Digital environments – Version 2 (consolidated environment with interoperability connectors & KPIs)

Deliverable D1.3

Distribution level: Public

Version: 1.0

Date of delivery: 29/09/2025

AI4REALNET has received funding from European Union's Horizon Europe Research and Innovation programme under the Grant Agreement No 101119527, and from the Swiss State Secretariat for Education, Research and Innovation.

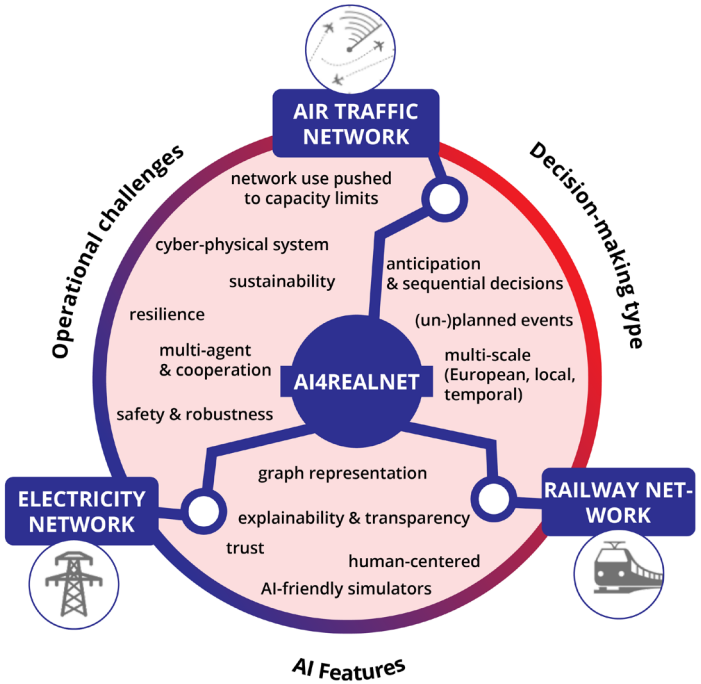
List of authors

Authors	Institution
Giulia Leto	Delft University of Technology
Clark Borst	Delft University of Technology
Joost Ellerbrook	Delft University of Technology
Benjamin Donnot	RTE
Geoffroy Jamgotchian	RTE
Antoine Marot	RTE
Bruno Lemetayer	RTE
Maroua Meddeb	IRT SystemX
Milad Leyli-Abadi	IRT SystemX
Abderrahman Ait Said	IRT SystemX
Manuel Schneider	Flatland (FLAT)
Christian Eichenberger	Flatland (FLAT)
Manuel Meyer	Flatland (FLAT)

Background: AI4REALNET

GOALS

- Develop the next generation of decision-making methods powered by supervised and reinforcement learning, which aim at trustworthiness in AI-assisted human control, human-AI co-learning, and autonomous AI
- Boost the development and validation of novel AI algorithms via 3 existing open-source AI-friendly digital environments



List of software modules, data & resources

- Grid2op:
  - AI4REALNET/grid2op: platform to model sequential decision making in power systems
  - AI4REALNET/pypowsybl2grid: integration between Grid2op and PyPowSyBl backend (Python library for power grid modelling and simulation)
- Flatland:
  - AI4REALNET/flatland-rl: Simulation environment for modeling railway scheduling and rescheduling problems using reinforcement learning and operation s research. Support for Gymnasium[1], RLlib[2], and PettingZoo[3] Environments. Generation of synthetic railway environments.
  - AI4REALNET/flatland-baselines: Baseline models for flatland.
  - AI4REALNET/flatland-scenarios: Scenarios for the railway use cases.
  - AI4REALNET/flatland-book: Conceptual and technical documentation of the Flatland environment.
- BlueSky:
  - AI4REALNET/BlueSky-Gym: A Gymnasium[1] style library for standardized Reinforcement Learning research in Air Traffic Management.
  - Use cases in BlueSky scenarios (proprietary data stored in the AI4REALNET GitLab, available upon request)
  - AI4REALNET/ATMSectorization: an interface for transparent decision making for sectorization, supporting use case 1.
  - AI4REALNET/bluesky, readme: plugins for testing BlueSky-Gym RL models on realistic scenarios
  - FAB ATM test runner: sample test runner that connects the testing of RL in BlueSky with the FAB client
- InteractiveAI:
  - AI4REALNET/InteractiveAI : Generic platform for human interaction with complex networks
  - Specific HMI modules for each industrial domain (ATM, Power grid, Railway)
  - Interfaces with each digital environment (BlueSky, Grid2Op, Flatland)
  - https://github.com/AI4REALNET/InteractiveAI-API Generic interoperability APIs to ensure communication with different digital environment
  - Instantiation of the generic interoperability APIs with the Railway Use Case using Maze Flatland AI agent

[1] https://gymnasium.farama.org/
[2] https://docs.ray.io/en/latest/rllib/index.html
[3] https://pettingzoo.farama.org/index.html

Grid2Op

Developer: RTE

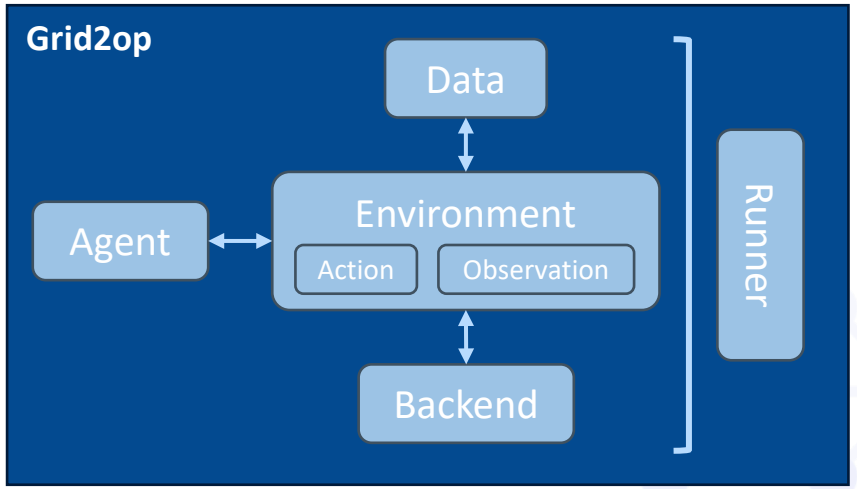
General description

Develop / train / evaluate performances of an agent acting on a Power grid

- Simulates model sequential decision making made by human operators to ease the research on sequential decision making applied to power systems
- Fully compatible with standard Gymnasium Reinforcement Learning API
- Abstracts the computation of the cascading failures
- Allows running same code with multiple power flows simulators (PandaPower, PyPowsybl, LightSim2Grid)
- Execute one agent on multiple independent scenarios in parallel
- Can be fully customized to serve different purposes (and not only Reinforcement Learning)
- Developed in Python
- Full documentation available at https://grid2op.readthedocs.io/en/latest/

Block diagram

- **Environment** is a representation of the power grid with which an Agent interacts (Generators, Loads, Powerlines, Shunts, Storage units, Substations)
- **Agents** can perform **Actions** on the environment, and receive **Observations** that correspond to its current or forecasted state
- Environment state evolves using **Data** (Time series) corresponding to injections (generators, loads) and structural information (planned outage, hazards)
- **Backend** allows calculating the physical state of the environment (Power grid, flow, intensity, etc.)
- The evaluation of agents can be performed by **Runner**, allowing corresponding data to be logged



Other experimental modules exists as part of the "GridAlive" ecosystem, but are not used in the project:

- Grid2Game (basic human GUI)
- Grid2Viz (visualization GUI)
- Grid2Bench (KPIs and benchmark)

Input data

- Environment
  - environment time series can be read from CSV (possibly by chunks to improve IO runtime)
  - “grid.json” is the full description of the powergrid characteristics, to be used by the backend
  - “grid_layout.json” is the geographic description of the powergrid for display
  - “config.py” is the configuration file
- Agent
  - Abstract class grid2op.Action.BaseAgent API can be implemented/extended to create custom agents
  - To perform their actions, Agents
    - receive grid2op.Reward.BaseReward and grid2op.Observation.BaseObservation data from the grid2op.Environment API
    - interacts with the grid2op.Action.ActionSpace API from the grid2op.Environment API
  - Abstract class grid2op.Action.BaseAction API can be implemented/extended to create custom actions
- Backends : Since release v1, PyPowsybl backend is available thanks to pypowsybl2grid package

Output data

- Environment
  - State of the environment is described by the grid2op.Observation.BaseObservation object
  - Observations are described by more than 30 attributes available in Observation API
  - Visualized using basic plotting features

    Grid plot image can be generated using backend functionalities

    Grid2Op allows for more advanced display features to be developed, for example in an advanced HMI framework
  - Exported in standard file formats (for example, iidm(*) thanks to the PyPowsybl backend)
- Runner : results of an evaluation are saved in standard JSON and NumPy (.npz) formats
- EnvRecorder added in v2:
  - Environment recorder for capturing and storing environment data
  - Observations saved into Parquet files for later analysis

(*) internal grid model initially developed under the iTesla research project that was funded by the European Union 7th Framework Programme (FP7)

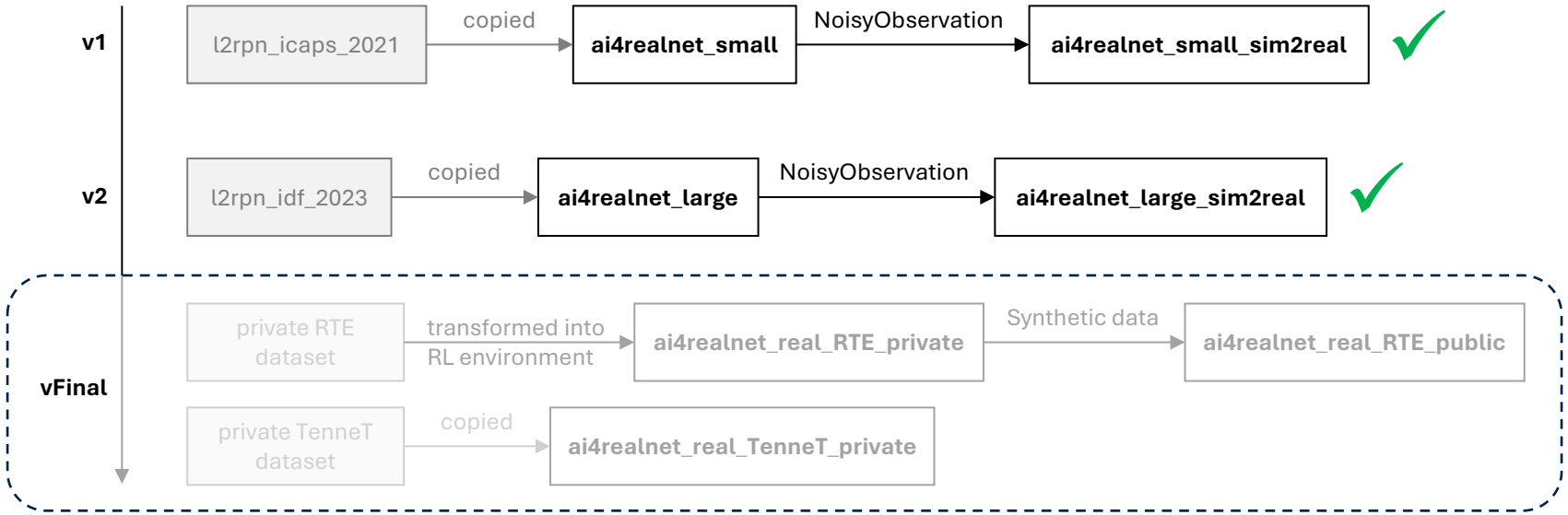
Grid2Evaluate

KPI calculation based on data produced by Grid2Op

- KPI-CF-012 Carbon Intensity
- KPI-TF-034 Topological Action Complexity
- KPI-NF-024 Network Utilization
- KPI-OF-036 Operation Score
- KPI-AF-008 Assistant Alert Accuracy

Grid2op – Environments

Implemented Power Grid environments



Expected for vFinal : real data environments

Grid2op – Links to repositories

https://github.com/AI4REALNET/grid2op

grid2op main repo

https://github.com/AI4REALNET/grid2evaluate

grid2op KPIs

https://github.com/AI4REALNET/pypowsybl2grid

An integration between Grid2op and PyPowSybl backend

https://github.com/AI4REALNET/grid2op-scenario

Scenarios to be used

Flatland

Developer: Flatland Association

General description

The Flatland environment provides simplistic representation of a rail network on a grid world to address the vehicle rescheduling problem (VRSP)

- Flatland is used to develop reinforcement learning (RL) solutions to the VRSP
- Trains are agents with a limited action space (↑ , ← , → , ✕ , ⟳)
- Agents have schedules for their origin, destination and intermediate stops
- Railway network includes switches, slips, crossings and over-/underpasses
- Translation from grid representation of the network to a graph representation is implemented
- Agents can have speed profiles, reflecting different train classes (passenger, freight, etc.)
- Agents can be disrupted (in malfunction)

flatland-rl flatland-scenarios flatland-baselines flatland-book

New Features

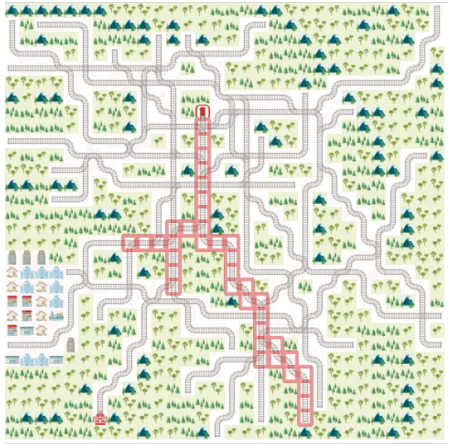
- New type of disruption (previous disruption: train breakdowns)
  - Departure delays (disruptions at stops)
- Observation Perturbations
- Multi objectives capability
  - 3 objectives as examples implemented (previous reward function for delays, late arrivals, collisions, journeys not started and stops not served, and 2 new rewards for energy efficiency and smoothness of travel)
- Improved Simulation stepper aka trajectory API
  - The state of the simulation can be read and manipulated at every timestep

Input data

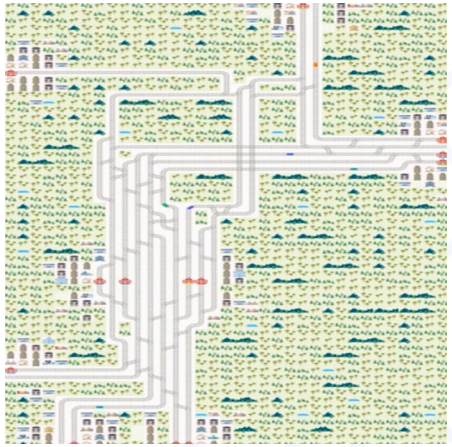
- Environment
  - Environments, i.e., railway network, train stations, schedules, lines, etc., can be specified manually or being generated based on parameters (grid size, number of agents, network characteristics, number of stations, etc.) in Python scripts.
  - Environments can be loaded from PKL (Pickled Python Objects) files.
  - Simulation runs, i.e., a sequence of environment states and actions (trajectories), can be loaded from PKL and TSV (tab-separated values) files.
- Agent
  - Agents interact with the environment by providing one of 5 actions (move forward, move left, move right, stop, do nothing) at each time step. With variable speed: acceleration by move forward action, deceleration by stop action.
  - Agents have access to an observation and rewards at each time step.
  - Models for the Agents can be loaded from PKL files.
  - Gymnasium, RLlib, PettingZoo APIs are available for agent-environment interaction.

