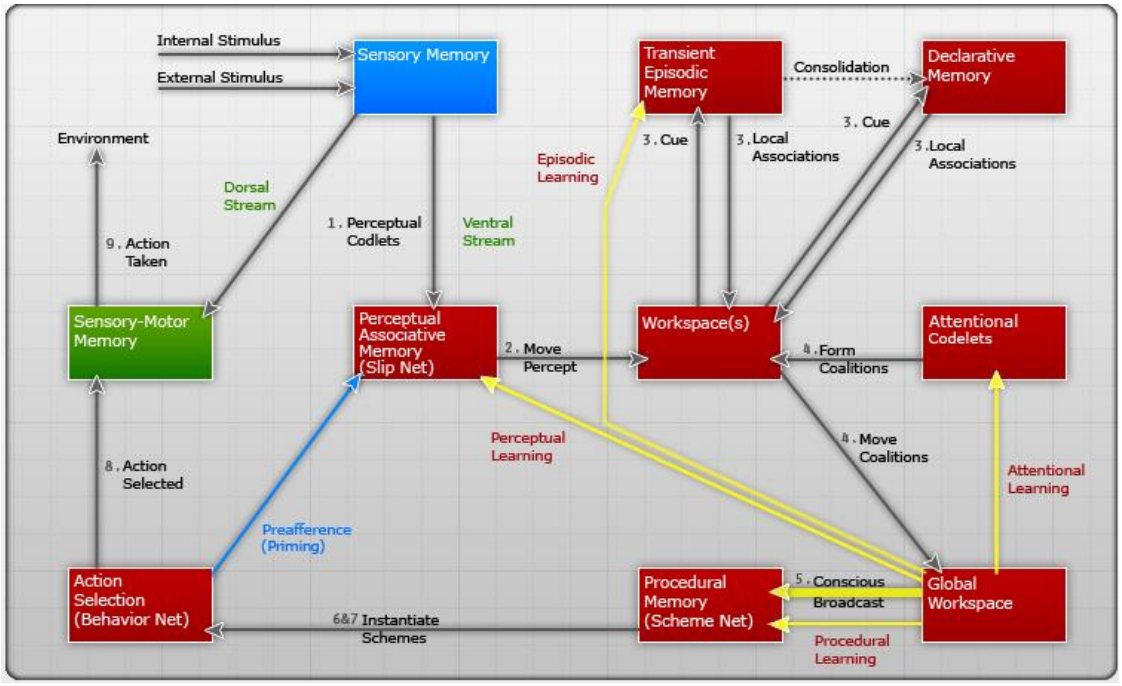
Aula 12 - LIDA 1: Entendendo a Arquitetura

Atividade 1

LIDA (*Learning Intelligent Distribution Agent*) é uma arquitetura cognitiva que abrange todo o espectro da cognição. Da percepção e ação até raciocínio de alto nível. Ela reflete o processo cognitivo encontrado em sistemas cognitivos naturais e se propõe a ser uma teoria cognitiva completa.

A arquitetura LIDA é baseada na teoria do Workspace Global proposta por Bernard Baars que explica o processo de funcionamento da consciência e foi implementada pelo grupo *Cognitive Computing Research Group*, liderado pelo professor Stan Franklin, da Universidade de Memphis.



Cada ciclo cognitivo da arquitetura LIDA pode ser dividido em três fases: compreensão, atenção e seleção de ação.

Compreensão: onde são recebidos estímulos internos e externos que são passados para a memória sensorial. Em seguida, é enviada para a memória associativa perceptual que faz a categorização e envia para o workspace.

Atenção: Inicia-se no ciclo cognitivo por meio dos *codelets*, que formam coalizões de partes selecionadas do modelo da situação atual e transportam as informações para a consciência.

Seleção de ação: Onde vários processo ocorrem paralelamente, e várias ações possíveis são enviadas para o mecanismo de seleção de ação. Esse mecanismo escolhe a ação mais adequada para a situação atual que está na consciência.

Atividade 2

A arquitetura LIDA foi desenvolvida em um framework de mesmo nome que utiliza a linguagem JAVA. Este framework implementa uma interface gráfica configurável e seu funcionamento é todo configurável através de arquivos XML.

