

# LIDA Tutorial Exercises

Version 1.0

Cognitive Computing Research Group, CCRG

The University of Memphis

Authors: Javier Snaider, Ryan McCall

Reviewer: Steve Strain

## Introduction

These tutorial exercises are intended to be part of a larger tutorial that includes an introduction to both the LIDA model of cognition and the LIDA software framework. The exercises have been divided into two main projects titled BasicAgent and ALifeAgent. The exercises introduce the participant to the framework by demonstrating several of its main features, the various ways it can be customized, and the use of its GUI. Connections to the LIDA cognitive model are made throughout. Several advanced take-home exercises are also provided.

This tutorial is a brief introduction to the capabilities of the LIDA Model and its software Framework. For further information visit: <http://ccrg.cs.memphis.edu/> and don't hesitate to contact us at [ccrg@cs.memphis.edu](mailto:ccrg@cs.memphis.edu). We hope you enjoy this tutorial as much as we did preparing it.

Javier and Ryan

## Tutorial Distribution Contents

### Contents used for the tutorial exercises

- |                                |                                                                                       |
|--------------------------------|---------------------------------------------------------------------------------------|
| 1. readme.txt                  | explanation of the distribution contents                                              |
| 2. LIDA-Tutorial-Exercises.pdf | this pdf                                                                              |
| 3. tutorialProjects            | folder of <i>ready-made</i> NetBeans projects specifically for the tutorial exercises |

### Contents to create your own project (see Appendix)

- |                                           |                                              |
|-------------------------------------------|----------------------------------------------|
| 4. LIDA-framework-non-commercial-v1.0.pdf | LIDA Framework non-commercial license        |
| 5. lida-framework-v1.1b.jar               | LIDA framework jar file                      |
| 6. lida-framework-v1.1b-doc.zip           | archive of framework's javadoc               |
| 7. lida-framework-v1.1b-src.zip           | archive of framework's source files          |
| 8. libs                                   | directory of jars required by LIDA framework |

## 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 <i>by name</i> , 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 is 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.*

