# Intelligent Verification/Validation for XR Based Systems

**Research and Innovation Action**

Grant agreement no.: 856716

# D3.4 – Final version of Functional Test Agents (FTAs)

**iv4XR – WP3 – D3.4**

**Version 1.10**

**December 2022**

| | |
|---|---|
| Project Reference | EU H2020-ICT-2018-3 - 856716 |
| Due Date | 31/12/2022 |
| Actual Date | 30/12/2022 |
| Document Author/s | Tanja Vos (UPV), Fernando Pastor Ricós (UPV), Borja Davó Gelardo (UPV), Wishnu Prasetya (UU), Fitsum Kifetew (FBK), Davide Prandi (FBK), Victor Gabillon (THA-SIX), Joseph Davidson (GA), Pedro Fernandes (INESC-ID), Inês Carvalho (INESC-ID), Jason Lander (GWE) |
| Version | 1.10 |
| Dissemination level | Public |
| Status | Final |
| Type | OTHER |

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 | 20/10/2022 | Initial document structure and contents | FP |
| 1.1 | 21/11/2022 | Discuss content and add tasks description | TV, WP, FK, FP |
| 1.2 | 9/12/2022 | Add exploratory and distributed sections | FP |
| 1.3 | 9/12/2022 | Add INESC-ID quality-diversity to Coverage section | PF |
| 1.4 | 9/12/2022 | Add QR-RL coverage and multi-agent THALES approaches | VG |
| 1.5 | 9/12/2022 | Add Space Engineers multi-agent section | JD |
| 1.6 | 9/12/2022 | Add LiveSite multi-site agents section | JL |
| 1.7 | 13/12/2022 | Add EvoMBT and RLbT sections | FK |
| 1.8 | 14/12/2022 | Add Augmented Reality section | BD, IC, FP |
| 1.9 | 15/12/2022 | Add LTL and Solvers sections | WP |
| 1.10 | 30/12/2022 | Final arrangements for submission | RP |

| Document Quality Control | | | |
|---|---|---|---|
| **Version QA** | **Date** | **Comments (and if appropriate reason for change)** | **Initials of QA Person** |
| 1.9 | 20/12/2022 | Initial comments and typo fixes | ML |

| | | | |
|---|---|---|---|
| 1.9 | 25/12/2022 | English grammar, sentence structure | JD |

| Document Authors and Quality Assurance Checks | | |
|---|---|---|
| **Author Initials** | **Name of Author** | **Institution** |
| TV | Tanja Vos | UPV |
| FP | Fernando Pastor | UPV |
| BD | Borja Davó | UPV |
| WP | Wishnu Prasetya | UU |
| FK | Fitsum Kifetew | FBK |
| DP | Davide Prandi | FBK |
| VG | Victor Gabillon | THA-SIX |
| JD | Joseph Davidson | GA |
| PF | Pedro M. Fernandes | INESC-ID |
| IC | Inês Carvalho | INESC-ID |
| JL | Jason Lander | GWE |
| ML | Manuel Lopes | INESC-ID |
| RP | Rui Prada | INESC-ID |

## TABLE OF CONTENTS

## EXECUTIVE SUMMARY

This deliverable D3.4 is of type OTHER, it describes the progress made in the final year related to the Functional Test agents (FTAs). The real delivery is the software that is available on the iv4XR GitHub repository: https://github.com/iv4xr-project

In this deliverable, we will summarize the overall concepts, architecture, design, and technical choices. With the intent to give a clear overview for the reviewers of the work that has been done in WP3, we will describe per task:

- Short introduction to the task.
- What has been done in the last year.
- Where the result can be found (link to GitHub, Zenodo, videos) and how to use them.

### ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| **FTA** | Functional Test Agent |
| **SUT** | System Under Test |
| **XR** | Extended Reality |
| **AI** | Artificial Intelligence |
| **WOM** | World Object Model |
| **API** | Application Programming Interface |
| **RL** | Reinforcement Learning |
| **DRL** | Deep Reinforcement Learning |
| **DSL** | Domain Specific Language |
| **TSL** | Test Specification Language |
| **LTL** | Linear Temporal Logic |
| **MDP** | Markov Decision Process |
| **MBT** | Model Based Testing |
| **ASM** | Action Selection Mechanism |
| **AR** | Augmented Reality |
| **QDRL** | Quality-Diversity Reinforcement Learning |