Output data

- Environment
  - Environments can be saved as PKL files.
  - State, observations and rewards are provided for each time step.
  - Trajectories can be saved as PKL and TSV files.
  - Renderings (videos / images) of simulation runs including observations and other information can be generated and saved in standard media file formats.
  - Events compatible with InteractiveAI framework can be exported or sent to live InteractiveAI instance.
- Agent
  - Actions can be exported as part of the trajectories.
  - Trained models can be saved as PKL files.
  - Training data is logged (e.g. rewards).



Visualization of the standard tree-observation



Real-world scenario demo (Olten, Switzerland): https://github.com/AI4REALNET/flatland-scenarios/raw/refs/heads/main/scenario_olten/img/olten_thumb.mp4

BlueSky

Developer: Delft University of Technology

New features

- Added support for ATM-UC 1 in a sectorization interface
- Added new simplified environments to BlueSky-Gym, representative of ATM-UC (use case) 2
  - StaticObstacleCR
  - StaticObstacleSector
  - StaticObstacleSectorCR
  - CentralisedStaticObstacle
  - CentralisedStaticObstacleCR
  - CentralisedStaticObstacleSectorCR
- Added BlueSky plugin to deploy the trained models on realistic, full-scale scenarios, enabling the testing of KPIs
- Added support for KPI testing platform FAB

ATM UC 1: sectorization

BlueSky-Gym

Context: ATM use cases on sectorization

Dynamic Airspace Sectorization (DAS): "a continuous restructuring of airspace sectors that ensures efficient allocation of scarce resources (e.g., Air Traffic Controllers) considering operational, economic and ecological constraints in both nominal and variable air traffic conditions." (Gerdes *et al.*, 2018)

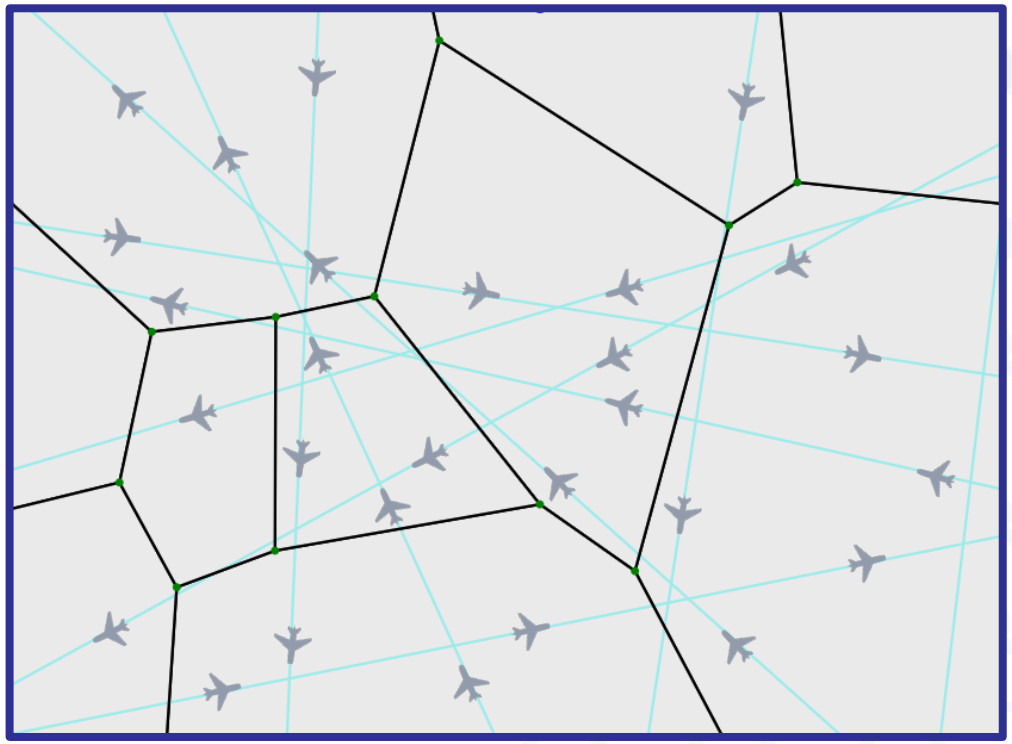
Human Role: flow and capacity manager

Objectives:

- Keep predicted controller workload within acceptable bounds (min/max)
- Balance predicted workload across sectors (low standard deviation)
- Minimize unnecessary additional flown track miles (even in the presence of disturbances, such as weather cells, contrail-sensitive areas)

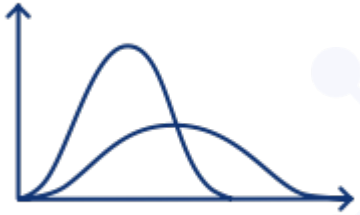
Operational constraints:

- Number of airspace sectors equal the number of available controllers
- Ensure convex airspace sectors
- Sectors should not be too small
- Avoid route crossings too close to sector boundaries
- Ensure sufficient route lengths within sectors
- Avoid route segments to coincide with sector vertices
- Avoid shallow crossing angles between routes and sector boundaries



Modeling methods – WP2

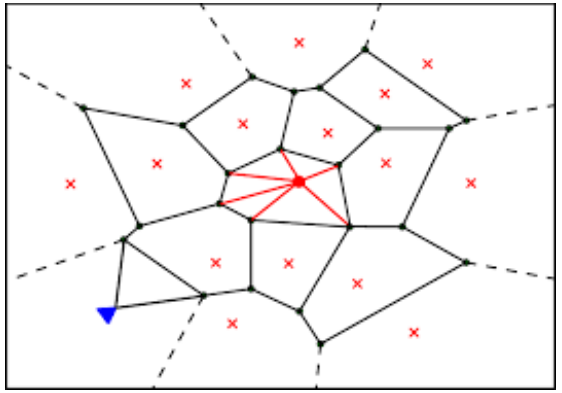
- **Trajectory generation:**
  - Interpolation of flight plan data (geometric calculations)
  - Historical flight data (distributions on FIR airspace entry/exit times)
  - Incorporate trajectory uncertainty (uncertainty distributions on position, arrival times and flight speed)



- **Workload prediction**: (weighted) sum of complexity (CX) factors
  - CX_1: Number of route crossings
  - CX_2: Number of coordination points (= route-airspace crossings)
  - CX_3: Number of predicted flights per sector, per time slice
  - CX_4: Number of predicted conflicts per sector, per time slice
  - …

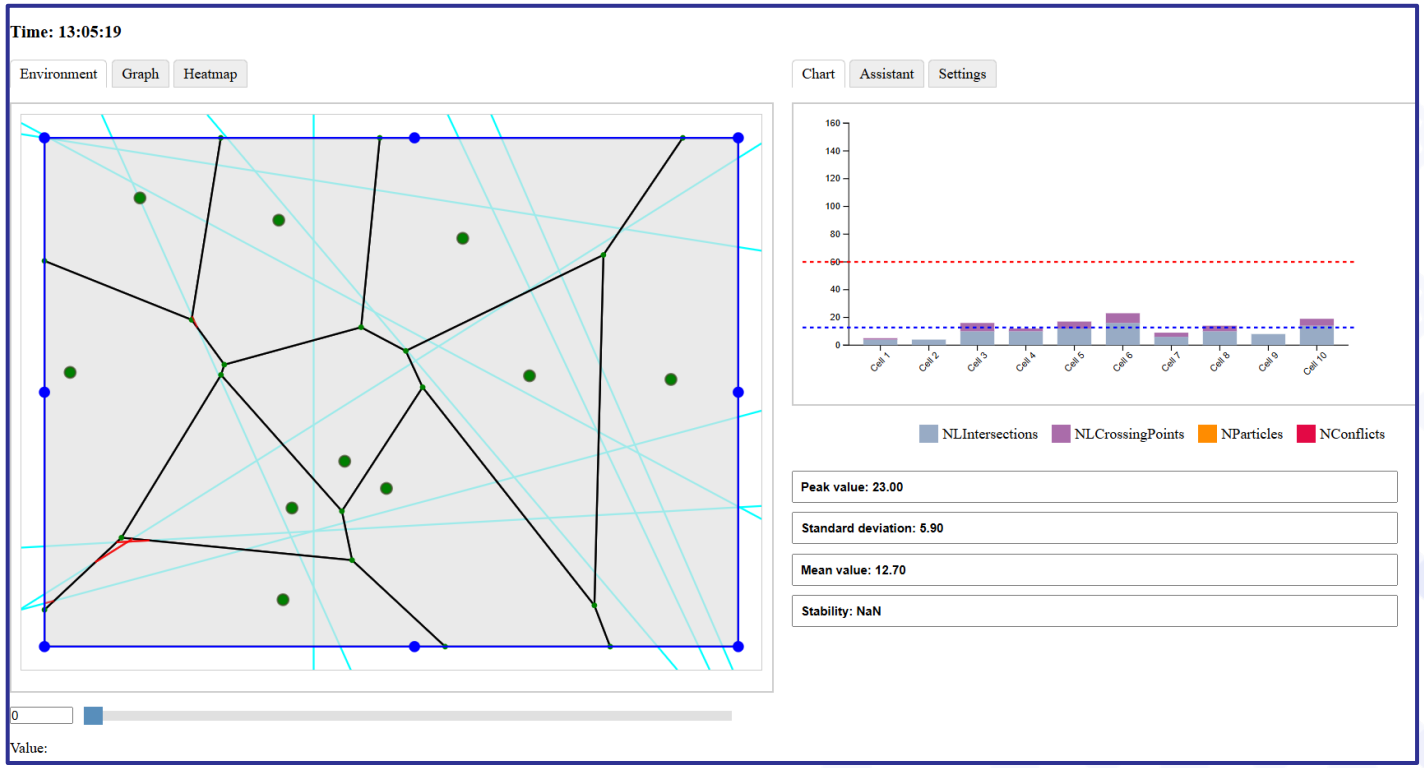
$$WL = \sum w_i \cdot CX_i \qquad (w_i \text{ from literature})$$

- **Convex sector geometry**:
  - Voronoi partitioning
  - Fortune's algorithm (fast!)
  - Number of centers = number of available controllers



Full human control (assisted by algorithms) - WP2

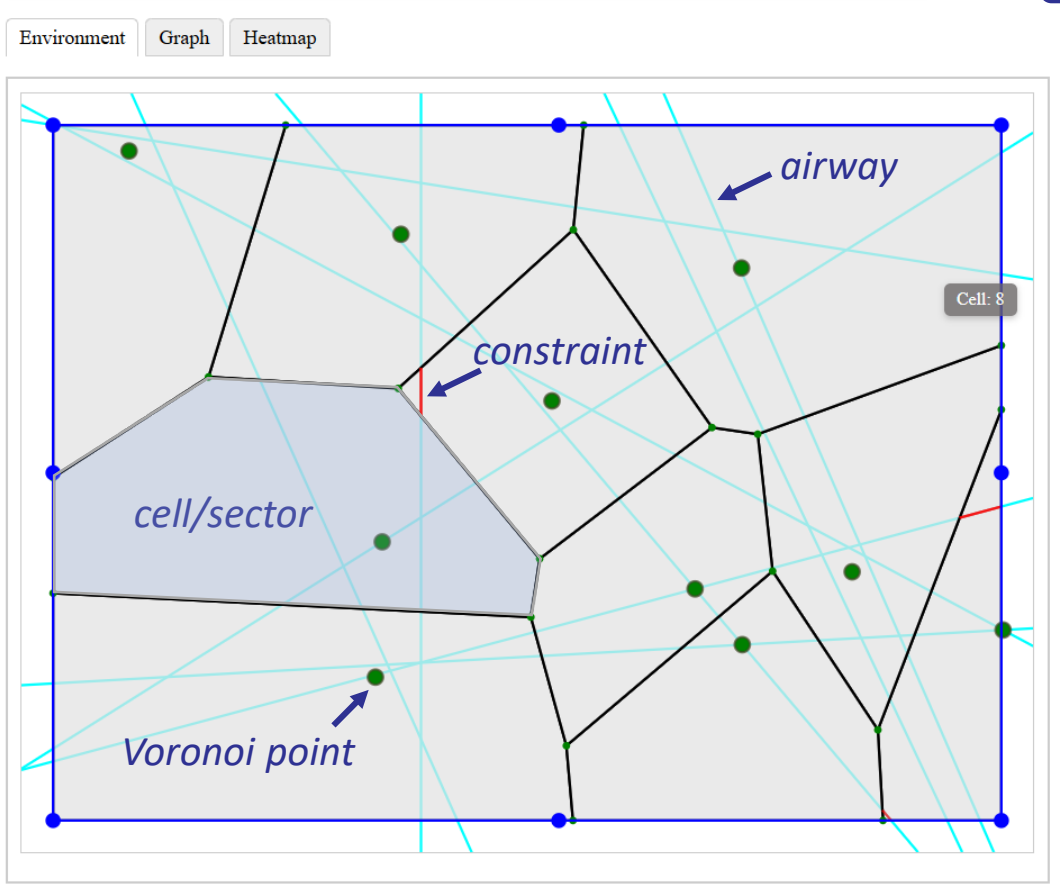
- Baseline HMI for sectorization
- Visual (algorithmic) transparency:
  - Portray sectorization goals
  - Portray sectorization constraints
- Support continuous re-sectorization of airspace by human interaction (i.e., add, remove and move Voronoi points)
- Sector complexity monitor
- Adjustable constraint parameters
- Calculates complexity per sector, per time slice in real time while interacting with Voronoi points
- K-means clustering to ‘kick start’ sectorization and subsequent optimization



