# Intelligent Verification/Validation for XR Based Systems

**Research and Innovation Action**

Grant agreement no.: 856716

# D5.2 – Intermediate integration of the pilots

**iv4XR – WP5 – D5.2**

**Version 1.5**

**March 2021**

| | |
|---|---|
| Project Reference | EU H2020-ICT-2018-3 - 856716 |
| Due Date | 31/03/2021 |
| Actual Date | 31/03/2021 |
| Document Author/s | Joseph Davidson (GA), Jean-Yves Donnart (THA-AVS), Fernando Pastor (UPV), Karel Hovorka (GA), Ian Saunter (GWE), Alexandre Kazmierowski (THA-SIX) |
| Version | 1.5 |
| Dissemination level | Public |
| Status | Final |

This project has received funding from the European Union's Horizon 2020 Research and innovation programme under grant agreement No 856716

| Document Version Control | | | |
|---|---|---|---|
| **Version** | **Date** | **Change Made (and if appropriate reason for change)** | **Initials of Commentator(s) or Author(s)** |
| 1.0 | 01/03/2021 | Initial document structure and contents | JD |
| 1.1 | 10-11/03/2021 | Added GA scenario and results | JD, FP, KH |
| 1.2 | 16/03/2021 | Added THA-AVS scenario and results | JYD |
| 1.3 | 17/03/2021 | Added conclusion draft, headers/footers | JD |
| 1.4 | 21/03/2021 | Added GWE sections | IS |
| 1.5 | 30/03/2020 | Add short description of THA-SIX tools and first results | AK, JYD |

| Document Quality Control | | | |
|---|---|---|---|
| **Version QA** | **Date** | **Comments (and if appropriate reason for change)** | **Initials of QA Person** |
| 1.4 | 25/03/2020 | Formatting changes | JYD |
| 1.4 | 29/03/2020 | Review of text, language, grammar | JD |
| 1.5 | 31/03/2021 | Final edits before submission | RP |

| Document Authors and Quality Assurance Checks | | |
|---|---|---|
| **Author Initials** | **Name of Author** | **Institution** |
| JD | Joseph Davidson | GA |
| JYD | Jean-Yves Donnart | THA-AVS |
| FP | Fernando Pastor | UPV |
| KH | Karel Hovorka | GA |
| IS | Ian Saunter | GWE |
| AK | Alexandre Kazmierowski | THA-SIX |

| | | |
|---|---|---|
| RP | Rui Prada | INESC-ID |

**TABLE OF CONTENTS**

## EXECUTIVE SUMMARY

This report details the progress of the integration of the iv4XR framework with the pilots supplied by the industrial partners. It explains what intermediate integration entails for each of the pilots, presents an overview of the technical details concerning each pilots' connection to the framework, and finally describes the current integration status and framework capabilities for each pilot.

The previous deliverable concerning basic integration showed communication between the pilots and the iv4XR framework. This communication took the form of sending observations and performing some basic unconditioned actions if applicable. For intermediate integration, the framework shall host an agent or set of tests which can make use of the observations to decide on which actions to take.

All the pilots are able to communicate with an agent hosted in the iv4XR framework and allow the agent to make decisions and to interact with the pilot based on observations supplied by the pilot systems. With the current level of maturity, the rest of the consortium are now able to create their own rudimentary agents to investigate research avenues, such as, coverage and goal solving using the pilot systems as environments.

