711:         "mean_risk": mean_risk,
1712:         "max_risk": max_risk,
1713:         "free_cells": free_cells
1714:     }
1715:     # return dictionary (caller may save)
1716:     return report
1717:
1718:
1719: def helper_mapping_report_39(grid: GridMap, rf: RiskField):
1720:     """Generate a small JSON-like report for mapping snapshot #39 (for demo & testing)."""
1721:     rm = rf.compute()
1722:     # sample a few statistics
1723:     mean_risk = float(rm.mean())
1724:     max_risk = float(rm.max())
1725:     free_cells = int((~grid.obstacles).sum())
1726:     report = {
1727:         "snapshot": 39,
1728:         "mean_risk": mean_risk,
1729:         "max_risk": max_risk,
1730:         "free_cells": free_cells
1731:     }
1732:     # return dictionary (caller may save)
1733:     return report
1734:
1735:
1736: def helper_mapping_report_40(grid: GridMap, rf: RiskField):
1737:     """Generate a small JSON-like report for mapping snapshot #40 (for demo & testing)."""
1738:     rm = rf.compute()
1739:     # sample a few statistics
1740:     mean_risk = float(rm.mean())
1741:     max_risk = float(rm.max())
1742:     free_cells = int((~grid.obstacles).sum())
1743:     report = {
1744:         "snapshot": 40,
1745:         "mean_risk": mean_risk,
1746:         "max_risk": max_risk,
1747:         "free_cells": free_cells
1748:     }
1749:     # return dictionary (caller may save)

```

```

1750:     return report
1751:
1752:
1753: def helper_mapping_report_41(grid: GridMap, rf: RiskField):
1754:     """Generate a small JSON-like report for mapping snapshot #41 (for demo & testing)."""
1755:     rm = rf.compute()
1756:     # sample a few statistics
1757:     mean_risk = float(rm.mean())
1758:     max_risk = float(rm.max())
1759:     free_cells = int((~grid.obstacles).sum())
1760:     report = {
1761:         "snapshot": 41,
1762:         "mean_risk": mean_risk,
1763:         "max_risk": max_risk,
1764:         "free_cells": free_cells
1765:     }
1766:     # return dictionary (caller may save)
1767:     return report
1768:
1769:
1770: def helper_mapping_report_42(grid: GridMap, rf: RiskField):
1771:     """Generate a small JSON-like report for mapping snapshot #42 (for demo & testing)."""
1772:     rm = rf.compute()
1773:     # sample a few statistics
1774:     mean_risk = float(rm.mean())
1775:     max_risk = float(rm.max())
1776:     free_cells = int((~grid.obstacles).sum())
1777:     report = {
1778:         "snapshot": 42,
1779:         "mean_risk": mean_risk,
1780:         "max_risk": max_risk,
1781:         "free_cells": free_cells
1782:     }
1783:     # return dictionary (caller may save)
1784:     return report
1785:
1786:
1787: def helper_mapping_report_43(grid: GridMap, rf: RiskField):
1788:     """Generate a small JSON-like report for mapping snapshot #43 (for demo & testing)."""
1789:     rm = rf.compute()
1790:     # sample a few statistics
1791:     mean_risk = float(rm.mean())
1792:     max_risk = float(rm.max())
1793:     free_cells = int((~grid.obstacles).sum())
1794:     report = {
1795:         "snapshot": 43,
1796:         "mean_risk": mean_risk,
1797:         "max_risk": max_risk,
1798:         "free_cells": free_cells
1799:     }
1800:     # return dictionary (caller may save)
1801:     return report
1802:

```

```

1803:
1804: def helper_mapping_report_44(grid: GridMap, rf: RiskField):
1805:     """Generate a small JSON-like report for mapping snapshot #44 (for demo & testing)."""
1806:     rm = rf.compute()
1807:     # sample a few statistics
1808:     mean_risk = float(rm.mean())
1809:     max_risk = float(rm.max())
1810:     free_cells = int((~grid.obstacles).sum())
1811:     report = {
1812:         "snapshot": 44,
1813:         "mean_risk": mean_risk,
1814:         "max_risk": max_risk,
1815:         "free_cells": free_cells
1816:     }
1817:     # return dictionary (caller may save)
1818:     return report
1819:
1820:
1821: def helper_mapping_report_45(grid: GridMap, rf: RiskField):
1822:     """Generate a small JSON-like report for mapping snapshot #45 (for demo & testing)."""
1823:     rm = rf.compute()
1824:     # sample a few statistics
1825:     mean_risk = float(rm.mean())
1826:     max_risk = float(rm.max())
1827:     free_cells = int((~grid.obstacles).sum())
1828:     report = {
1829:         "snapshot": 45,
1830:         "mean_risk": mean_risk,
1831:         "max_risk": max_risk,
1832:         "free_cells": free_cells
1833:     }
1834:     # return dictionary (caller may save)
1835:     return report
1836:
1837:
1838: def helper_mapping_report_46(grid: GridMap, rf: RiskField):
1839:     """Generate a small JSON-like report for mapping snapshot #46 (for demo & testing)."""
1840:     rm = rf.compute()
1841:     # sample a few statistics
1842:     mean_risk = float(rm.mean())
1843:     max_risk = float(rm.max())
1844:     free_cells = int((~grid.obstacles).sum())
1845:     report = {
1846:         "snapshot": 46,
1847:         "mean_risk": mean_risk,
1848:         "max_risk": max_risk,
1849:         "free_cells": free_cells
1850:     }
1851:     # return dictionary (caller may save)
1852:     return report
1853:
1854:
1855: def helper_mapping_report_47(grid: GridMap, rf: RiskField):

```

```

1856: """Generate a small JSON-like report for mapping snapshot #47 (for demo & testing)."""
1857: rm = rf.compute()
1858: # sample a few statistics
1859: mean_risk = float(rm.mean())
1860: max_risk = float(rm.max())
1861: free_cells = int((~grid.obstacles).sum())
1862: report = {
1863:     "snapshot": 47,
1864:     "mean_risk": mean_risk,
1865:     "max_risk": max_risk,
1866:     "free_cells": free_cells
1867: }
1868: # return dictionary (caller may save)
1869: return report
1870:
1871:
1872: def helper_mapping_report_48(grid: GridMap, rf: RiskField):
1873:     """Generate a small JSON-like report for mapping snapshot #48 (for demo & testing)."""
1874:     rm = rf.compute()
1875:     # sample a few statistics
1876:     mean_risk = float(rm.mean())
1877:     max_risk = float(rm.max())
1878:     free_cells = int((~grid.obstacles).sum())
1879:     report = {
1880:         "snapshot": 48,
1881:         "mean_risk": mean_risk,
1882:         "max_risk": max_risk,
1883:         "free_cells": free_cells
1884:     }
1885:     # return dictionary (caller may save)
1886:     return report
1887:
1888:
1889: def helper_mapping_report_49(grid: GridMap, rf: RiskField):
1890:     """Generate a small JSON-like report for mapping snapshot #49 (for demo & testing)."""
1891:     rm = rf.compute()
1892:     # sample a few statistics
1893:     mean_risk = float(rm.mean())
1894:     max_risk = float(rm.max())
1895:     free_cells = int((~grid.obstacles).sum())
1896:     report = {
1897:         "snapshot": 49,
1898:         "mean_risk": mean_risk,
1899:         "max_risk": max_risk,
1900:         "free_cells": free_cells
1901:     }
1902:     # return dictionary (caller may save)
1903:     return report
1904:
1905:
1906: def helper_mapping_report_50(grid: GridMap, rf: RiskField):
1907:     """Generate a small JSON-like report for mapping snapshot #50 (for demo & testing)."""
1908:     rm = rf.compute()

```