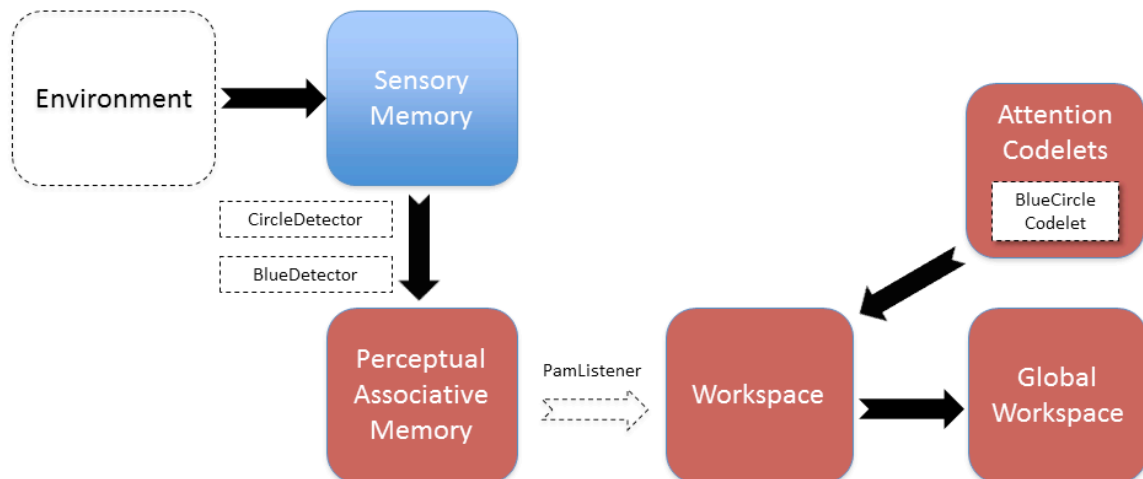
## Basic Agent Exercise 3

### 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

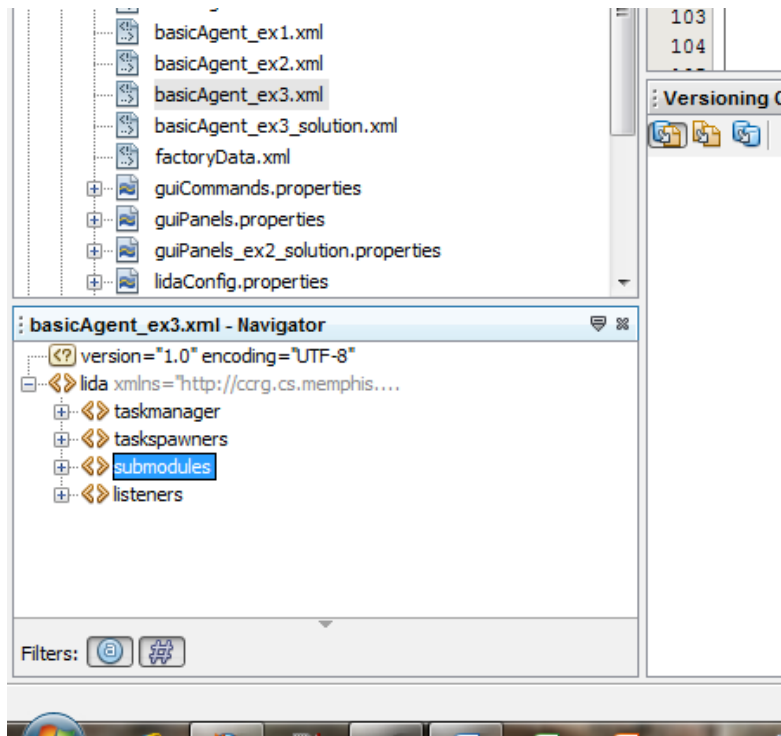


## Instructions

### 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<sup>1</sup> 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:

---

<sup>1</sup> 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.ccrq.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



## Tutorial Project II





This tutorial project involves an agent, “our fearless hero”, in the “Hamburger Jungle” environment. This is a grid world environment where locations are represented as discrete cells. In each cell 0 or more objects can exist, however, cells have a capacity which cannot be exceeded. In this particular implementation there are the following objects: an agent, evil monkeys, trees, rocks, and delicious hamburgers. The evil monkeys move randomly and will try to harm the agent if they are in the same cell. Rocks occupy an entire cell so that no other object may enter that cell. The agent can only sense the objects that are in its current cell and the objects in the cell in front of it in the direction it is facing. It can also sense its own health. The agent can perform the following actions: move forward one cell, turn left, turn right, turn around, eat, and flee (turn and move forward). The agent’s health diminishes a small amount every tick. Also, health is decreased if a monkey attacks the agent; the agent tries to move into a cell with a rock, or tries to move outside of the world’s boundaries. Eating a hamburger increases the agent’s health substantially. These properties of the agent and its environment are summarized in the Table below.

### Project II Environment & Agent

Environment Attributes	<p>Agent, evil monkeys, trees, rocks, hamburgers</p> <p>10 x 10 grid (cells can contain zero or more objects)</p> <p>Monkeys move randomly and cannot enter cells with rocks or trees</p> <p>Agent cannot enter cells with rocks</p>
Sensing	Objects in current and facing cell, health
Actions	Move forward, turn left, turn right, turn around, eat, flee
Agent Health	<p>+ eat hamburger</p> <p>- attacked by monkey, moves into rock, moves out of bounds, time</p>

### Environment Icons

Agent	
Monkey	

Food	
Tree	
Rock	
Multiple Objects	

The perceptual processes used in this project are simplified for pedagogical reasons. They do not represent a full implementation of the perceptual processes of the current LIDA model.

## Setup

- With NetBeans running select **File → Open Project...**
- Browse to the 'tutorialProjects' directory and select **alifeAgentExercises**
- Click open project. Right-click on the project in the Project view and select **Set as Main Project**.

## Agent Exercise 1

### Goals

- Explore a functional ALife agent

### Preparation

- Open the **alifeAgentExercises** project in NetBeans. Right-click on the project in the Project view and select **Set as Main Project**. Run the project (F6).