O framework pode ser dividido em: módulos, tarefas, interface gráfica configurável, fábricas e interpretador XML.

Módulos: A arquitetura LIDA possui inúmeros módulos, como a memória sensorial, a memória perceptiva associada e o Workspace. Os módulos podem conter submódulos.

Listeners: Permitem que um módulo envie as informações para outros módulos sem precisar conhecer os detalhes sobre os módulos *Listener* que receberão a informação.

Tasks: Os módulos precisam executar várias tarefas para conseguir atingir suas funcionalidades. Então, o LIDA usa tarefas para implementar seus processos.

TaskManager: Usa um conjunto de threads para a execução das tarefas e controla o tempo interno do aplicativo.

Nós e Links: Nós e Links são as principais estruturas no LIDA. Ambos possuem ativação que podem ser excitadas de várias maneiras e se decaem com o tempo. Um nó pode representar objetos, eventos, conceitos, sentimentos, ações, etc.. Um link é conecta um nó a outro nó ou a outro link.

Atividade 3

- **Arquitetura baseada em Codelets (o que é isso ?)**

Assim como a arquitetura Copycat, (Hofstadter e Mitchell), uma arquitetura baseada em codelet utilizam pequenos elementos capazes de executar uma função (os codelets). Estes elementos trabalham de forma independente e assíncrona e carregam a informação da sua utilidade ajustada através de feedbacks. No LIDA, estes codelets carregam informações de atenção e servem para carregar as informações mais importantes para a consciência.

- **PAM - Perceptual Associative Memory**

Analogamente como na arquitetura Copycat, o PAM funciona como uma rede semântica Slipnet. O objetivo desta rede é encontrar características do interesse de cada codelet e traduzir em um estímulo para a consciência.

- **Behavior Network (Rede de Comportamentos)**

São redes de seleção de ação, como o proposto por Patty Maes. Os comportamentos que formam a rede competem entre si para serem selecionados. E a escolha do comportamento depende da situação atual do agente.

- **Global Workspace Theory (Mecanismo de Consciência do LIDA)**

Proposta por Bernard Baars, esta teoria explica os processos cognitivos conscientes e inconscientes. Reforçada pelas idéias de Daniel Dannett, que diz que toda a consciência vem de processos inconscientes, nesta teoria o cérebro é visto como uma rede distribuída de processos inconscientes chamadas de contexto.

Neste workspace, grupos neurais competem entre si pelo spotlight. Uma vez iluminados, eles têm suas informações propagadas em broadcast. Ou seja, o workspace global é um processador central e palco da consciência. O modelo de Baars pode ser visto também como um 'teatro da mente', em que os observadores também são atores que competem pelo spotlight e quando iluminados, ocorre a propagação da informações para toda mente.

- **Como a arquitetura LIDA implementa a percepção do ambiente ?**

O ambiente é implementado no LIDA pelo usuário utilizando uma classe abstrata chamada *EnvironmentImpl. Deve-se implementar os métodos para obter o estado do ambiente (getState), e o processamento das ações (processAction).*

- **Como a arquitetura LIDA guarda/armazena informações de memória ?**

A arquitetura LIDA armazena informações de varias maneiras: Memória associativa (slipnet), Memória episódica temporária, Memória procedural (schemas) e Memória declarativa.

- **Como a arquitetura LIDA seleciona as ações que serão implementadas no ambiente ?**

As ações são selecionadas por uma rede de comportamentos no módulo de Action Selection. Este módulo é responsável por selecionar a melhor ação na situação em que o agente se encontra.

- **Como a arquitetura LIDA executa a ação selecionada no ambiente ?**

As ações são executadas pelo Sensory-Motor-Memory. Neste módulo, as informações dos sensores são utilizadas para decidir qual ação será realizada dentro de um ciclo cognitivo.