```

1909: # sample a few statistics
1910: mean_risk = float(rm.mean())
1911: max_risk = float(rm.max())
1912: free_cells = int((~grid.obstacles).sum())
1913: report = {
1914:     "snapshot": 50,
1915:     "mean_risk": mean_risk,
1916:     "max_risk": max_risk,
1917:     "free_cells": free_cells
1918: }
1919: # return dictionary (caller may save)
1920: return report
1921:
1922:
1923: def helper_mapping_report_51(grid: GridMap, rf: RiskField):
1924:     """Generate a small JSON-like report for mapping snapshot #51 (for demo & testing)."""
1925:     rm = rf.compute()
1926:     # sample a few statistics
1927:     mean_risk = float(rm.mean())
1928:     max_risk = float(rm.max())
1929:     free_cells = int((~grid.obstacles).sum())
1930:     report = {
1931:         "snapshot": 51,
1932:         "mean_risk": mean_risk,
1933:         "max_risk": max_risk,
1934:         "free_cells": free_cells
1935:     }
1936:     # return dictionary (caller may save)
1937:     return report
1938:
1939:
1940: def helper_mapping_report_52(grid: GridMap, rf: RiskField):
1941:     """Generate a small JSON-like report for mapping snapshot #52 (for demo & testing)."""
1942:     rm = rf.compute()
1943:     # sample a few statistics
1944:     mean_risk = float(rm.mean())
1945:     max_risk = float(rm.max())
1946:     free_cells = int((~grid.obstacles).sum())
1947:     report = {
1948:         "snapshot": 52,
1949:         "mean_risk": mean_risk,
1950:         "max_risk": max_risk,
1951:         "free_cells": free_cells
1952:     }
1953:     # return dictionary (caller may save)
1954:     return report
1955:
1956:
1957: def helper_mapping_report_53(grid: GridMap, rf: RiskField):
1958:     """Generate a small JSON-like report for mapping snapshot #53 (for demo & testing)."""
1959:     rm = rf.compute()
1960:     # sample a few statistics
1961:     mean_risk = float(rm.mean())

```

```

1962:     max_risk = float(rm.max())
1963:     free_cells = int((~grid.obstacles).sum())
1964:     report = {
1965:         "snapshot": 53,
1966:         "mean_risk": mean_risk,
1967:         "max_risk": max_risk,
1968:         "free_cells": free_cells
1969:     }
1970:     # return dictionary (caller may save)
1971:     return report
1972:
1973:
1974: def helper_mapping_report_54(grid: GridMap, rf: RiskField):
1975:     """Generate a small JSON-like report for mapping snapshot #54 (for demo & testing)."""
1976:     rm = rf.compute()
1977:     # sample a few statistics
1978:     mean_risk = float(rm.mean())
1979:     max_risk = float(rm.max())
1980:     free_cells = int((~grid.obstacles).sum())
1981:     report = {
1982:         "snapshot": 54,
1983:         "mean_risk": mean_risk,
1984:         "max_risk": max_risk,
1985:         "free_cells": free_cells
1986:     }
1987:     # return dictionary (caller may save)
1988:     return report
1989:
1990:
1991: def helper_mapping_report_55(grid: GridMap, rf: RiskField):
1992:     """Generate a small JSON-like report for mapping snapshot #55 (for demo & testing)."""
1993:     rm = rf.compute()
1994:     # sample a few statistics
1995:     mean_risk = float(rm.mean())
1996:     max_risk = float(rm.max())
1997:     free_cells = int((~grid.obstacles).sum())
1998:     report = {
1999:         "snapshot": 55,
2000:         "mean_risk": mean_risk,
2001:         "max_risk": max_risk,
2002:         "free_cells": free_cells
2003:     }
2004:     # return dictionary (caller may save)
2005:     return report
2006:
2007:
2008: def helper_mapping_report_56(grid: GridMap, rf: RiskField):
2009:     """Generate a small JSON-like report for mapping snapshot #56 (for demo & testing)."""
2010:     rm = rf.compute()
2011:     # sample a few statistics
2012:     mean_risk = float(rm.mean())
2013:     max_risk = float(rm.max())
2014:     free_cells = int((~grid.obstacles).sum())

```

```

2015:     report = {
2016:         "snapshot": 56,
2017:         "mean_risk": mean_risk,
2018:         "max_risk": max_risk,
2019:         "free_cells": free_cells
2020:     }
2021:     # return dictionary (caller may save)
2022:     return report
2023:
2024:
2025: def helper_mapping_report_57(grid: GridMap, rf: RiskField):
2026:     """Generate a small JSON-like report for mapping snapshot #57 (for demo & testing)."""
2027:     rm = rf.compute()
2028:     # sample a few statistics
2029:     mean_risk = float(rm.mean())
2030:     max_risk = float(rm.max())
2031:     free_cells = int((~grid.obstacles).sum())
2032:     report = {
2033:         "snapshot": 57,
2034:         "mean_risk": mean_risk,
2035:         "max_risk": max_risk,
2036:         "free_cells": free_cells
2037:     }
2038:     # return dictionary (caller may save)
2039:     return report
2040:
2041:
2042: def helper_mapping_report_58(grid: GridMap, rf: RiskField):
2043:     """Generate a small JSON-like report for mapping snapshot #58 (for demo & testing)."""
2044:     rm = rf.compute()
2045:     # sample a few statistics
2046:     mean_risk = float(rm.mean())
2047:     max_risk = float(rm.max())
2048:     free_cells = int((~grid.obstacles).sum())
2049:     report = {
2050:         "snapshot": 58,
2051:         "mean_risk": mean_risk,
2052:         "max_risk": max_risk,
2053:         "free_cells": free_cells
2054:     }
2055:     # return dictionary (caller may save)
2056:     return report
2057:
2058:
2059: def helper_mapping_report_59(grid: GridMap, rf: RiskField):
2060:     """Generate a small JSON-like report for mapping snapshot #59 (for demo & testing)."""
2061:     rm = rf.compute()
2062:     # sample a few statistics
2063:     mean_risk = float(rm.mean())
2064:     max_risk = float(rm.max())
2065:     free_cells = int((~grid.obstacles).sum())
2066:     report = {
2067:         "snapshot": 59,

```