# OVERALL CONCEPTS, ARCHITECTURE, DESIGN OF FUNCTIONAL TEST AGENTS (FTAS)

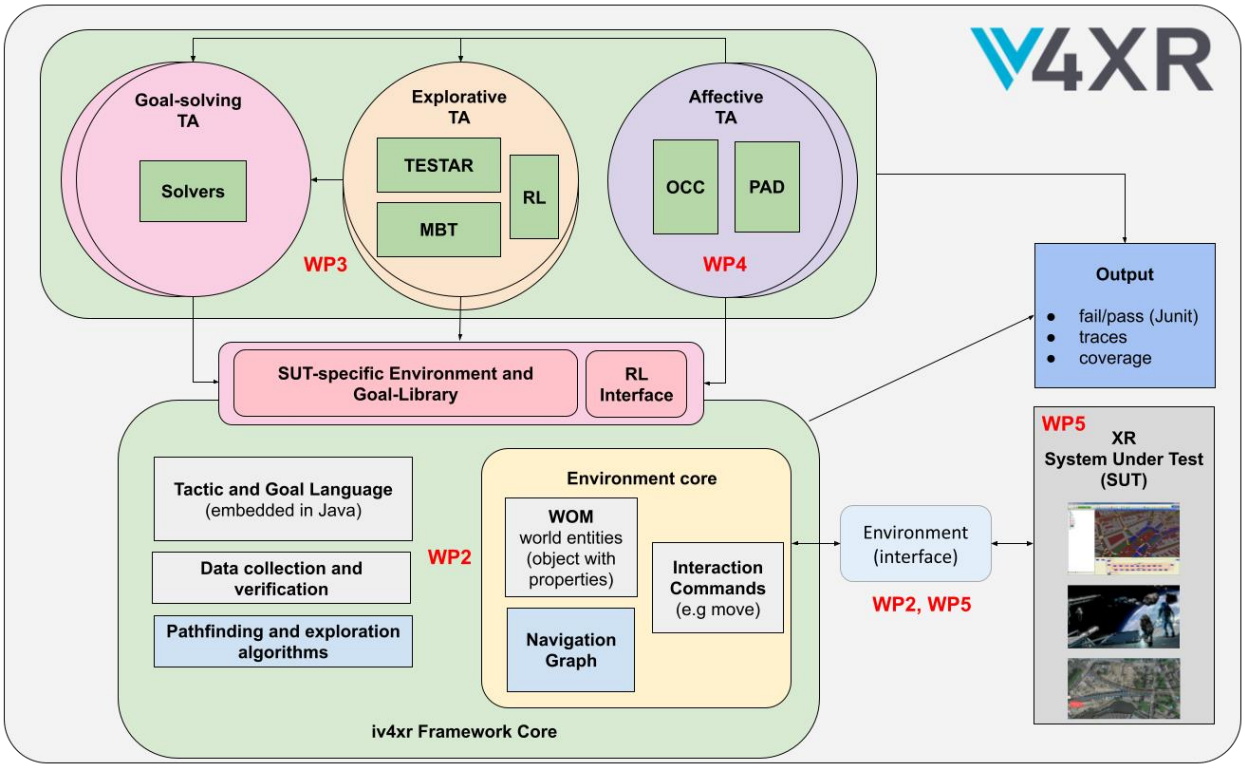


**Figure 1**: Different types of Test Agents in iv4XR.

We distinguish between four sub-types of FTAs in WP3, of which three of them follow similar exploratory capabilities:

- The first type of agent makes deliberations to choose appropriate strategies to allow it to solve goals (**Goal-solving** TA circle in **Figure 1**).
- The second type of agent does not follow specific goal structures but learns from the executed goal interactions to verify if it is possible to achieve a final state (**RL** section from **Explorative** TA circle in **Figure 1**).
- The third type of agent explores the XR environment with a *Scriptless* approach while verifying if the system is robust enough to respond to multiple and unexpected user interactions (**TESTAR** section from **Explorative** TA circle in **Figure 1**).
- The fourth type of agent can follow the space of interactions that are abstractly represented in a model to cover all the transitions (**MBT** section from **Explorative** TA circle in **Figure 1**).

These FTAs are able to test the SUTs from WP5, by using the **Framework-Core** from WP2, as is also shown in **Figure 1**.

