Intelligent Verification/Validation for XR Based Systems

Research and Innovation Action

Grant agreement no.: 856716

D3.2 – 1st prototype of Functional Test Agents (FTAs)

iv4XR – WP3 – D3.2

Version 1.5

December 2020

Project Reference	EU H2020-ICT-2018-3 - 856716
Due Date	31/12/2020
Actual Date	30/12/2020
Document Author/s	Fernando Pastor (UPV), Tanja Vos (UPV), Wishnu Prasetya (UU), Angelo Susi (FBK), Rui Prada (INESC-ID), Jeremy Cook (GWE), Manuel Lopes (INESC-ID)
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	11/11/2020	Initial document structure and contents	FP
1.1	3/12/2020	Add LabRecruits Agents content	WP
1.2	3/12/2020	Add TESTAR content	FP
1.3	10/12/2020	General modifications	WP, TV, FP
1.4	27/12/2020	End modifications and solve comments	WP
1.5	30/12/2020	Final arrangements for submission	RP

Document Quality Control			
Version QA	**Date**	**Comments (and if appropriate reason for change)**	**Initials of QA Person**
1.0	13/11/2020	Initial comments and section assignment	TV
1.2	7/12/2020	Content review and comments	TV
1.3	14/12/2020	Initial comments	RP
1.3	16/12/2020	Grammar and continuity review	JC
1.3	17/12/2020	Review	AS
1.3	21/12/2020	Comments and minor edits	RP
1.3	22/12/2020	Review	ML

Document Authors and Quality Assurance Checks		
Author Initials	**Name of Author**	**Institution**
TV	Tanja Vos	UPV
RP	Rui Prada	INESC-ID
WP	Wishnu Prasetya	UU

FP	Fernando Pastor	UPV
AS	Angelo Susi	FBK
JC	Jeremy Cook	GWE
ML	Manuel Lopes	INESC-ID

TABLE OF CONTENTS

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

EXECUTIVE SUMMARY

This deliverable D3.2 reports the progress related to the Functional Test Agents (FTAs). It explains the relevant characteristics of the first prototype and version of the FTAs, describes the integration with the iv4XR framework and the demo System Under Test (SUT), and finally presents a Goal Solving demonstration.

1. INTRODUCTION TO FUNCTIONAL TEST AGENTS (FTAS)

The purpose of this document is to reveal the first prototype of FTA's and explain how they use their intelligent skills to test an Extended Reality (XR) system.

To help explain the concepts behind the FTA's we sometimes explain them in terms of examples set in a 3D game called *Lab Recruits*. The game was constructed within our project to serve as a configurable test field for AI. The idea is similar to that of *Open-AI Gym* and *Unity Obstacle Tower*. However, *Lab Recruits* allows researchers to define their own game levels, and hence allowing much more control on the kind of experiments they want to do. Some screenshots of the game are shown in Figure 1 below.

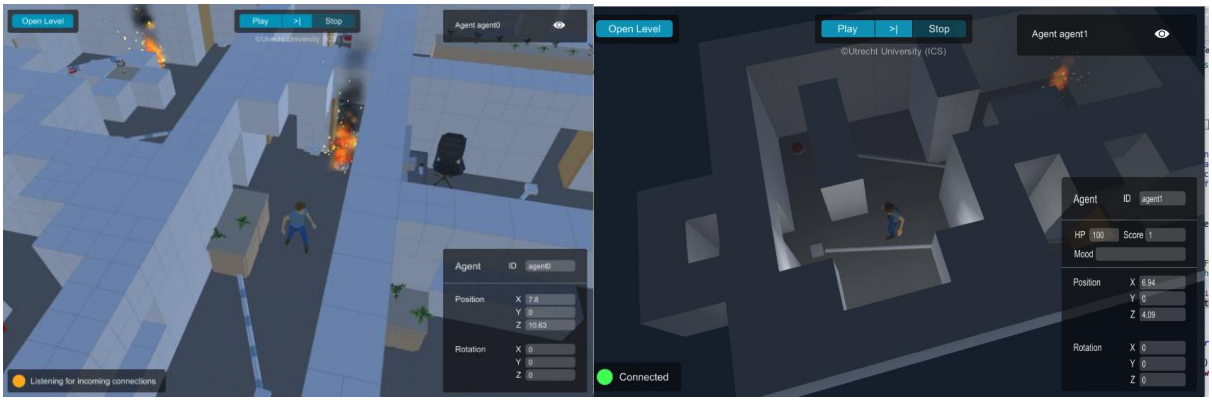

Figure 1: some screenshots of a game called *Lab Recruits* that we created for the purpose of testing our AI. Researchers can create custom game-levels using CSV files, to create custom challenges for AI.