Aula 13 - LIDA 2: Exemplos de Implementação Prática

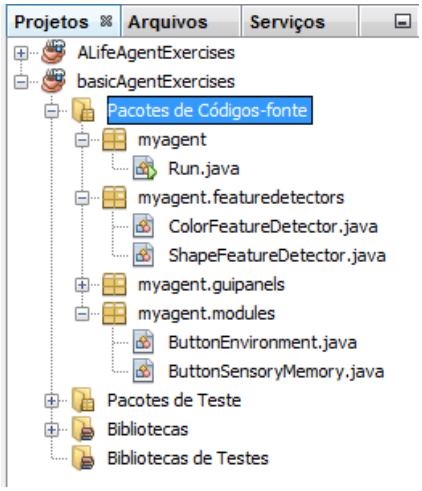
Atividade 1

Foi realizado o download do tutorial prático do LIDA. Este tutorial representa um agente onde se exibe dois tipos de formas geométricas em duas cores possíveis: Quadrado ou círculo, Azul ou vermelho. O Agente deve pressionar o botão 1 deve ser pressionado pelo agente quando um quadrado vermelho for apresentado, ou o botão 2 se a figura for um círculo azul.

Atividade 2 - Tutorial Project I

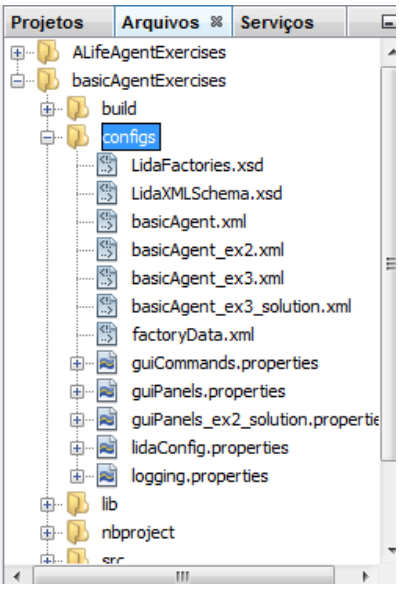
Basic Agent Exercise 0

Neste exercício foi possível conhecer a estrutura de classes do Basic Agent. Na figura abaixo podemos ver as principais classes utilizadas.



- *myagent.run*: Classe inicial que executa a aplicação;
- *myagent.modules.ButtonEnvironment*: implementa o ambiente;
- *myagent.modules.ButtonSensoryMemory*: implementa a memória sensorial do agente;
- *myagent.featuredetectors.ColorDetector*: detecta cores;
- *myagent.featuredetectors.ShapeDetector*: detecta formas.

Na árvore de diretório podemos encontrar a pasta configs, onde são armazenadas as configurações do Agente.

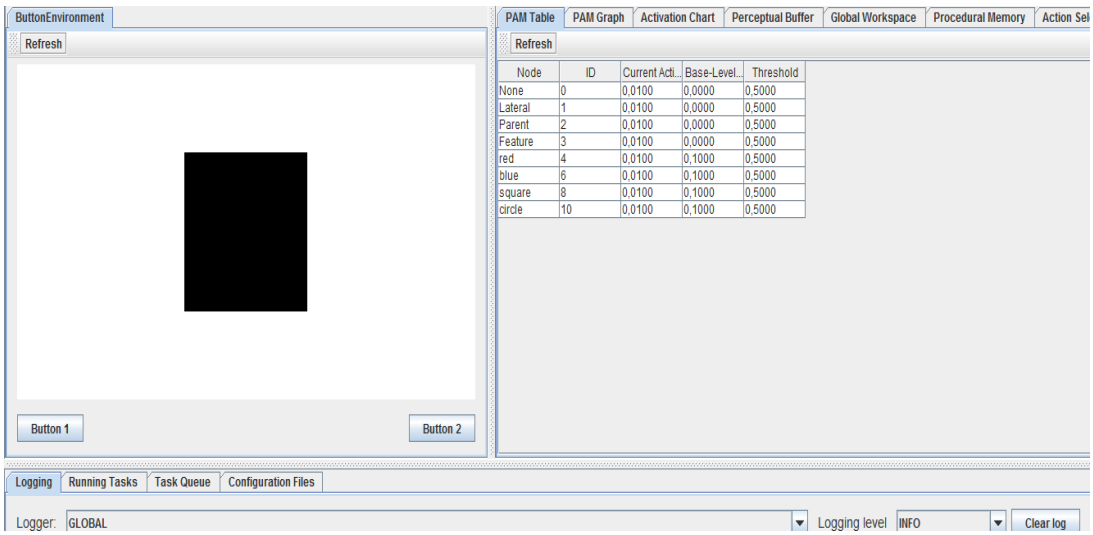


Os arquivos de configuração são os seguintes:

- *lidaConfig.properties*: Arquivo principal de configuração principal que indica onde estão os outros arquivos XML de configuração;
- *basicAgent.xml*: Arquivo onde é definida a arquitetura do agente, os módulos e processos que serão usados;
- *factoryData.xml* : Arquivo que define os elementos que podem ser criados através do do*ElementFactory:* nós, links, estratégias e tipos de tarefas;
- *guiPanels.properties*: Arquivo de configuração das abas da interface gráfica.

Basic Agent Exercise 1

O objetivo deste exercício é entender o funcionamento básico da interface gráfica, suas abas e painéis.



Na parte superior temos as opções:

Start/Pause: Inicia ou pausa a simulação

Current Tick: Indicador de unidade de tempo atual

Step Mode: Realiza a execução passo a passo.

Tick Duration: Modifica a duração de cada tick.

Na parte inferior exitem as abas:

Logging: Exibe as mensagens de log geradas pelo agente

Running Tasks: Exibe as tarefas em execução

Tasks Queue: Exibe a fila de tarefas a serem executadas

Configuration Files: Exibe os arquivos de configuração em uso pelo agente

Basic Agent Exercise 2

Neste exercício alteramos a configuração da arquitetura do agente informando um outro xml:

Código-Fonte | Histórico

```
1  ########################################################################################
2  # Copyright (c) 2009, 2011 The University of Memphis.  All rights reserved.
3  # This program and the accompanying materials are made available
4  # under the terms of the LIDA Software Framework Non-Commercial License v1.0
5  # which accompanies this distribution, and is available at
6  # http://ccrg.cs.memphis.edu/assets/papers/2010/LIDA-framework-non-commercial-v1.0.pdf
7  ########################################################################################
8
9  #Agent properties
10 lida.agentdata=configs/basicAgent_ex2.xml
11 lida.elementfactory.data=configs/factoryData.xml
12
13 #Gui properties
14 lida.gui.panels=configs/guiPanels.properties
15 lida.gui.commands=configs/guiCommands.properties
16 lida.gui.enable=true
17
18 #Logging properties
19 lida.logging.configuration=configs/logging.properties
```

Task1

Nesta tarefa mudamos o tempo do tick para 20 e percebemos que nesta configuração, o círculo azul não dispara nenhuma ativação. Já o quadrado vermelho, faz com que a ativação dos nós Red e Square passem para 1.0 e comecem a decair quando a imagem desaparece, até chegar a 0.

PAM Table | PAM Graph | Activation Chart | Perceptual Buffe

Refresh

Node	ID	Current Acti...	Base-Level...	Threshold
None	0	0,0000	0,0000	0,5000
Lateral	1	0,0000	0,0000	0,5000
Parent	2	0,0000	0,0000	0,5000
Feature	3	0,0000	0,0000	0,5000
red	4	1,0000	0,1000	0,5000
blue	6	0,0000	0,1000	0,5000
square	8	1,0000	0,1000	0,5000
circle	10	0,0000	0,1000	0,5000

Task 2

O Painel Perceptual Buffer mostra os nós que estão sendo percebidos pelo agente.