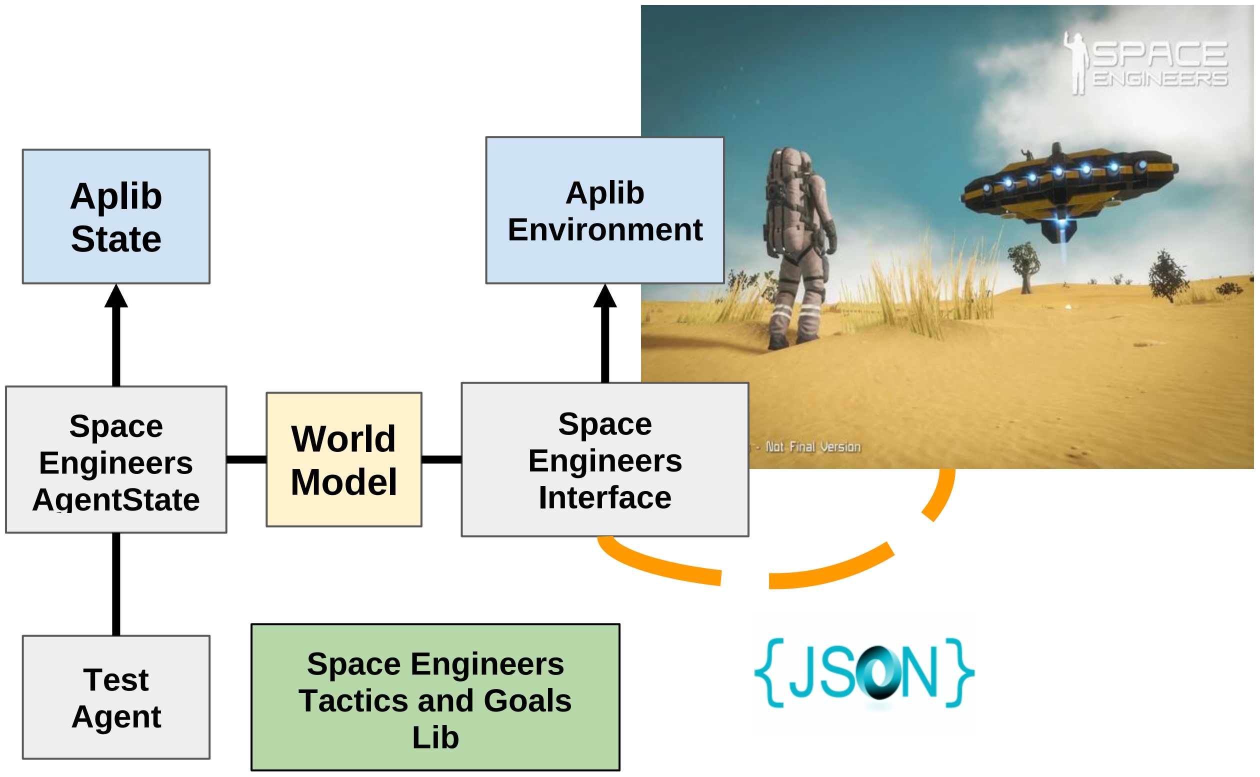
## Acronyms and Abbreviations

| | |
|---|---|
| **XR** | Extended Reality |
| **GA** | GoodAI |
| **SE** | Space Engineers |
| **CGE** | Computer Generated Entities |
| **CGF** | Computer Generated Forces |
| **FTA** | Functional Test Agent |

## 2 INTRODUCTION

One of the objectives of the iv4XR project is to permit external organisations to use the framework so that their extended reality environments can be tested or monitored with less human interaction than is required by the testing methods of today. For adoption to be effective, prospective developers need 1.) Some demonstration of the benefits of using iv4XR, and 2.) A measure of guidance on how to integrate the framework into their development lifecycle. The pilots are one of the methods that the consortium is using to deliver this knowledge.

There are three integration deliverables in this project, of which this is the second. The work in this deliverable consists of building on the interfaces of the first deliverable and producing agents which can interact using the interface in a simple manner. We strive in this deliverable to produce agents which can perform some of the simpler test cases.


Aplib State
Aplib Environment
Space Engineers AgentState
World Model
Space Engineers Interface
Test Agent
Space Engineers Tactics and Goals Lib
{JSON}
SPACE ENGINEERS
Not Final Version

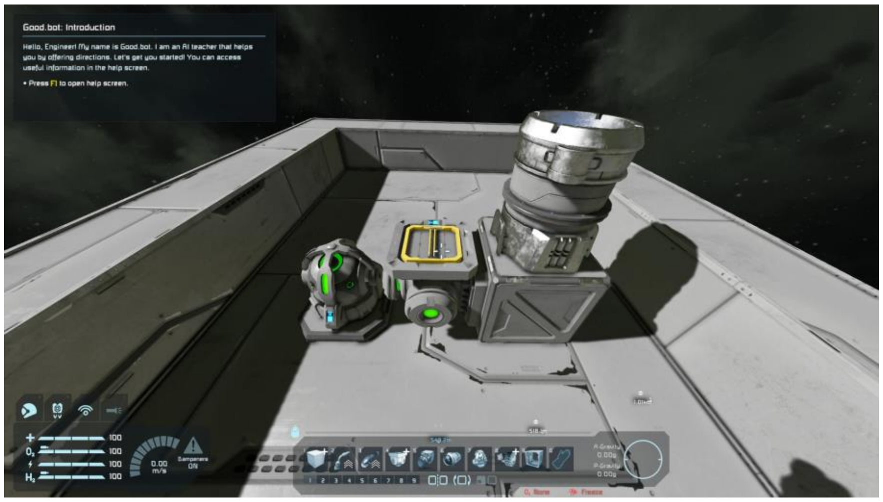
Block Diagram of how the iv4XR system interfaces with Space Engineers

# 3 Intermediate Integration definition

In the previous deliverable, basic integration was broadly expressed as “one way communication” between the framework and pilot where the pilot sends some observation to the framework via the interface. Intermediate integration has been similarly defined as “two-way communication” between the framework and pilots. For each of the pilots, we aim to host an agent in the framework which is able to process observations from the pilot and translate them to actions which can affect the pilot or render a judgement on whether the pilot is operating correctly.

## 3.1 Space Engineers (Good AI)

As a game under continued development, Space Engineers requires thorough and frequent testing. Given the complexity of the game world, the testing team performs the majority of the tests manually. One of these tests is in the creation of game entities known as “blocks”. Blocks are voxels from which the game world is created and blocks vary in functionality from generic armour blocks to more complex storage and medical unit blocks.



Good.bot: Introduction

Hello, Engineer! My name is Good.bot. I am an AI teacher that helps you by offering directions. Let's get you started! You can access useful information in the help screen.

• Press F1 to open help screen.

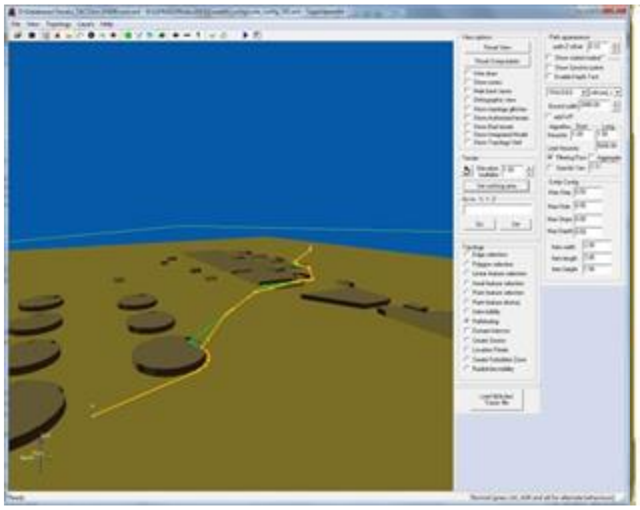
A collection of different blocks; A gyroscope (left), a reactor (centre), and a thruster (right) all attached to a platform of armour blocks.