First, beneath FTA's ability to do automated function XR testing, we build the ability to ***navigate*** through the XR system's virtual world. Without this ability, the world is just too large and too fine grained to be searched randomly. To do this efficiently, spatial navigation is implemented through path finding algorithms over the navigation mesh[1] of the XR's virtual World Object Model (WOM). Besides this, we can interact with entities in the virtual world, using the WOM's interaction commands, or specific goal commands.

[1] Navigation Mesh or NavMesh is a component that represents the walkable area for an Agent in an XR system.

Subsequently, two types of FTA's are being developed:

- The first type of agent makes deliberations to choose the appropriate strategies that will allow him to do ***goal-solving*** (Section 4).
- The second type of agent is intended to test the functionality of the XR system using ***exploration*** (Section 5).

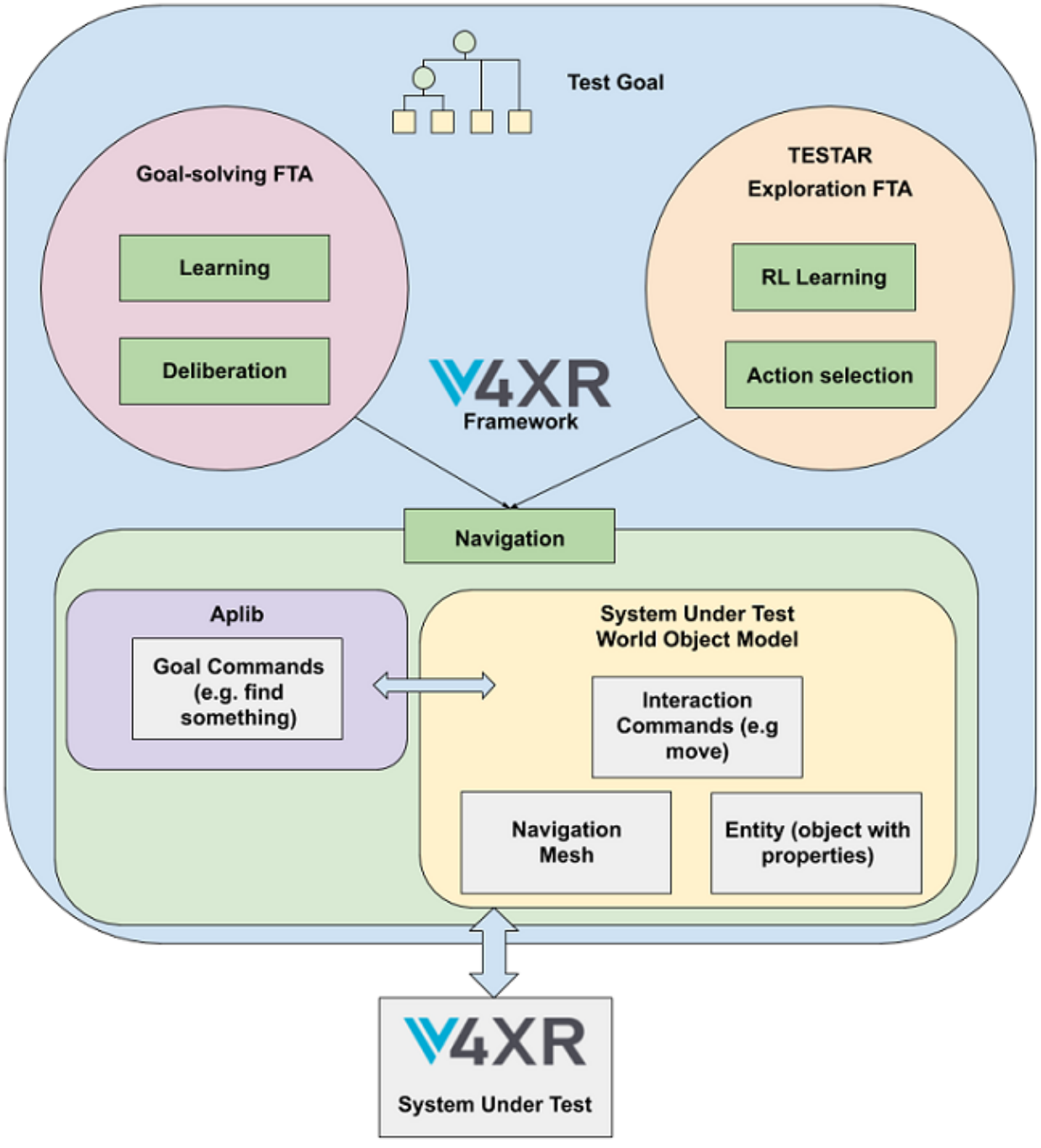

***Figure 2**: Different types of FTAs in iv4XR.*

2. NAVIGATION

When testing some functionality *F*(*x*) in a more traditional setup such as when *F* is a service, we typically proceed as follows:

- provide an instance of *x* as input,
- invoke *F*, passing to it the input we devised,
- check *F*''s output.

Even in a more complicated setup e.g. when *F* is a functionality of a mobile-app, the workflow is still largely the same: we locate the corresponding widget in the app, and interact on it to trigger *F* and then we check the response of the app.

In the XR setup things are fundamentally different. To test realistically (from the actual user's perspective), "invoking" F often involves **navigating** the XR's virtual (or mixed) world to actually get to a location where *F* can physically be invoked. Similarly, to check its result might require the test agent, simulating a real user, to navigate to a different location where *F*'s results can be witnessed. Since a virtual world is often made to simulate physical worlds, restrictions in physical worlds often apply in virtual worlds as well, e.g. it is not possible for an agent to walk through a solid door, or to jump over a 3m fence. Moreover, most virtual worlds are very fine grained. The 3D space that they define is in principle infinite. No automated testing approach can deal with this headlong.