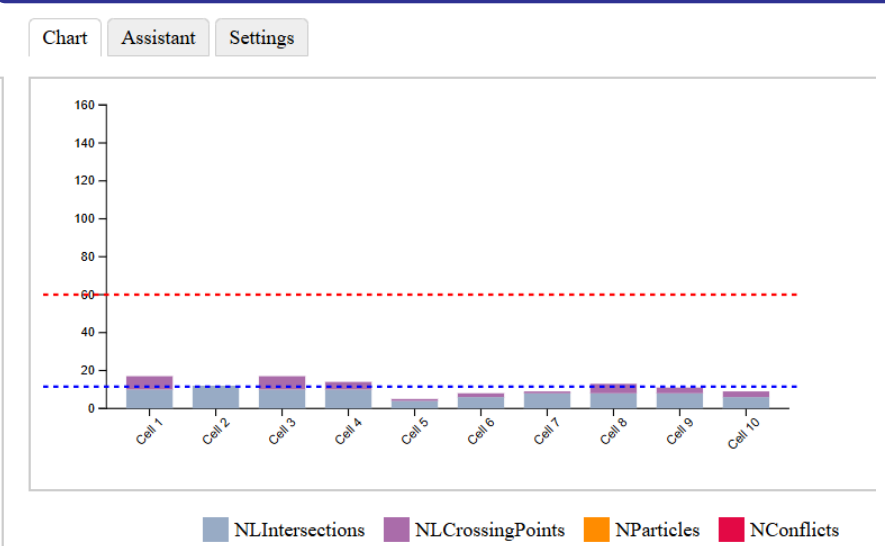
JavaScript application (runs in web browser)
https://github.com/clarkborst/ATMSectorization