### Instructions

#### Task 1

Click on an occupied cell in the AlifeEnvironment GuiPanel, which appears in the upper left. Notice that the window just to the right now displays the cell's contents. To the right of that the information about the cell and the objects it contains are displayed. You may have to resize the dividers to see the information. Clicking on an object in the objects list will display its attributes in the table below. Additionally all objects can be individually selected from the drop down list in this section to view object attributes.

*STUDY QUESTION 2.1: Compare the GUI of the alifeAgent to that of the basicAgent. What are the differences, and why are they different?*



## **Task 2**

Inspect the “PAM Graph” GuiPanel. Try pressing the relax button to spread out the nodes and links. Zoom in and out by scrolling with mouse wheel. Also see the “PAM Table” GuiPanel for a different view of PAM’s nodes and links.

## **Task 3**

Start the simulation. The agent navigates the world avoiding the evil monkeys and exploring when its health gets low. It eats the hamburgers when it can. Click on the “Task Queue” GuiPanel and pause the application. Look at the tasks that are scheduled to run at various ticks. Now click on the “Running Tasks” GuiPanel to view a list of all of the currently active tasks and their data.

## ALife Agent Exercise 2

### Goals

- Create a feature detector class
- Add a task definition to the `factoryData.xml` file
- Add a task declaration to the agent xml file.
- Add a new task declaration to the agent xml file based on an existing one.

### Preparation

- From the Files view in NetBeans open the `lidaConfig.properties` file in the 'configs' folder of the **alifeAgentExercises** project. Change the `lida.agentdata` property to **configs/alifeAgent\_ex2.xml**. Save the file.
- Also in the 'configs' folder find and open the `objects.properties` file. Change the quantity value (QTY) for food to 0 (it's just to the right of the first equals sign).

### Instructions

#### Task 0

Run the application and start the simulation. Choose the agent from the drop-down list in the GUI and watch the agent's health as it behaves in its environment. Notice how the agent fails to move when it has low health, i.e., health below 0.33. Quit the application.

#### Task 1

Go to the Projects view in NetBeans and right-click on the icon for the `alifeagent.featuredetectors` package. Now select **New → Java Class...** to create a new class and name it **BadHealthDetector**. It will extend from `BasicDetectionAlgorithm.java` and should detect bad health, i.e., when health is below 0.33. You can simply copy the code of `GoodHealthDetector.java` changing the class name to **BadHealthDetector** and also modifying the `if` statement appropriately.

#### Task 2

Add a **BadHealthDetector** task definition to `factoryData.xml` modeled after the other health detectors. First, open the `factoryData.xml` file. Go to the `<tasks>` section. Find the comment **INSERT YOUR CODE HERE**. Create a new task entry similar that of the `FairHealthDetector`. Give it the name **BadHealthDetector** and set its `<class>` tag value to be the qualified (package + classname) of the class you created in Task 1. The other tag values are exactly the same as the `FairHealthDetector`.

Xml files can be validated at any time by right clicking on the xml file body and selecting **Validate XML**. Validate and save the file.

### Task 3

Add a task to the declaration of the agent's `PerceptualAssociativeMemory` module in `alifeAgent_ex2.xml` file. First go to the `<initialtasks>` tag of the `PerceptualAssociativeMemory` module declaration. Create a new task entry with the name **BadHealthDetector**. The `tasktype` is **BadHealthDetector** (you just created this type in the last task), the `ticksperrun` is **3**, and add a node parameter of type `string` with value **badHealth**. The entry should look similar to the one for `GoodHealthDetector`.

Validate and save the file.

### Task 4

Also in the agent xml file, add a **predatorFrontDetector** to the initial tasks of the `PerceptualAssociativeMemory` module. This task is similar to the previous one. **Hint:** model it after the `predatorOriginDetector`. The `position` parameter is `0` for *origin* and `1` for *front*. Since this is of `tasktype ObjectDetector`, you will not need to create a new Java class for this feature detector.

