Tutorial

Daniel Puschmann

17. Mai 2011

1 VISP

This document will first introduce you to the VISP-framework which you will use to implement the final state machine representing the whole behavior of the agent.

1.1 Final state machine

The class MachineImpl represents the final state machine (**fsm**). It includes a static Hashmap as an attribute in which you can add, get and change the value of „global variables“ which will be visible by all the nodes contained in the **fsm**. It also contains a nonstatic Hashmap for creating channels and getting the input from the channels. The channels in combinations with Events (see section 2.4 Event) are used to trigger state changes in the **fsm**.
A **fsm** contains nodes representing the current state of the machine and edges defining the transitions between the nodes. One node in the **fsm** must be set as the startnode through the setStartNode method of the **fsm**.

1.2 Nodes

There exist two types of nodes, supernodes and nodes. NodeImpl represents a node in the final state machine to which actions can be added. SuperNodeImpl is used to group multiple nodes together. Like in the **fsm** one of the nodes needs to be set as the startnode.

1.3 Edges

An edge represents one possible transition between two states, to which actions can be added. There are six different types of edges:

- **InterruptEdge**: defines an edge between two supernodes. This transition is possible from all nodes in the current supernode to the startnode of the target supernode. It is triggered by the condition which is set through the constructor when creating the InterruptEdge .

- **EpsilonEdge**: defines an epsilon edge between two nodes, a transition which is triggered immediately without a condition.
- **ConditionalEdge**: defines an edge with an condition. If the condition is met in a given time the transition to the targeted node is executed.
- **ConditionalElseEdge**: if the condition of the ConditionalEdge is not met in the given time this transition is executed.
- **ProbabilisticEdge**: defines an edge with a given probability. The probabilities of all ProbabilisticEdges in one node combined should be 1. It is possible to neglect the ConditionalElseEdge and instead add ProbabilisticEdges to a node containing an ConditionalEdge if you don't want always the same transition in the else case.
- **RandomWaitEdge**: waits for a random amount of time between a given minimum and maximum before the transition to the target node is executed.

1.4 Conditions

Conditions are used in the ConditionalEdges. There exist ConditionAnd, ConditionEquals and ConditionOr. Conditions are predicates which can be True or False, you can use conditions to trigger the behavior of ConditionalEdges.

2 Getting started with your project

The first thing to do is creating a new package, e.g. de.dfki.embots.framework.ui.weathercast in which your classes will be written.
This section shows you how to implement your own **fsm** controlled behavior component into the semaine API.

2.1 Component

The EMBOTS framework utilizes the semaine API, an open source framework for building emotion-oriented systems in which a message-oriented middleware (ActiveMQ) is used for communication between components. So to integrate your project in the EMBOTS framework you need to write a class which extends component and overrides the act and/or the react method. The act method is used for acting without first receiving a message and is called at configurable intervals, while the react method is only triggered if a message was sent to the component. This class acts as your main class, here you create your **fsm**, initialize your strategies (see section 2.2 Strategy), the senders and receivers for semaine messages and the simulator. You can orientate on the class Gaze in de.dfki.embots.framework.ui.eyetracking to see how its done.

2.2 Strategy

A strategy defines a supernode in the **fsm**. You can implement the interface GazeStrategy in de.dfki.embots.framework.ui.eyetracking or write a new interface if you feel that more or other methods are needed than provided by the interface. Strategies define the overall behavior of the agents, that is to say they specify which behavior is triggered when. Take a look at DominantGazeStrategy and ShyGazeStrategy for example strategies.

2.3 Behavior

Defines a certain behavior of the agent by generating EMBRScript using the current data of the eyetracker (see LogData in de.dfki.carmina.eyeTrackerLogger.dataProcessor or LogDataSingleton in the eyetracking package). You can implement the interface GazeBehavior for writing these classes. Several behaviors exist in the eyetracking package where you can see how the EMBRScript is generated.

2.4 Event

Some of the transitions in the **fsm** are eventdriven. An event only needs to override the hashcode (best with a constant hashfunction) and the equals method. When the given event is happening you simply need to put it in a beforehand created channel of your **fsm**. The corresponding edges need to check if the event is in the channel via ConditionEquals (see section 1.4 Conditions). Examples for events are UserStartsLookingAwayEvent and UserStartsLookingAtEvent in the eyetracking package.

2.5 Action

Actions can be added to nodes or edges. These will be triggered if the corresponding node is reached or the corresponding edge is traversed. For your purpose GenerateScriptAction and GenerateBMLAction in de.dfki.embots.framework.ui.eyetracking.actions should suffice, but feel free to write new actions if needed.

2.6 Simulator

I wrote a new simpler Simulator without any buttons or the possibility to change if the user is looking at the agent. It is called SimpleGazeInputWindow and is located in de.dfki.embot.framework.gaze.eyetrackersim. To use it you first need the change all occurences of Gaze to your component and then initialize it in your component. It produces LogData like the eyetracker, the fields x_gazepos_lefteye, y_gazepos_lefteye, x_gazepos_righteye and y_gazepos_righteye are the screencoordinates of the users gaze.

3 Running your project

To test your project you need to copy and rename the embots.bat file (or embots.sh on linux/mac) and create a xml configurations file with the same name as your .bat/.sh-file in the config folder, where you specify which components shall be used. Just take a look at the given config-files for orientation.