Environment & Interface - WP2

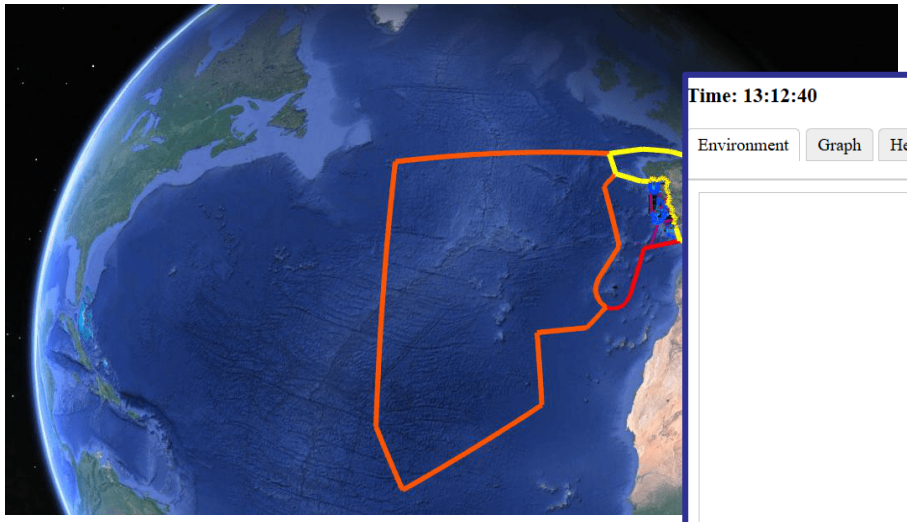
Environment monitor

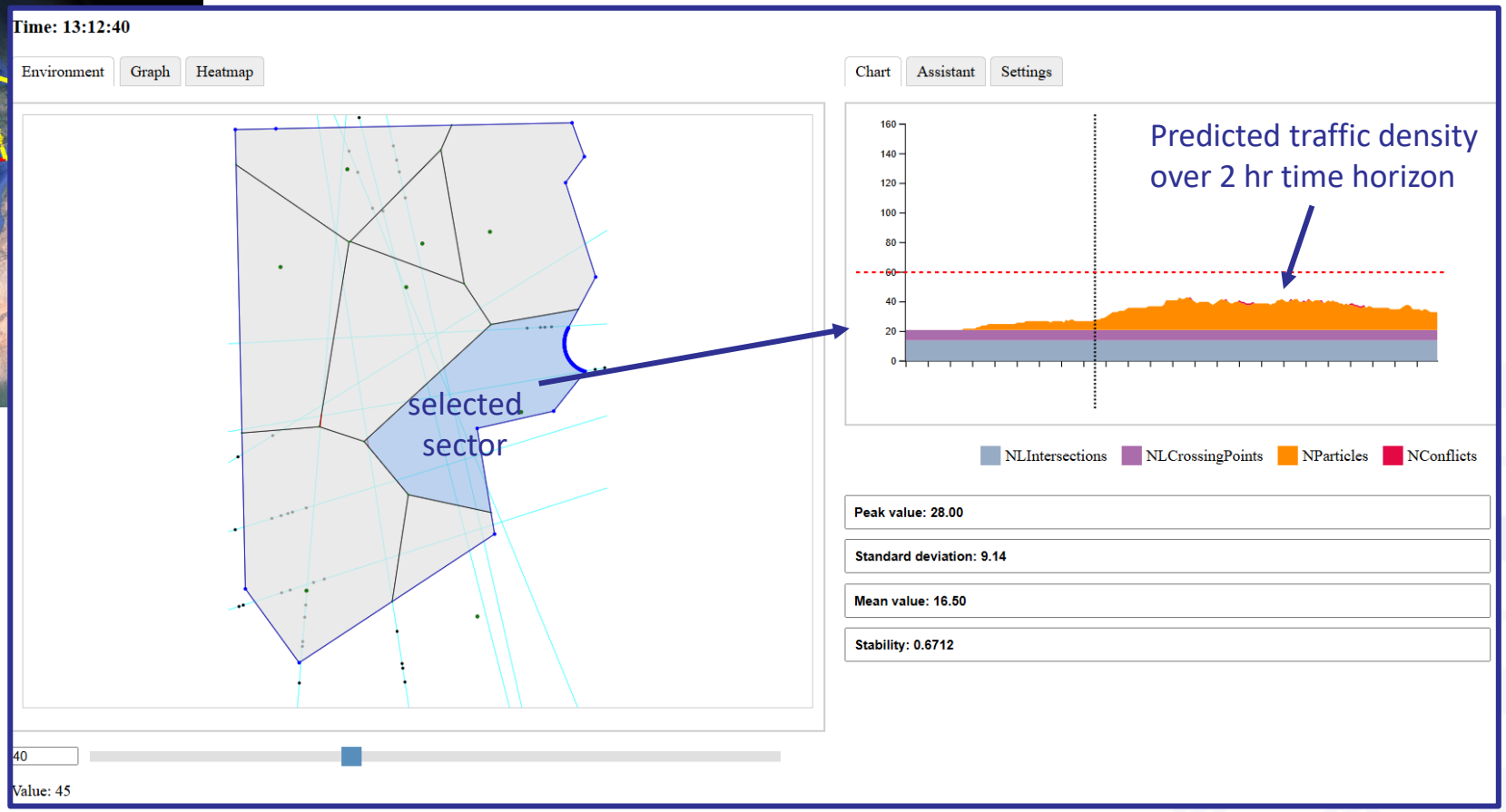


Sector complexity monitor

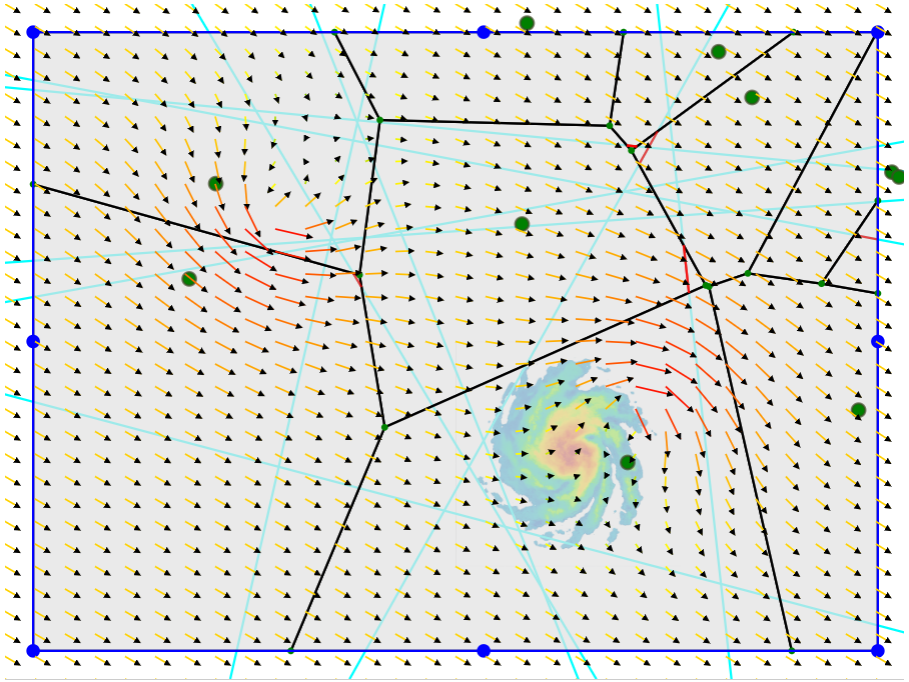


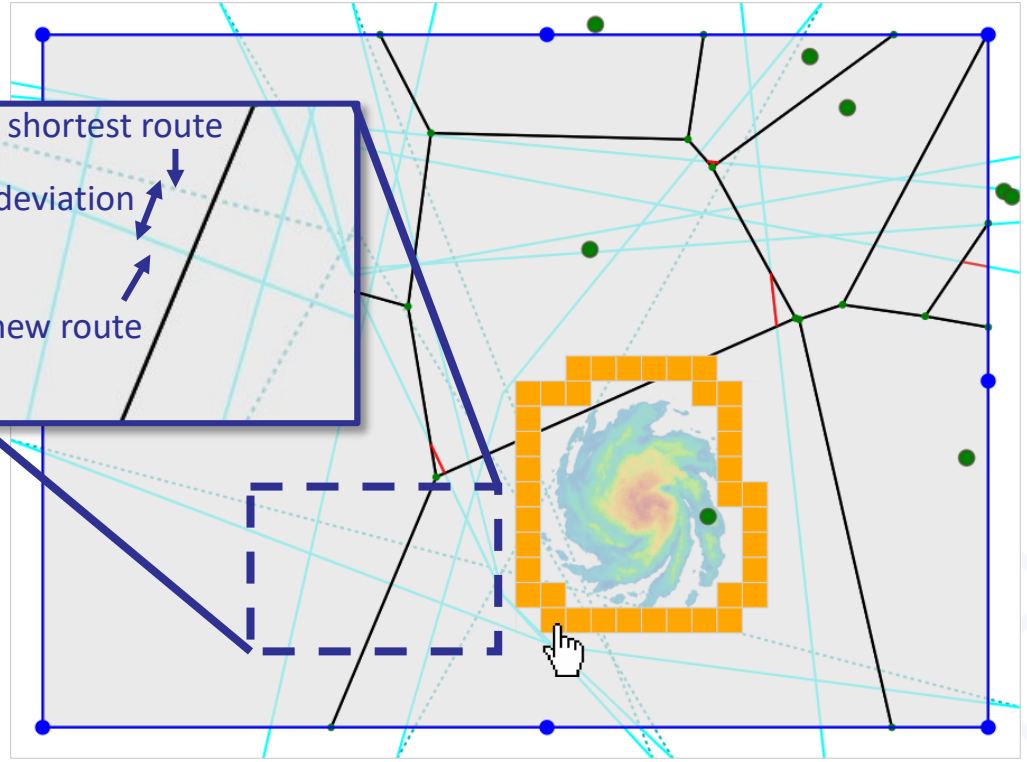
Santa Maria FIR





Handling environmental disturbances

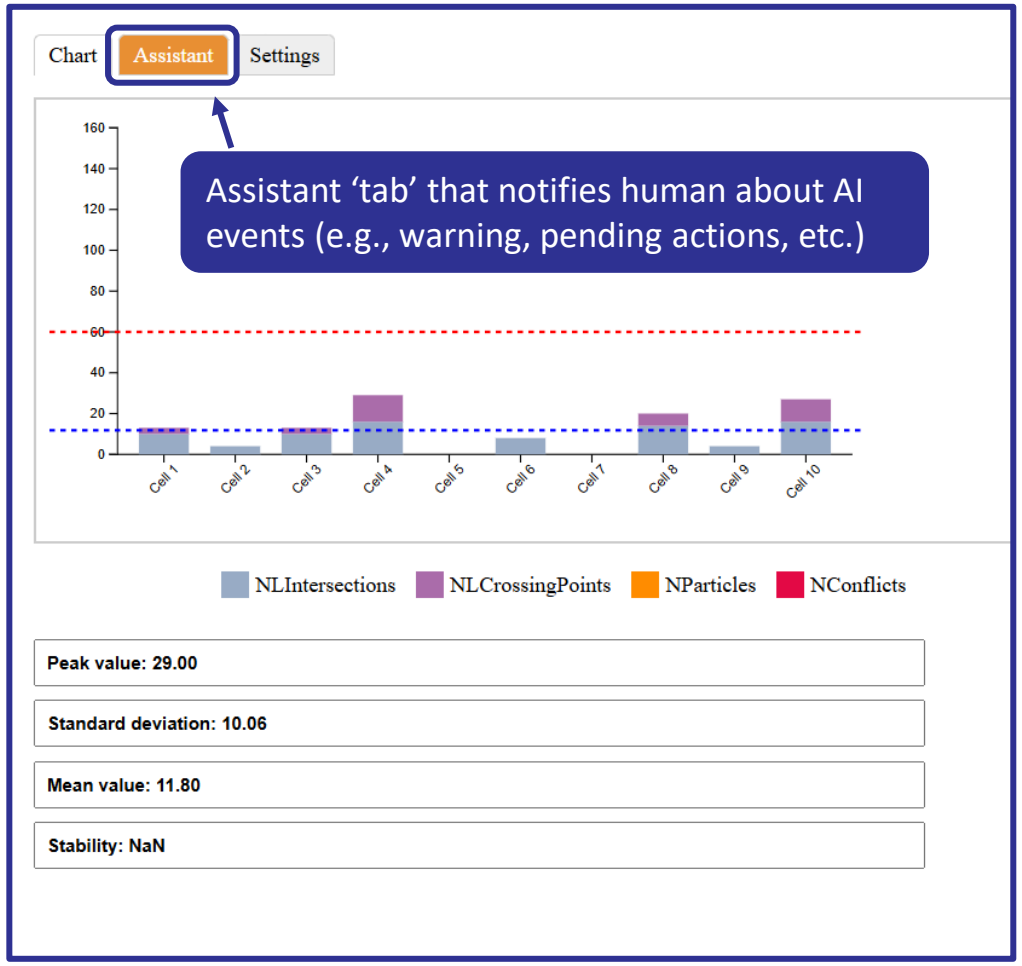


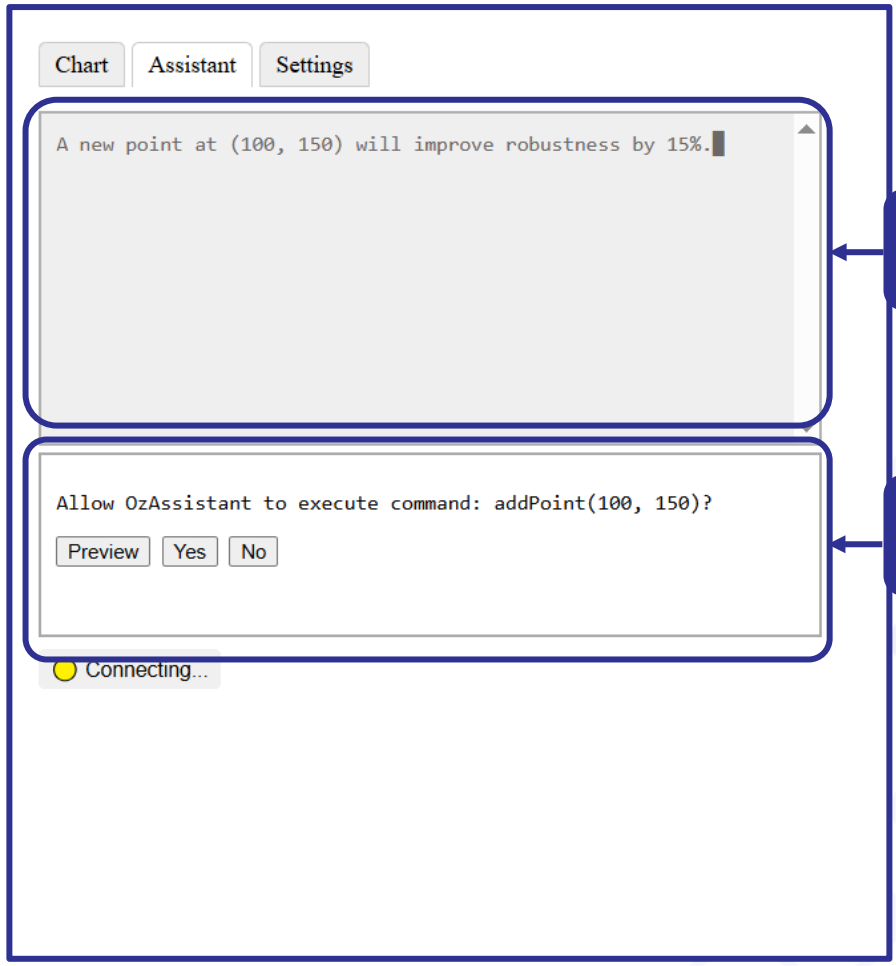


- **Wind field visualization** (direction & magnitude)
- **Disruptive weather cells** (location & size)

1. **Human:** activates ‘no-go’ areas (grid cells)
2. **AI:** replan aircraft routes (avoid ‘no-go’ area, using Theta* algorithm)
3. **Human/AI**: re-partition airspace based on new routing structure

AI assistant interface (work in progress)





time slice

time slider

Oz Assistant (work in progress)

- Scripted ‘Wizard-of-Oz’ assistant to test-drive Human/AI interaction
- Replays action items (i.e., display text messages, execute commands) defined in JSON text file
- Flexible JSON structure for defining how/when action items should be displayed and their options (e.g., ability to preview command, accept/reject command, etc.)

```
{
  "options": {
    "sequential": true,
    "stepDelay": 1500
  },
  "items": [
    {
      "time": 3000,
      "message": "Hello! I’m Oz, your scripted assistant here to support your sectorization efforts.\n\
      "repeat": 0,
      "interval": 7000,
      "maxRuntime": 300000
    },
    {
      "time": 13000,
      "command": "addPoint",
      "args": [100, 150],
      "requiresConfirmation": true,
      "explanation": "A new point at (100, 150) will improve robustness by 15%.",
      "previewable": false
    },
    {
      "time": 25000,
      "message": "That's it! No more items are scheduled."
    }
  ]
}
```

Live Assistant (work in progress)

- Live Assistant enables a client (e.g., Python application) to connect to the sectorization application (server)
- Bi-directional, real-time communication via JSON strings over WebSocket
- Uses same flexible JSON structure as Oz Assistant for defining how/when action items should be displayed and their options (e.g., ability to preview command, accept/reject command, etc.)

Python

```python
import asyncio
import websockets
import json

async def talk_to_browser():
    uri = "ws://localhost:8080/ws"   # must match config.json in JS

    async with websockets.connect(uri) as websocket:
        print("[Python] Connected to browser assistant")

        # Example 1: send a text message
        await websocket.send(json.dumps({
            "message": "Hello from Python client!"
        }))

        # Example 2: send a valid command
        await websocket.send(json.dumps({
            "command": "addPoint",
            "args": [200, 150],
            "requiresConfirmation": true,
            "explanation": "A new point at (200, 150)",
            "previewable": false
        }))

        # Listen for replies from the browser
        while True:
            try:
                response = await websocket.recv()
                data = json.loads(response)
                print("[Python] Received from browser:", json.dumps(data, indent=2))

                if data.get("type") == "error":
                    print(f"❌ Browser reported error: {data['reason']}")
                elif data.get("type") == "confirmation":
                    print(f"✅ Browser confirmed command {data['command']} with args {data['args']}")

            except websockets.exceptions.ConnectionClosed:
                print("[Python] Connection closed by browser")
                break

if __name__ == "__main__":
    asyncio.run(talk_to_browser())
```

Live Assistant: RL agent connection (work in progress)

Inputs (= state S_t)

- Planned (aircraft) trajectories
- Number of Voronoi centers
- …

(requires 'feature engineering' that describes the environment state in a concise way)

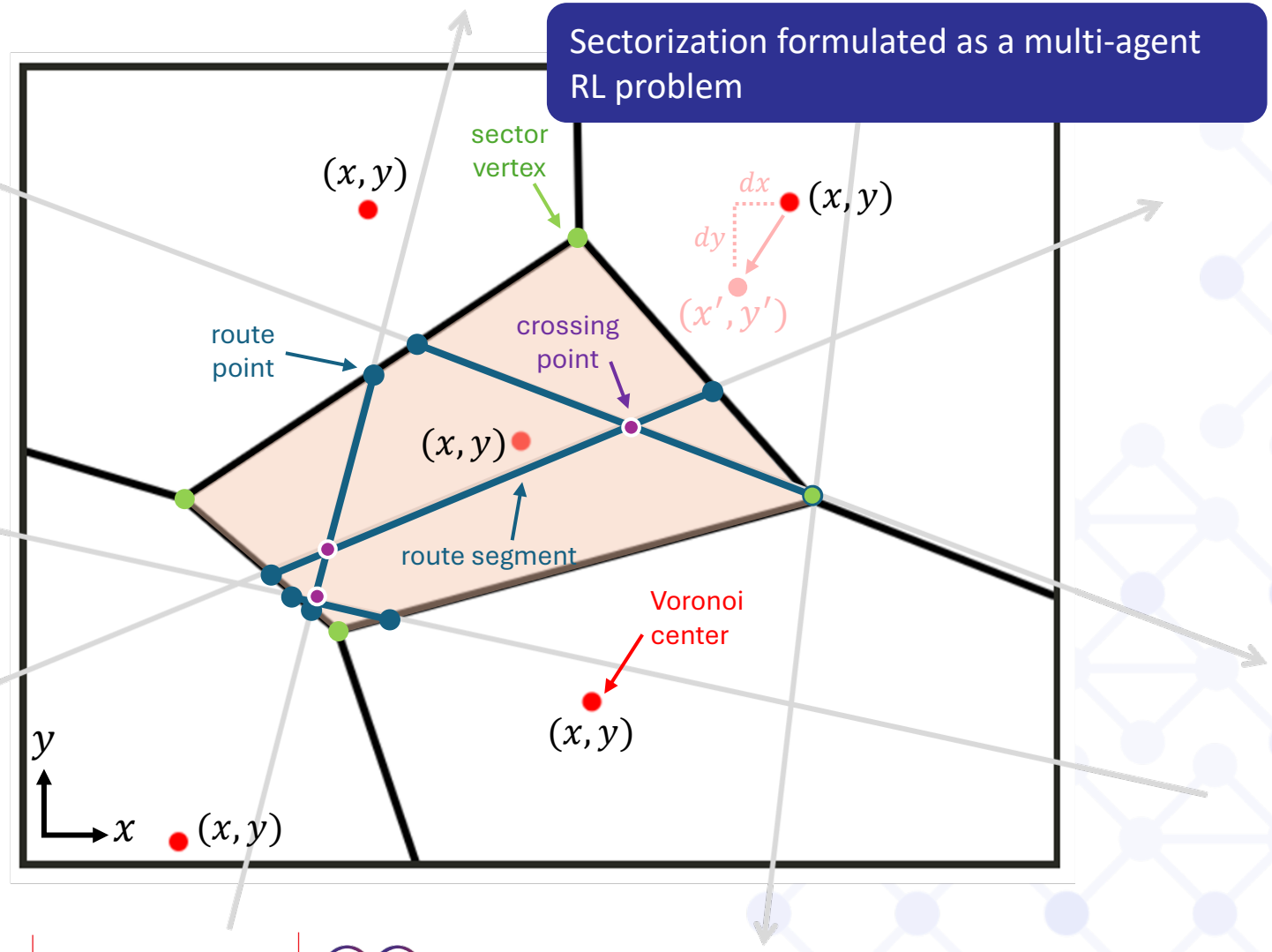
Actions A_t

- (dx, dy) displacement of Voronoi centers

Rewards R_t (and penalties)

- Distance crossing points with sector boundaries
- Route segment lengths in a sector
- Distance route points with sector vertices
- Sector area
- Taskload per sector should not exceed thresholds
- Taskload standard deviation across sectors should not exceed threshold

Sectorization formulated as a multi-agent RL problem



BlueSky-Gym

Deploy-RL plug-in

FAB-ATM test runner

ATM UC 2: path planning

BlueSky-Gym

BlueSky-Gym - General description

A Gymnasium style library for standardized Single-Agent Reinforcement Learning research in Air Traffic Management

- A platform to accelerate RL research in ATM, bridging gap between fundamental research and full-scale experiments
- Fully compatible with standard Gymnasium Reinforcement Learning API
- Compatible with RL libraries (StableBaselines-3, RLLib), allows for own algorithms
- Developed in Python

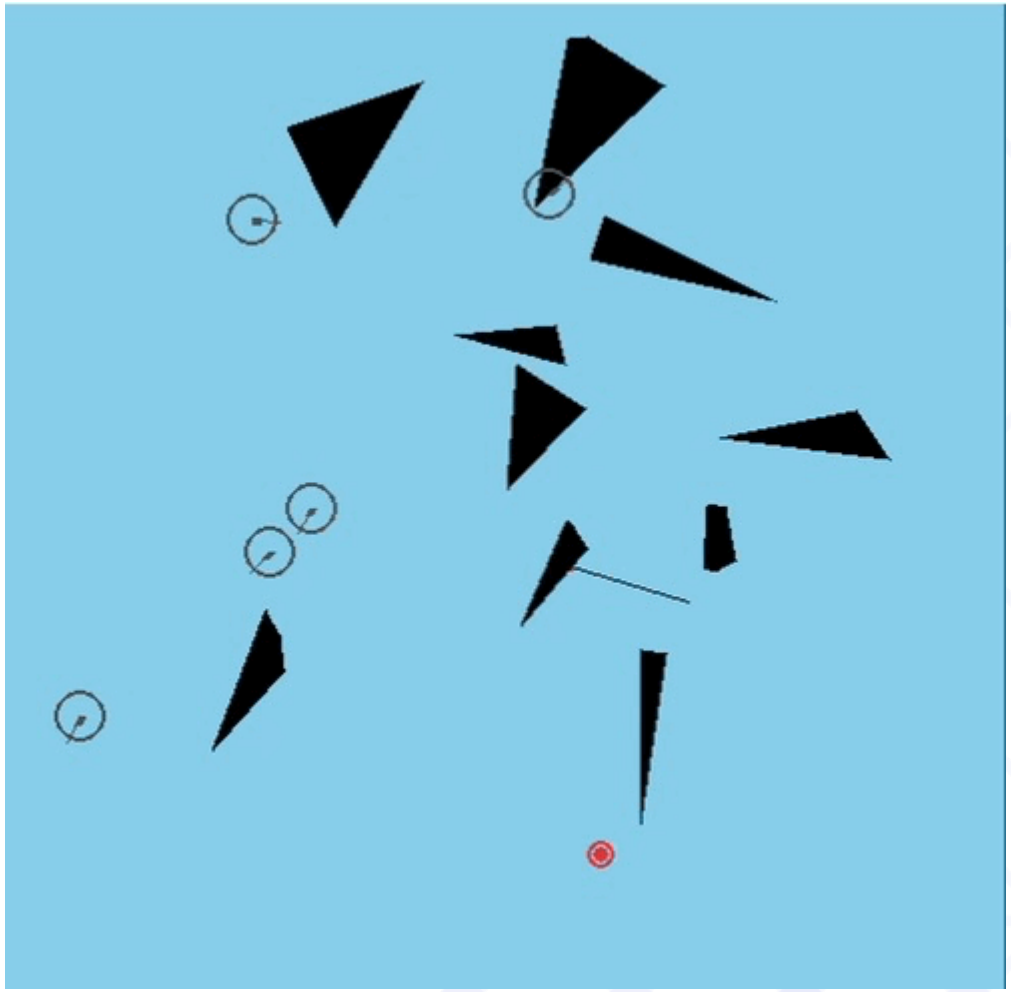
BlueSky-Gym - General description

Environments:

- Environments included in the release can be used as benchmark for comparing results
- Scenarios are simplified, but representative of full-scale problems
- Customizable complexity and traffic types thanks to a modular design, providing standard observation and action spaces, reward functions for:
  - Horizontal environments for lateral navigation (observations: drift and distance from destination. Distance, bearing and speed with respect to intruders; actions: heading changes; reward: intrusions, drift, destination reached)
  - Vertical environments for vertical navigation (observations: altitude, vertical speed, target distance, runway distance. Bearing, speed and altitude with respect to intruders; actions: intrusion and target altitude distance, crash penalty)

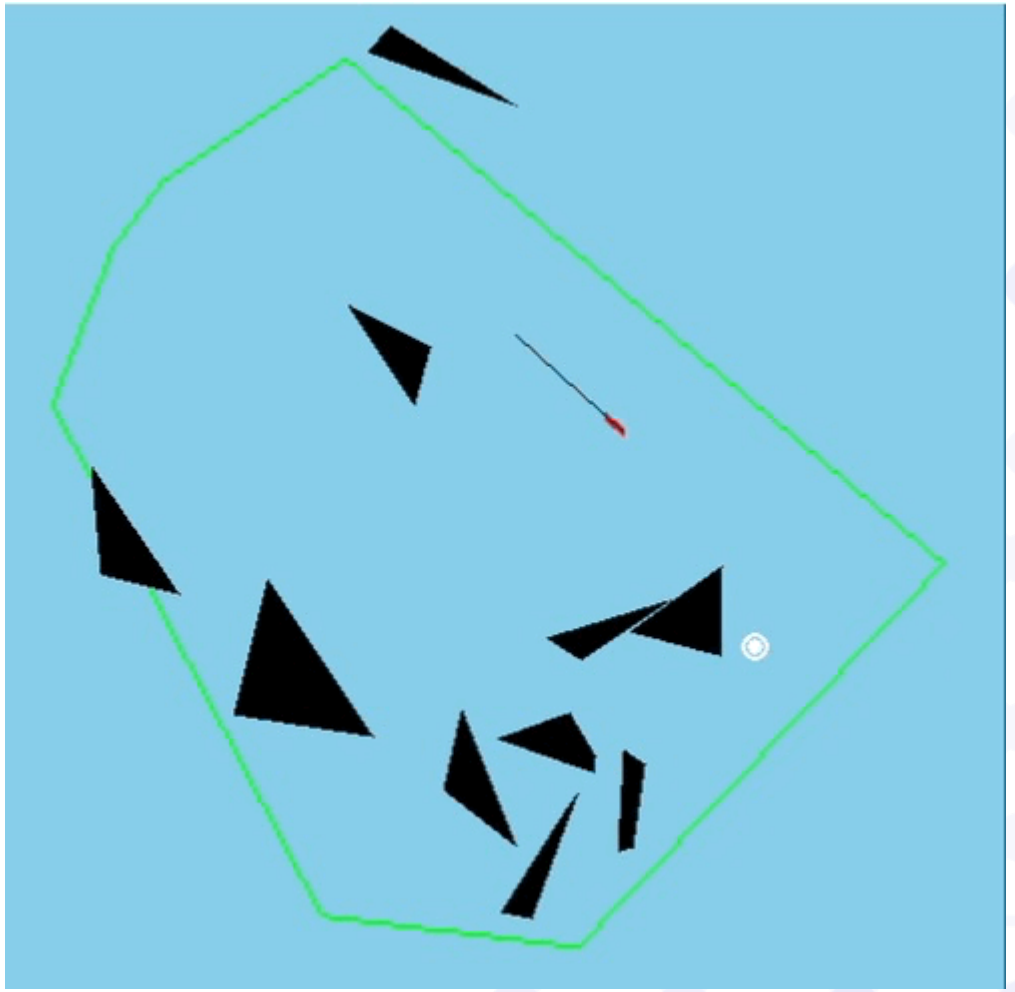
Static Obstacle CR Environment

- Single Agent, 1 aircraft controlled
- Elements:
  - Ownship aircraft + destination
  - Aircraft with pre-established route
  - Obstacles (restricted areas)
- Action: speed and heading
- Observation:
  - Drift and distance to destination waypoint
  - Distance, relative heading and relative speed to other aircraft
  - Distance and relative heading to obstacles
- Reward:
  - Reaching destination (+)
  - Drifting away from destination (-)
  - Intruding other aircraft protected zone (-)
  - Intruding restricted areas (-)



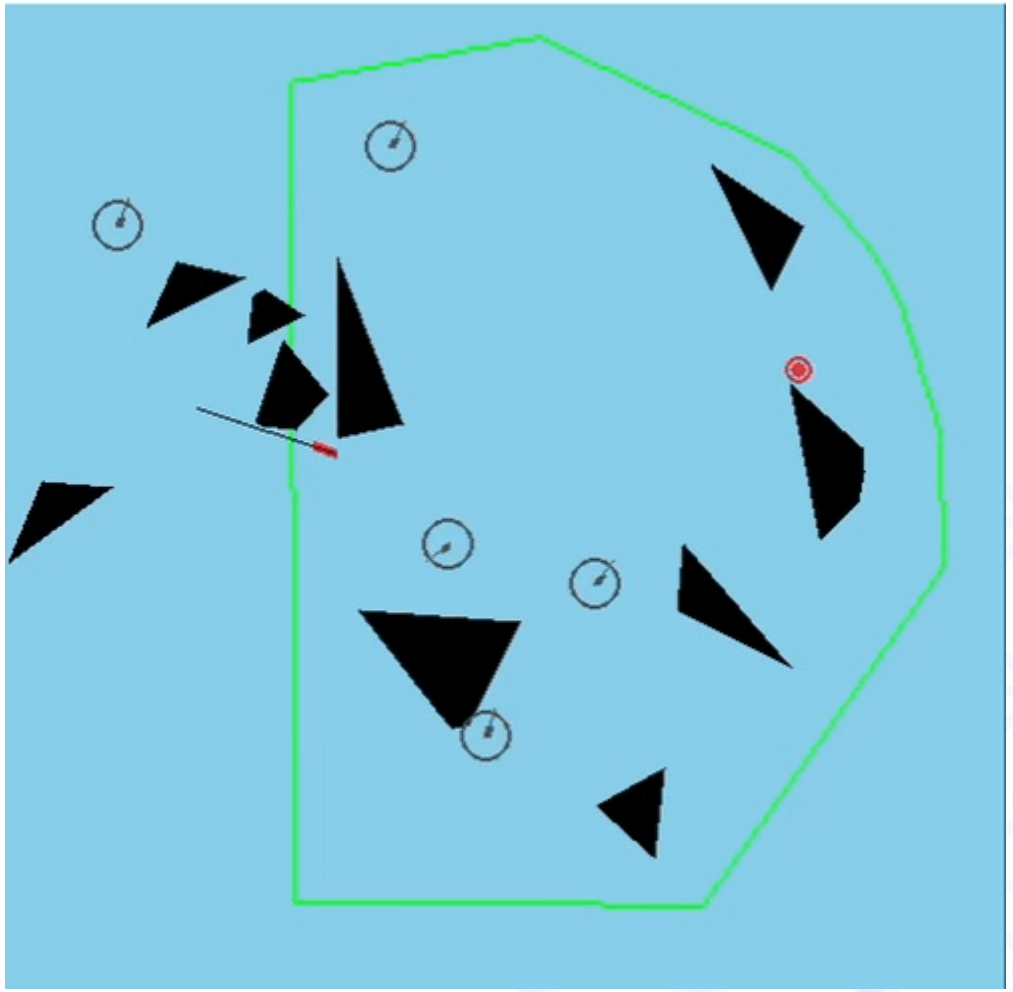
Static Obstacle Sector Environment

- Single Agent, 1 aircraft controlled
- Elements:
  - Ownship aircraft + destination
  - Obstacles (restricted areas)
  - Sector
- Action: speed and heading
- Observation:
  - Drift and distance to destination waypoint
  - Distance and relative heading to obstacles
  - Sector border points distance and relative heading
- Reward:
  - Reaching destination (+)
  - Drifting away from destination (-)
  - Intruding restricted areas (-)
  - Exiting the sector (-)



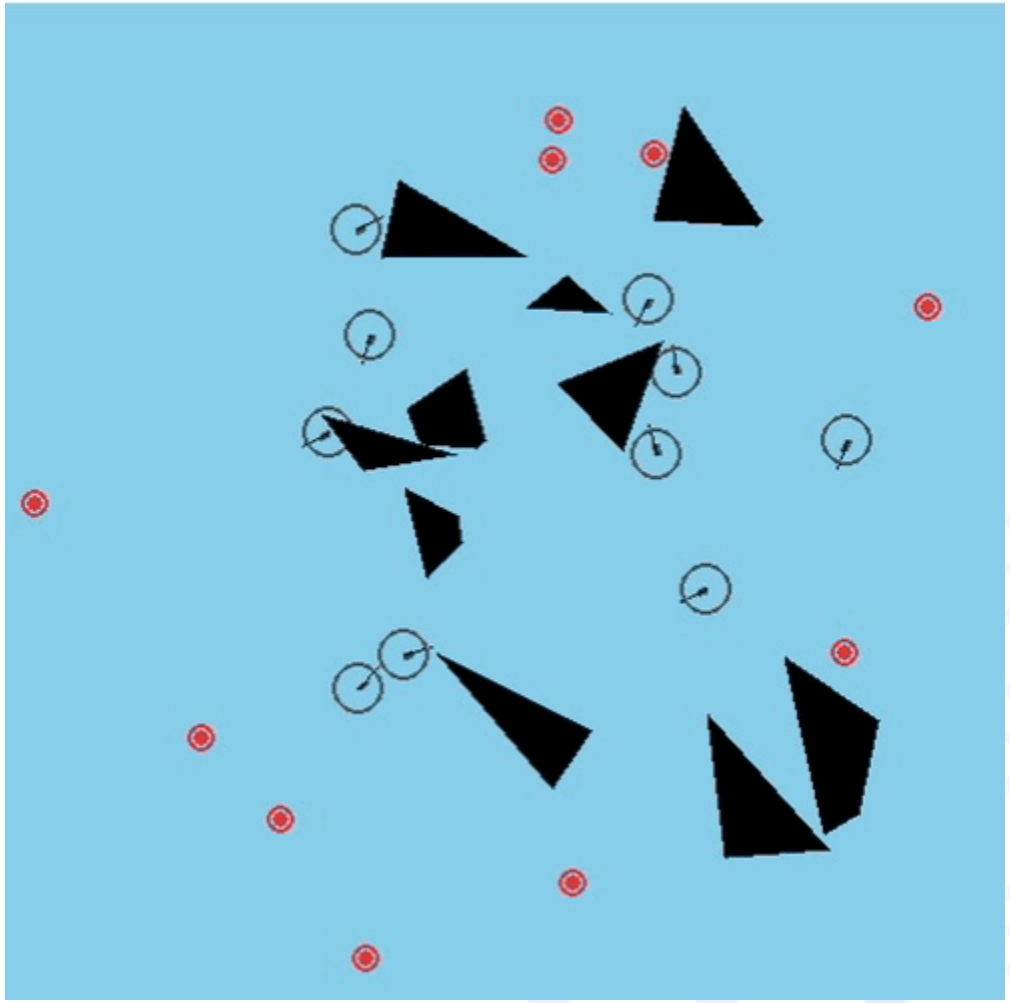
Static Obstacle Sector CR Environment

- Single Agent, 1 aircraft controlled
- Elements:
  - Ownship aircraft + destination
  - Aircraft with pre-established route
  - Obstacles (restricted areas)
  - Sector
- Action: speed and heading
- Observation:
  - Drift and distance to destination waypoint
  - Distance, relative heading and relative speed to other aircraft
  - Distance and relative heading to obstacles
  - Sector border points distance and relative heading
- Reward:
  - Reaching destination (+)
  - Drifting away from destination (-)
  - Intruding other aircraft protected zone (-)
  - Intruding restricted areas (-)
  - Exiting the sector (-)