Save the file and run the application again. The agent now has two new feature detectors, and two new perceptive capabilities. Observe its behavior; it should now detect when it has bad health (see `Perceptual Buffer GuiPanel`) and it should begin moving when it has bad health. Also the agent should run away from the predator when it has a predator in front of it. Previously the agent would only flee if the predator was in the same cell as it.

*STUDY QUESTION 2.2: Why does the agent's behavior change in this way after the new feature detectors were added?*

## ALife Agent Exercise 3

### Goals

- Create and modify attention codelets
- Learn the effects of changing attention codelet parameters

### Preparation

- In `lidaConfig.properties` file change the `lida.agentdata` property to **`configs/alifeAgent_ex3.xml`**. Save the file. Some of the agent's attentional mechanisms have been removed; they will be restored during this exercise.
- In the 'configs' folder find and open the `objects.properties` file. Change the quantity value (QTY) for food back to **10**.

### Instructions

#### Task 0

Run the application and start the simulation. Notice that the agent doesn't react to predators at all.

*STUDY QUESTION 2.3: What are the possible reasons the agent doesn't flee?*

#### Task 1

In this task you will create a **`predatorAttentionCodelet`** task in the `AttentionCodelet` module's initial tasks. Open the `alifeAgent_ex3.xml` file, now (you can use the Netbeans Navigator) go to the `AttentionModule` declaration, go to the `<initialtasks>` tag and create a new task entry similar to `RockAttentionCodelet`.

The name will be **`PredatorAttentionCodelet`**, the `tasktype` will be **`NeighborhoodAttentionCodelet`**, `ticksPerRun` will be **5**; `nodes` will be **`predator`**; `refractoryPeriod` will be **50** and `initialActivation` will be **1.0**.

Save the file and run the application, now the agent will create coalitions (see the Global Workspace Panel) with predator nodes inside the coalitions' content.

*STUDY QUESTION 2.4: How might this affect the agent's cognition and behavior?*

#### Task 2

Open the agent xml file and change the `initialActivation` parameter for the `FoodAttentionCodelet` to **0.01**. (This task declaration is also in the `<initialtasks>` tag of the `AttentionModule` declaration.) Run the application until it starts moving and see if the agent ever reacts to (i.e., eats) any food it comes across. (It most likely won't react.)

*STUDY QUESTION 2.5: How does this parameter change affect the agent's cognition?*

### Task 3

Open the agent xml file and reset the `initialActivation` parameter for the `FoodAttentionCodelet` to **1.0**. Save the file and run the application. Set the *tick duration* to **40** in the GUI to slow the simulation speed.

Look at the upper half of the Global Workspace GuiPanel and notice how often coalitions containing 'goodHealth' appear there. It may help to pause the application and resize the table columns.

Now exit the application and look for the `GoodHealthAttentionCodelet` declaration in the `<initialtask>` tag of the `AttentionModule` module. Set the `refractoryPeriod` parameter to **10**. Run the application again and notice how coalitions containing 'goodHealth' appears. The frequency should have increased.

## ALife Agent Exercise 4

### Goals

- Modify schemes in Procedural Memory
- Use a custom Initializer for the Perceptual Associative Memory module
- Obtain non-default decay strategy from the ElementFactory

### Preparation

- In `lidaConfig.properties` file change the `lida.agentdata` property to `configs/alifeAgent_ex4.xml`. Save the file.

### Instructions

#### Task 0

Run the application and start the simulation. Notice that the agent doesn't move from its cell unless its health drops below 0.66. (However, it will still move if an evil monkey comes close.)

#### Task 1

In the `alifeAgent_ex4.xml` file and find the `ProceduralMemory` module. Inside this module's declaration find the `<param>` tag named `scheme.10b`. Change the action name, which appears after the second pipe (`|`), to `action.moveAgent`. Optionally change the description of the scheme (the first part of the tag value). Save the file.

Run the agent and notice that it now moves even when it has good health.