## TASK 3.1: SPECIFYING TESTS

| | |
|---|---|
| **Short introduction to the task** | This task aims to develop a Test Specification Language (TSL) that would allow developers to abstractly specify complex testing tasks.<br><br>This Deliverable 3.4 focuses on the TSL improvements made in the last year of the project, while ***Deliverable 3.5*** details the usage of the TSL for the FTAs of the iv4XR framework. |
| **What has been done in the last year** | • ***Linear Temporal Logic extension***<br>An LTL formula is a predicate over sequences of states, allowing us to capture a requirement over a whole execution rather than just a requirement over some states. For example, applying LTL predicates on an example game as the SUT, we can check that the total amount of points collected by the agent/player during a certain test scenario should never exceed some upper bound *N*.<br><br>We collaborated with the socio-emotional agents team from WP4 to extend the usage of LTL predicates to allow designers to specify the emotion-based user experiences, such as fear, hope, and joy, that the agents need to verify over time and space.<br><br>• ***Specifying tests for Augmented Reality systems***<br>The Espresso testing framework, which allows realizing user interactions with mobile devices, has been integrated within the iv4xr framework to enable the declaration of tactics and goals by using the TSL. This allows stakeholders to formulate the declarative goals that the agents follow to test the virtual characteristics of Augmented Reality (AR) systems. |
| **Where are these results and how to use them** | **User Documents**. The Test Specification Language is actually part of the DSL for formulating goals and tactics for agents, so they share the same underlying concepts. More on the concepts of iv4xr agent programming can be found in its Documentation page:<br><br>https://github.com/iv4xr-project/aplib/blob/master/README.md<br><br>**User Reference.**<br>1. The syntax of the DSL test specification language: https://github.com/iv4xr-project/aplib/blob/master/docs/manual/DSL.md<br><br>2. APIs Reference is provided as part of the APIs reference of the Framework-core: : |

| | |
|---|---|
| | http://www.staff.science.uu.nl/~prase101/research/projects/iv4xr/aplib/apidocs/<br><br>The key classes implementing the DSL are AplibEDSL and iv4xrEDSL in the package nl.uu.cs.aplib and eu.iv4xr.framework. The implementation of LTL can be found in the class LTL and its subclasses in the package eu.iv4xr.framework.extensions.ltl. We also provide an extension of LTL called Bounded-LTL, which can be found in the class BoundedLTL in the same package.<br><br>3. An extension of the LTL from the Framework Core that is more specialized for expressing user experience can be found here: https://github.com/iv4xr-project/ltl-pxevaluation<br><br>4. Augmented Reality demo project https://github.com/iv4xr-project/iv4ARDemo/tree/tests<br><br>**Usage Examples**<br>1. A starting (simple) example of using the DSL is provided in the above Documentation page of the Framework. Direct link: https://github.com/iv4xr-project/aplib/blob/master/docs/iv4xr/testagent_tutorial_1.md<br><br>2. Examples of the use of LTL are provided in the above Documentation page of the Framework. Direct link: https://github.com/iv4xr-project/aplib/blob/master/docs/iv4xr/testagent_tutorial_3.md<br><br>3. A full example in the context of a realistic SUT can be found as part of the iv4xrDemo project that demonstrates the use of iv4xr to test a 3D maze-puzzle game. The Demo project can be found here: (https://github.com/iv4xr-project/iv4xrDemo. Look for example in the test-example src/test/java/agents/demo/RoomReachabilityTest.java.<br><br>**Implementation**.<br>The Test Specification Language is part of the Framework-core, which can be obtained here: https://github.com/iv4xr-project/aplib<br><br>For inspecting the code, the key classes that implement the DSL are AplibEDSL and iv4xrEDSL. The implementation of LTL and Bounded-LTL can be found in the classes LTL and BoundedLTL.<br><br>**Publications:**<br>*Aplib: Tactical Agents for Testing Computer Games*. Paper by Prasetya, Dastani, Prada et al. Published in EMAS 2021<br>https://link.springer.com/chapter/10.1007/978-3-030-66534-0_2 |

## TASK 3.2: GOAL SOLVING AGENTS

| | |
|---|---|
| **Short introduction to the task** | This task aims to develop agents that follow test-related goals to automatically navigate to certain locations in a virtual world that contains the entities that must be tested. These goals are composed of testing tasks consisting of a sequence of high-level steps followed by an assertion of the correctness condition to check. Each high-level step is essentially a goal for the test agent, which needs to be solved by the agent to accomplish the testing objective. The following types of solvers are being developed in T3.2:<br><br>● **Graph-based pathfinding and exploration algorithm**. A *pathfinding* algorithm is used to auto-navigate from one location to another and complete the testing task.<br>● **Reasoning-based solver** by using Prolog which allows reasoning rules to be formulated, thus allowing a running agent to use them for making decisions.<br>● **Model checker solvers**. This kind of solver can solve a goal provided a LTL bounder model checker *M* is given.<br>● **Online solvers**. An online solver tries to solve a goal by actually trying different interactions on the SUT, by following a certain heuristic without the need of a model.<br>● **Learning algorithm**. Developers can express a SUT-specific problem as an environment for Reinforcement Learning by formalizing it as a Markov Decision Process (MDP) with states, actions, and rewards. Then, a Deep Reinforcement Learning (DRL) agent can be used to solve the SUT-specific goal.<br><br>This Deliverable 3.4 focuses on the goal solver improvements and summarizes the new AR development made in the last year of the project. Then ***Deliverable 3.5*** details all the solver capabilities that goal-solving agents provide to the iv4XR framework and extends the AR agent capabilities. |
| **What has been done in the last year** | ● ***Model checker solvers.*** An LTL model checker has been implemented for iv4xr. It implements a similar double DFS algorithm as in the SPIN model checker[1], however unlike SPIN, our model checker is a bounded model checker (BMC) and hence can deal with unbounded state space. A decided to implement our own LTL BMC tool to facilitate better integration with the rest of the Framework modules. As far as we know ours is the only LTL BMC implementation in Java. |