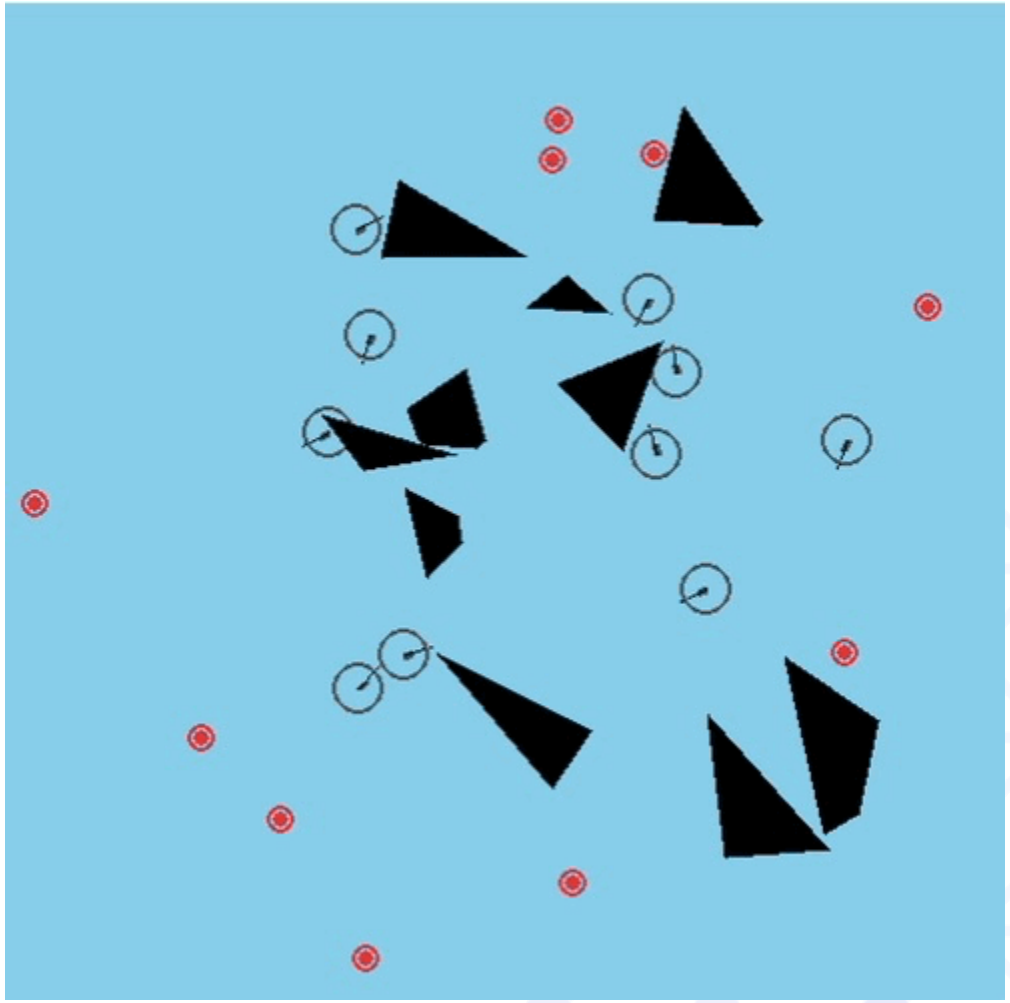
Centralised Static Obstacle Environment

- Single Agent, multiple aircraft controlled
- Elements:
  - Multiple ownship aircraft + destinations
  - Obstacles (restricted areas)
- Action: speed and heading
- Observation:
  - Drift and distance to destination waypoint
  - Distance and relative heading to obstacles
- Reward:
  - Reaching destination (+)
  - Drifting away from destination (-)
  - Intruding restricted areas (-)



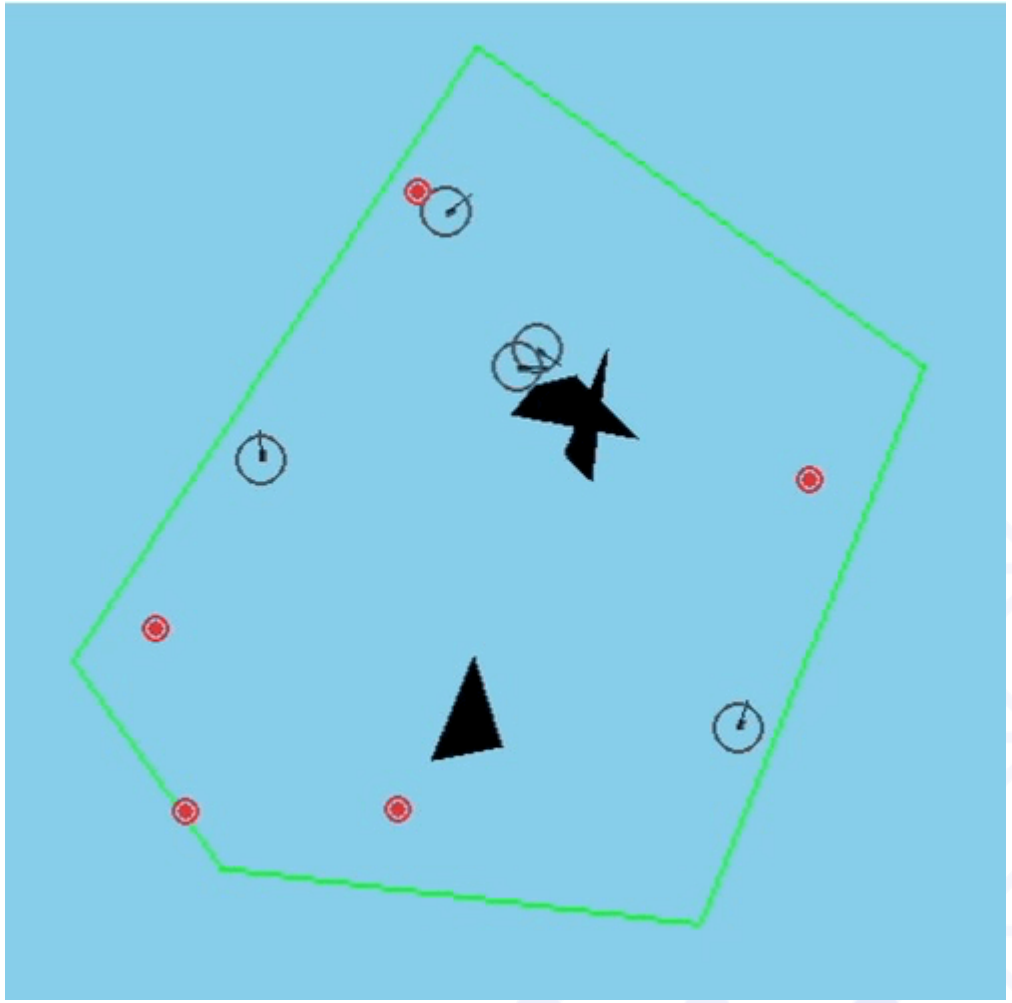
Centralised Static Obstacle CR Environment

- Single Agent, multiple aircraft controlled
- Elements:
  - Multiple ownship aircraft + destinations
  - Obstacles (restricted areas)
- Action: speed and heading
- Observation:
  - Drift and distance to destination waypoint
  - Distance, relative heading and relative speed to other aircraft
  - Distance and relative heading to obstacles
- Reward:
  - Reaching destination (+)
  - Drifting away from destination (-)
  - Intruding other aircraft protected zone (-)
  - Intruding restricted areas (-)



Centralised Static Obstacle Sector CR Environment

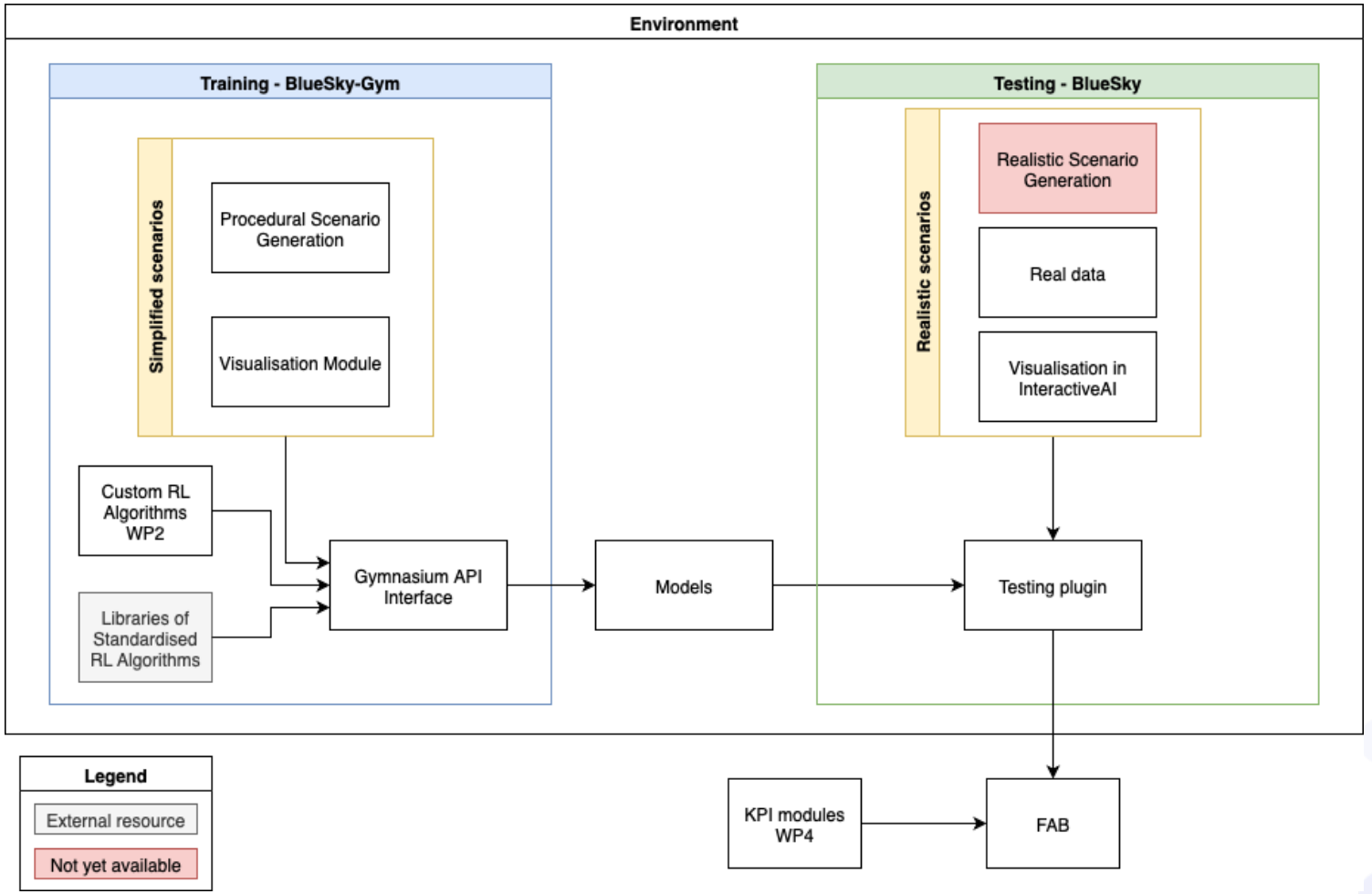
- Single Agent, multiple aircraft controlled
- Elements:
  - Multiple ownship aircraft + destinations
  - Obstacles (restricted areas)
  - Sector
- Action: speed and heading
- Observation:
  - Drift and distance to destination waypoint
  - Distance, relative heading and relative speed to other aircraft
  - Distance and relative heading to obstacles
  - Sector border points distance and relative heading
- Reward:
  - Reaching destination (+)
  - Drifting away from destination (-)
  - Intruding other aircraft protected zone (-)
  - Intruding restricted areas (-)
  - Exiting the sector (-)



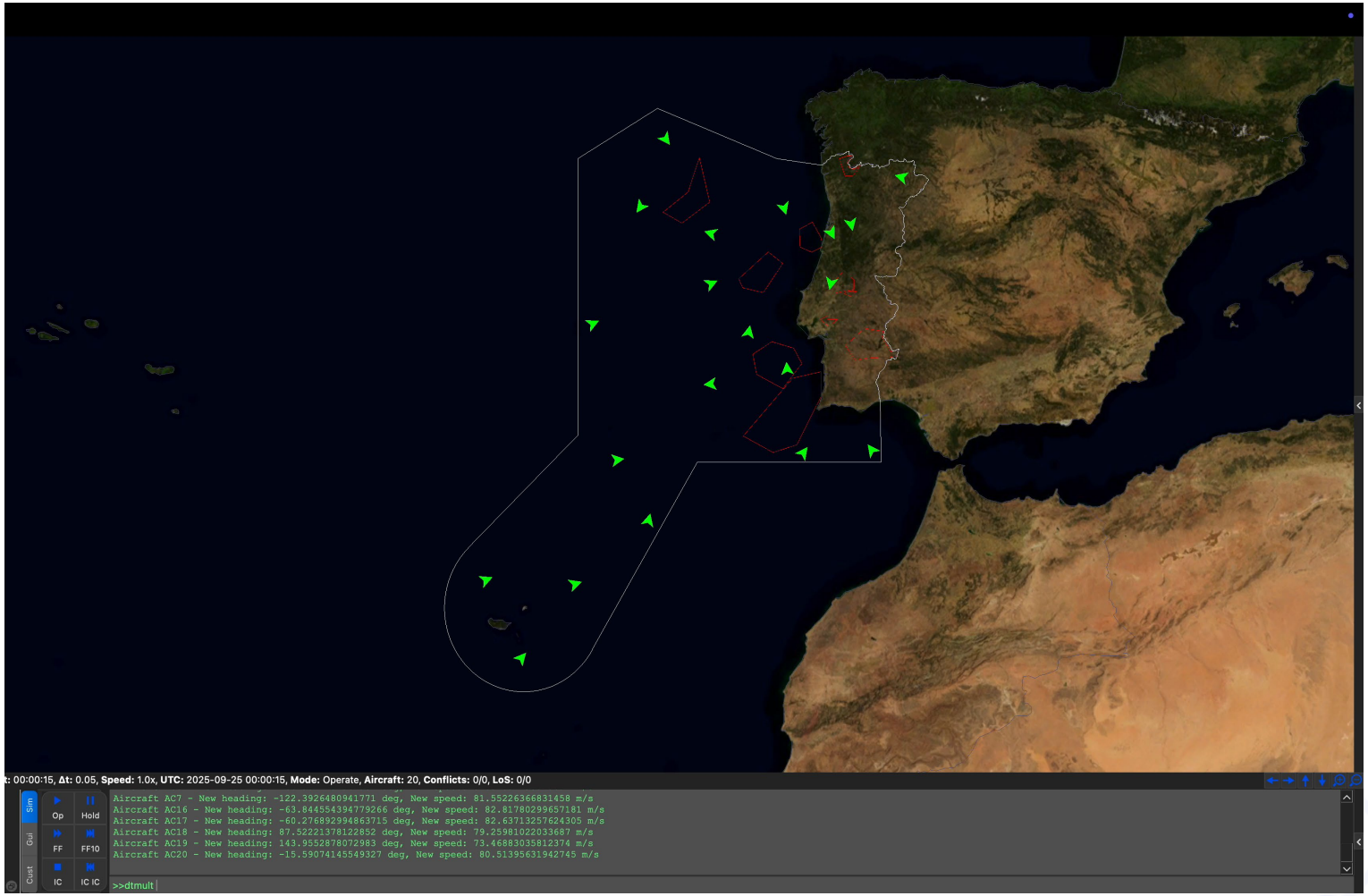
Plugin for deploying RL

- Environment specific
- Input:
  - Realistic sector configurations,
  - Restricted areas,
  - Flight entry points and destination waypoints
- Output:
  - Action, as inferred by the trained models
- Current limitations:
  - Flattened altitudes
  - Flattened entry times of aircraft in the sectors (fixed size of the action space)