#### Task 2

Now we will customize the initialization of some of the elements in PAM. The default Initializer for PAM, `BasicPamInitializer`, reads two parameters named "nodes" and "links" and creates Nodes and Links in PAM based on the specifications in these parameters. Nodes and Links created in this way will use default excite and decay strategies. In order to obtain elements with non-default strategies, a custom Initializer is required. Most other modules can also be initialized with a custom Initializer.

Open the `alifeAgent_ex4.xml` xml file and find the `PerceptualAssociativeMemory` module declaration. In this declaration change the `<initializerclass>` tag value to `alifeagent.initializers.CustomPamInitializer`. Save the file. Now go to the Project view in NetBeans and open the `CustomPamInitializer` class in package `alifeagent.initializers`. This class is a partial implementation of a custom initializer for the PAM of an agent in the aLife environment. This initializer will enable the agent to add customized nodes and links to PAM, which is not possible using the default initializer. In its current form, this class does two things:

1. Adds a new Node to PAM labeled "object"

2. Adds a new Link to PAM from the “rock” Node to the new “object” node.

The implication of this Link is that any time the agent perceives a rock; the object node in PAM will receive a portion of the rock node’s activation.

For this task you will be creating a new link in PAM from “food” to “object”. This means that the “object” node will also receive activation when food is perceived. You can follow the instructions given in the comments of this class to complete this task.

Run the agent and look at the PAM table GuiPanel. There should be an entry for the “object” node. Also in the PAM graph GuiPanel you can find the “object” node with these two links attached to it. Before, this Node only received activation when the agent detected a rock. Now it will receive activation whenever the agent detects a rock or food. This “object” node may enter the PerceptualBuffer as part of percept if it has sufficient activation, just like any other PAM node.

(Note: It is possible to create a Node with two children using the default initializer for PAM. However, adding a non-default decay strategy, as we will do in Task 3, requires the custom Initializer.)

### Task 3

Continuing in the `CustomPamInitializer` class, we want to adjust the `DecayStrategy` used by the “object” Node. The `DecayStrategy` governs how a Node’s activation is decayed over time. The available `DecayStrategy` types are defined in the `factoryData.xml` file. Go to this file now and in the `<strategies>` tag look for a strategy definition named `slowDecay`. Notice that the parameter for this strategy is very small which implies a slow decay rate. You are going to use this strategy for the “object” node. To do this copy the following code and add it in the `CustomPamInitializer` class. A comment in this class will indicate where to insert it.

```
DecayStrategy decayStrategy = factory.getDecayStrategy("slowDecay");  
objectNode.setDecayStrategy(decayStrategy);
```

This code obtains the “slowDecay” strategy from the `ElementFactory` and sets it as the “object” Node’s decay strategy. Save this file and run the application. Find the object node entry in the PAM table GuiPanel. Now when the object Node is detected its current activation will remain high (1.0) for a very long period compared to the rock and food nodes (as well as most other nodes in this agent’s PAM).

## Advanced Exercises

### *Advanced Exercise 1*

The agent currently does not do anything when facing a tree. When the agent is in a cell with a tree he will be safe from the evil monkeys. Add a new attention codelet declaration to agent xml file that attends to the “tree” node. Add a new scheme to Procedural Memory with a context of **treeFront** node, and the action **action.moveAgent**.

### *Advanced Exercise 2*

The Action Selection module used for this project is fairly simple. Based on it, create a new Action Selection class, which extends `FrameworkModuleImpl` and implements the `ActionSelection` and `BroadcastListener` interfaces. Modify the agent xml file adding a new listener declaration where the **ActionSelection** is a **BroadcastListener** of the **GlobalWorkspace**. Change the declaration of the ActionSelection module in the agent xml file so that it uses the class that you have created. In this module try your own algorithm for selecting an action. For example, you can require that a Behavior’s context must be present in the current broadcast before it is eligible for selection.

## Conclusion

This concludes the exercises for the `alifeAgent` project. In these exercises we have focused on these topics:

- Creating a feature detector class, adding a task definition to `factoryData.xml`, adding a task declaration to the agent xml file
- Creating attention codelets and changing their parameters
- Working with schemes in Procedural Memory
- Customizing modules using Initializers and elements with the `ElementFactory`



## Appendix

### ***Creating a New LIDA Framework Project***

The description that follows is for installing and running the framework with NetBeans 6.9 or above.