[1] https://spinroot.com/spin/whatispin.html

| | |
|---|---|
| | • ***Online solvers.*** Online solvers have been extended with a new systematic algorithm that allows agents to autonomously explore a level to complete all the high-level transitions (basic goals) that compose the complete goal structure. Using this approach, the agent requires more time to explore the XR level, but the developer no longer needs to specify how to solve all the composed transitions of a goal.<br><br>• ***Goal-solving agents for Augmented Reality systems.*** The extension of the iv4xr framework with the Espresso testing framework allows the usage of goal-solving FTAs to test different characteristics of AR systems. These agents can observe the environment of different SUTs based on the Google ARCore software development kit to obtain the WOM and to test that the surface position, the orientation of the entities, or the collision between them are correct when objects are placed.<br><br>• ***Applying RL solver to MAEV scenario***. We worked on scaling the TD3 algorithm from small mazes with a small number of guards in the guarding patrol to larger mazes with a larger number of guards. We reached an almost 100% successful intrusion strategy in the final environment, that is the powerplant use case with the SE-STAR simulator. Our techniques to enable the scaling involved reward-shaping and curriculum learning. When transferring the strategy to the MAEV simulator (more realistic than SE-STAR) we noticed that the strategy would only have a success rate of 50%. |
| **Where are these results and how to use them** | • **Graph-based pathfinding, Reasoning-based, Model checker and Online solvers** are implemented in the Framework-core. The Framework-core can be obtained here: https://github.com/iv4xr-project/aplib<br><br>Documentation and tutorials that helps for the integration and usage of solvers can be found here: https://github.com/iv4xr-project/aplib/blob/master/docs/agentprogramming.md<br><br>https://github.com/iv4xr-project/aplib/blob/master/docs/iv4xr/solvers.md<br><br>• **Agent-based approach that contains goal solving FTAs publication:**<br>I. S. W. B. Prasetya, Fernando Pastor Ricós, Fitsum Meshesha Kifetew, Davide Prandi, Samira Shirzadehhajimahmood, Tanja E. J. Vos, Premysl Paska, Karel Hovorka, Raihana Ferdous, Angelo Susi, and Joseph Davidson. An agent-based approach to automated game testing: an experience report. In Proceedings of the 13th International |

| | |
|---|---|
| | Workshop on Automating Test Case Design, Selection and Evaluation (A-TEST 2022).<br><br>https://doi.org/10.1145/3548659.3561305<br><br>● **Online solver approach publication**:<br><br>Samira Shirzadehhajimahmood, Wishnu Prasetya, Frank Dignum, Mehdi Dastani, An Online Agent-based Search Approach in Automated Computer Game Testing with Model Construction. In Proceedings of the 13th International Workshop on Automating TEST Case Design, Selection, and Evaluation. 2022.<br><br>https://zenodo.org/record/7230140#.Y5ulhOzMIUp<br><br>● The iv4XR plugin for defining Reinforcement Learning environments with the SUT as well as the Python connector for DRL Agents and the implementation of the TD3 algorithm as a goal solver are available in the GitHub repository:<br>GitHub - iv4xr-project/iv4xrl: iv4XR RL Environment library<br><br>Details about the approach and illustrations of usage are available in the README and Wiki<br><br>● Augmented Reality demo project<br>https://github.com/iv4xr-project/iv4ARDemo/tree/tests |