Testing & KPI measuring architecture



Plugin visualisation



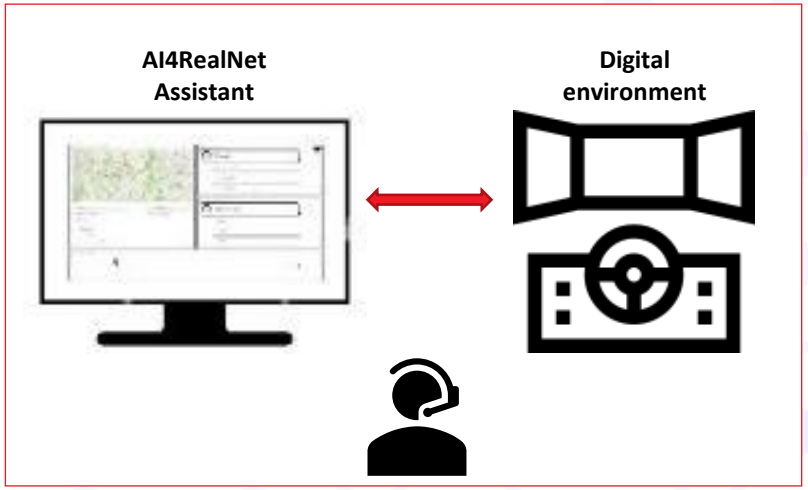
https://youtu.be/n7gW4iz6Dv0

InteractiveAI

Developer: IRT SystemX

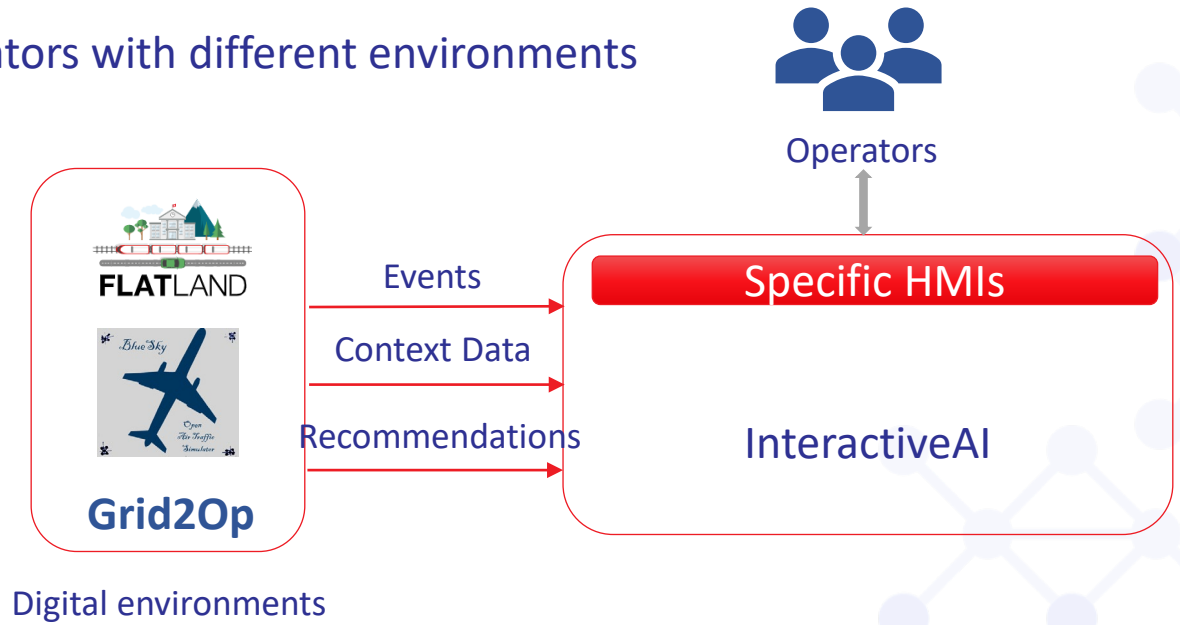
InteractiveAI Platform - Objectives

- The **InteractiveAI** platform is a **generic** technical base used to build interactive assistants for different industrial sectors
- The assistant objectives:
  - Display a user friendly context view about the digital environment : real time state
  - Notify the operator in case of incidents
  - Allow the operator to interact with his environment
  - Ask for help
  - Display recommendations proposed by AI agents



InteractiveAI Platform - Description

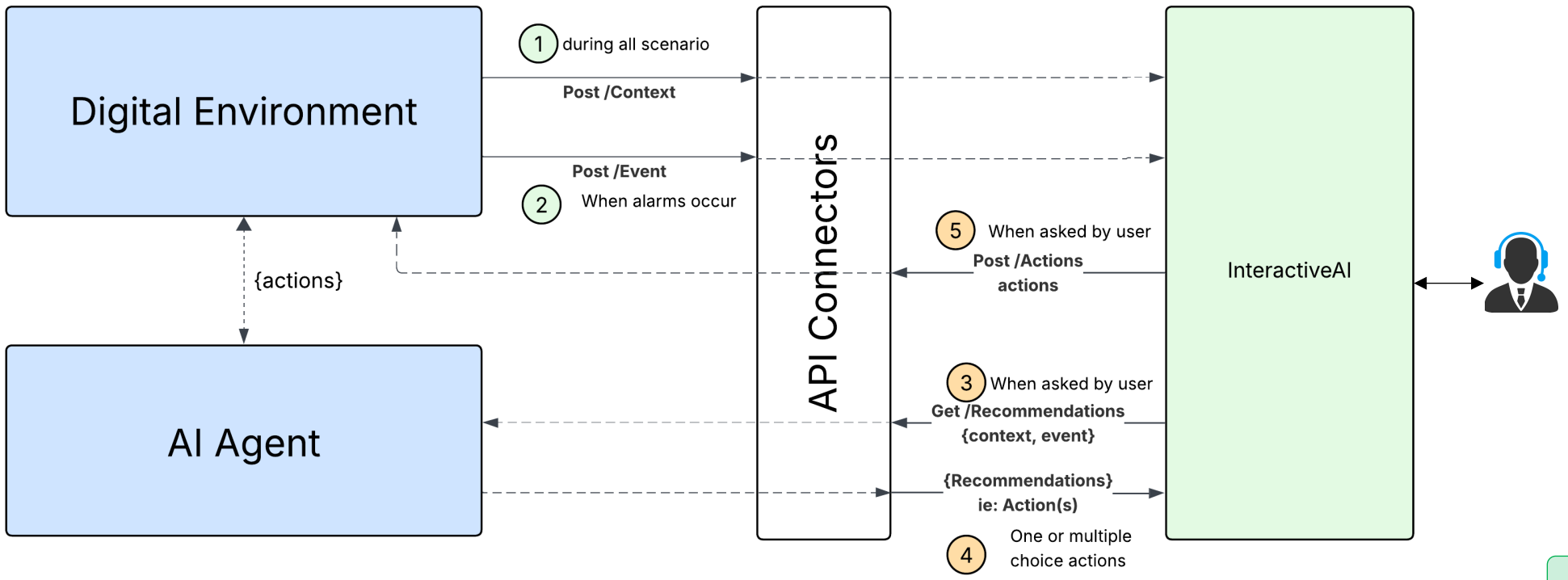
- **An interactive AI Assistant Platform for Real Time operations**
  - A generic technical base for all the industrial sector
  - Specific HMIs for each application domain
  - Communication with the digital environments
  - Facilitate the interaction of the operators with different environments



Inputs

- Generic InteractiveAI Platform : Open source
- Different use case scenarios
- Specific needs for HMIs
- Generic API connectors specifications

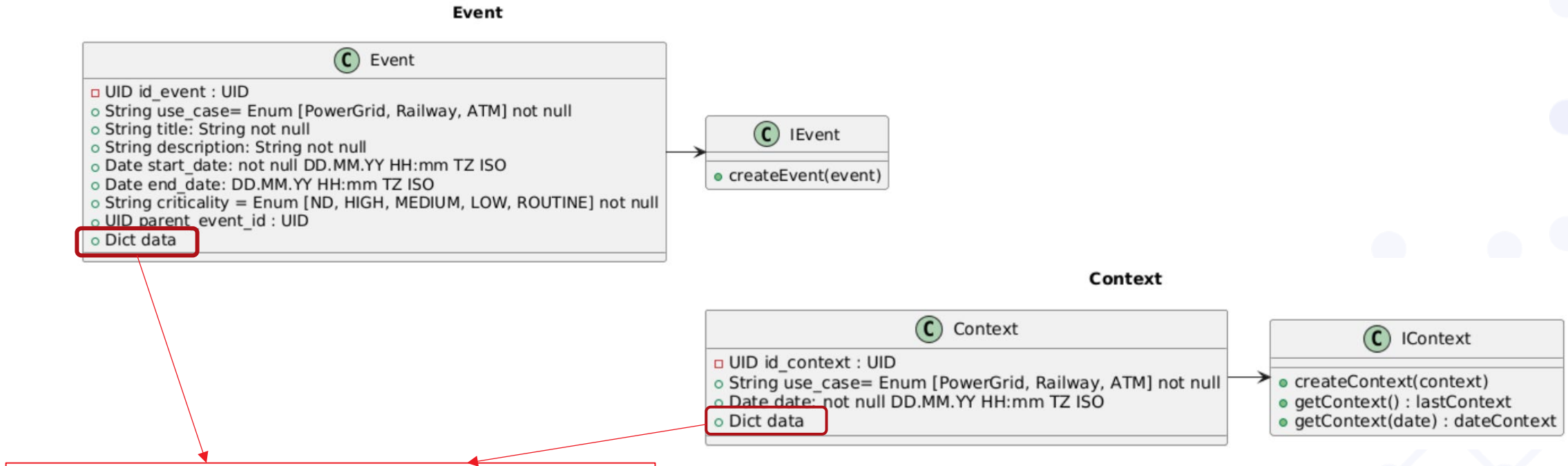
Block diagram



Release#1

Release#2

InteractiveAI generic entities – Context and Event

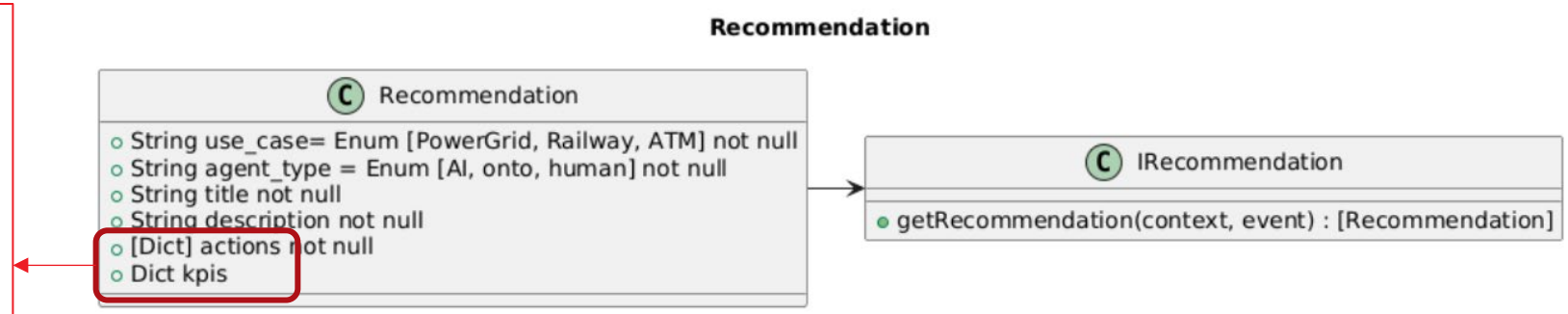


- Specific data from different digital environment
- This data will serve as input for AI agents

Interoperabilty using InteractiveAI

Recommendation: Generic entity used to display a solution on the assitant dashboard

