Tutorial Project I

This first project involves a simple agent in the “Button” environment. This environment randomly displays a red square, a blue circle, or is blank. It also has two buttons: *button 1*, which should be pressed *by the agent* when a red square appears, and *button 2*, which should be pressed when a blue circle appears. Note that the buttons that will appear in the GUI are merely visualizations and *pressing them has no effect*.

This project was inspired by agents developed for the purpose of replicating human data from a reaction time experiment; however, it was mainly chosen for this tutorial because of its simplicity. The perceptual processes used in this project are simplified for pedagogical reasons. As such they do not represent a complete implementation of the perceptual processes of the current LIDA model.

Setup

1. If you haven’t already, install the Java JDK and NetBeans 6.9 or above
2. Unzip the tutorial distribution to a folder on your computer
3. With NetBeans running select **File → Open Project…**
4. Browse to the ‘tutorialProjects’ directory in the distribution folder and select **basicAgentExercises**
5. Click open project
6. Right-click on the project in the Project view and select **Set as Main Project**

Basic Agent Exercise 0

Goals

- Explore a functional, basic agent

Preparation

- Open the **basicAgentExercises** project, find the 'Source Packages' folder and open it

Task 1

Explore structure of the project using the Projects view in NetBeans. *Briefly* browse through the packages of the project looking for the following Java classes:

myagent.Run	Runs the application
myagent.modules.ButtonEnvironment	Button Environment implementation
myagent.modules.ButtonSensoryMemory	Agent's SensoryMemory implementation
myagent.featuredetectors.ColorDetector	Color feature detector
myagent.featuredetectors.ShapeDetector	Shape feature detector

Now switch to the Files view (tab just next to the Projects view) in NetBeans. Browse to the 'configs' folder of the **basicAgentExercises** project. Find the following config files, which you will be working with throughout these exercises:

lidaConfig.properties	Main configuration file that serves as a directory for the other configuration files needed to build an agent application.
basicAgent.xml	Agent declaration file. This file defines the agent's architecture including the modules and processes the agent application will use.
factoryData.xml	Definition of the elements that can be obtained from the ElementFactory. This file defines several Node, Link, Strategy, and Task types. These element types are referenced *by name*, e.g. "defaultDecay", in the agent declaration file. (These same

	names are also used by several framework classes to obtain new elements.)
guiPanels.properties	Configuration file for the GuiPanels used by the framework's GUI. These panels can be added, removed, and/or customized using this file.

Note: None of these file *names* are mandatory. `lidaConfig.properties` is the default name of the main configuration file. However, any name could be used provided it appears as the first command line argument of the AgentStarter.java class. The rest of the file names in this table are defined in the `lidaConfig.properties` file.

Basic Agent Exercise 1

Goals

- Become familiarized with the framework's GUI
- Understand the tool bar and its functionality including start/pause, ticks, step mode, and tick duration
- Become familiar with the **Button Environment**
- Learn about the **Logging** and **ConfigurationFiles** GuiPanels

Preparation

- If you haven't already, open the `basicAgentExercises` project
- Right-click, select **Set as Main Project**
- Now clicking the Run button (or pressing F6) will run Run.java in this project

Instructions

Task 1

Find the tool bar section of the GUI. It appears directly below the File, Panels, and Help menus. Press the 'Start/Pause' button a few times to toggle the running of the simulation. The run status appears just to the right of this button. To the right of this is the current tick text field. Notice how it advances as the simulation runs.

Task 2

Next find the 'Step Mode' button and click it. The button will darken signifying that the system is now in "Step Mode". In this mode the application can be run for a specified amount of ticks at a time. Try it now; enter **`100`** in the tick text field that appears to the right. Now click 'Run ticks' and observe that the application runs 100 ticks and stops. Each time you press 'Run ticks' the application will run for a period of 100 ticks.

Task 3

Leave step mode by clicking the 'Step mode' button again. Find the tick duration control in the far right of the tool bar. With the system running, use the arrows to adjust the tick duration to 20 and then to 0. Notice how the speed of the application changes.

Task 4:

With the application running find the "Logging" GuiPanel at the bottom. Each line in this panel is a logger entry. The first column is the number of the log entry; the second is the tick at the time of the log; next, the level of severity; then the name of the logger; finally, the message.