```

2068:     "mean_risk": mean_risk,
2069:     "max_risk": max_risk,
2070:     "free_cells": free_cells
2071: }
2072: # return dictionary (caller may save)
2073: return report
2074:
2075:
2076: def helper_mapping_report_60(grid: GridMap, rf: RiskField):
2077:     """Generate a small JSON-like report for mapping snapshot #60 (for demo & testing)."""
2078:     rm = rf.compute()
2079:     # sample a few statistics
2080:     mean_risk = float(rm.mean())
2081:     max_risk = float(rm.max())
2082:     free_cells = int((~grid.obstacles).sum())
2083:     report = {
2084:         "snapshot": 60,
2085:         "mean_risk": mean_risk,
2086:         "max_risk": max_risk,
2087:         "free_cells": free_cells
2088:     }
2089:     # return dictionary (caller may save)
2090:     return report
2091:
2092:
2093: def helper_mapping_report_61(grid: GridMap, rf: RiskField):
2094:     """Generate a small JSON-like report for mapping snapshot #61 (for demo & testing)."""
2095:     rm = rf.compute()
2096:     # sample a few statistics
2097:     mean_risk = float(rm.mean())
2098:     max_risk = float(rm.max())
2099:     free_cells = int((~grid.obstacles).sum())
2100:     report = {
2101:         "snapshot": 61,
2102:         "mean_risk": mean_risk,
2103:         "max_risk": max_risk,
2104:         "free_cells": free_cells
2105:     }
2106:     # return dictionary (caller may save)
2107:     return report
2108:
2109:
2110: def helper_mapping_report_62(grid: GridMap, rf: RiskField):
2111:     """Generate a small JSON-like report for mapping snapshot #62 (for demo & testing)."""
2112:     rm = rf.compute()
2113:     # sample a few statistics
2114:     mean_risk = float(rm.mean())
2115:     max_risk = float(rm.max())
2116:     free_cells = int((~grid.obstacles).sum())
2117:     report = {
2118:         "snapshot": 62,
2119:         "mean_risk": mean_risk,
2120:         "max_risk": max_risk,

```



```

2121:     "free_cells": free_cells
2122: }
2123: # return dictionary (caller may save)
2124: return report
2125:
2126:
2127: def helper_mapping_report_63(grid: GridMap, rf: RiskField):
2128:     """Generate a small JSON-like report for mapping snapshot #63 (for demo & testing)."""
2129:     rm = rf.compute()
2130:     # sample a few statistics
2131:     mean_risk = float(rm.mean())
2132:     max_risk = float(rm.max())
2133:     free_cells = int((~grid.obstacles).sum())
2134:     report = {
2135:         "snapshot": 63,
2136:         "mean_risk": mean_risk,
2137:         "max_risk": max_risk,
2138:         "free_cells": free_cells
2139:     }
2140:     # return dictionary (caller may save)
2141:     return report
2142:
2143:
2144: def helper_mapping_report_64(grid: GridMap, rf: RiskField):
2145:     """Generate a small JSON-like report for mapping snapshot #64 (for demo & testing)."""
2146:     rm = rf.compute()
2147:     # sample a few statistics
2148:     mean_risk = float(rm.mean())
2149:     max_risk = float(rm.max())
2150:     free_cells = int((~grid.obstacles).sum())
2151:     report = {
2152:         "snapshot": 64,
2153:         "mean_risk": mean_risk,
2154:         "max_risk": max_risk,
2155:         "free_cells": free_cells
2156:     }
2157:     # return dictionary (caller may save)
2158:     return report
2159:
2160:
2161: def helper_mapping_report_65(grid: GridMap, rf: RiskField):
2162:     """Generate a small JSON-like report for mapping snapshot #65 (for demo & testing)."""
2163:     rm = rf.compute()
2164:     # sample a few statistics
2165:     mean_risk = float(rm.mean())
2166:     max_risk = float(rm.max())
2167:     free_cells = int((~grid.obstacles).sum())
2168:     report = {
2169:         "snapshot": 65,
2170:         "mean_risk": mean_risk,
2171:         "max_risk": max_risk,
2172:         "free_cells": free_cells
2173:     }

```

```

2174: # return dictionary (caller may save)
2175: return report
2176:
2177:
2178: def helper_mapping_report_66(grid: GridMap, rf: RiskField):
2179:     """Generate a small JSON-like report for mapping snapshot #66 (for demo & testing)."""
2180:     rm = rf.compute()
2181:     # sample a few statistics
2182:     mean_risk = float(rm.mean())
2183:     max_risk = float(rm.max())
2184:     free_cells = int((~grid.obstacles).sum())
2185:     report = {
2186:         "snapshot": 66,
2187:         "mean_risk": mean_risk,
2188:         "max_risk": max_risk,
2189:         "free_cells": free_cells
2190:     }
2191:     # return dictionary (caller may save)
2192:     return report
2193:
2194:
2195: def helper_mapping_report_67(grid: GridMap, rf: RiskField):
2196:     """Generate a small JSON-like report for mapping snapshot #67 (for demo & testing)."""
2197:     rm = rf.compute()
2198:     # sample a few statistics
2199:     mean_risk = float(rm.mean())
2200:     max_risk = float(rm.max())
2201:     free_cells = int((~grid.obstacles).sum())
2202:     report = {
2203:         "snapshot": 67,
2204:         "mean_risk": mean_risk,
2205:         "max_risk": max_risk,
2206:         "free_cells": free_cells
2207:     }
2208:     # return dictionary (caller may save)
2209:     return report
2210:
2211:
2212: def helper_mapping_report_68(grid: GridMap, rf: RiskField):
2213:     """Generate a small JSON-like report for mapping snapshot #68 (for demo & testing)."""
2214:     rm = rf.compute()
2215:     # sample a few statistics
2216:     mean_risk = float(rm.mean())
2217:     max_risk = float(rm.max())
2218:     free_cells = int((~grid.obstacles).sum())
2219:     report = {
2220:         "snapshot": 68,
2221:         "mean_risk": mean_risk,
2222:         "max_risk": max_risk,
2223:         "free_cells": free_cells
2224:     }
2225:     # return dictionary (caller may save)
2226:     return report

```

```

2227:
2228:
2229: def helper_mapping_report_69(grid: GridMap, rf: RiskField):
2230:     """Generate a small JSON-like report for mapping snapshot #69 (for demo & testing)."""
2231:     rm = rf.compute()
2232:     # sample a few statistics
2233:     mean_risk = float(rm.mean())
2234:     max_risk = float(rm.max())
2235:     free_cells = int((~grid.obstacles).sum())
2236:     report = {
2237:         "snapshot": 69,
2238:         "mean_risk": mean_risk,
2239:         "max_risk": max_risk,
2240:         "free_cells": free_cells
2241:     }
2242:     # return dictionary (caller may save)
2243:     return report
2244:
2245:
2246: def helper_mapping_report_70(grid: GridMap, rf: RiskField):
2247:     """Generate a small JSON-like report for mapping snapshot #70 (for demo & testing)."""
2248:     rm = rf.compute()
2249:     # sample a few statistics
2250:     mean_risk = float(rm.mean())
2251:     max_risk = float(rm.max())
2252:     free_cells = int((~grid.obstacles).sum())
2253:     report = {
2254:         "snapshot": 70,
2255:         "mean_risk": mean_risk,
2256:         "max_risk": max_risk,
2257:         "free_cells": free_cells
2258:     }
2259:     # return dictionary (caller may save)
2260:     return report
2261:
2262:
2263: def helper_mapping_report_71(grid: GridMap, rf: RiskField):
2264:     """Generate a small JSON-like report for mapping snapshot #71 (for demo & testing)."""
2265:     rm = rf.compute()
2266:     # sample a few statistics
2267:     mean_risk = float(rm.mean())
2268:     max_risk = float(rm.max())
2269:     free_cells = int((~grid.obstacles).sum())
2270:     report = {
2271:         "snapshot": 71,
2272:         "mean_risk": mean_risk,
2273:         "max_risk": max_risk,
2274:         "free_cells": free_cells
2275:     }
2276:     # return dictionary (caller may save)
2277:     return report
2278:
2279:

```