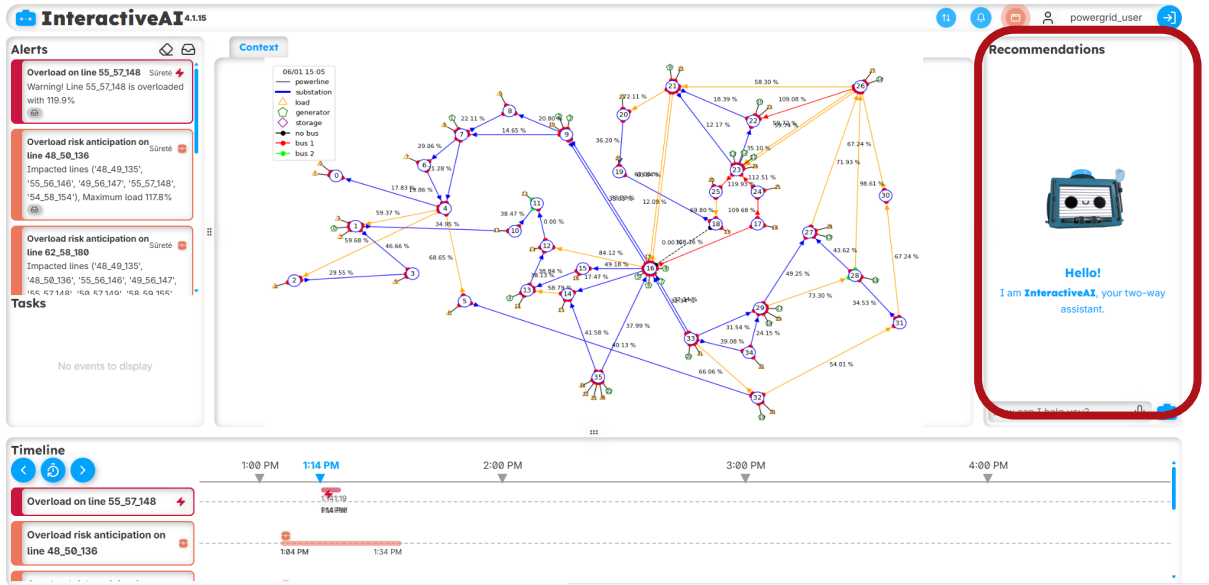
- **Action:** Specific entity used to display apply a solution on the digital environment
- Specific actions related to each digital environment
- Specific technical KPIs for each recommendations

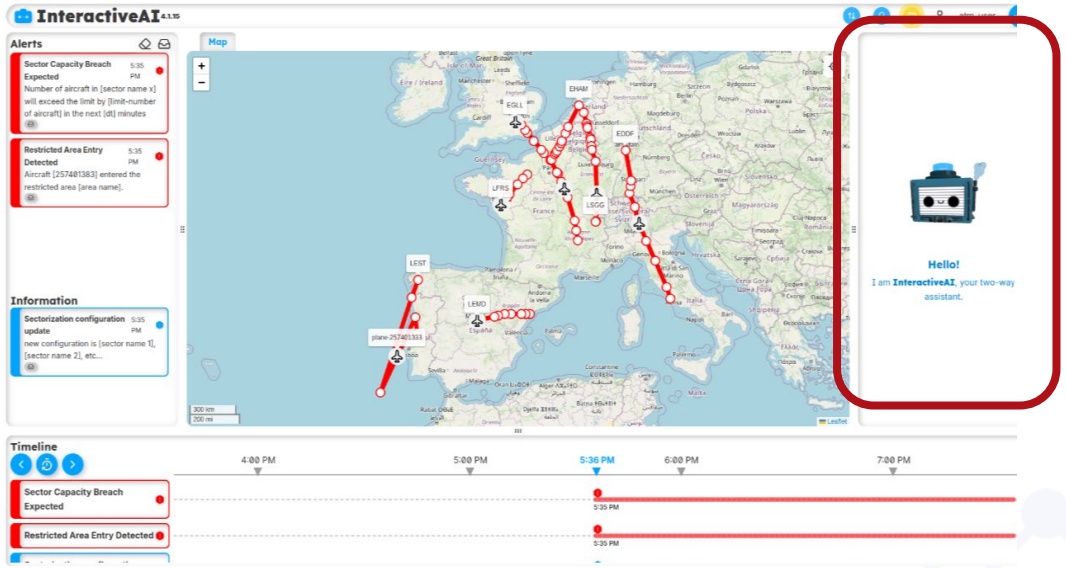


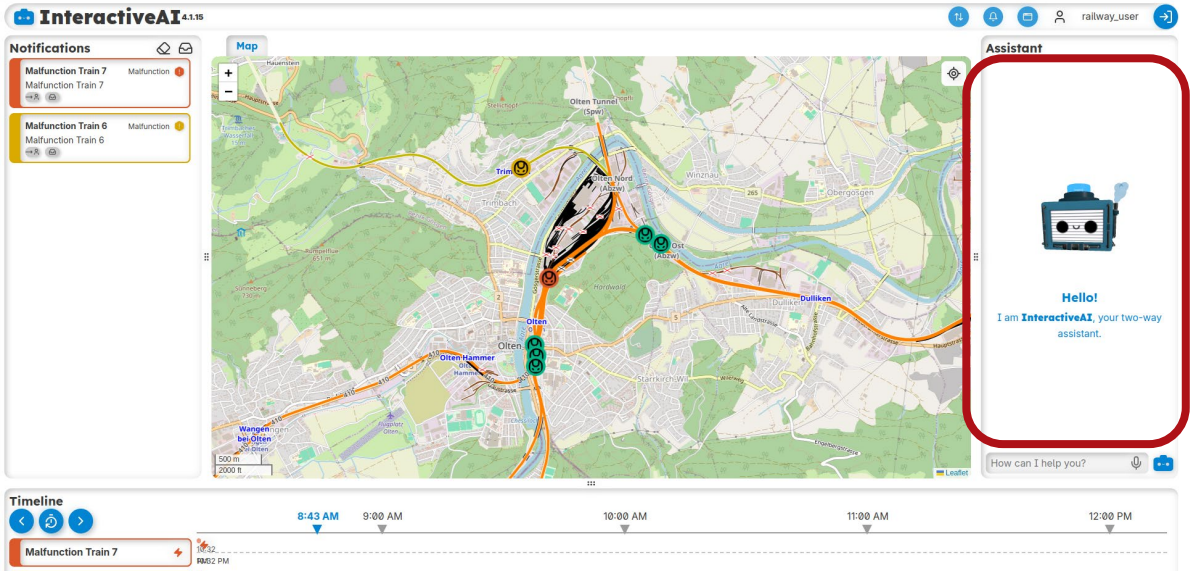
This is an exemple of a recommendation

{ "title": "Title displayed on each reco",
"description": "Detailed description to help operator understand the proposed reco",
"use_case": "Railway",
"agent_type": "AI",
"actions": What is needed by the DE to apply the reco, NOT to be displayed
"kpis": {
"kpi1_name": "kpi1_value",
"kpi2_name": "kpi2_value"}}

InteractiveAI – Release #1

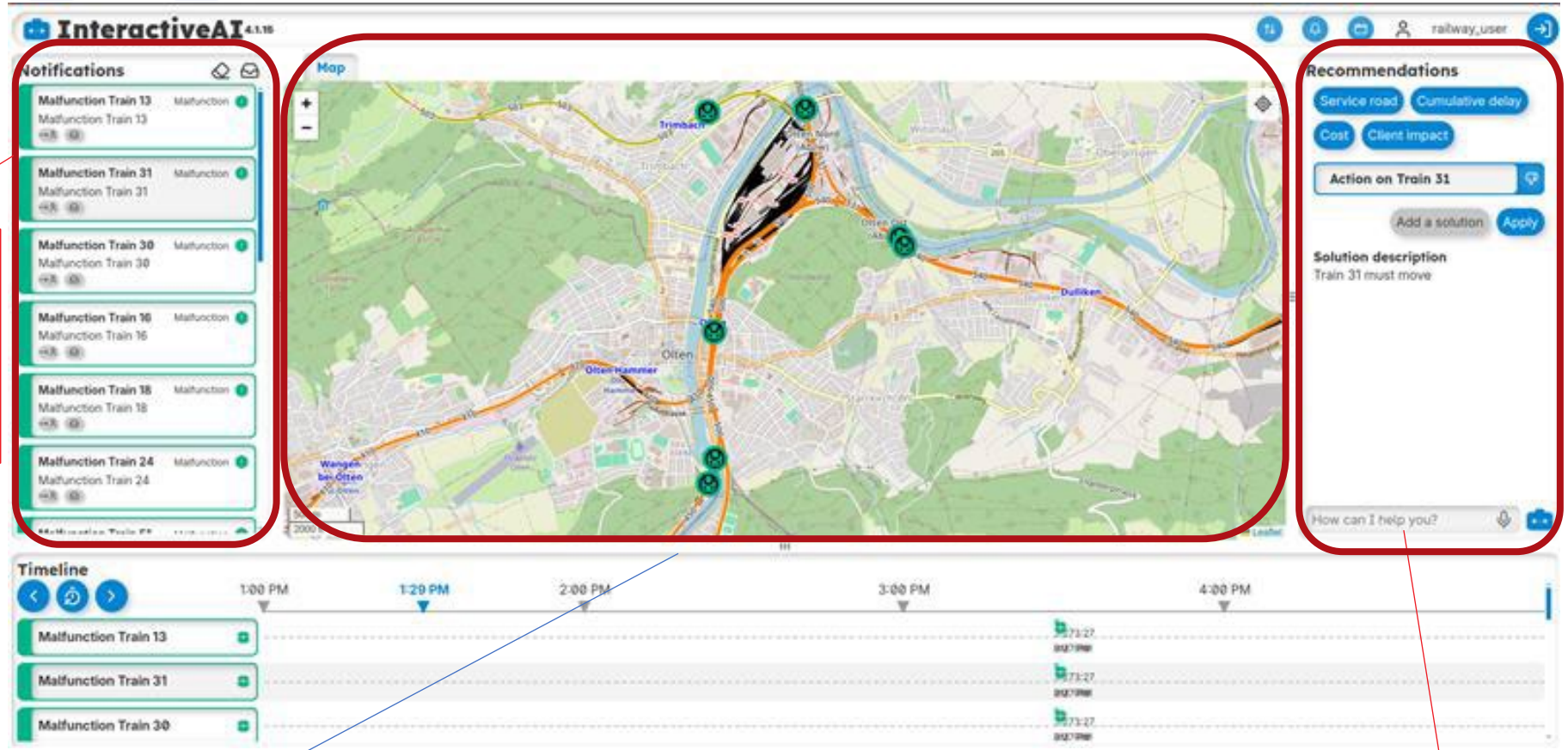






Release#2:
AI agents outputs

Release #2 Output - GUI - Railway



Notifications:

- Malfunctions on trains

- Communication with Flatland

Railway context:

-Trains' GPS positions

-Colored trains with issues

-Communication with Flatland

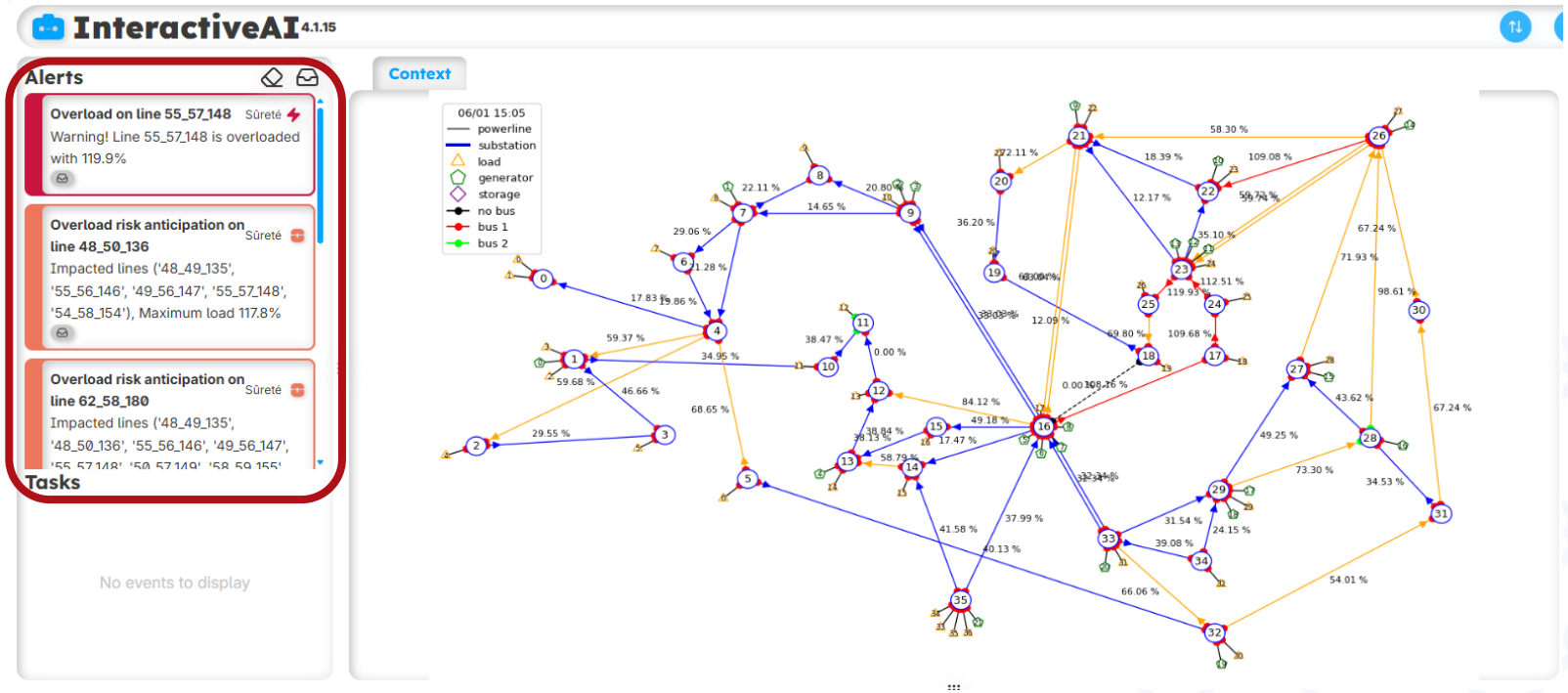
AI agent recommendation:

- Generic interoperability API
- Communication with Maze Flatland Agent

Power Grid Simulator

Improvement of context display and interaction

- correlate the line name in a card with the grid plot
- Grid plot reflects the grid state after the N-1 line disconnection of an anticipation card
- Integrate alerts on lines in addition to the existing alarm on a zone.



AI4REALNET

AI4REALNET has received funding from European Union's Horizon Europe Research and Innovation programme under the Grant Agreement No 101119527 and from the Swiss State Secretariat for Education, Research and Innovation

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.