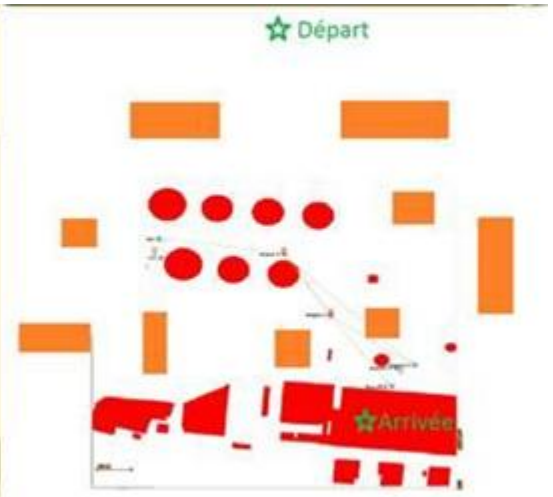
For this integration deliverable we have endeavoured to construct a scenario and agent to automatically perform these block creation and destruction tests. The agent will load into a scenario, place a block, grind it with a tool to damage it until a certain health threshold is passed, then weld it with another tool to repair it fully.

In parallel, UPV has performed an initial integration of their TESTAR tool which tries a number of stochastic strategies in order to determine the state space of space engineers and discover edge cases which require testing.

## 3.2 Nuclear plant intrusion simulation (Thales AVS)

Today, the verification of a simulation with artificial agents (non-player characters) often relies on human operators which play the opposite force (e.g., the intruders) with the objective to challenge the scenario integrated into the simulation. Because this task is cost consuming, we sometimes use the scenario functionalities of our CGE MAEV to model the different strategies of the opposite force and analyze, afterwards, the results of the different confrontations. In most cases, this kind of verification is not exhaustive enough to be efficient.
Our main objective in iv4XR Project is to test an alternative solution for scenario verification where AI agents can learn how to challenge the CGE scenario simulating the defense of a nuclear plant. These AI agents will have access to all the necessary information (the perceived situation of each intruder) and will be allowed to decide which actions will be performed by their corresponding artificial agents into the simulation.





During the "Basic Integration" phase, we defined an API plugin (AIEngine) that allows our CGE MAEV to communicate with AI agents, whatever AI technology they rely on, and developed the communication modules both in the CGE and in the iv4XR Framework in order to ensure the genericity of the communication.

Our objective, during the "Intermediate Integration" phase, is to test the capacity for an external module, developed by another partner of the consortium (Thales SIX), to control some of the MAEV agents through the iv4XR Framework. It means that this external module will receive the states and detections of the MAEV agents under its control and will be able to give high level commands, such as "MoveTo", to these agents.

This step prefigures the "Final Integration" phase where actual AI tools, such as Thales SIX Reinforcement Learning algorithms, will be used to test the defense strategy, implemented in MAEV, by finding a sequence of actions to reach the objective. To achieve this objective, the CGE should also be able to run the simulation much quicker than real time in order to exploit the efficient discovery capacities of Machine Learning algorithms.

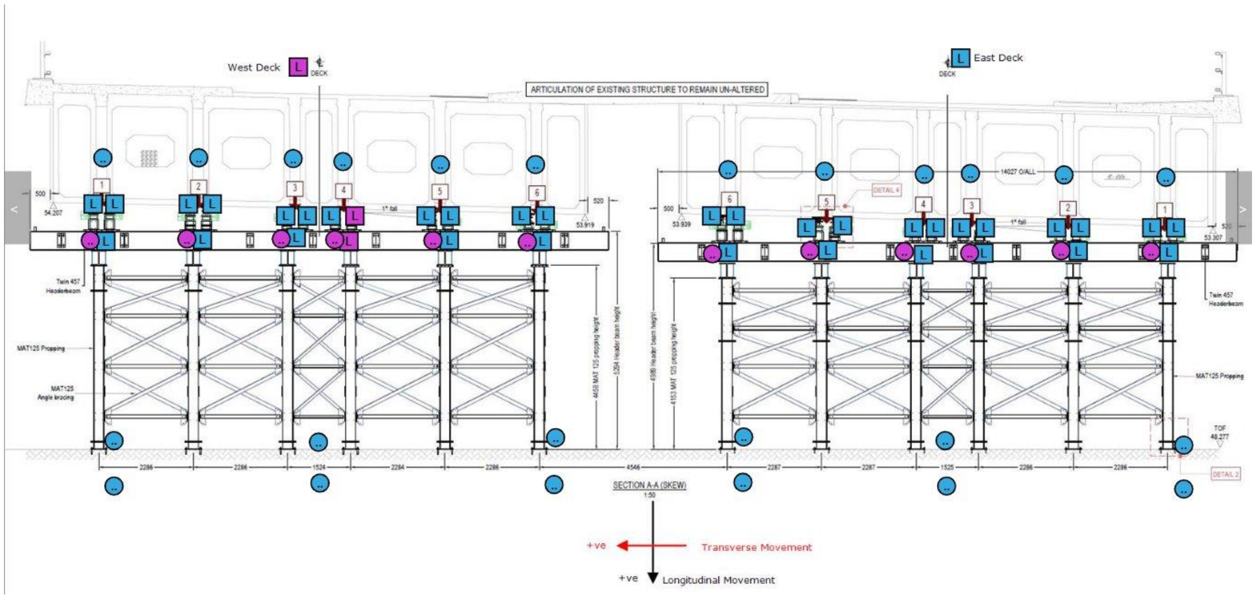
## 3.3 LiveSite (Gameware)

For the basic integration phase, we developed a server-side tool which can interface with the iv4XR framework. Its inputs are monitoring projects with sensor definitions, thresholds, and their varying requirements, and it uses the IV4XR framework to test parameters within the definition of the given sensors.