Find the first drop down list called "Logger". This controls the logger whose logs are displayed in this panel. In this menu, select **myagent.modules.ButtonEnvironment.** In the second drop down list, "Level", select a level of FINEST. This controls the level of logs that are created by this particular logger (see java logger API for details). Notice now that there are additional logs from the environment. A large volume of logs may slow down the speed of the application. Each time the application is run the logging levels are returned to their default settings.

Task 5

Study the "ConfigurationFiles" GuiPanel which appears as a Tab in the same section as the Logging Panel. This panel displays the configuration files currently used by the application. Its table displays the content of the `LidaConfig.properties` file.

Basic Agent Exercise 2

Goals

- Become familiar with the functionality of a feature detector
- Become familiar with the functionality of a running attention codelet
- Study several GuiPanels in the GUI, including "PAM Table", "Perceptual Buffer" and "GlobalWorkspace"
- Customize the GUI by adding a GuiPanel to visualize the contents of the Workspace's Current Situational Model

Preparation

- Close the running application by selecting **File → Exit**
- Switch to the Files view in Netbeans, then open the 'config' folder in basicAgentExercises
- Open the lidaConfig.properties file and set the `lida.agentdata` property to **`configs/basicAgent_ex2.xml`**. Save the file. The agent created by this new configuration file will have some missing parts, which we will be filling in for this and the following exercises
- Run the application as before. Toggle the Start/Pause button. Note that this agent does not perform "Press button 2" action

Instructions

Task 1

In the tool bar, set the *tick duration* value to `20`. Make sure the application is not paused. Now go to the "PAM table" GuiPanel. This table shows what the agent currently perceives as a result of its recent sensory input. Observe the nodes 'square' and 'red'; their activation values should be changing. Their activation is 1.0 when a red square as sensed, and decays to 0.0 otherwise. If there isn't a red square present in the environment, you may need to wait a few moments before the values change. On the other hand, the activation of the 'blue' and 'circle' nodes should NOT be changing — this agent doesn't have a feature detector for these PAM nodes yet. In the next exercise, you will add feature detectors to the agent to allow it to perceive the blue circles.

Task 2

Open the "PerceptualBuffer" GuiPanel. Observe how it changes over time. (Scrolling with your mouse or trackpad zooms in and out. Also the window area can be clicked and dragged.) What causes the panel to become empty? (Notice that the nodes here are not *exactly* the same ones listed in the PAM Graph and Table, but rather instantiations of some of them.) Mouse-over the nodes that appear here and their values will appear the tooltip (increase the tick duration to make this easier). Note that these tooltip values are not updated in real-time.

STUDY QUESTION 1.1: What module of the LIDA Model is the Perceptual Buffer in? Where do the nodes in this buffer come from? How do they differ from the nodes in other modules?

Task 3

Open the "GlobalWorkspace" GuiPanel and observe what is displayed here: In the top half of this panel is a table that displays the Coalitions currently in the GlobalWorkspace and their attributes. In the bottom half of this panel there is a table that displays the recent history (with the top being most recent) of winning coalitions and their data.

STUDY QUESTION 1.2: Where do the coalitions in the Global Workspace come from? Where do they go? Think about how the answer to these questions depends on whether we are discussing the LIDA Model or a specific agent implemented using the Framework.

Task 4

Quit the application. Now we will add a GuiPanel to the GUI to display the contents of the CurrentSituationalModel (a submodule of the Workspace Module). You will add a single line to the 'guiPanels.properties' file. Model it after the line that defines the `perceptualBufferGraph`. A comment appears in this file that explains the structure of variable declarations. It should look like this:

```
#name = panelTitle, className, Position, tabOrder, refreshAfterLoad, parameters
```

Make a copy of the perceptualBufferGraph panel declaration. Now change the copied line; the name to the left of the equal sign can be changed to **`csm`**, panelTitle to "**`CSM`**", the className will be the same, the position will be the same, tabOrder will be **`7`**, and most importantly, the parameter at the end should be: **`Workspace.CurrentSituationalModel.`**

Save the file, run the application, start the simulation, and adjust the tick duration if necessary. The new CSM GuiPanel should appear among the tabs in the right section. If you used a tab order of 7, it should appear as the last tab. Click the CSM tab and verify that nodes appear and disappear in a manner similar to the Perceptual Buffer panel.

STUDY QUESTION 1.3: How are the contents of the Current Situational Model related to the contents of the Perceptual Buffer? to the contents of PAM? Think about how the answer to these questions depends on whether we are discussing the LIDA Model, or a specific agent implemented in the Framework.

