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

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