```

2280: def helper_mapping_report_72(grid: GridMap, rf: RiskField):
2281:     """Generate a small JSON-like report for mapping snapshot #72 (for demo & testing)."""
2282:     rm = rf.compute()
2283:     # sample a few statistics
2284:     mean_risk = float(rm.mean())
2285:     max_risk = float(rm.max())
2286:     free_cells = int((~grid.obstacles).sum())
2287:     report = {
2288:         "snapshot": 72,
2289:         "mean_risk": mean_risk,
2290:         "max_risk": max_risk,
2291:         "free_cells": free_cells
2292:     }
2293:     # return dictionary (caller may save)
2294:     return report
2295:
2296:
2297: def helper_mapping_report_73(grid: GridMap, rf: RiskField):
2298:     """Generate a small JSON-like report for mapping snapshot #73 (for demo & testing)."""
2299:     rm = rf.compute()
2300:     # sample a few statistics
2301:     mean_risk = float(rm.mean())
2302:     max_risk = float(rm.max())
2303:     free_cells = int((~grid.obstacles).sum())
2304:     report = {
2305:         "snapshot": 73,
2306:         "mean_risk": mean_risk,
2307:         "max_risk": max_risk,
2308:         "free_cells": free_cells
2309:     }
2310:     # return dictionary (caller may save)
2311:     return report
2312:
2313:
2314: def helper_mapping_report_74(grid: GridMap, rf: RiskField):
2315:     """Generate a small JSON-like report for mapping snapshot #74 (for demo & testing)."""
2316:     rm = rf.compute()
2317:     # sample a few statistics
2318:     mean_risk = float(rm.mean())
2319:     max_risk = float(rm.max())
2320:     free_cells = int((~grid.obstacles).sum())
2321:     report = {
2322:         "snapshot": 74,
2323:         "mean_risk": mean_risk,
2324:         "max_risk": max_risk,
2325:         "free_cells": free_cells
2326:     }
2327:     # return dictionary (caller may save)
2328:     return report
2329:
2330:
2331: def helper_mapping_report_75(grid: GridMap, rf: RiskField):
2332:     """Generate a small JSON-like report for mapping snapshot #75 (for demo & testing)."""

```

```

2333: rm = rf.compute()
2334: # sample a few statistics
2335: mean_risk = float(rm.mean())
2336: max_risk = float(rm.max())
2337: free_cells = int((~grid.obstacles).sum())
2338: report = {
2339:     "snapshot": 75,
2340:     "mean_risk": mean_risk,
2341:     "max_risk": max_risk,
2342:     "free_cells": free_cells
2343: }
2344: # return dictionary (caller may save)
2345: return report
2346:
2347:
2348: def helper_mapping_report_76(grid: GridMap, rf: RiskField):
2349:     """Generate a small JSON-like report for mapping snapshot #76 (for demo & testing)."""
2350:     rm = rf.compute()
2351:     # sample a few statistics
2352:     mean_risk = float(rm.mean())
2353:     max_risk = float(rm.max())
2354:     free_cells = int((~grid.obstacles).sum())
2355:     report = {
2356:         "snapshot": 76,
2357:         "mean_risk": mean_risk,
2358:         "max_risk": max_risk,
2359:         "free_cells": free_cells
2360:     }
2361:     # return dictionary (caller may save)
2362:     return report
2363:
2364:
2365: def helper_mapping_report_77(grid: GridMap, rf: RiskField):
2366:     """Generate a small JSON-like report for mapping snapshot #77 (for demo & testing)."""
2367:     rm = rf.compute()
2368:     # sample a few statistics
2369:     mean_risk = float(rm.mean())
2370:     max_risk = float(rm.max())
2371:     free_cells = int((~grid.obstacles).sum())
2372:     report = {
2373:         "snapshot": 77,
2374:         "mean_risk": mean_risk,
2375:         "max_risk": max_risk,
2376:         "free_cells": free_cells
2377:     }
2378:     # return dictionary (caller may save)
2379:     return report
2380:
2381:
2382: def helper_mapping_report_78(grid: GridMap, rf: RiskField):
2383:     """Generate a small JSON-like report for mapping snapshot #78 (for demo & testing)."""
2384:     rm = rf.compute()
2385:     # sample a few statistics

```

```

2386: mean_risk = float(rm.mean())
2387: max_risk = float(rm.max())
2388: free_cells = int((~grid.obstacles).sum())
2389: report = {
2390:     "snapshot": 78,
2391:     "mean_risk": mean_risk,
2392:     "max_risk": max_risk,
2393:     "free_cells": free_cells
2394: }
2395: # return dictionary (caller may save)
2396: return report
2397:
2398:
2399: def helper_mapping_report_79(grid: GridMap, rf: RiskField):
2400:     """Generate a small JSON-like report for mapping snapshot #79 (for demo & testing)."""
2401:     rm = rf.compute()
2402:     # sample a few statistics
2403:     mean_risk = float(rm.mean())
2404:     max_risk = float(rm.max())
2405:     free_cells = int((~grid.obstacles).sum())
2406:     report = {
2407:         "snapshot": 79,
2408:         "mean_risk": mean_risk,
2409:         "max_risk": max_risk,
2410:         "free_cells": free_cells
2411:     }
2412:     # return dictionary (caller may save)
2413:     return report
2414:
2415:
2416: def helper_mapping_report_80(grid: GridMap, rf: RiskField):
2417:     """Generate a small JSON-like report for mapping snapshot #80 (for demo & testing)."""
2418:     rm = rf.compute()
2419:     # sample a few statistics
2420:     mean_risk = float(rm.mean())
2421:     max_risk = float(rm.max())
2422:     free_cells = int((~grid.obstacles).sum())
2423:     report = {
2424:         "snapshot": 80,
2425:         "mean_risk": mean_risk,
2426:         "max_risk": max_risk,
2427:         "free_cells": free_cells
2428:     }
2429:     # return dictionary (caller may save)
2430:     return report
2431:
2432:
2433: def helper_mapping_report_81(grid: GridMap, rf: RiskField):
2434:     """Generate a small JSON-like report for mapping snapshot #81 (for demo & testing)."""
2435:     rm = rf.compute()
2436:     # sample a few statistics
2437:     mean_risk = float(rm.mean())
2438:     max_risk = float(rm.max())

```