Basic Agent Exercise 3

Goals

- Modify the agent declaration file to enhance the agent's functionality. NOTE: Every time the agent declaration file is changed, the application must be restarted before the changes will take effect.
- Add a module declaration to the agent xml file
- Add a listener declaration to the agent xml file
- Add feature detector declaration to the agent xml file
- Add an attention codelet declaration to the agent xml file
- Explore the agent xml file

Preparation

- In the `lidaConfig.properties` file set the `lida.agentdata` property to **`configs/basicAgent_ex3.xml`** and save the file. *Don't* run the application.
- Study the diagram below. The dashed parts of the diagram represent the Framework elements that you will be adding to the agent application during this exercise. Note that *not all* modules and elements of the agent are shown here. Also note that this diagram, while similar, is slightly different to that conceptual LIDA model for these modules.

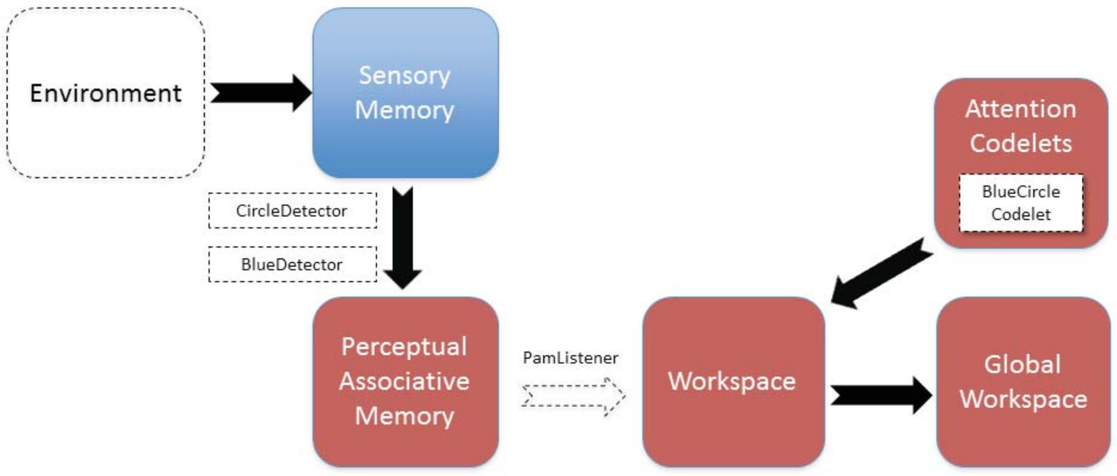


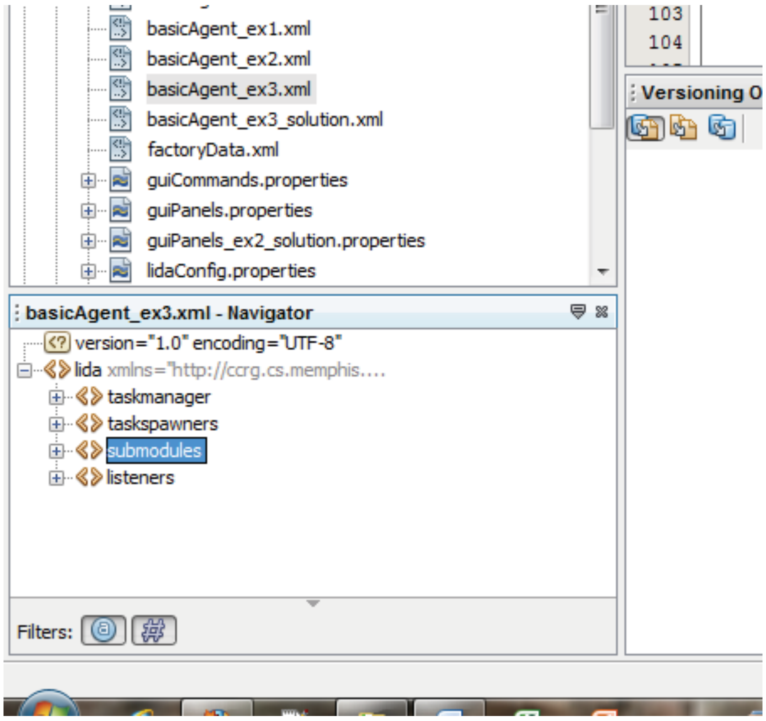
Figure 1: Partial diagram of ButtonAgent's architecture in Exercise 3
Dashed elements will be added during this exercise

Instructions

Task 1

In the 'configs' folder of the project open the `basicAgent_ex3.xml` file. This file has an agent declaration similar to the previous exercise, but is missing some elements (a module and a listener). In this task and those that follow, you will extend this agent's functionality by adding the missing parts as well as some new ones (a feature detector and an attention codelet).