1. With NetBeans running select **File → New Project...**
  - a. Select Java for *categories* and select Java Application for *projects*
2. Give the project a name
  - a. Uncheck **Create main class**
  - b. Check **Set as main project**
3. Include required Libraries
  - a. Right-click project, select **Properties**
  - b. Select Libraries
  - c. Add JAR / folder
    - i. Add all jars from lib
    - ii. Add framework jar
4. Set references to source code and Javadoc for LIDA framework Library
  - a. Go to Properties → Libraries → Select **lida-framework-v1.0b.jar**
    - i. Edit framework jar and select the respectively zip files.
5. Create a 'config' folder and add all necessary configuration files.
  - a. In NetBeans go to the Files view and create the 'config' folder inside the Project from here.
  - b. Add the following configuration files (you can copy those from a tutorial project):
    - i. agent xml file
    - ii. factoryData xml file
    - iii. guiPanels properties
    - iv. guiCommands properties
    - v. lidaConfig properties

- vi. LidaXMLSchema.xsd
- vii. LidaFactories.xsd

### **Study Question Answers**

**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?*

**A:** The Perceptual Buffer is in the Workspace (therefore, it is implemented as a submodule of the Workspace in the Framework). The nodes in the buffer come asynchronously from PAM nodes that have activation above the threshold. The nodes in the Workspace are *Activatable*, but not *Learnable*; they have current activation but not base-level activation (like PAM nodes). However, they are copies of PAM nodes, and represent the same data as those nodes, and they have a reference to the original PAM node. They can have different excite and decay strategies from the PAM nodes, and different linkage patterns.

**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.*

**A:** Coalitions come from the Workspace. In the Model, one coalition wins the competition and is chosen for broadcast to most other modules (see the arrows in the LIDA Model diagram). In a particular implementation of an agent using the Framework, the broadcast will only go to modules that implement BroadcastListener and have registered as listeners with the agent's GlobalWorkspace module.

**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.*

**A:** In the LIDA Model, CSM should contain the agent's idea about what is currently happening in the external and/or internal environments. It will be built by Structure Building Codelets, primarily out of input from PAM and Declarative Memory Modules. In a specific Framework implementation, the CSM's contents will depend on the specifications in the agent declaration and other configuration files. In the basicAgent exercise, there are no Declarative Memory modules, and all of the CSM contents are from the Perceptual Buffer by way of PAM.

**STUDY QUESTION 2.1:** *Compare the GUI for the alifeAgent to the GUI of basicAgent. What are the differences, and why are they different?*

**A:** The differences in the GUI between the two agents are due to differences in the `guiPanels.properties` file. Inspecting the files shows the differences. The two agents have different environments, and the `alifeAgent` does not have a CSM panel (which was added to `basicAgent` in the exercises of Project I).

***STUDY QUESTION 2.2:*** *Why does the agent's behavior change in this way after the new feature detectors were added?*

A: The agent already has Schemes in Procedural Memory to respond to bad health and predators (see the declarations for Schemes 6 and 8 in the `ProceduralMemory` section of the agent declaration file). When the new perceptive capabilities are added, it begins to use these Schemes appropriately.

***STUDY QUESTION 2.3:*** *What are the possible reasons the agent doesn't flee?*

A: It may be that the agent cannot perceive the predator (feature detector absent); it cannot bring such a perception to consciousness (attention codelet missing); and/or it does not have an action mechanism to allow it to flee from predators (Scheme and/or Behavior missing). In the case of `alifeAgent_Exercise 3`, the agent is missing the attention codelet for predators, so it cannot pay attention to them.

***STUDY QUESTION 2.4:*** *How does this affect the agent's cognition and behavior?*

A: The agent can now bring the "concept" of a predator to "consciousness" and act appropriately.

***STUDY QUESTION 2.5:*** *How does this parameter change affect the agent's cognition?*

A: It becomes less "interested" in food, and less likely to "notice" it.