The server-side tool is written in JAVA and processes command files which include zero or more data items from the project itself. Since the monitoring projects from which we sample frequently have large amounts of data (often gigabytes), the LiveSite tool processes only a small amount of it at any one time.

For the intermediate integration phase, our objective has been to further enhance this tool to allow both processing and navigation of the project, by allowing the tool to control which sections of the data it is looking at. In effect, it is an agent navigating through the data.

Also, for some engineering projects, for example, where there is monitoring of a train going over a bridge or section of track, there is a physical moving entity as well as the structure itself. For this phase we have been adding support to enable monitoring of moving entities in the project.



For the final integration phase, we will enhance the tool further so that visualisation of the progress of the agent within the monitoring project scenario is visible on a virtual timeline, as well as adding more parameters from various projects that can be run inside the tool.

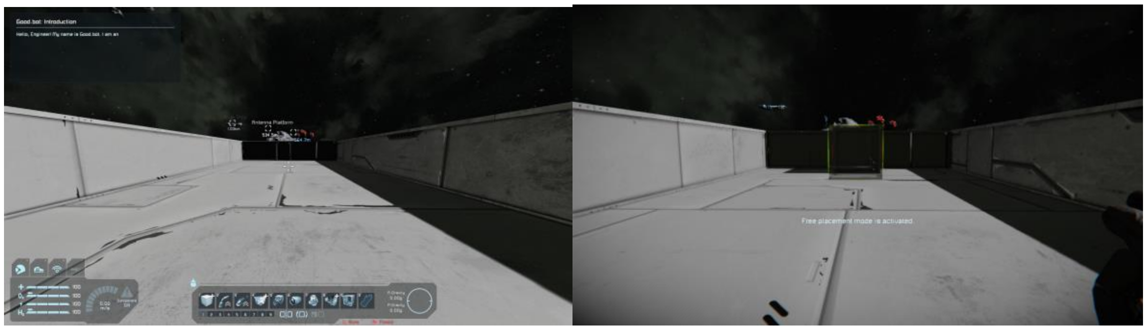
There are many types of sensors and we have concentrated up to now on positional movements and expansions (LVDTs). During the final phase we will integrate further sensor monitoring, such as, support for noise, dust and temperature.

# 4 Interaction between environments and agents

## 4.1 Space Engineers (Good AI)

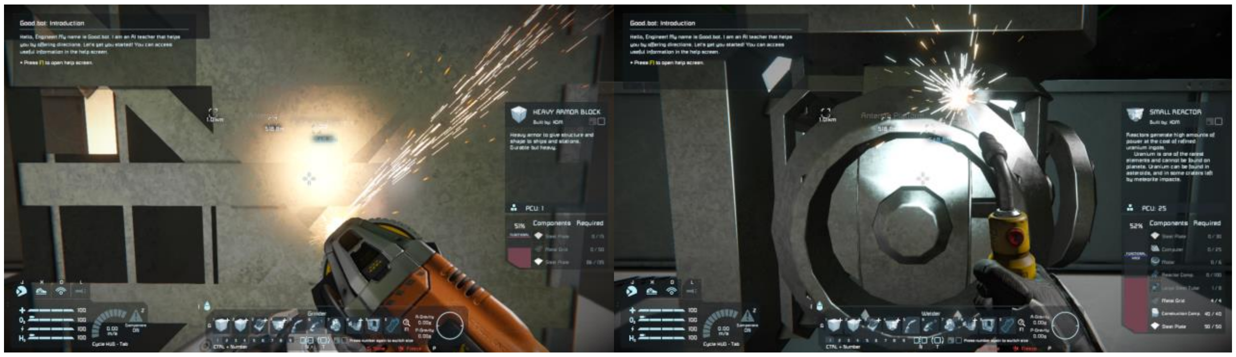
The scenario that we have developed is in the construction of a block and its reaction to the handheld tools that the player can wield. The player has a grinder for destroying blocks and a welder for building or repairing blocks. The objective of the scenario is to determine that a block can indeed be damaged or repaired by these tools.

The scenario is set up where the agent is spawned on a platform with no block or tool equipped. The agent equips the block to be placed and places the block in front of itself.



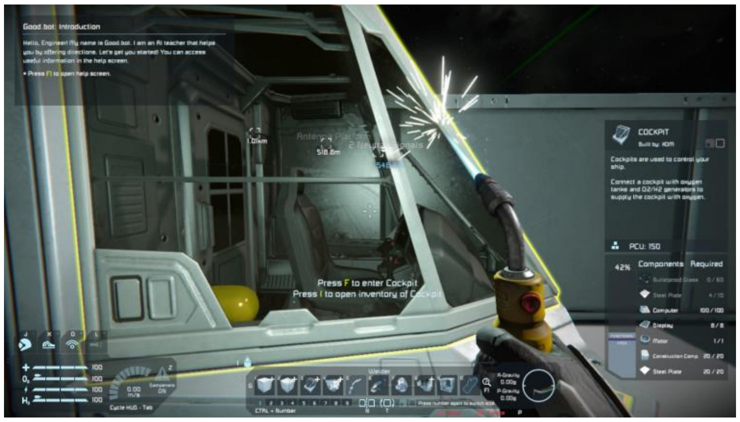
The initial scenario (left). A block that the agent has placed (right).

Once this has been done, the agent equips its grinder and approaches the block. The agent uses the grinder to drop the health of the block to 10% health and stops. It uses observations provided by the pilot to determine the health of the block, the materials in the block and the functional state of the block.



The agent grinding an armour block (left). The agent welding a reactor (right). Note the panel on the right side of the screen in both images that details the health of the block and the materials that are required to further construct. Faded entries indicate items that are still required to finish construction of the block.