```

2439: free_cells = int((~grid.obstacles).sum())
2440: report = {
2441:     "snapshot": 81,
2442:     "mean_risk": mean_risk,
2443:     "max_risk": max_risk,
2444:     "free_cells": free_cells
2445: }
2446: # return dictionary (caller may save)
2447: return report
2448:
2449:
2450: def helper_mapping_report_82(grid: GridMap, rf: RiskField):
2451:     """Generate a small JSON-like report for mapping snapshot #82 (for demo & testing)."""
2452:     rm = rf.compute()
2453:     # sample a few statistics
2454:     mean_risk = float(rm.mean())
2455:     max_risk = float(rm.max())
2456:     free_cells = int((~grid.obstacles).sum())
2457:     report = {
2458:         "snapshot": 82,
2459:         "mean_risk": mean_risk,
2460:         "max_risk": max_risk,
2461:         "free_cells": free_cells
2462:     }
2463:     # return dictionary (caller may save)
2464:     return report
2465:
2466:
2467: def helper_mapping_report_83(grid: GridMap, rf: RiskField):
2468:     """Generate a small JSON-like report for mapping snapshot #83 (for demo & testing)."""
2469:     rm = rf.compute()
2470:     # sample a few statistics
2471:     mean_risk = float(rm.mean())
2472:     max_risk = float(rm.max())
2473:     free_cells = int((~grid.obstacles).sum())
2474:     report = {
2475:         "snapshot": 83,
2476:         "mean_risk": mean_risk,
2477:         "max_risk": max_risk,
2478:         "free_cells": free_cells
2479:     }
2480:     # return dictionary (caller may save)
2481:     return report
2482:
2483:
2484: def helper_mapping_report_84(grid: GridMap, rf: RiskField):
2485:     """Generate a small JSON-like report for mapping snapshot #84 (for demo & testing)."""
2486:     rm = rf.compute()
2487:     # sample a few statistics
2488:     mean_risk = float(rm.mean())
2489:     max_risk = float(rm.max())
2490:     free_cells = int((~grid.obstacles).sum())
2491:     report = {

```

```

2492:     "snapshot": 84,
2493:     "mean_risk": mean_risk,
2494:     "max_risk": max_risk,
2495:     "free_cells": free_cells
2496: }
2497: # return dictionary (caller may save)
2498: return report
2499:
2500:
2501: def helper_mapping_report_85(grid: GridMap, rf: RiskField):
2502:     """Generate a small JSON-like report for mapping snapshot #85 (for demo & testing)."""
2503:     rm = rf.compute()
2504:     # sample a few statistics
2505:     mean_risk = float(rm.mean())
2506:     max_risk = float(rm.max())
2507:     free_cells = int((~grid.obstacles).sum())
2508:     report = {
2509:         "snapshot": 85,
2510:         "mean_risk": mean_risk,
2511:         "max_risk": max_risk,
2512:         "free_cells": free_cells
2513:     }
2514:     # return dictionary (caller may save)
2515:     return report
2516:
2517:
2518: def helper_mapping_report_86(grid: GridMap, rf: RiskField):
2519:     """Generate a small JSON-like report for mapping snapshot #86 (for demo & testing)."""
2520:     rm = rf.compute()
2521:     # sample a few statistics
2522:     mean_risk = float(rm.mean())
2523:     max_risk = float(rm.max())
2524:     free_cells = int((~grid.obstacles).sum())
2525:     report = {
2526:         "snapshot": 86,
2527:         "mean_risk": mean_risk,
2528:         "max_risk": max_risk,
2529:         "free_cells": free_cells
2530:     }
2531:     # return dictionary (caller may save)
2532:     return report
2533:
2534:
2535: def helper_mapping_report_87(grid: GridMap, rf: RiskField):
2536:     """Generate a small JSON-like report for mapping snapshot #87 (for demo & testing)."""
2537:     rm = rf.compute()
2538:     # sample a few statistics
2539:     mean_risk = float(rm.mean())
2540:     max_risk = float(rm.max())
2541:     free_cells = int((~grid.obstacles).sum())
2542:     report = {
2543:         "snapshot": 87,
2544:         "mean_risk": mean_risk,

```



```

2545:     "max_risk": max_risk,
2546:     "free_cells": free_cells
2547: }
2548: # return dictionary (caller may save)
2549: return report
2550:
2551:
2552: def helper_mapping_report_88(grid: GridMap, rf: RiskField):
2553:     """Generate a small JSON-like report for mapping snapshot #88 (for demo & testing)."""
2554:     rm = rf.compute()
2555:     # sample a few statistics
2556:     mean_risk = float(rm.mean())
2557:     max_risk = float(rm.max())
2558:     free_cells = int((~grid.obstacles).sum())
2559:     report = {
2560:         "snapshot": 88,
2561:         "mean_risk": mean_risk,
2562:         "max_risk": max_risk,
2563:         "free_cells": free_cells
2564:     }
2565:     # return dictionary (caller may save)
2566:     return report
2567:
2568:
2569: def helper_mapping_report_89(grid: GridMap, rf: RiskField):
2570:     """Generate a small JSON-like report for mapping snapshot #89 (for demo & testing)."""
2571:     rm = rf.compute()
2572:     # sample a few statistics
2573:     mean_risk = float(rm.mean())
2574:     max_risk = float(rm.max())
2575:     free_cells = int((~grid.obstacles).sum())
2576:     report = {
2577:         "snapshot": 89,
2578:         "mean_risk": mean_risk,
2579:         "max_risk": max_risk,
2580:         "free_cells": free_cells
2581:     }
2582:     # return dictionary (caller may save)
2583:     return report
2584:
2585:
2586: def helper_mapping_report_90(grid: GridMap, rf: RiskField):
2587:     """Generate a small JSON-like report for mapping snapshot #90 (for demo & testing)."""
2588:     rm = rf.compute()
2589:     # sample a few statistics
2590:     mean_risk = float(rm.mean())
2591:     max_risk = float(rm.max())
2592:     free_cells = int((~grid.obstacles).sum())
2593:     report = {
2594:         "snapshot": 90,
2595:         "mean_risk": mean_risk,
2596:         "max_risk": max_risk,
2597:         "free_cells": free_cells

```