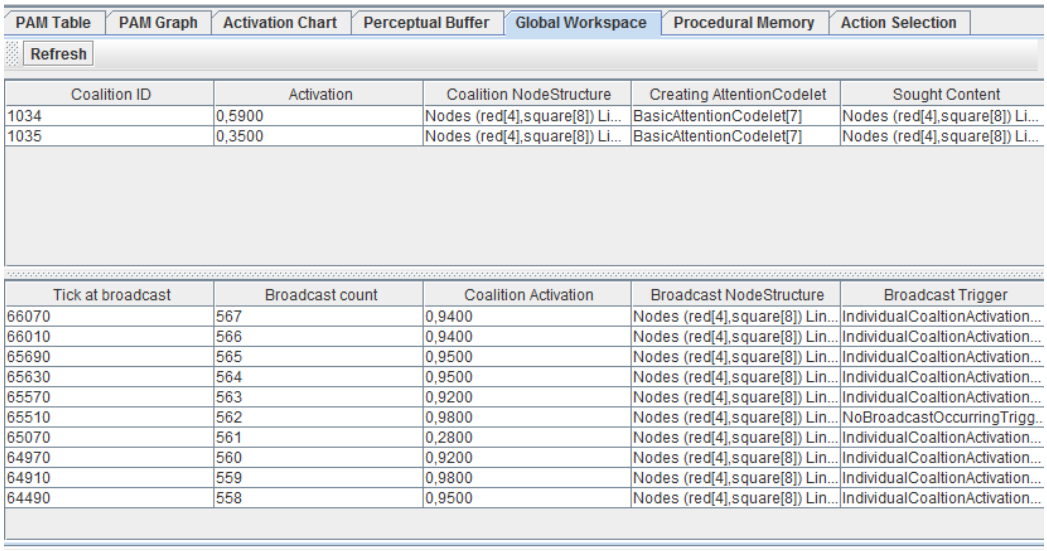
Questão 1.1:

O Perceptual Buffer pertence ao módulo Workspace. Estes nós vêm da memória associativa perceptual.

Task 3

Na aba Global Workspace são exibidas as coalizões que atualmente estão no workspace global.

Na parte inferior é exibido o histórico das últimas coalisões.



Coalition ID	Activation	Coalition NodeStructure	Creating AttentionCodelet	Sought Content
1034	0,5900	Nodes (red[4],square[8]) Li...	BasicAttentionCodelet[7]	Nodes (red[4],square[8]) Li...
1035	0,3500	Nodes (red[4],square[8]) Li...	BasicAttentionCodelet[7]	Nodes (red[4],square[8]) Li...

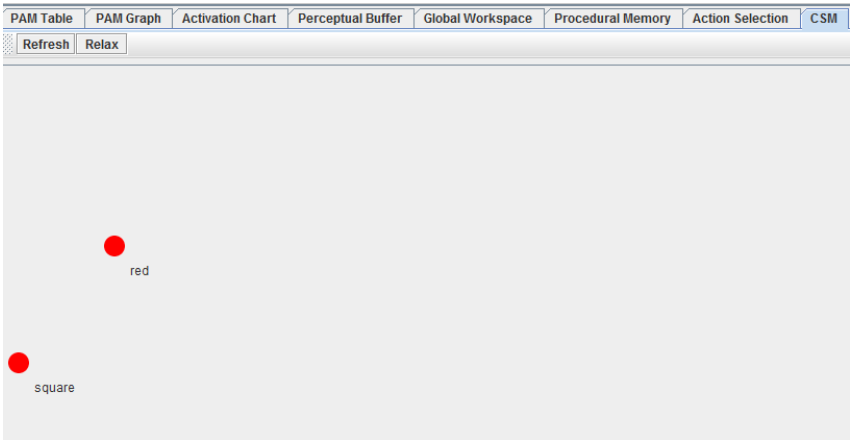
Tick at broadcast	Broadcast count	Coalition Activation	Broadcast NodeStructure	Broadcast Trigger
66070	567	0,9400	Nodes (red[4],square[8]) Lin...	IndividualCoaltionActivation...
66010	566	0,9400	Nodes (red[4],square[8]) Lin...	IndividualCoaltionActivation...
65690	565	0,9500	Nodes (red[4],square[8]) Lin...	IndividualCoaltionActivation...
65630	564	0,9500	Nodes (red[4],square[8]) Lin...	IndividualCoaltionActivation...
65570	563	0,9200	Nodes (red[4],square[8]) Lin...	IndividualCoaltionActivation...
65510	562	0,9800	Nodes (red[4],square[8]) Lin...	NoBroadcastOccurringTrigg...
65070	561	0,2800	Nodes (red[4],square[8]) Lin...	IndividualCoaltionActivation...
64970	560	0,9200	Nodes (red[4],square[8]) Lin...	IndividualCoaltionActivation...
64910	559	0,9800	Nodes (red[4],square[8]) Lin...	IndividualCoaltionActivation...
64490	558	0,9500	Nodes (red[4],square[8]) Lin...	IndividualCoaltionActivation...