Once the block has been appropriately damaged, the agent will equip the welder and repair the block back to 100% health before concluding the test. Again, the agent monitors the health of the block through observations so that it knows when to start and stop welding.

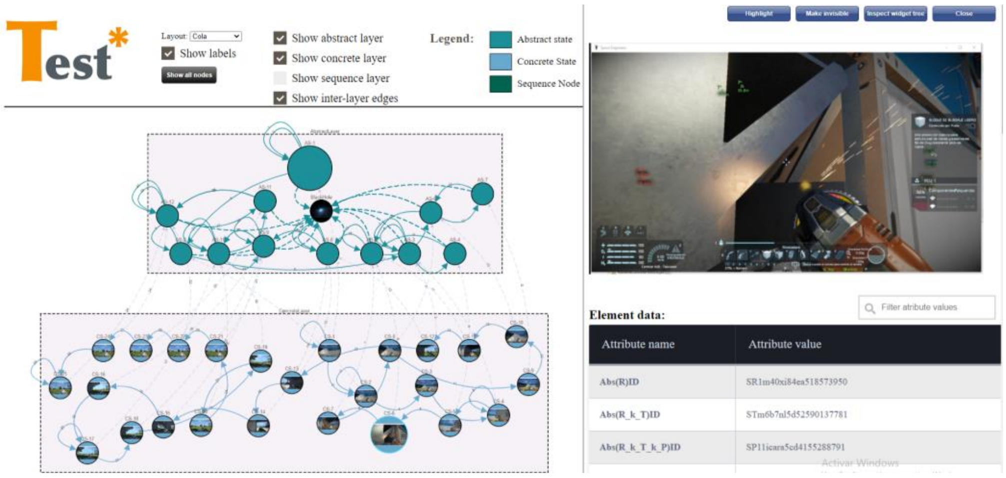

Welding the cockpit. Even though the block health is not yet at 100%, it is still functional is indicated by the health level being above the red line which is at 40%

A video of the agent performing tests is available here[1]. In this video the agent performs the above test on 4 different blocks. The agent involved in the test can be found in the iv4XR repository[2].

### 4.1.1 TESTAR Integration

The TESTAR tool, an exploratory Functional Test Agent (FTA), has integrated a first version[3,4] that is able to connect with the SE System Under Test (SUT) using the SE iv4XR plugin to get the agent observation with all the information of the available widgets (SE blocks), and that allows the execution of actions (movement and tools usage) on the Space Engineers game environment.

As the TESTAR tool executes actions in a semi-random way trying to prioritize not-executed actions, the inference of a State Model is carried out (see Figure T1), a navigable graph that allows the tool to remember where TESTAR has been and what TESTAR has done.



| Attribute name | Attribute value |
|---|---|
| Abs(R)ID | SR1m40xi84ea518573950 |
| Abs(R_k_T)ID | STm6b7nl5d52590137781 |
| Abs(R_k_T_k_P)ID | SP11icarn5ed4155288791 |

Space Engineers State Model inferred by TESTAR tool.

[1] https://www.youtube.com/watch?v=nJSvfUujci8
[2] https://github.com/iv4xr-project/iv4xrDemo-space-engineers
[3] TESTAR iv4xr v3.0 release: https://github.com/iv4xr-project/TESTAR_iv4xr/releases/tag/v3.0
[4] TESTAR agent interacting with the SE environment: https://www.youtube.com/watch?v=yxpud4LY1zw

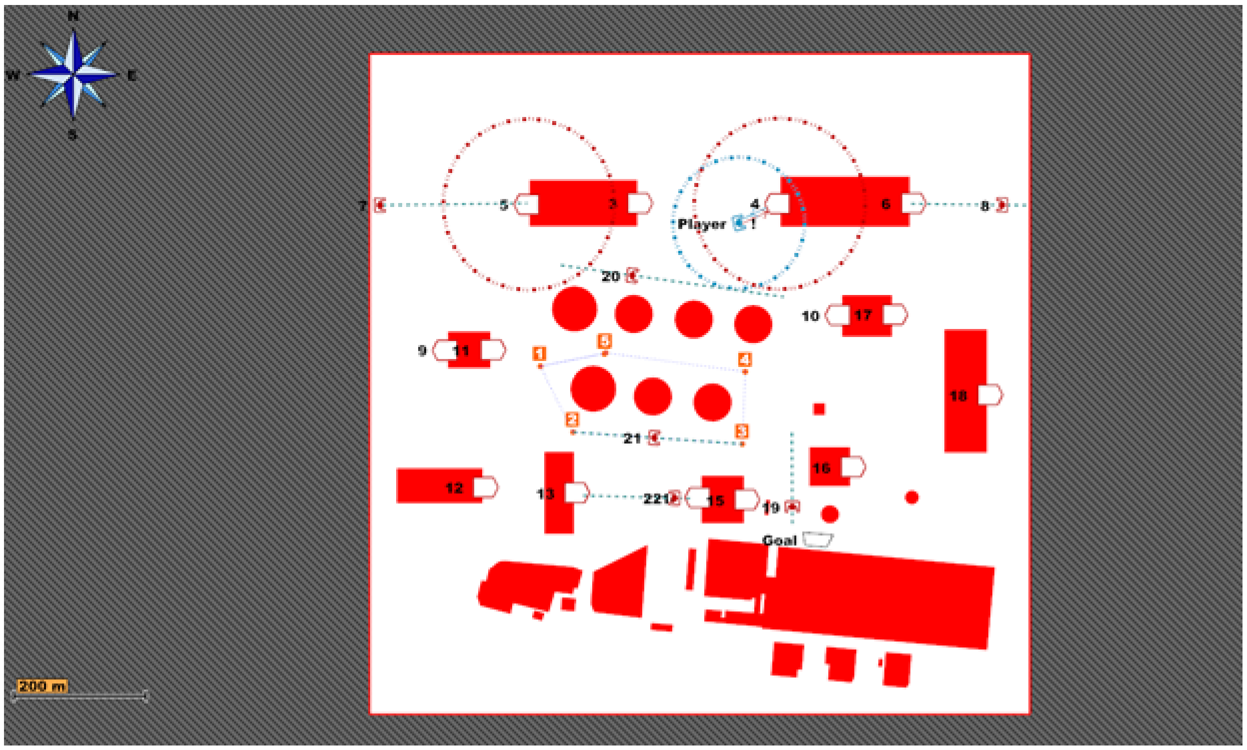
## 4.2 Nuclear plant intrusion simulation (Thales AVS)