```

2598: }
2599: # return dictionary (caller may save)
2600: return report
2601:
2602:
2603: def helper_mapping_report_91(grid: GridMap, rf: RiskField):
2604:     """Generate a small JSON-like report for mapping snapshot #91 (for demo & testing)."""
2605:     rm = rf.compute()
2606:     # sample a few statistics
2607:     mean_risk = float(rm.mean())
2608:     max_risk = float(rm.max())
2609:     free_cells = int((~grid.obstacles).sum())
2610:     report = {
2611:         "snapshot": 91,
2612:         "mean_risk": mean_risk,
2613:         "max_risk": max_risk,
2614:         "free_cells": free_cells
2615:     }
2616:     # return dictionary (caller may save)
2617:     return report
2618:
2619:
2620: def helper_mapping_report_92(grid: GridMap, rf: RiskField):
2621:     """Generate a small JSON-like report for mapping snapshot #92 (for demo & testing)."""
2622:     rm = rf.compute()
2623:     # sample a few statistics
2624:     mean_risk = float(rm.mean())
2625:     max_risk = float(rm.max())
2626:     free_cells = int((~grid.obstacles).sum())
2627:     report = {
2628:         "snapshot": 92,
2629:         "mean_risk": mean_risk,
2630:         "max_risk": max_risk,
2631:         "free_cells": free_cells
2632:     }
2633:     # return dictionary (caller may save)
2634:     return report
2635:
2636:
2637: def helper_mapping_report_93(grid: GridMap, rf: RiskField):
2638:     """Generate a small JSON-like report for mapping snapshot #93 (for demo & testing)."""
2639:     rm = rf.compute()
2640:     # sample a few statistics
2641:     mean_risk = float(rm.mean())
2642:     max_risk = float(rm.max())
2643:     free_cells = int((~grid.obstacles).sum())
2644:     report = {
2645:         "snapshot": 93,
2646:         "mean_risk": mean_risk,
2647:         "max_risk": max_risk,
2648:         "free_cells": free_cells
2649:     }
2650:     # return dictionary (caller may save)

```

```

2651:     return report
2652:
2653:
2654: def helper_mapping_report_94(grid: GridMap, rf: RiskField):
2655:     """Generate a small JSON-like report for mapping snapshot #94 (for demo & testing)."""
2656:     rm = rf.compute()
2657:     # sample a few statistics
2658:     mean_risk = float(rm.mean())
2659:     max_risk = float(rm.max())
2660:     free_cells = int((~grid.obstacles).sum())
2661:     report = {
2662:         "snapshot": 94,
2663:         "mean_risk": mean_risk,
2664:         "max_risk": max_risk,
2665:         "free_cells": free_cells
2666:     }
2667:     # return dictionary (caller may save)
2668:     return report
2669:
2670:
2671: def helper_mapping_report_95(grid: GridMap, rf: RiskField):
2672:     """Generate a small JSON-like report for mapping snapshot #95 (for demo & testing)."""
2673:     rm = rf.compute()
2674:     # sample a few statistics
2675:     mean_risk = float(rm.mean())
2676:     max_risk = float(rm.max())
2677:     free_cells = int((~grid.obstacles).sum())
2678:     report = {
2679:         "snapshot": 95,
2680:         "mean_risk": mean_risk,
2681:         "max_risk": max_risk,
2682:         "free_cells": free_cells
2683:     }
2684:     # return dictionary (caller may save)
2685:     return report
2686:
2687:
2688: def helper_mapping_report_96(grid: GridMap, rf: RiskField):
2689:     """Generate a small JSON-like report for mapping snapshot #96 (for demo & testing)."""
2690:     rm = rf.compute()
2691:     # sample a few statistics
2692:     mean_risk = float(rm.mean())
2693:     max_risk = float(rm.max())
2694:     free_cells = int((~grid.obstacles).sum())
2695:     report = {
2696:         "snapshot": 96,
2697:         "mean_risk": mean_risk,
2698:         "max_risk": max_risk,
2699:         "free_cells": free_cells
2700:     }
2701:     # return dictionary (caller may save)
2702:     return report
2703:

```

```

2704:
2705: def helper_mapping_report_97(grid: GridMap, rf: RiskField):
2706:     """Generate a small JSON-like report for mapping snapshot #97 (for demo & testing)."""
2707:     rm = rf.compute()
2708:     # sample a few statistics
2709:     mean_risk = float(rm.mean())
2710:     max_risk = float(rm.max())
2711:     free_cells = int((~grid.obstacles).sum())
2712:     report = {
2713:         "snapshot": 97,
2714:         "mean_risk": mean_risk,
2715:         "max_risk": max_risk,
2716:         "free_cells": free_cells
2717:     }
2718:     # return dictionary (caller may save)
2719:     return report
2720:
2721:
2722: def helper_mapping_report_98(grid: GridMap, rf: RiskField):
2723:     """Generate a small JSON-like report for mapping snapshot #98 (for demo & testing)."""
2724:     rm = rf.compute()
2725:     # sample a few statistics
2726:     mean_risk = float(rm.mean())
2727:     max_risk = float(rm.max())
2728:     free_cells = int((~grid.obstacles).sum())
2729:     report = {
2730:         "snapshot": 98,
2731:         "mean_risk": mean_risk,
2732:         "max_risk": max_risk,
2733:         "free_cells": free_cells
2734:     }
2735:     # return dictionary (caller may save)
2736:     return report
2737:
2738:
2739: def helper_mapping_report_99(grid: GridMap, rf: RiskField):
2740:     """Generate a small JSON-like report for mapping snapshot #99 (for demo & testing)."""
2741:     rm = rf.compute()
2742:     # sample a few statistics
2743:     mean_risk = float(rm.mean())
2744:     max_risk = float(rm.max())
2745:     free_cells = int((~grid.obstacles).sum())
2746:     report = {
2747:         "snapshot": 99,
2748:         "mean_risk": mean_risk,
2749:         "max_risk": max_risk,
2750:         "free_cells": free_cells
2751:     }
2752:     # return dictionary (caller may save)
2753:     return report
2754:
2755:
2756: def helper_mapping_report_100(grid: GridMap, rf: RiskField):

```

```

2757: """Generate a small JSON-like report for mapping snapshot #100 (for demo & testing)."""
2758: rm = rf.compute()
2759: # sample a few statistics
2760: mean_risk = float(rm.mean())
2761: max_risk = float(rm.max())
2762: free_cells = int((~grid.obstacles).sum())
2763: report = {
2764:     "snapshot": 100,
2765:     "mean_risk": mean_risk,
2766:     "max_risk": max_risk,
2767:     "free_cells": free_cells
2768: }
2769: # return dictionary (caller may save)
2770: return report
2771:
2772:
2773: def helper_mapping_report_101(grid: GridMap, rf: RiskField):
2774:     """Generate a small JSON-like report for mapping snapshot #101 (for demo & testing)."""
2775:     rm = rf.compute()
2776:     # sample a few statistics
2777:     mean_risk = float(rm.mean())
2778:     max_risk = float(rm.max())
2779:     free_cells = int((~grid.obstacles).sum())
2780:     report = {
2781:         "snapshot": 101,
2782:         "mean_risk": mean_risk,
2783:         "max_risk": max_risk,
2784:         "free_cells": free_cells
2785:     }
2786:     # return dictionary (caller may save)
2787:     return report
2788:
2789:
2790: def helper_mapping_report_102(grid: GridMap, rf: RiskField):
2791:     """Generate a small JSON-like report for mapping snapshot #102 (for demo & testing)."""
2792:     rm = rf.compute()
2793:     # sample a few statistics
2794:     mean_risk = float(rm.mean())
2795:     max_risk = float(rm.max())
2796:     free_cells = int((~grid.obstacles).sum())
2797:     report = {
2798:         "snapshot": 102,
2799:         "mean_risk": mean_risk,
2800:         "max_risk": max_risk,
2801:         "free_cells": free_cells
2802:     }
2803:     # return dictionary (caller may save)
2804:     return report
2805:
2806:
2807: def helper_mapping_report_103(grid: GridMap, rf: RiskField):
2808:     """Generate a small JSON-like report for mapping snapshot #103 (for demo & testing)."""
2809:     rm = rf.compute()

```