Questão 1.2:

Estas coalizões vêm da ligação com os nós do *Perceptual Buffer. Elas se propagam via broadcast para todos os módulos que estiverem "ouvindo", ou seja, que tenham implementado um BroadcastListener.*

Tarefa 4

Nesta tarefa modificamos o arquivo de configuração da interface gráfica e incluímos um novo painel "*CurrentSituationalModel*"



Questão 1.3:

O conteúdo do *CSM* vêm do *Perceptual Buffer*. A memória associativa perceptual (PAM) é de onde vêm os nós para o Perceptual Buffer..

Basic Agent Exercise 3

Modificamos a arquitetura para utilizar o arquivo basicAgent_ex3.xml

Tarefa 1

Verificamos no arquivo xml onde deve-se colocar a declaração dos módulos.

Tarefa 2

Incluimos a declaração para o módulo Environment

```
</taskspawners>
<submodules>
    <!-- TASK2 INSERT YOUR CODE HERE ************************ -->
    <module name="Environment">
        <class>myagent.modules.ButtonEnvironment</class>
        <param name="height" type="int">10</param>
        <param name="width" type="int">10</param>
        <taskspawner>defaultTS</taskspawner>
    </module>
```

Executando o agente percebemos que os nós Red e Square passam a receber ativação, mas o Perceptual Buffer não exibe nada.

Tarefa 3

Incluímos a declaração de um listener para o PAM

```
<listeners>
    <!-- TASK 3 INSERT YOUR CODE HERE ************************ -->
    <listener>
        <listenertype>edu.memphis.ccrg.lida.pam.PamListener</listenertype>
        <modulename>PerceptualAssociativeMemory</modulename>
        <listenername>Workspace</listenername>
    </listener>
```

Ao executar o agente percebe-se que o Perceptual Buffer passa a exibir os nós Red e Square.

Tarefa 4

Incluimos agora dentro da declaração do PAM, tasks para detectar a cor azul.

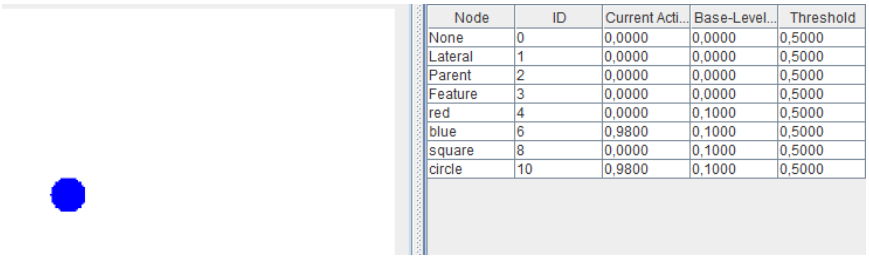
```
<!-- INSERT YOUR CODE HERE ************************ -->
<task name="BlueDetector">
    <tasktype>ColorDetector</tasktype>
    <ticksperrun>3</ticksperrun>
    <param name="color" type="int">-16776961</param>
    <param name="node" type="string">blue</param>
</task>
```

Tarefa 5

Agora incluímos task para detectar círculos

```
    <task name="CircleDetector">
        <tasktype>ShapeDetector</tasktype>
        <ticksperrun>3</ticksperrun>
        <param name="area" type="int">31</param>
        <param name="backgroundColor" type="int">-1</param>
        <param name="node" type="string">circle</param>
    </task>
```

Ao executar o agente, vemos que há ativação para quadrados vermelhos e também para círculos azuis:



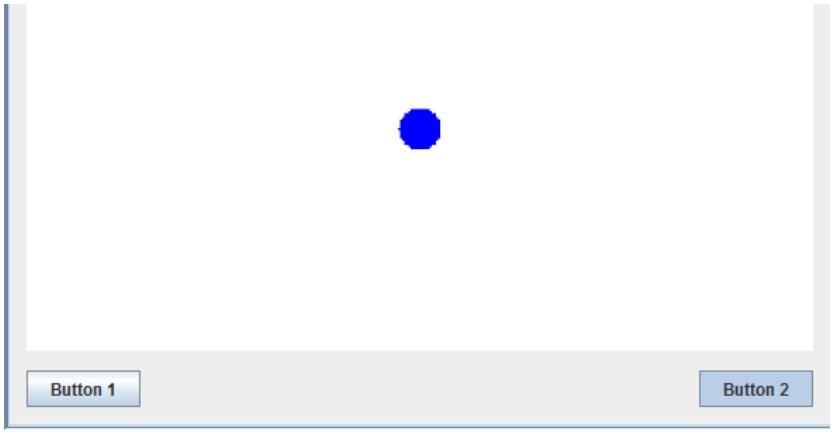
Node	ID	Current Acti...	Base-Level...	Threshold
None	0	0,0000	0,0000	0,5000
Lateral	1	0,0000	0,0000	0,5000
Parent	2	0,0000	0,0000	0,5000
Feature	3	0,0000	0,0000	0,5000
red	4	0,0000	0,1000	0,5000
blue	6	0,9800	0,1000	0,5000
square	8	0,0000	0,1000	0,5000
circle	10	0,9800	0,1000	0,5000

Tarefa 6

Para que o agente passe a pressionar o botão 2 quando encontrar círculos azuis, precisa-se adicionar uma declaração no AttentionModule:

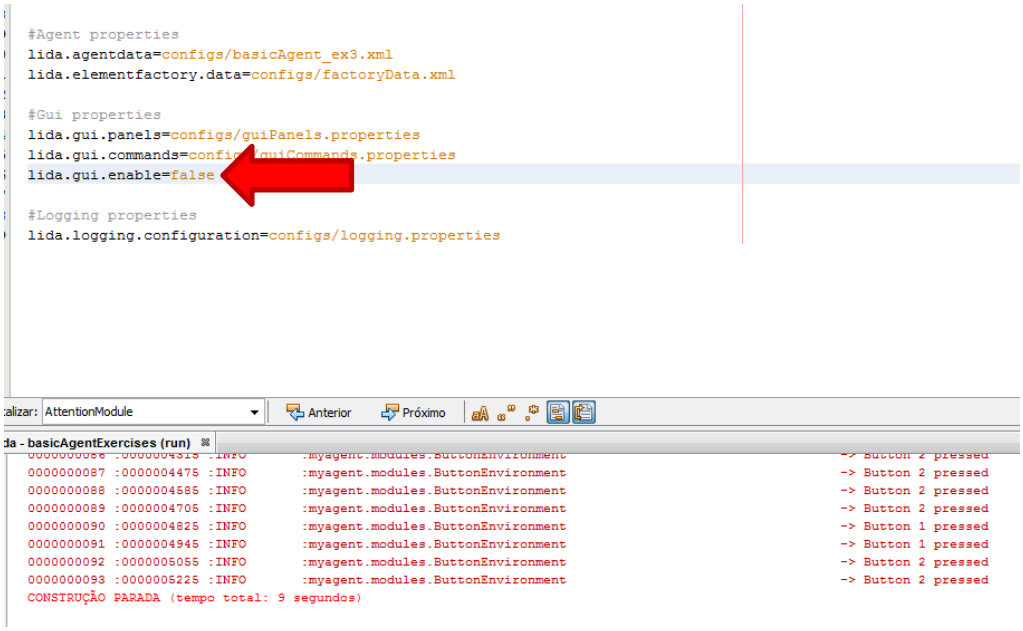
```
<!-- TASK 6 INSERT YOUR CODE HERE ************************ -->
<task name="BlueCircleCodelet">
    <tasktype>BasicAttentionCodelet</tasktype>
    <ticksperrun>5</ticksperrun>
    <param name="nodes" type="string">blue,circle</param>
    <param name="refractoryPeriod" type="int">30</param>
    <param name="initialActivation" type="double">1.0</param>
</task>
```

Executando o agente agora pode-se perceber que o botão 2 é pressionado quando exibe um círculo azul.