We therefore adopt an approach known from computer games where developers provide a finite approximation of the navigable parts of a virtual world. Typically this is provided as a graph of adjacent triangles called a **navigation mesh**; see the Figure below. This mesh can often be extracted automatically from the utilised graphic engine of the XR system. When this mesh is given, auto-navigating from A to B can be solved with standard path finding algorithms such as A*. Such capability is supported by our FTAs.

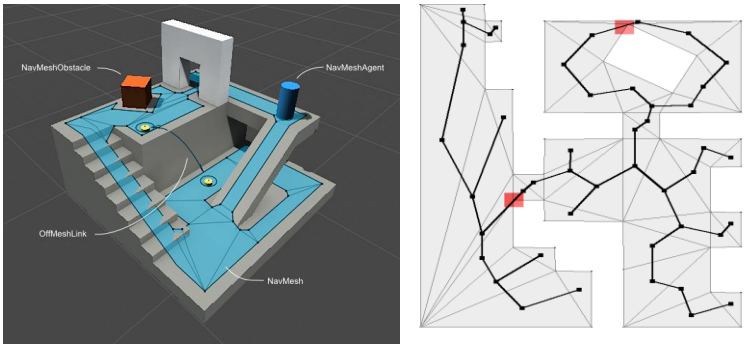

***Figure 3**: To the left we see a 3D world. The blue colored part indicates the part of this world that is physically navigable by a virtual agent. This navigable surface can then be fragmented into connected triangles as also shown in the left-picture. The picture on the right shows an example of such a surface. Connected triangles form a graph (illustrated by thick lines). An agent can navigate from one triangle to another by traversing this graph.*

3. FTA'S: GOAL SOLVING

Having automated the navigation, testers are subsequently enabled to focus on formulating **goal-solving** strategies at a more logical level and to use an FTA. These intelligent agents use the framework developed in WP2 and add a new set of abilities and modules that produce solutions to solve test goals.

For example, imagine again a small level in Lab Recruits shown in Figure 3 below. Suppose we want to verify whether interacting with an in-game *button3* is expected to open *door2*. The high level testing task would be as in the previous sentence, so that is also the logical level where we would like to be able to formulate our test strategies. Concretely though, the FTA that would execute the task will have to navigate to *button3*'s location in the world to interact with it, and then it needs to navigate towards *door2*'s location to at least be able to see it to confirm its state. The automated navigation part should be hidden from the testers (they should not concern themselves with solving such a problem). Note that automating "spatial navigation" may entail more than just pathfinding mentioned in the previous section. e.g. the virtual agent may get stuck in some sticking corner, due to some non-determinism in the XR system, or due to inaccuracy in the navigation mesh. This needs to be solved by the navigation layer as well, so that testers can focus on the logical level.

