Architecture design

WhySoSerious

April 29, 2016

Contents

1 Introduction

This document provides a high-level description of our final product and the systems it uses. It divides the product into systems and subsystems which are explained in section 2.

1.1 Design goals

Performance

Our virtual human should not be overwhelmed with all the information it gets from the environment. When this does happen it is possible that virtual human is busy calculating actions which should have been executed a half minute ago (as example). The agent is then acting on old information which can be imprecise. There for our agent must perform well and react fast.

Availability

It is important to deliver a working system every week so Tygron can see what our plans are for the next sprint and what we implemented this sprint. If Tygron does not like a new feature we can remove it instead of spending a lot of useful time extending this feature.

Code quality

The software has to be properly coded. The code should follow the general code guidelines, and because we are programming in GOAL, we should divide our code into coherent modules with descriptive names. It is also important to create clear documentation and have comments at the appropriate places between pieces of code.

Reliability

It is very important to continually test our code to fix bugs and unwanted behaviour. We need unit tests, integration tests and regression tests. This is the most effective way to improve the reliability of our program.

2 Software architecture views

2.1 Subsystem decomposition

A user accessing the Tygron engine does so with client software on his or her computer. This client communicates with the Tygron servers where all of the processing is done. Our agent will communicate with the simulation via the EIS connector. This connector gives the agent percepts which it will process into beliefs. Based on these our agent will return actions to the EIS connector sending them through the Tygron API directly into the Tygron servers.

- Servers

  The servers are managed by Tygron and this is where the simulation takes place, the clients use this data to visualise the simulation

- Tygron API

  The API is part of the server software, and allows a user to write their own application to send instructions and/or display data[1]. The EIS connector uses this API to connect our goal agent to the server.

- EIS connector

  This is the connection between the Tygron API and the GOAL language which we use to code our agent.

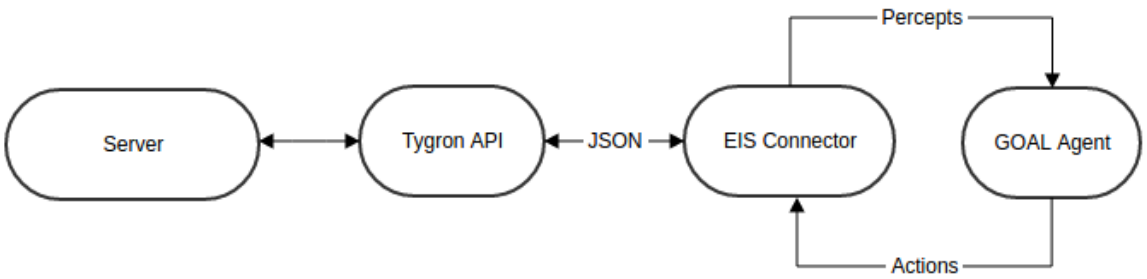
- GOAL agent

  The GOAL agent is our final product, it simulates a human stakeholder in the Tygron game

[1] `http://support.tygron.com/wiki/REST_API`

2.2 Hardware/software mapping

The Tygron API is run on Tygron's servers, while the EIS connector and the GOAL agent are both on the user's computer. This graph illustrates the connection between all parts of the process.



A graph of the interaction between the server and the agent.

2.3 Persistent data management

As our software will be a real-time bot, there will be no persistent data on our side. All persistent data is the game state which will be kept by the server at all times.

2.4 Concurrency

Bots will have to be able to communicate with each other. This means that we will have to work together with the other teams in our context to develop a language which all of the bots can use and understand. Agents should also remember their actions so they won't get stuck in a loop.

3 Glossary

- API

  API means application programming interface. Programs use an API to communicate with external software, usually servers.

- GOAL

  A programming language created at the TU Delft that enables effective programming of logical agents.

- Tygron

  Tygron is the company that developed the Tygron Engine[2], which we use in our project.

- Tygron Engine

  A program run on the Tygron server that enables multi-user simulated city planning with customizable roles.

- Virtual human

  Virtual human is another term for the agent that we are creating.

- Unit test

  A unit test tests a single component to verify its behaviour.

- Integration test

  An integration test tests whether multiple components are able to cooperate as expected.

- Regression test

  A regression test tests whether new bugs have been introduced due to updated code.

[2] `http://www.tygron.com`