## TASK 3.2: EXPLORATION AGENTS

| | |
|---|---|
| **Short introduction to the task** | The open-source **`TESTAR`** tool takes an **`exploratory`** FTA role within the iv4xr framework. This FTA does not follow specific goals or crafted models to interact with and test the XR system. TESTAR executes non-sequential actions to test that the System Under Test (SUT) and its functional aspects are robust enough to respond to different and unexpected user interactions.<br><br>TESTAR is a tool that existed previous to the project to test Graphical User Interface (GUI) systems, and that has been extended with the iv4XR plugins to observe the SUT states, execute actions and apply oracles. ***Deliverables 3.2*** and ***3.3*** explained the work done in the first and second years of the project, respectively. This Deliverable 3.4 focuses on the new developments and improvements made in the last year of the project, while ***Deliverable 3.5*** details the final integration as an exploratory agent within the iv4XR framework. |
| **What has been done in the last year** | **1.** The extension of the **`State`** classes was already implemented in TESTAR to allow the tool to use the WOM information of the plugin to fetch the Widgets and States that are used to derive actions and apply oracles. This last year, these classes have been updated with the new versions of the iv4XR plugins, which allows the TESTAR tool to obtain additional properties of the virtual entities and create new states of the existing terminals of some XR systems, as in the case of Space Engineers (SE).<br><br>**2.** Regarding the **`Actions`** that TESTAR is capable of deriving and executing, this last year, the effort was mainly focused on improving the navigability aspect of the movements for the SE system. This allows the agent to realize a long and robust exploration of the SUT by reaching the entities to test while dealing with obstructive elements.<br><br>This navigation improvement, together with the changes made that include a new set of properties in the different types of widgets, have allowed the definition of a new set of actions focused on testing functional aspects of the XR system (e.g., test that all the medical rooms restore the agent's health, that all cryo chambers restore agent's energy, etc.)<br><br>**3.** Due to the importance of testing the functional aspect of the virtual entities in XR systems, a new set of **`Oracles`** that test *functional robustness* were included in the TESTAR tool. These new oracles focused on testing the functional aspects of the SE environment allowed us to verify that an exploratory FTA approach can detect existing documented bugs. |

| | |
|---|---|
| **Where are these results and how to use them** | • The wiki section of the TESTAR_iv4xr GitHub repository contains detailed information regarding the technical extension and instructions to download and use the tool. https://github.com/iv4xr-project/TESTAR_iv4xr/wiki<br><br>• Videos are here:<br>a. TESTAR usage instructions<br>i. https://www.youtube.com/watch?v=WxMFVnh5Uso<br>b. TESTAR for LabRecruits<br>i. https://www.youtube.com/watch?v=st4FL_mMflE<br>ii. https://www.youtube.com/watch?v=3T4v3STVMVU<br>c. TESTAR for Space Engineers<br>i. https://www.youtube.com/watch?v=C-y-jV82K50<br>ii. https://www.youtube.com/watch?v=ho1EMVtr8C4<br><br>• Published papers in the third year are:<br>a. *Mulders, A., Valdes, O.R., Ricós, F.P., Aho, P., Marín, B., Vos, T.E.J. (2022). State Model Inference Through the GUI Using Run-Time Test Generation. In: Guizzardi, R., Ralyté, J., Franch, X. (eds) Research Challenges in Information Science. RCIS 2022. Lecture Notes in Business Information Processing, vol 446. Springer, Cham. https://doi.org/10.1007/978-3-031-05760-1_32*<br><br>b. *Pastor Ricós, F. (2022). Scriptless Testing for Extended Reality Systems. In: Guizzardi, R., Ralyté, J., Franch, X. (eds) Research Challenges in Information Science. RCIS 2022. Lecture Notes in Business Information Processing, vol 446. Springer, Cham. https://doi.org/10.1007/978-3-031-05760-1_56*<br><br>c. *Thorn Jansen, Fernando Pastor Ricós, Yaping Luo, Kevin van der Vlist, Robbert van Dalen, Pekka Aho, and Tanja E. J. Vos, Scriptless GUI testing on mobile applications, QRS 2022, 22nd IEEE International Conference on Software Quality, Reliability, and Security. (pending to be published)* |

## TASK 3.3: DEALING WITH HAZARDOUS ELEMENTS

| | |
|---|---|
| **Short introduction to the task** | Hazardous entities are entities in the virtual world that may block or even sabotage an agent's progress or even cause it to fall into an inescapable stuck state. To deal with them the agent needs to be actively aware of their threat and apply countermeasures. |
| **What has been done in the last year** | We made a study on how to program tactics for test agents to deal with enemies. In general, the approach works by extending regular tactics (e.g. tactics for traveling/navigating) with tactics that handle enemies. The latter is used to, for example, handle combat as well acquiring items that would improve the agent's survival. A master thesis describing the study can be found in the results row. |
| **Where are these results and how to use them** | • LabRecruits tactic library: https://github.com/iv4xr-project/iv4xrDemo/blob/master/src/main/java/agents/tactics/TacticLib.java<br>• *Latos, Anastasios. (2022). Automated Playtesting on 2D Video Games An Agent-Based Approach on NethackClone Game via Iv4XR Framework.* https://studenttheses.uu.nl/handle/20.500.12932/510 |