```

2810: # sample a few statistics
2811: mean_risk = float(rm.mean())
2812: max_risk = float(rm.max())
2813: free_cells = int((~grid.obstacles).sum())
2814: report = {
2815:     "snapshot": 103,
2816:     "mean_risk": mean_risk,
2817:     "max_risk": max_risk,
2818:     "free_cells": free_cells
2819: }
2820: # return dictionary (caller may save)
2821: return report
2822:
2823:
2824: def helper_mapping_report_104(grid: GridMap, rf: RiskField):
2825:     """Generate a small JSON-like report for mapping snapshot #104 (for demo & testing)."""
2826:     rm = rf.compute()
2827:     # sample a few statistics
2828:     mean_risk = float(rm.mean())
2829:     max_risk = float(rm.max())
2830:     free_cells = int((~grid.obstacles).sum())
2831:     report = {
2832:         "snapshot": 104,
2833:         "mean_risk": mean_risk,
2834:         "max_risk": max_risk,
2835:         "free_cells": free_cells
2836:     }
2837:     # return dictionary (caller may save)
2838:     return report
2839:
2840:
2841: def helper_mapping_report_105(grid: GridMap, rf: RiskField):
2842:     """Generate a small JSON-like report for mapping snapshot #105 (for demo & testing)."""
2843:     rm = rf.compute()
2844:     # sample a few statistics
2845:     mean_risk = float(rm.mean())
2846:     max_risk = float(rm.max())
2847:     free_cells = int((~grid.obstacles).sum())
2848:     report = {
2849:         "snapshot": 105,
2850:         "mean_risk": mean_risk,
2851:         "max_risk": max_risk,
2852:         "free_cells": free_cells
2853:     }
2854:     # return dictionary (caller may save)
2855:     return report
2856:
2857:
2858: def helper_mapping_report_106(grid: GridMap, rf: RiskField):
2859:     """Generate a small JSON-like report for mapping snapshot #106 (for demo & testing)."""
2860:     rm = rf.compute()
2861:     # sample a few statistics
2862:     mean_risk = float(rm.mean())

```

```

2863: max_risk = float(rm.max())
2864: free_cells = int((~grid.obstacles).sum())
2865: report = {
2866:     "snapshot": 106,
2867:     "mean_risk": mean_risk,
2868:     "max_risk": max_risk,
2869:     "free_cells": free_cells
2870: }
2871: # return dictionary (caller may save)
2872: return report
2873:
2874:
2875: def helper_mapping_report_107(grid: GridMap, rf: RiskField):
2876:     """Generate a small JSON-like report for mapping snapshot #107 (for demo & testing)."""
2877:     rm = rf.compute()
2878:     # sample a few statistics
2879:     mean_risk = float(rm.mean())
2880:     max_risk = float(rm.max())
2881:     free_cells = int((~grid.obstacles).sum())
2882:     report = {
2883:         "snapshot": 107,
2884:         "mean_risk": mean_risk,
2885:         "max_risk": max_risk,
2886:         "free_cells": free_cells
2887:     }
2888:     # return dictionary (caller may save)
2889:     return report
2890:
2891:
2892: def helper_mapping_report_108(grid: GridMap, rf: RiskField):
2893:     """Generate a small JSON-like report for mapping snapshot #108 (for demo & testing)."""
2894:     rm = rf.compute()
2895:     # sample a few statistics
2896:     mean_risk = float(rm.mean())
2897:     max_risk = float(rm.max())
2898:     free_cells = int((~grid.obstacles).sum())
2899:     report = {
2900:         "snapshot": 108,
2901:         "mean_risk": mean_risk,
2902:         "max_risk": max_risk,
2903:         "free_cells": free_cells
2904:     }
2905:     # return dictionary (caller may save)
2906:     return report
2907:
2908:
2909: def helper_mapping_report_109(grid: GridMap, rf: RiskField):
2910:     """Generate a small JSON-like report for mapping snapshot #109 (for demo & testing)."""
2911:     rm = rf.compute()
2912:     # sample a few statistics
2913:     mean_risk = float(rm.mean())
2914:     max_risk = float(rm.max())
2915:     free_cells = int((~grid.obstacles).sum())

```

```

2916:     report = {
2917:         "snapshot": 109,
2918:         "mean_risk": mean_risk,
2919:         "max_risk": max_risk,
2920:         "free_cells": free_cells
2921:     }
2922:     # return dictionary (caller may save)
2923:     return report
2924:
2925:
2926: def helper_mapping_report_110(grid: GridMap, rf: RiskField):
2927:     """Generate a small JSON-like report for mapping snapshot #110 (for demo & testing)."""
2928:     rm = rf.compute()
2929:     # sample a few statistics
2930:     mean_risk = float(rm.mean())
2931:     max_risk = float(rm.max())
2932:     free_cells = int((~grid.obstacles).sum())
2933:     report = {
2934:         "snapshot": 110,
2935:         "mean_risk": mean_risk,
2936:         "max_risk": max_risk,
2937:         "free_cells": free_cells
2938:     }
2939:     # return dictionary (caller may save)
2940:     return report
2941:
2942:
2943: def helper_mapping_report_111(grid: GridMap, rf: RiskField):
2944:     """Generate a small JSON-like report for mapping snapshot #111 (for demo & testing)."""
2945:     rm = rf.compute()
2946:     # sample a few statistics
2947:     mean_risk = float(rm.mean())
2948:     max_risk = float(rm.max())
2949:     free_cells = int((~grid.obstacles).sum())
2950:     report = {
2951:         "snapshot": 111,
2952:         "mean_risk": mean_risk,
2953:         "max_risk": max_risk,
2954:         "free_cells": free_cells
2955:     }
2956:     # return dictionary (caller may save)
2957:     return report
2958:
2959:
2960: def helper_mapping_report_112(grid: GridMap, rf: RiskField):
2961:     """Generate a small JSON-like report for mapping snapshot #112 (for demo & testing)."""
2962:     rm = rf.compute()
2963:     # sample a few statistics
2964:     mean_risk = float(rm.mean())
2965:     max_risk = float(rm.max())
2966:     free_cells = int((~grid.obstacles).sum())
2967:     report = {
2968:         "snapshot": 112,

```



