1 避難者 = 1 台の soft-Xinu の 4 エージェント（アクター）ソースコード

phone_node_fire.abcl —— 多賀 2021 第 6 章 火災シナリオ / OCaml AIPL

① 情報 InfoAgent・② ノード管理 NodeManager・③ 情報拡散 DiffusionAgent(push,hop1)・④ 情報収集 CollectingAgent(pull)

```
// phone_node_fire.abcl —— 多賀(2021) 第6章 大規模火災シナリオ用ノード
// ============================================================================
// 「1 避難者 = 1 台の soft-Xinu / AIPL ノード」。各携帯端末に第4章の 4 エージェントを
// 載せ、web_expose で公開し web_listen(PORT) で Xinu 互換のアクター HTTP ゲートウェイ
// (POST /api/json/call) に載せる。実機 Embedded-Xinu(rpi3/4/5) と同一の配線なので、
// この .abcl をそのまま実機へ載せても動く。
//
//   4 エージェント(多賀論文 第4章 / 第6章 火災シナリオ):
//    ┌ 静的エージェント (端末に常駐) ──────────────────────────────────────────┐
//    │  ① 情報エージェント        InfoAgent       "info"                       │
//    │       端末の頭脳。通行不能点(火災)の発見/学習、移動エージェントの      │
//    │       生成・派遣、避難経路の再構成(既知の通行不能点を避ける)。         │
//    │  ② ノード管理エージェント   NodeManager     "nodemgr"                   │
//    │       端末が「共有可能」として公開する通行不能点テーブルを保持。       │
//    │       隣端末から来た収集エージェントはここを読みに来る。               │
//    └──────────────────────────────────────────────────────────────────────┘
//    ┌ 移動エージェント (端末間を渡り歩く) ──────────────────────────────────┐
//    │  ③ 情報拡散エージェント     DiffusionAgent "diffusion" (push)          │
//    │       通行不能点 ID を載せて電波内(通信範囲50m)の隣端末へ「移動」し、  │
//    │       着いた先で情報エージェントへ預ける。ホップ回数は 1(第6章 図6.2)。│
//    │  ④ 情報収集エージェント     CollectingAgent "collecting" (pull)        │
//    │       隣端末のノード管理エージェントを巡回訪問し、未知の通行不能点     │
//    │       ID を持ち帰って自端末へ取り込む。                                │
//    └──────────────────────────────────────────────────────────────────────┘
//
//   通信方式(実機 Xinu と同じ AIPL ランタイム特性):
//     ・同一端末内 (info<->nodemgr<->mobile) …… in-process の非同期 send。
//     ・端末をまたぐ移動 (隣端末への migration) …… now remote(host,actor).method()
//       によるネイティブ現在型(同期)リモート送信。着信ハンドラは即 reply。
//
//   世界(各端末の位置・移動・電波到達=隣接・時計・火災の発生)は外部コントローラ
//   が司り、毎tick set_step() で「今この瞬間に電波が届く隣人(host:port)」を渡す。
//   端末は自分の隣人しか知らない —— 基地局の無い MANET の現実そのもの。
//   __PORT__ はコントローラが起動時にノードごとに書き換える。
// ============================================================================

// 配列 arr に id が含まれるか(集合を線形走査で表現)
function has(arr, id) {
  var i = 0;
  var n = array_len(arr);
  while (i < n) do {
    if (arr[i] == id) { return 1; }
    i = i + 1;
  }
  return 0;
}

// ----------------------------------------------------------------------------
// ② ノード管理エージェント (静的) — 端末が共有公開する通行不能点テーブル
```

```
//    純データアクター(onward call 無し)なので、いつ同期着信しても安全。
// ---------------------------------------------------------------------------
class NodeManager {
  var known = [];              // この端末が「共有可能」として公開する通行不能点 ID 群

  method store(id) {           // 情報エージェントが学習するたびに反映(in-process send)
    var isnew = 0;
    if (has(known, id) == 0) { array_push(known, id); isnew = 1; }
    reply(isnew);              // 1=新規 / 0=既知
  }
  method all()   { reply(known); }              // 巡回してきた収集エージェントが読む
  method count() { reply(array_len(known)); }
}

// ---------------------------------------------------------------------------
// ① 情報エージェント (静的) — 端末の頭脳
//    通行不能点の発見/学習・移動エージェント派遣・避難経路の再構成。
// ---------------------------------------------------------------------------
class InfoAgent {
  var nm      = 0;             // 同端末のノード管理エージェント
  var diff    = 0;             // 同端末の情報拡散エージェント
  var coll    = 0;             // 同端末の情報収集エージェント
  var nbrs    = [];            // 今この瞬間に電波が届く隣人 host:port(世界が毎tick更新)
  var known   = [];            // 学習済みの通行不能点 ID(避難経路の再計算に使う)
  var pending = [];            // 次tickで拡散エージェントに載せる通行不能点 ID
  var contacts = 0;            // この端末が通行不能点に接触した合計回数(図6.2)

  method wire(node_mgr, diffusion, collecting) {