## TASK 3.4: COVERAGE

| | |
|---|---|
| **Short introduction to the task** | This task aims to measure the coverage of the tests to see how good they are in exercising the system under test (SUT). Furthermore, the FTA's functionality is enhanced to increase its efficacy in achieving higher levels of coverage.<br><br>The activities in this task are focused along two main lines: 1) define reasonable metrics for measuring coverage on which different notions of coverage have been explored for the WP5 use cases, and 2) develop test generation strategies that increase the coverage obtained from the tests, on which different approaches based on search as well as reinforcement learning have been explored.<br><br>This Deliverable 3.4 focuses on the new developments and improvements made in the last year of the project, while ***Deliverable 3.5*** details the final coverage approaches integrated into the FTAs and ***Deliverable 5.4*** their validation in the project use cases. |

| | |
|---|---|
| **What has been done in the last year** | ● ***EvoMBT***<br>EvoMBT applies a combination of model based and search based testing approaches for generating tests for XR based systems. In the last year, EvoMBT has been improved by adding a new coverage criterion, i.e., *k-transition coverage*, which allows the generation of test cases that are more thorough with respect to the functionalities of the system they exercise. Furthermore, EvoMBT has been improved to support the possibility to use an externally built model, conforming to the interface provided by EvoMBT, for the generation of test cases. It also supports more search algorithms for the generation of test cases. The tool documentation has been significantly improved, including a wiki that details how to instantiate and use the tool complete with working examples. To assess empirically the effectiveness of different search algorithms, we have carried out a large scale experimental study comparing the performances of the algorithms on a large number of models with different characteristics. The results have been submitted to the TOSEM (ACM Transactions on Software Methodology) Journal. Furthermore, EvoMBT also took part in a tool competition in which it was applied for generating tests for self-driving cars.<br><br>● ***RLbT***<br>In contexts where models of the system under test are not available, iv4xr has developed tools that perform testing applying different techniques. One of these is the RLbT tool which makes use of multi-agent reinforcement learning to test the system. In the last year of the project, RLbT implementation has been completed with support for applying multiple agents running in a collaborative manner to achieve effective testing of the system. RLbT reports coverage of elements it was able to interact with in the environment as well as the percentage of connections between elements exercised. Furthermore, it also produces the sequence of successful actions performed by the active agent for later re-execution. Finally, spatial coverage of the environment is also reported by means of a heatmap plot showing the positions the agent has been at and with what frequency. The RLbT approach and initial results have been presented at the ASE4GAMES workshop, an event dedicated to works on the applications of automated software engineering techniques to development and testing of games.<br><br>● ***Area-based coverage***<br>Test agents can be configured to record properties of the SUT states into trace files. Such a trace file would then contain values of these properties over time, as the agent progresses in its execution. Among these properties we can also include the agent's positions. Based on this information, it is then possible to calculate how well the runs of a test suite cover different areas in the SUT's virtual world. |

| | |
|---|---|
| | To do this, we extended the implementation of LTL in iv4xr with the concept of "*areas*", so that we can then infer, from a trace file of a test agent, which areas were visited by the agent, and where exactly in the area it visited. To be able to calculate the coverage over some or all areas, LTL is extended with an aggregation semantic. More details can be found in the implementation of the extended LTL; a link is provided in the next row of this table.<br><br>● ***Spatial coverage***<br>Some systems, such as SE, employ files to store the configuration of the existing scenarios on which the FTAs execute the testing process. The agents can compare the information obtained while observing the environment at runtime with the file data that contains information regarding the position of the entities to obtain spatial coverage metrics that indicate the number and percentage of observed and interacted blocks and navigated positions.<br><br>● ***Code coverage***<br>We have been integrating open-source code coverage tools to obtain metrics that indicate how the FTA interactions realized with the virtual entities invoke the different internal methods that compose the code of XR systems. Due to these systems can be developed with various programming languages, it was necessary to research the feasibility of multiple tools such as Unity framework[2], OpenCover[3], or dotCover[4].<br><br>● ***Thales QDRL***<br>Quality-Diversity Reinforcement Learning (QDRL) is a meta-algorithm built on top of another base RL algorithm which in our case was TD3. Our work this year went two ways. First, we worked on making sure TD3 would successfully solve the original set of maze problems. This was successfully achieved with curriculum learning. For the QDRL algorithm we played with different metrics for diversity. Finally, we settled for an averaged through time Euclidean distance of the respective trajectories of each agent. Going beyond a deterministic scenario, we now train our agent against multiple patrol strategy patterns.<br><br>We successfully trained the method in medium size mazed with up to 3 guards. However, the method did not converge when trained on the powerplant scenario. |

[2] Unity code coverage: https://docs.unity3d.com/Packages/com.unity.testtools.codecoverage@1.1
[3] OpenCover: https://github.com/OpenCover/opencover
[4] dotCover: https://www.jetbrains.com/dotcover/