Tip: To easily navigate through the xml files, you can use the Navigator window in NetBeans (see screenshot). You can see the structure of the xml file in this window and navigate to the corresponding tag in the file by just clicking on the tag in the Navigator window.



Task 2

Open the `submodules` tag. Add a module declaration[1] to the agent declaration file nested within the `<submodules>` tag (there are comments indicating where to insert the new declaration). Model the syntax of the new module declaration after the `ActionSelection` module declaration in this section (you can copy this and edit the copy). The declaration must be within a <module></module> tag pair, and should have the following tag values:

[1] We use the term 'declaration' when an element is added to an agent declaration file, and 'definition' when a new element is added to ElementFactory definition file.

name: **`Environment`**

class: **`myagent.modules.ButtonEnvironment`**

Also include the following two `param` tags noting the structure of parameter declarations. They have a name, a type (`int, boolean, string, double`), and a value inside the tag. If not specified, the default type of the parameter will be `string`:

Param1: **`name="height" type="int"`**, tag value is **`10`**

Param2: **`name="width"  type="int"`**, tag value is **`10`**

Finally include another tag as follows:

taskspawner is **`defaultTS`**

Save the file. Xml files can be validated at any time by right-clicking on the xml file body and selecting **Validate XML**. (Alternatively there is a button in the toolbar with two triangles that does the same thing.) Try it now to validate your changes. Look at the output tab below the XML editor window. This tab will have a subtab that reads "XML check." If your code has proper syntax, the XML check subtab will read, "XML validation finished" with no error messages. If there are errors in your syntax, error messages will appear to help you identify the errors. (NOTE: the XML validation will not check the class names or parameter values, only the XML syntax of the file).

Now run the application by pressing the NetBeans run button, and then click the Start/Pause button to begin the simulation. The Button environment should appear running in the GUI. Set the tick duration to 10 or so to slow the simulation down enough so that you can see the activation values changing in the PAM Table GuiPanel. Notice that the "red" and "square" nodes are receiving activation, but not the "blue" and "circle" nodes. This is because feature detectors for "blue" and "circle" have not yet been added to the PAM module. This will be done below in Tasks 4 & 5.

However even though "red" and "square" have feature detectors that activate the appropriate PAM nodes when these features are present in the ButtonEnvironment, these nodes do not appear in the PerceptualBuffer GuiPanel. This is because the connection between PAM and the Workspace has not yet been defined in this agent's declaration file (this connection corresponds to the arrow labeled "Move Percept" in the LIDA Model diagram). Therefore, this agent can perceive red and square, but these nodes never enter the Workspace. In order to make this connection, a `PamListener` must be added to the agent declaration file. This will be done in the next task.

Task 3

Now open the `<listeners>` tag in the agent xml file. Add a new listener declaration at the beginning of this section, modeling it after the other listener declarations. The declaration should have the following tag values

listenertype is **`edu.memphis.ccrg.lida.pam.PamListener`**

modulename is **`PerceptualAssociativeMemory`**

listenername is **`Workspace`**

Save the file. Rerun the application. Now the PerceptualBuffer should display nodes "red" and "square" when the agent perceives a red square in the environment. If these nodes are not appearing in the PerceptualBuffer when there is a red square in the ButtonEnvironment GuiPanel, carefully check the declaration file and make sure that the new listener declaration has the correct information. Also note that nodes for "blue" and "circle" do not appear in the PerceptualBuffer, even when a blue circle is present in the environment. As mentioned above, this is because there are no feature detectors in PAM for these features yet.

Task 4

Find the declaration of the `PerceptualAssociativeMemory` module. Now find the `<initialtasks>` tag in this module declaration. Find the `redDetector` task declaration. Add a new task declaration modeled after `redDetector` task.

Set its tag values as follows:

name: **`blueDetector`**

tasktype: **`ColorDetector`**

ticksperrun: **`3`**

Set its parameters as follows:

Parameter 1

`name="color" type="int"`, tag value is **`-16776961`**

Parameter 2

`name="node" type="string"`, tag value is **`blue`**

Task 5

Add another task declaration for a detection algorithm that detects a circle.