    nm = node_mgr; diff = diffusion; coll = collecting; reply(1);
  }

  // 避難者がこの端末で通行不能点(火災)に接触した(=図6.2 の接触1回)
  method contact(id) {
    contacts = contacts + 1;
    if (has(known, id) == 0) {
      array_push(known, id);
      array_push(pending, id);   // 次tickで拡散エージェントが周囲へ通知
      send nm.store(id);         // 共有テーブル(ノード管理)へ反映
    }
    reply(contacts);
  }

  // 移動エージェントが運んできた通行不能点 ID を取り込む(非同期コールバック)
  method learn(id) {
    if (has(known, id) == 0) {
      array_push(known, id);
      array_push(pending, id);
      send nm.store(id);
    }
    reply(1);
  }

  // 世界が毎 tick、現在電波が届く隣人(host:port)リストを渡す
  method set_step(list) { nbrs = list; reply(array_len(nbrs)); }

  // 1 シミュレーションステップ: 2 種の移動エージェントを派遣
  method tick() {
```

```
    send diff.dispatch(pending, nbrs);    // ③ 拡散(push): 待機中の通行不能点を隣人へ
    pending = [];
    send coll.dispatch(nbrs);             // ④ 収集(pull): 隣人のノード管理を巡回
    reply(array_len(known));
  }

  // コントローラが取得: [ 既知通行不能点数, 接触回数 ]
  method status() {
    var s = [];
    array_push(s, array_len(known));
    array_push(s, contacts);
    reply(s);
  }
  method known_points() { reply(known); }   // 経路再構成のため既知点を返す
}

// ---------------------------------------------------------------------------
// ③ 情報拡散エージェント (移動 / push) — ホップ 1(第6章 図6.2 の条件)
//    HOME 側 dispatch(): 載せた通行不能点 ID を電波内の各隣人へ migration。
//    着信側 receive():   運ばれてきた ID を自端末の情報エージェントへ預ける。
// ---------------------------------------------------------------------------
class DiffusionAgent {
  var info = 0;
  method wire(info_agent) { info = info_agent; reply(1); }

  // HOME 側: pending の各 ID を、電波内の全隣人へ 1 ホップ移動させる
  method dispatch(ids, nbrs) {
    var i = 0; var ni = array_len(ids);
    while (i < ni) do {
      var id = ids[i];
      var j = 0; var nj = array_len(nbrs);
      while (j < nj) do {
        var nbr = nbrs[j];
        var _r = now remote(nbr, "diffusion").receive(id);  // 隣端末へ「移動」(hop 1)
        j = j + 1;
      }
      i = i + 1;
    }
    reply(1);
  }

  // 着信側: 拡散エージェントがこの端末へ移動してきた(ID を運搬)。hop=1 なので再拡散しない。
  method receive(id) { send info.learn(id); reply(1); }
}

// ---------------------------------------------------------------------------
// ④ 情報収集エージェント (移動 / pull)
//    HOME 側 dispatch(): 電波内の各隣人のノード管理を巡回し、未知の通行不能点を持ち帰る。
// ---------------------------------------------------------------------------
class CollectingAgent {
  var info = 0;
  method wire(info_agent) { info = info_agent; reply(1); }

  method dispatch(nbrs) {
    var j = 0; var nj = array_len(nbrs);
    while (j < nj) do {
      var nbr = nbrs[j];
```

```
      var got = now remote(nbr, "nodemgr").all();   // 隣端末の公開テーブルを読む
      var g = 0; var gn = array_len(got);
      while (g < gn) do {
        send info.learn(got[g]);   // 持ち帰った情報を自端末へ取り込む（重複は無視）
        g = g + 1;
      }
      j = j + 1;
    }
    reply(1);
  }
}

// ---------------------------------------------------------------------------
// 端末起動：4 エージェントを生成 → 相互配線 → 公開 → ゲートウェイ起動
// ---------------------------------------------------------------------------
var nodemgr    = new NodeManager();
var info       = new InfoAgent();
var diffusion  = new DiffusionAgent();
var collecting = new CollectingAgent();

send info.wire(nodemgr, diffusion, collecting);
send diffusion.wire(info);
send collecting.wire(info);

web_expose("nodemgr",    nodemgr);     // ② ノード管理エージェント
web_expose("info",       info);        // ① 情報エージェント
web_expose("diffusion",  diffusion);   // ③ 情報拡散エージェント（移動）
web_expose("collecting", collecting);  // ④ 情報収集エージェント（移動）
web_listen(__PORT__);

print("[phone] up http://localhost:__PORT__  agents=info,nodemgr,diffusion,collecting  (Xinu-
    compatible fire-scenario MANET node)");
```