| | |
|---|---|
| | ● ***Quality-Diversity Optimisation for Testing Space Engineers***<br>In the game Space Engineers, one of the pilots of the iv4XR project, there exist many different types of blocks, which can be placed and interacted with in many different ways. The placement of blocks next to one another can also change their properties, which makes the task of testing the interactions with these blocks a complex one. One of the biggest problems is ensuring relevant coverage of the interaction space when the amount of blocks and their possible combinations makes it infeasible to test every scenario.<br><br>To tackle this problem, we developed a tool for action generation for two players that promotes diversity for groups of sequences of actions in a given Space Engineers scenario. By implementing a version of the Quality-Diversity optimisation algorithm to generate grids of action sequences, our tool has shown to be capable of creating grids with very good total diversity values, ensuring that various interactions are covered and that redundant testing is minimized.<br><br>In order to evaluate the bug detection capabilities of this tool, we created a game simulator, accompanied by a bug generator, where the actions from the grid are performed and generate the expected outcomes that would be generated in the real game. Our results showed that the framework is capable of detecting a good number of single-player bugs as well as multiplayer bugs. |
| **Where are these results (paper, github, etc)** | ● The combined application of model-based and search-based test generation for maximizing coverage has been presented at the 13th Symposium on Search-Based Software Engineering (SSBSE). The data and replication material accompanying the publication are available in the project Zenodo repository: https://zenodo.org/record/5140432<br><br>● The EvoMBT tool source code is available in the project github repository, together with all the necessary resources and documentation to execute it: https://github.com/iv4xr-project/iv4xr-mbt<br><br>● The use of reinforcement learning for curiosity driven coverage testing has been presented at the ASE4GAMES workshop co-located with the International Conference on Automated Software Engineering (ASE). The publication is available in the project Zenodo repository: https://zenodo.org/record/7224960<br><br>● The RLbT tool source code is available in the project github repository, together with all the necessary resources and documentation needed to execute it: https://github.com/iv4xr-project/iv4xr-rlbt |

| | |
|---|---|
| | • The implementation of reinforcement learning for test generation applied to Lab Recruits is also available in the project github repository: https://github.com/iv4xr-project/iv4XR-FTA-RL<br>• The implementation for calculating area-coverage can be found here: https://github.com/iv4xr-project/ltl-pxevaluation<br>• The implementation of the QD-RL algorithm for behavioral coverage is available in the github repository: GitHub - iv4xr-project/iv4xrl: iv4XR RL Environment library Details about the approach and illustrations of usage are available in the project README and Wiki<br>• The implementation of the quality-diversity optimisation tool design for testing block interactions in Space Engineers can be found in the github repository: https://github.com/iv4xr-project/se-action-grid-generator |
| **How can they evaluate/use these results (what is there and what should they do to evaluate or use it)** | • Usage instructions for the EvoMBT tool are detailed in the wiki: https://github.com/iv4xr-project/iv4xr-mbt/wiki Moreover, EvoMBT provides help on the various options available at runtime, suffices to simply run the tool without any parameters and a help page should be displayed.<br>• EvoMBT performs:<br>  ○ Coverage driven abstract test generation from a given EFSM model<br>  ○ Concretization of the abstract tests and execution on the system under test (Lab Recruits, SpaceEngineers)<br>  ○ Mutation analysis of the generated tests to assess their fault finding potential.<br>• Experimental results are available from our Zenodo repository: https://zenodo.org/record/4769901 |

## TASK 3.5: MULTI AGENT TESTING

| | |
|---|---|
| **Short introduction to the task** | The simultaneous interaction of multiple users in the same environment is an essential feature of XR systems. It is important to verify the correct interaction of multiple users since they can influence each other. This task is focused on extending the iv4XR framework to allow multiple agents to communicate and realize simultaneous interaction in runtime. There are two main objectives for this task:<br><br>- Allow the definition of test cases that involve simultaneous interactions, collaboration, or confrontation of multiple agents.<br><br>- Research the integration of Reinforcement learning (RL) algorithms and strategies that diversify the agent's workload and speed up the exploratory and training procedures.<br><br>***Deliverable 3.3*** explained the multi-agent development done in the project's second year. This Deliverable 3.4 focuses on the new multi-agent improvements made in the last year of the project. Then, ***Deliverable 3.5*** contains an overall vision of the FTAs and ***Deliverable 5.4*** their validation in the project use cases. |
| **What has been done in the last year** | ● ***Multi-agent RL with Lab Recruits***<br>One of the tools developed for generating test cases is RLbT which uses reinforcement learning to explore the system with the purpose of maximizing coverage of the system. RLbT can be run in multi-agent mode where it deploys a couple of agents that work in a collaborative manner, inline with the second objective stated above. In multi-agent mode, RLbT deploys an *active agent* that is guided by a curiosity-driven reward mechanism to explore the system and produce a set of actions that cover the various elements in the system as well as functional aspects such as the connection between doors and buttons, for example. To help the active agent, RLbT deploys another *passive agent* that is responsible for scouting the environment and reporting its observations to the active agent. This enables the active agent to be aware of the effects of its actions in an efficient way, especially in systems where the environment is large and complex. For instance, in Lab Recruits where a button in one room could open a door in another room, the fact that there is a second agent, possibly far from the active agent, allows us to observe changes to the environment triggered by the actions of the active agent.<br><br>The RLbT multi-agent approach is currently applied on Lab Recruits, exploiting its multiplayer feature. However, it could be applied to similar systems that support multiple players. |

- ***Thales Diversity multi-agent RL***

With its Diversity RL approach, Thales has trained multiple agents, each agent interacts in its own environment but is guided by reward that depends on the interaction of other agents in their respective environment.