```

2969:     "mean_risk": mean_risk,
2970:     "max_risk": max_risk,
2971:     "free_cells": free_cells
2972: }
2973: # return dictionary (caller may save)
2974: return report
2975:
2976:
2977: def helper_mapping_report_113(grid: GridMap, rf: RiskField):
2978:     """Generate a small JSON-like report for mapping snapshot #113 (for demo & testing)."""
2979:     rm = rf.compute()
2980:     # sample a few statistics
2981:     mean_risk = float(rm.mean())
2982:     max_risk = float(rm.max())
2983:     free_cells = int((~grid.obstacles).sum())
2984:     report = {
2985:         "snapshot": 113,
2986:         "mean_risk": mean_risk,
2987:         "max_risk": max_risk,
2988:         "free_cells": free_cells
2989:     }
2990:     # return dictionary (caller may save)
2991:     return report
2992:
2993:
2994: def helper_mapping_report_114(grid: GridMap, rf: RiskField):
2995:     """Generate a small JSON-like report for mapping snapshot #114 (for demo & testing)."""
2996:     rm = rf.compute()
2997:     # sample a few statistics
2998:     mean_risk = float(rm.mean())
2999:     max_risk = float(rm.max())
3000:     free_cells = int((~grid.obstacles).sum())
3001:     report = {
3002:         "snapshot": 114,
3003:         "mean_risk": mean_risk,
3004:         "max_risk": max_risk,
3005:         "free_cells": free_cells
3006:     }
3007:     # return dictionary (caller may save)
3008:     return report
3009:
3010:
3011: def helper_mapping_report_115(grid: GridMap, rf: RiskField):
3012:     """Generate a small JSON-like report for mapping snapshot #115 (for demo & testing)."""
3013:     rm = rf.compute()
3014:     # sample a few statistics
3015:     mean_risk = float(rm.mean())
3016:     max_risk = float(rm.max())
3017:     free_cells = int((~grid.obstacles).sum())
3018:     report = {
3019:         "snapshot": 115,
3020:         "mean_risk": mean_risk,
3021:         "max_risk": max_risk,

```

```

3022:     "free_cells": free_cells
3023: }
3024: # return dictionary (caller may save)
3025: return report
3026:
3027:
3028: def helper_mapping_report_116(grid: GridMap, rf: RiskField):
3029:     """Generate a small JSON-like report for mapping snapshot #116 (for demo & testing)."""
3030:     rm = rf.compute()
3031:     # sample a few statistics
3032:     mean_risk = float(rm.mean())
3033:     max_risk = float(rm.max())
3034:     free_cells = int((~grid.obstacles).sum())
3035:     report = {
3036:         "snapshot": 116,
3037:         "mean_risk": mean_risk,
3038:         "max_risk": max_risk,
3039:         "free_cells": free_cells
3040:     }
3041:     # return dictionary (caller may save)
3042:     return report
3043:
3044:
3045: def helper_mapping_report_117(grid: GridMap, rf: RiskField):
3046:     """Generate a small JSON-like report for mapping snapshot #117 (for demo & testing)."""
3047:     rm = rf.compute()
3048:     # sample a few statistics
3049:     mean_risk = float(rm.mean())
3050:     max_risk = float(rm.max())
3051:     free_cells = int((~grid.obstacles).sum())
3052:     report = {
3053:         "snapshot": 117,
3054:         "mean_risk": mean_risk,
3055:         "max_risk": max_risk,
3056:         "free_cells": free_cells
3057:     }
3058:     # return dictionary (caller may save)
3059:     return report
3060:
3061:
3062: def helper_mapping_report_118(grid: GridMap, rf: RiskField):
3063:     """Generate a small JSON-like report for mapping snapshot #118 (for demo & testing)."""
3064:     rm = rf.compute()
3065:     # sample a few statistics
3066:     mean_risk = float(rm.mean())
3067:     max_risk = float(rm.max())
3068:     free_cells = int((~grid.obstacles).sum())
3069:     report = {
3070:         "snapshot": 118,
3071:         "mean_risk": mean_risk,
3072:         "max_risk": max_risk,
3073:         "free_cells": free_cells
3074:     }

```

```

3075: # return dictionary (caller may save)
3076: return report
3077:
3078:
3079: def helper_mapping_report_119(grid: GridMap, rf: RiskField):
3080:     """Generate a small JSON-like report for mapping snapshot #119 (for demo & testing)."""
3081:     rm = rf.compute()
3082:     # sample a few statistics
3083:     mean_risk = float(rm.mean())
3084:     max_risk = float(rm.max())
3085:     free_cells = int((~grid.obstacles).sum())
3086:     report = {
3087:         "snapshot": 119,
3088:         "mean_risk": mean_risk,
3089:         "max_risk": max_risk,
3090:         "free_cells": free_cells
3091:     }
3092:     # return dictionary (caller may save)
3093:     return report
3094:
3095:
3096:
3097:
3098: # =====
3099: # Extended demo runner: batch runs for reproducible screenshots
3100: # =====
3101: @timeit
3102: def batch_demo_runs(out_prefix="batch_demo", runs=6):
3103:     results = []
3104:     for r in range(runs):
3105:         seed = 100 + r*7
3106:         W,H = DEFAULT_CONFIG["map"]["width"], DEFAULT_CONFIG["map"]["height"]
3107:         grid = GridMap(W,H); grid.random_corridors(blocks=7, seed=seed)
3108:         rf = RiskField(W,H, base=DEFAULT_CONFIG["risk"].get("base_level",0.0))
3109:         batch_add_sources(rf, DEFAULT_CONFIG["risk"]["sources"])
3110:         rf.step() # initialize dynamics
3111:         detector = SimulatedYOLO(rf)
3112:         slam = SLAMMock()
3113:         start = (1 + (r%3), 1 + (r%4)); goal = (W-2, H-2 - (r%2))
3114:         robot = Robot(start=start, goal=goal, grid=grid, risk=rf, slam=slam, detector=detector)
3115:         robot.plan()
3116:         sim = Simulator(grid, rf, robot)
3117:         steps = 0
3118:         while steps < 400:
3119:             cont = sim.step()
3120:             if not cont:
3121:                 break
3122:             steps += 1
3123:         # save a small report
3124:         rep = helper_mapping_report_1(grid, rf)
3125:         rep["run"] = r
3126:         results.append(rep)
3127:     try:

```

```

3128:         # export visuals for the first run
3129:         if r == 0:
3130:             viz = Visualizer(grid, rf, robot, sim)
3131:             viz.draw()
3132:             import matplotlib.pyplot as plt
3133:             plt.savefig(f"{out_prefix}_run{r}_result.png", dpi=200)
3134:         except Exception as e:
3135:             logger.warning("Visual save failed: %s", e)
3136:         # save combined results
3137:         save_json(f"{out_prefix}_summary.json", {"results": results})
3138:         return results
3139:
3140:
3141:
3142:
3143:
3144: # =====
3145: # Appendix: lightweight unit-test runner (not using unittest to keep single-file)
3146: # =====
3147: def run_unit_tests():
3148:     logger.info("Running unit tests...")
3149:     path = simple_planner_smoke_test()
3150:     if path:
3151:         logger.info("Smoke test produced path length %d", len(path))
3152:     perf_path = planner_perf_test()
3153:     if perf_path:
3154:         logger.info("Perf test path length %d", len(perf_path))
3155:     # run batch demo (short)
3156:     results = batch_demo_runs(out_prefix="unit_test_demo", runs=2)
3157:     logger.info("Batch demo results: %s", results)
3158:     logger.info("All unit tests completed (informal).")
3159:
3160: if __name__ == "__main__":
3161:     setup_file_logger("run_expanded.log")
3162:     print_config(DEFAULT_CONFIG)
3163:     run_unit_tests()
3164:
3165:
3166:
3167: # === END EXPANSION ===

```