Now back at the logical level, reaching the *button3* actually involves a series of interactions (e.g. *door0* and *door1* should be opened first). We prefer of course that the FTA can, on its own, find the right sequence of interactions, rather than that the tester having to spell them out. The latter would not only cost more labour, but also results in a less robust test when later the developers decide to change the world. Goal solving FTA's have skills such as deliberation and learning to observe the World Object Model (WOM), reason and decide the immediate strategy towards solving the given goal.

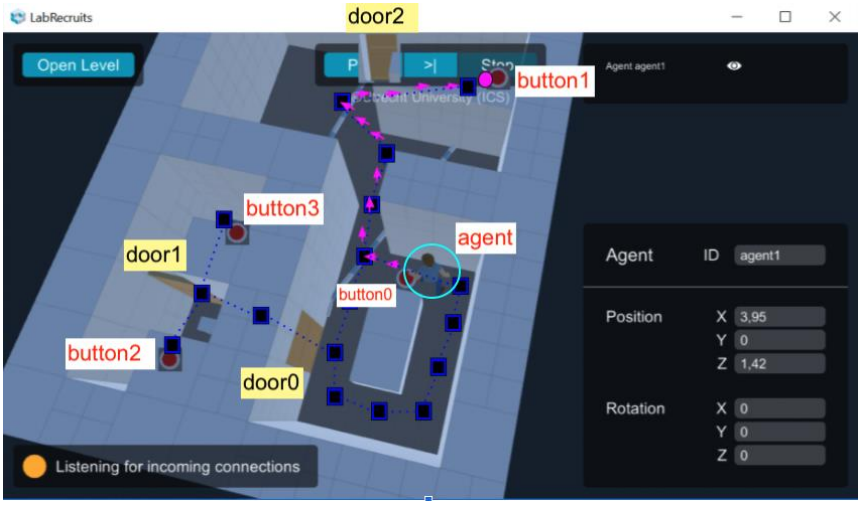

***Figure 4**: a small level in Lab Recruits, with 4 buttons and three doors. The starting position of the agent is shown in the blue circle.*

Since a goal may contain sub-goals which have to be solved first and in a certain order, a specific deliberation module keeps track of the current primitive sub-goal G (a goal without no further sub-goal) that it has to solve. This module maintains a dynamic collection of strategies to deal with various typical situations (e.g. when the agent becomes stuck or to counter a hostile entity) and decides which strategy to be tried to solve G. Such a strategy can simply be to invoke the navigation module to move the agent to a new position where the interactions required by G are within the agent's immediate reach. Or, it anticipates that solving G requires other interactions, which it then adds as new sub-goals of G and proceeds recursively to handle these sub-goals first.

To make strategies able to improve themselves over time, they will internally be represented as some data structure that can be improved by a learning algorithm implemented in the learning module.

3.1. LabRecruits Agents

LabRecruits FTA's are the direct evolution of plain agents to which navigation and deliberation abilities have been added for the resolution of goals. These agents accept goals expressing testing tasks, formulated using the Test Specification Language (TSL) developed in Task 3.1 (see D3.1), a declarative expression that allows testers/developers to define testing tasks. A simple example is shown below, useful for testing the Lab Recruits level shown in Figure 4:

```
SEQ(entityInteracted("button1"),
    entityStateRefreshed("door1"),
    entityInvariantChecked(...,
      "door1",
      "door1 should be open",
      (WorldEntity e) -> e.getBooleanProperty("isOpen"))
   )
```

For example, entityInteracted("button1") constructs a goal that would be solved if the test agent manages to interact with button1, and entityStateRefreshed("door1") constructs a goal that would be solved if the memorized state of *door1* has the same timestamp as the current time (in other words, when the state is up to date).

The whole SEQ construct above formulates a testing task that instructs the agent to interact with *button1*, then it needs to update its observation on door1, and then to check the orange assertion (expressed as a predicate) on *door1*. The predicate is essentially this:

e → e.isOpen

Which means that we are checking whether door1 is open.
The strategies that solve these goals are not shown. They indeed need to be created and indeed need to involve navigation, so that the agent can find e.g. the purple path shown Figure 4 that

leads it to *button1*. The actual programming of the needed strategies only need to be done once. After that we can keep using the strategies. At the top-level, testers only need to specify tasks at the goal-level as shown above.