In this pilot, the testing agents are AI agents that try to defeat the defense strategy of a power plant by finding a way to get an intruder to the core of the plant without being detected. Because the number of guards and cameras are limited, our objective is to use AI tools to test the quality of different defense strategies in order for the operator to find a compromise between efficiency and available resources.

Within this scenario, intruders may use a map of the plant but have no initial information about the locations of cameras and about the number and behavior of the guards.
The defense strategy is modelled by Thales' simulation tool MAEV thanks to its scenario module where one can position cameras and assign missions to the guards (such as patrol). The behaviour of the guards is modelled in MAEV thanks to traditional AI capacities like mission graphs, behavioral trees and pathfinding algorithms.

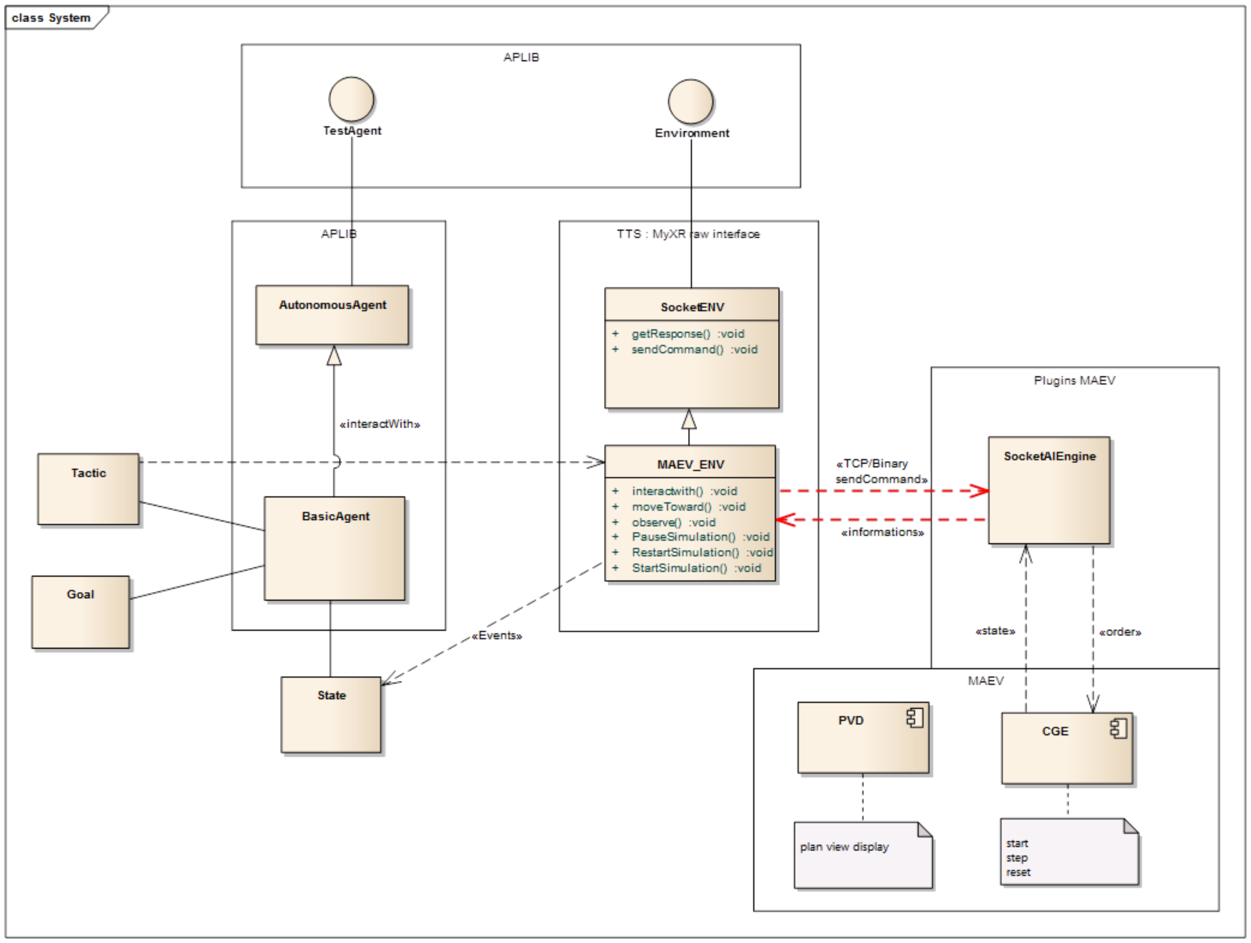
The following figure represents a map of the plant, and we can see a basic scenario that has been modeled for the "Intermediate Integration" phase:

- Agent 1 (the "Player") represents an intruder who tries to reach the "Goal". It is the only MAEV agent controllable by an external module.
- Agents 3, 4, 5, 6, 9, 10, 11, 12, 13, 15, 16, 17, 18 and 19 are 180° cameras;
- Agents 7, 8, 19, 20, 21 and 22 are guards with specific missions (e.g., agent 21 makes a circle patrol passing by locations identified by squares 1, 2, 3, 4 and 5 on the map).