Set its tag values as follows:

name: **`circleDetector`**

tasktype: **`ShapeDetector`**

ticksperrun: **`3`**

Set its parameters as follows:

Parameter 1

`name="area" type="int"`, tag value is **`31`**

Parameter 2

`name="backgroundColor" type="int"`, tag value is **`-1`**

Parameter 3

`name="node" type="string"`, tag value is **`circle`**

Save the file, run the application, start the simulation, and adjust the tick duration if necessary. Now view the Perceptual Buffer GuiPanel. Notice that the *blue* and *circle* nodes appear now. However, the agent still does not 'press button 2' in response to blue circles. This is because there is no attention codelet to add these nodes (as a Coalition) to the `GlobalWorkspace`.

Task 6

Go to the `AttentionModule` declaration in the agent xml file. Find its `<initialtasks>` tag. Add another task declaration similar to the `RedSquareCodelet`. It should have the following tag values:

name: **`blueCircleCodelet`**

tasktype: **`BasicAttentionCodelet`**

ticksperrun: **`5`**

Set its parameters as follows:

Parameter 1

`name="nodes" type="string"`, tag value is **`blue,circle`**

Parameter 2

`name="refractoryPeriod" type="int"` tag value is **`30`**

Parameter 3

`name="initialActivation" type="double"` tag value is **`1.0`**

Save the file. At this point run the application again and the agent should press button 2 in response to blue circles.

Optional task

Those interested may examine the Java classes for the simple feature detectors used in this project. They are located in the package `myagent.featuredetectors`

Advanced Exercises

Advanced Exercise 1

Logging & GUI Configuration. The LIDA Framework can be run without the GUI. In order to do so go to the `lidaConfig.properties` file and change the property `lida.gui.enable` to **`false`**. Now the application will run the agent without the GUI.

Optionally, the Logging can be configured so that the logging handlers will send the logs to a file or to the console. The LIDA framework uses the standard Java logging mechanism; see `java.util.logging` in the standard Java platform for details on this. The framework uses the `logging.properties` file defined in the `lidaConfig.properties` file. You can see an example in the configs folder of this project.

Advanced Exercise 2

This exercise will introduce you to the effects of tuning the `ticksPerRun` parameter.

Launch the application, set tickDuration to 10 before starting the simulation. Observe that several coalitions in the upper part of the Global Workspace panel appear when the red square appears in the environment. Now, exit the application…

- Set the lidaConfig.properties to run the basicAgent.xml agent declaration
- Open the agent xml file
- Go to the `RedSquareCodelet` declaration inside the `Attention` module's `<initialtasks>` tag
- Change `ticksPerRun` to **`50`**
- Change `refractoryPeriod` to **`300`**
- Launch application, set tickDuration to **`10`**. There will be relatively few coalitions in the upper part of the Global workspace panel when the red square appears. There should also be less coalitions being broadcast (appearing in the lower section of the same panel).

Advanced Exercise 3

In this exercise you will be creating a feature detector class and adding it, an attention codelet, and a scheme to the agent xml declaration file.

- Create your own feature detector to detect when the screen is white. Hint: copy ideas from `ShapeFeatureDetector`
- Add this new feature detector in the `factoryData.xml` as another *task* in the `<tasks>` section
- Find the 'nodes' parameter in the PerceptualAssociativeMemory module declaration. Add another node here for `empty`
- In `basicAgent.xml` define a task for attention codelet with a parameter for the `empty` node
- Create a new scheme declaration in `ProceduralMemory` following the format of the existing schemes. The action of scheme can be **`action.releasePress`**
- In the `SensoryMotorMemory` module declaration, uncomment the third action-algorithm association, a parameter named `smm.3`

Now when the agent is run it will detect the empty environment and perform the releasePress action in response. This action releases any button that is currently pressed.

Conclusion

This concludes the exercises for the **basicAgentExercises** project. In these exercises we have focused on these topics:

- The framework's GUI including the tool bar's basic functionality and several GuiPanels displaying the internal state of the agent.
- GuiPanels for logging and for configuration files
- The agent xml declaration file and how to modify it including adding a module, a listener, feature detectors, and attention codelets.
- Functionality and definition of feature detectors and attention codelets
- Adding a GuiPanel to the framework's GUI