4. FTA: EXPLORATION

4.1. TESTAR TOOL

TESTAR[2] is an open source tool for automated testing with a scriptless approach, that emerged to test desktop applications through the Graphical User Interface (GUI). The TESTAR approach can be easily extended to test any event-based system under test (SUT), as long as the tool knows how to interact with the environment. The underlying principle of TESTAR is: generate test sequences of (state,action)-pairs by connecting to the SUT in its initial state and continuously select an action to bring the SUT in another state and check oracles.

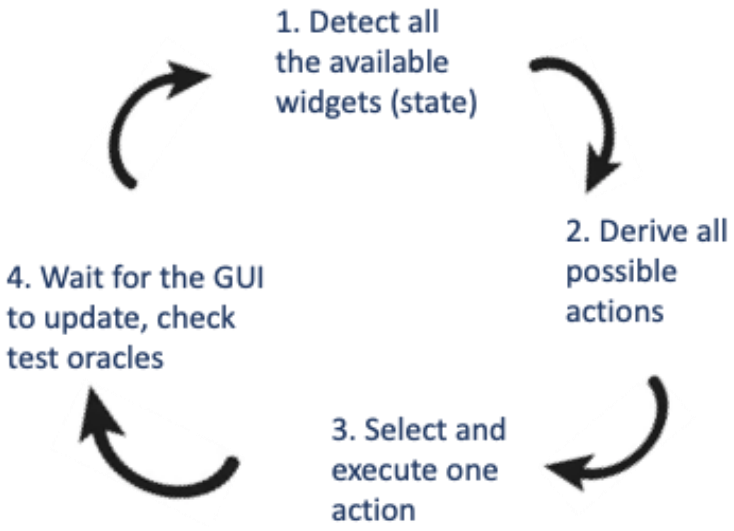

***Figure 5**: TESTAR execution flow.*

The information about the states and widgets that exist in the SUT can be extracted by integrating accessibility Application Programming Interfaces (APIs) (e.g. UIAAutomation for Windows) or automation frameworks (e.g. Selenium Webdriver). An integration with the first version of the iv4xr Framework has been developed that allows extracting information from virtual entities and execute actions that send instructions back to the Framework to launch commands or solve strategies. Essentially, the integration is achieved by treating the World Object Model (WOM) maintained by iv4XR test agent as if it describes a set of UI widgets, exposing their state and possible interactions on them. Figure 6 shows how the state of the Lab Recruits game would look like to TESTAR.