Each agent has a detection range, symbolized by a circle on the map, that parametrizes the detection capacity of the agents. For instance, we see in the figure that the “Player” is detected by Camera 4 but the camera is detected in return by this agent.
The pilot offers the possibility for an external module to control some of the MAEV agents through the iv4XR Framework. For instance, Agent 1 (The “Player”) receives its movement instructions from the iv4XR Framework and sends back its perception of the environment.

The following figure describes the integration of MAEV with iv4XR Framework. In particular, we have developed an environment class MAEV_ENV in the iv4XR framework (that inherits from the SocketEnv class) and an interface class SocketAIEngine in the AIEngine plugin which assure the communication between MAEV and the Framework and allow the reception of commands (MAEV high level actions) and the emission of the agent's state vectors (position, speed, detections).



This communication with the iv4XR Framework is performed by TCP sockets and use the following JSON format:

- For sending commands from the Framework to MAEV agents:
  *{"cmd":"AGENTCOMMAND","arg":{"cmd":"MOVETO","agentId":1,"targetId":1,"arg":{"x":1.0,"y":0.0,"z":0.0}}}*
  *{"cmd":"AGENTCOMMAND","arg":{"cmd":"DONOTHING","agentId":1,"targetId":1,"arg":{"t":1.0}}}*

- <u>For receiving MAEV agent's state vectors by the Framework</u>:

```
{"agentID":1,
"tick":13139012,
 "agentPosition":{"x":0;,"y":0;,"z":0;},
 "velocity":{"x":10;,"y":10;,"z":0;},
 "didNothing":"true",
 "entities":[
        {"isActive":"true",center":{"x":5;,"y":0;,"z":5;},"extents":{"x":00,"y":0.,"z":0.},"type":2,"tag":"",
        "id":2,"position":{"x":5;,"y":0;,"z":5;},"property":""},
        {"isActive":"true","center":{"x":3;,"y":0;,"z":2;},"extents":{"x":0.,"y":0.,"z":0.},"type":2,"tag":"",
        "id":3,"position":{"x":3;,"y":0;,"z":2;},"property":""},
        {"isActive":"true",center":{"x":4;,"y":0;,"z":2;},"extents":{"x":0.,"y":0.,"z":0.},"type":2,"tag":"","i
        d":4,"position":{"x":4;,"y":0;,"z":2;},"property":""}],
"navMeshIndices":[]}
```

We began the integration with Thales SIX in December 2020 by providing a standalone executable capable to load the AIEngine plugin and emulate the CGE (reception of commands and sending of agent's state vectors).

In February, we tested the AI Engine plugin thanks to a standalone executable capable of sending move commands to MAEV agent in place of the iv4XR framework and displaying the returned state vectors.

In March 2021, a first version of the pilot, integrating MAEV with the above scenario to the iv4XR Framework, was embedded on a PC and sent to Thales SIX for basic experimentations.
The results are in line with our expectations:

- The intruder (Agent 1) moves on the map according to commands sent by the Thales SIX module connected to the iv4XR Framework;
- The state vector of Agent 1 received by the Thales SIX module evolves according to its current position, speed and detections in MAEV simulation;
- The cameras still detect the intruder controlled by the Thales SIX module when it enters their detection perimeter.

A video of this agent performing tests can be found here: https://youtu.be/QPpnRloAr7o
In this experiment, the agent receives its current perceptions and state from MAEV Simulation through the IV4XR Framework and launches "MoveTo" commands to random destinations to drive the "Player" entity. Note that the agent's destination changes every 10 seconds to make it more demonstrative.

The code of the plugin is available on the project's GitHub page:
https://github.com/iv4xr-project/iv4XR-IntrusionSimulation

Thanks to these promising results, we will continue to improve the pilot by accelerating the simulation and by enhancing the scenario in order to allow Thales SIX to achieve AI experiments with their Reinforcement Learning tools.

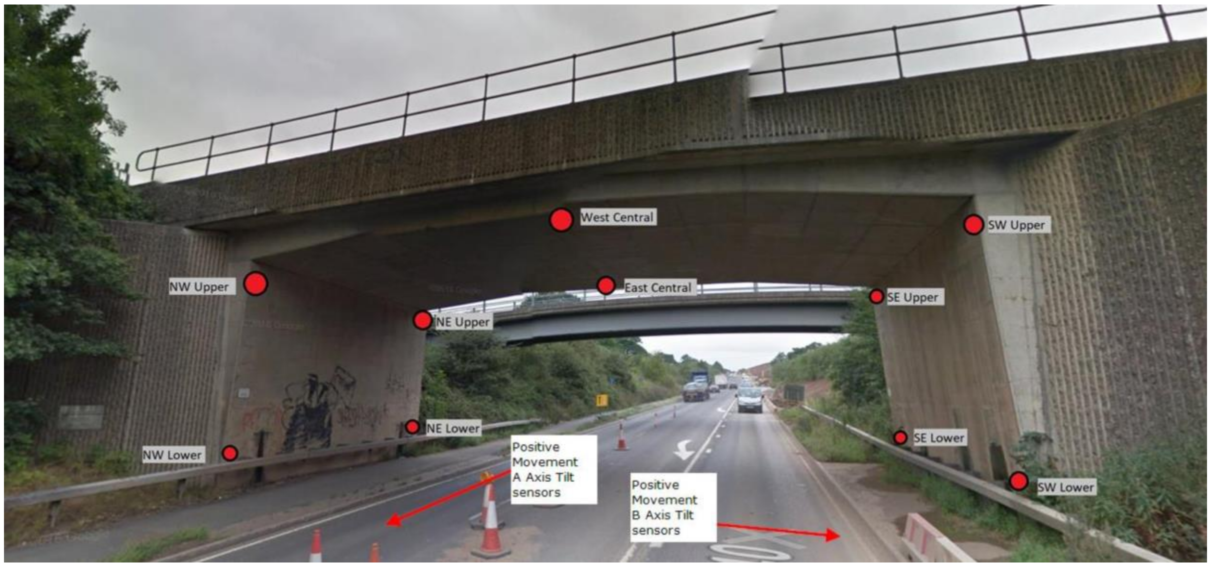
The code of Thales SIX tools is divided in two modules, available on the project's Github page:

- The **iv4xr-rl-env** module allows defining a Reinforcement Learning environment interface over iv4XR Systems Under Test. It also contains a connector to a Python Reinforcement Learning Agent training code that provides actions and receives states and rewards. https://github.com/iv4xr-project/iv4xr-rl-env
- The **iv4xr-rl-trainer** module allows training Deep Reinforcement Learning agents in the Python language on Reinforcement Learning environments run by the **iv4xr-rl-env** module within the iv4XR framework. For the intermediate integration, a placeholder that outputs random actions replaces the actual training code. https://github.com/iv4xr-project/iv4xr-rl-trainer

## 4.3 LiveSite (Gameware)

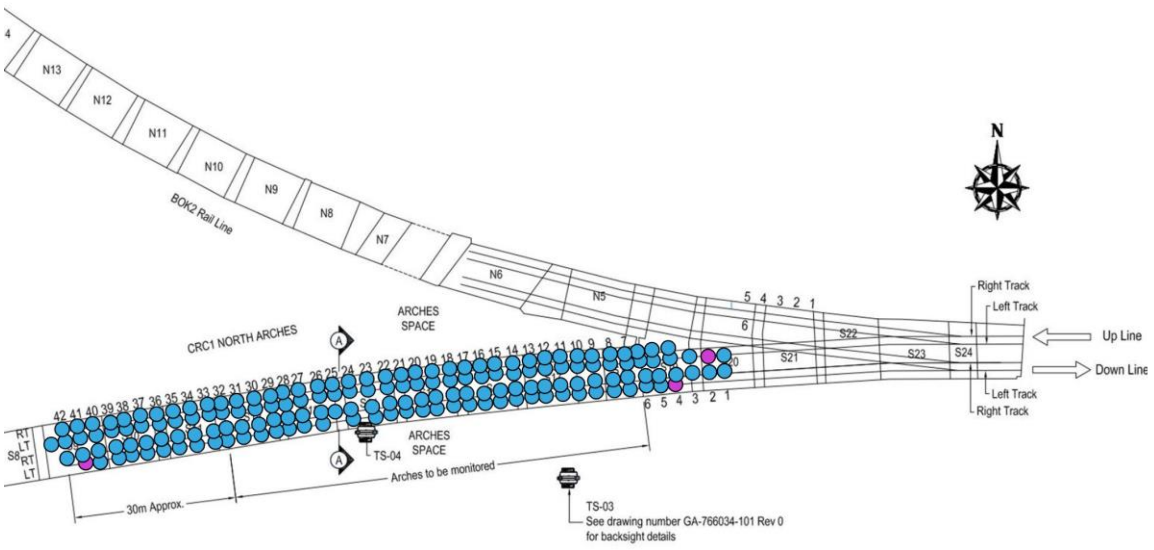
We are using 2 main scenarios for testing.

The first is a concrete bridge which can exhibit some movement over time. Movement can cause concrete to crack, so it is very important that all readings are accurate and the system needs to verify the integrity of the data of all sensors. Sometimes sensors fail or are erroneous or have been calibrated incorrectly.



Structural movement monitoring

The second scenario models a section of train track, which vibrates as trains pass over it.



Track monitoring

In both scenarios the tool is fed a time slice of data from the project and the tool needs to verify if any sensors are incorrect, exceed their given thresholds or may be missing or not working.

An example of sensor parameters for an LVDT (movement sensor) include (simplified:)

```
{
type: LVDT,
name: LVDT_NW_1,
value: 0.7,
Min:0.5,
Max:0.8,
position: {100,40,20}
}
```

The tool constructs a list of possible errors, and from these can decide to continue processing the project data (chronologically or arbitrarily) or look at a section of data in more detail (via binary search). In effect, the tool here creates a virtual agent which parses the data based on errors detected or suggested.

For the train moving over the bridge scenario, we know where the train is at any point in time and the vibrations occurring due to the train's position.

This allows further cross-referencing checking to be performed by the tool. So, the tool starts to look at the relationships between the different items in the project over time and at any given point in time.

The system will be further enhanced for the final integration phase, as the projects include formulae on how the sensors are intended to be interrelated. For example, sensor2 may have a formula which states that *sensor2_movement = sensor1_movement * 0.5*.

Normally the relationships are a function of multiple sensors being attached to connecting parts of a structure, we may move in a line or twist - such as piers along a bridge.

## 5 Conclusions

This deliverable has presented the intermediate integration of the project. Intermediate integration has been defined for each pilot, following along the general guidance of “two-way interaction”, and for each pilot we have stated and explained scenarios in which the iv4XR framework has been hosting an agent which is able to sense and react to the world in some fashion. In the Space Engineers and MAEV pilots, that reaction is further interaction with the world, in the LiveSite project we implemented two-way communication between the iv4XR framework and the sensor projects.

The pilots are being continually integrated with the iv4XR framework and at this juncture, the interfaces are developed enough for the other partners to start experimenting with the pilots. The pilot interfaces will become more advanced to support the agents as their capabilities are developed. Allowing the interfaces to accommodate test coverage, exploration, reinforcement learning, and socio-emotive properties.