The adapted QD-RL algorithm also manages the evolution of the agent population to have them explore the state space efficiently, thanks to the diversity criterion that emphasizes the difference between two agents' trajectories.

We manage our multi-agent by storing them in an archive of agents and constantly selecting the ones that are the most promising either in terms of performance or in terms of originality of their strategy (diversity).

- ***SE multi-agent approaches***

Space Engineers is a multiplayer game that allows a large group of clients to connect to a hosted server to play. In order to verify that the host server and the running level respond correctly to multiple agent interactions, the *multi-character* and *multiplayer* extensions were developed.

The *multi-character* feature allows the SE stakeholders to create basic tests to spawn "NPC" agents in one SE client to assign them tactics and goals that execute actions while asserting that the entity's features respond adequately. On the one hand, this feature requires less hardware resources because the testing is done in one SE instance. However, on the other hand, these NPCs are not client independent and capable as the "player" agent because the usage of some tools and the execution of some actions are restricted to the status of the player agent.

The *multiplayer* extension has been developed to emulate the testing environment of the developers with more fidelity by allowing the execution of multiple "player" agent instances simultaneously, which requires more hardware resources because each multi-agent runs in its own SE client. This extension allows SE stakeholders to prepare tests that synchronize the observation of the entity's properties over the network to verify the correctness of the system in two types of multiplayer connections: Lobby, when a player hosts the scenario, and a Dedicated Server (DS) with the scenario.

- ***GWE multi-sites agents***

LiveSite is a complex real-time instrumentation and monitoring system broken down into denominated sub-sites, each with its hosting server that connects to

| | |
|---|---|
| | the sensors for that area. The multi-agent approach allows triggering multiple agents for each sub-site, which can further analyze the sensors at that sub-site.<br><br>Until the final integration phase, the LiveSite testing was performed on mirrors of monitoring servers to ensure no disruption to actual monitoring systems happened. For this reason, during this last year, the multi-site testing agent approach was wholly integrated into LiveSite as a Diagnostics and warning module. Simple summaries of errors are then made available, highlighting problems at each sub-site.<br><br>The testing is performed with multiple instances, with each being given assigned tasks within the overall testing of a project. Although tasks are given to the agent in a hierarchical manner, sometimes a sub-task is more intensive/time-consuming than the parent task which requested it. For this reason, it has been optimized how tasks are broken down, terminated after partial completion, and then restarted and canceled. If a sensor is deemed invalid for some reason (such as no power going to the hub to which the sensor is connected), checking the sensor in more detail is a waste of bandwidth.<br><br>● ***TESTAR distributed approach***<br>The TESTAR agent could infer a State Model while exploring the SUT. However, this inference process was restricted to one model per TESTAR instance, which requires the execution of a large number and takes a long execution time. Multiple TESTAR instances can now connect to a centralized state model to share the knowledge of the observed environment. This is possible due to the usage of the same abstraction mechanism used in TESTAR to identify states and actions using the widget properties. A new Action Selection Mechanism (ASM) allows all TESTAR instances to coordinate their action selection by marking the target actions they pretend to execute. |
| **Where are these results and how to use them** | ● Multi-agent RL with Lab Recruits<br>https://github.com/iv4xr-project/iv4xr-rlbt<br><br>● SE multi-character and multiplayer documentation:<br>https://github.com/iv4xr-project/iv4xr-se-plugin/blob/main/JvmClient/docs/Multiple-Characters.MD<br><br>https://github.com/iv4xr-project/iv4xr-se-plugin/blob/main/JvmClient/docs/Multiplayer.MD<br><br>● TESTAR distributed approach:<br>https://github.com/iv4xr-project/TESTAR_iv4xr/wiki/TESTAR-iv4xr-distributed |

## TASK 3.6: INTEGRATION

| | |
|---|---|
| **Short introduction to the task** | In this task, we integrate the software module that conforms to the FTAs within the iv4XR framework from WP2 to interact and test WP5 use cases. |
| **What has been done in the last year** | The stakeholders involved in WP3 have been working in consonance with WP2, WP4, and WP5 stakeholders, to develop a framework that documents and integrates the features of the FTAs by following the Overall concepts, architecture, design of Functional Test Agents (FTAs). |
| **Where are these results and how to use them** | • The iv4XR framework repository that contains documentation and examples of how FTAs are integrated is available in the following: https://github.com/iv4xr-project/iv4xr-framework |