[2] TESTAR, Test your system from the GUI: https://testar.org/

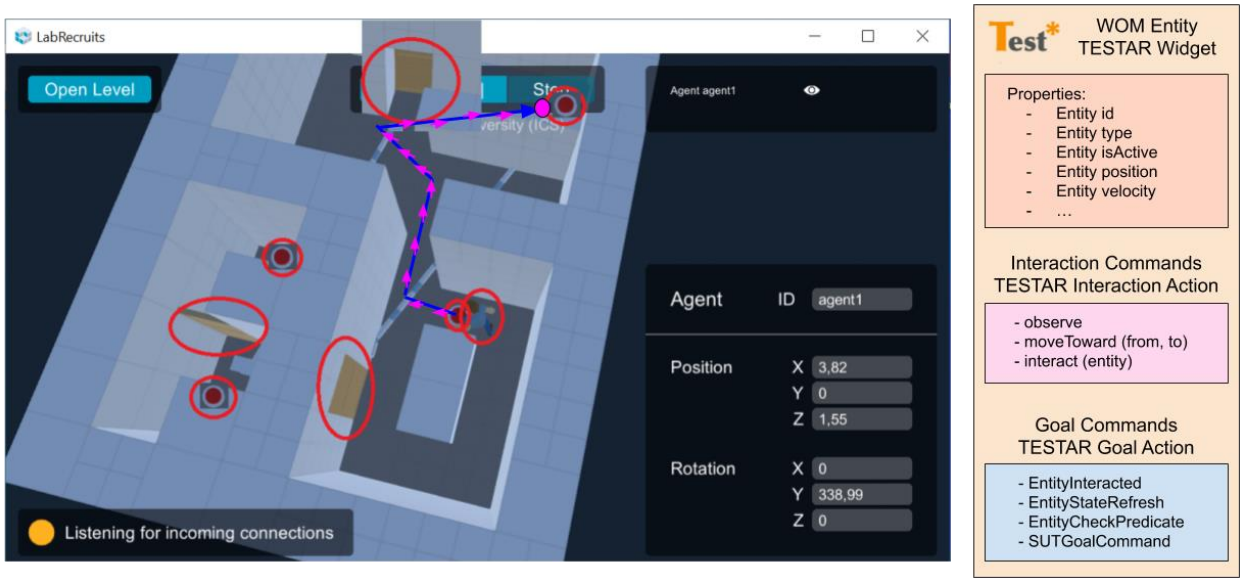

***Figure 6**: TESTAR WOM Integration.*

4.2. TESTAR FTA

TESTAR, and its possible decision making while discovering what happens in the system to select and execute an action, allows this tool to play the role of an exploring FTA. While LabRecruits agents are focused on achieving the subgoals sequence by sequence by navigating to the entities, the objective of TESTAR agent is to carry out a non-sequential exploration of the environment, trying to learn which are potentially interesting actions while verifying and validating the behavior of the system.

To learn about the state of the entities in the environment and their changes after executing actions, TESTAR is capable of inferring a State Model that contains the information about the behavior of the XR system (see Figure 7). Depending on the objectives desired by the partners regarding the exploration and testing of their SUTs (increase coverage, induce changes, reach goals, etc ...), different rewards will be assigned to the actions and/or states of the inferred model, which together with different Reinforcement Learning (RL) algorithms will allow us to improve the selection of potentially interesting actions. This functionality will be officially introduced in the second prototype (D3.3).

In addition, trying to improve the decision making and reduce possible RL training costs, TESTAR is capable of being launched in listening mode, which allows listening LabRecruits agents actions and inferring models of their navigation sequences and strategic deliberation.

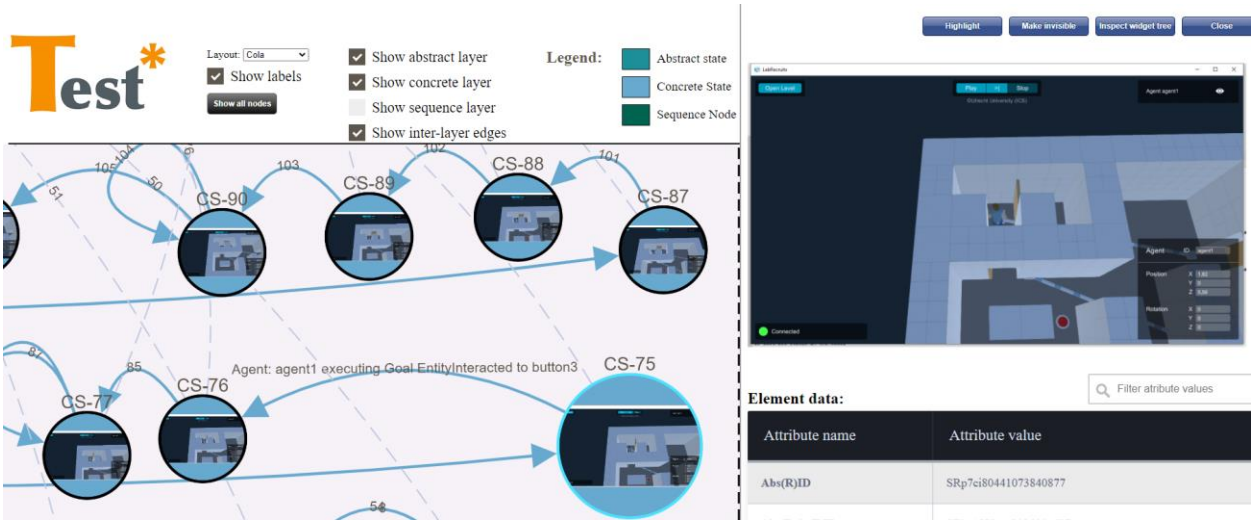

***Figure 7**: TESTAR Inferred State Model.*

5. OUTPUTS

1. Prototypes:
    a. ***Goal-solving*** Functional Test Agent for LabRecruits as part of the iv4xr Framework: https://github.com/iv4xr-project/iv4xrDemo
    b. ***Exploration*** Functional Test Agent integrating TESTAR: https://github.com/iv4xr-project/TESTAR_iv4xr

2. Videos:
    a. TESTAR Agent testing LabRecruits with State Model inference: https://www.youtube.com/watch?v=sZkVwX9m8_s

3. Papers:
    a. *Navigation and Exploration in 3D-Game Automated Play Testing*, by Prasetya, Volhol, Tanis, et al., in the proceedings of the 11th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation, co-located at the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2020. Pdf: https://arxiv.org/pdf/2009.07015