Advanced Exercise 1

Neste exercício vimos que é possível executar o agente sem interface gráfica. O resultado da execução pode ser visto através do log gerado pelo agente.



Advanced Exercise 2

Neste exercício executamos o basicAgent e verificamos que o número de coalizões enviadas ao workspace global era grande.

Fizemos a alteração abaixo proposta no arquivo xml do agente:

```
<task name="RedSquareCodelet">
    <tasktype>BasicAttentionCodelet</tasktype>
    <ticksperrun>50</ticksperrun>
    <param name="nodes" type="string">red,square</param>
    <param name="refractoryPeriod" type="int">300</param>
    <param name="initialActivation" type="double">1.0</param>
</task>
```

Definimos o RedSquareCodlet para 50 ticks per run e ao executar o agente verificamos que quando o quadrado vermelho era exibido, o número de coalizões era bem menor do que antes.

Advanced Exercise 3

Neste exercício criamos um detector para a tela em branco. Para isso criamos uma classe para a detecção, um codelet de atenção e um esquema no o xml do agente.

A Classe para detecção será responsável por contar os pixels da tela e retornar a ativação máxima se apenas existir o background:

```
15  public class EmptyFeatureDetector extends BasicDetectionAlgorithm {
        private int backgroundColor = 0xFFFFFFFF;
        private Map<String, Object> smParams = new HashMap<String, Object>();
18
19      @Override
        public void init() {
21          super.init();
22          smParams.put("mode","all");
23      }
24
25      @Override
        public double detect() {
27          int[] layer = (int[]) sensoryMemory.getSensoryContent("visual",smParams);
28          int area=0;
29          for(int i=0;i<layer.length;i++){
30              if(layer[i]!=backgroundColor){
31                  area++;
32              }
33          }
34          if(area == 0){
35              return 1.0;
36          }
37          return 0.0;
38      }
```

Adicionando o detector de característica no factoryData.xml

```
<task name="EmptyDetector">
        <class>myagent.featuredetectors.EmptyFeatureDetector</class>
        <ticksperrun>5</ticksperrun>
        <associatedmodule>SensoryMemory</associatedmodule>
        <associatedmodule>PerceptualAssociativeMemory</associatedmodule>
        <param name="backgroundColor" type="int">-1</param>
        <param name="node" type="string">empty</param>
</task>
```

No basicAgent.xml foi adicionado um novo nó ao módulo *PerceptualAssociativeMemory:*

```
<module name="PerceptualAssociativeMemory">
    <class>edu.memphis.ccrg.lida.pam.PerceptualAssociativeMemoryImpl</class>
    <param name="pam.Upscale" type="double">.7 </param>
    <param name="pam.Downscale" type="double">.6 </param>
    <param name="pam.Selectivity" type="double">.5 </param>
    <param name="nodes">red,blue,square,circle,empty</param>
    <taskspawner>defaultTS</taskspawner>
    <initialTasks>
        <task name="EmptyDetector">
            <tasktype>EmptyDetector</tasktype>
            <ticksperrun>3</ticksperrun>
            <param name="backgroundColor" type="int">-1</param>
            <param name="node" type="string">empty</param>
        </task>
```

Ainda no basicAgent.xml foi incluída uma nova tarefa ao codelet de atenção:

```
<module name="AttentionModule">
    <class>edu.memphis.ccrg.lida.attentioncodelets.AttentionCodeletModule</class>
    <associatedmodule>Workspace</associatedmodule>
    <associatedmodule>GlobalWorkspace</associatedmodule>
    <taskspawner>defaultTS</taskspawner>
    <initialTasks>
        <task name="EmptyCodelet">
            <tasktype>BasicAttentionCodelet</tasktype>
            <ticksperrun>5</ticksperrun>
            <param name="nodes" type="string">empty</param>
            <param name="refractoryPeriod" type="int">30</param>
            <param name="initialActivation" type="double">1.0</param>
        </task>
```

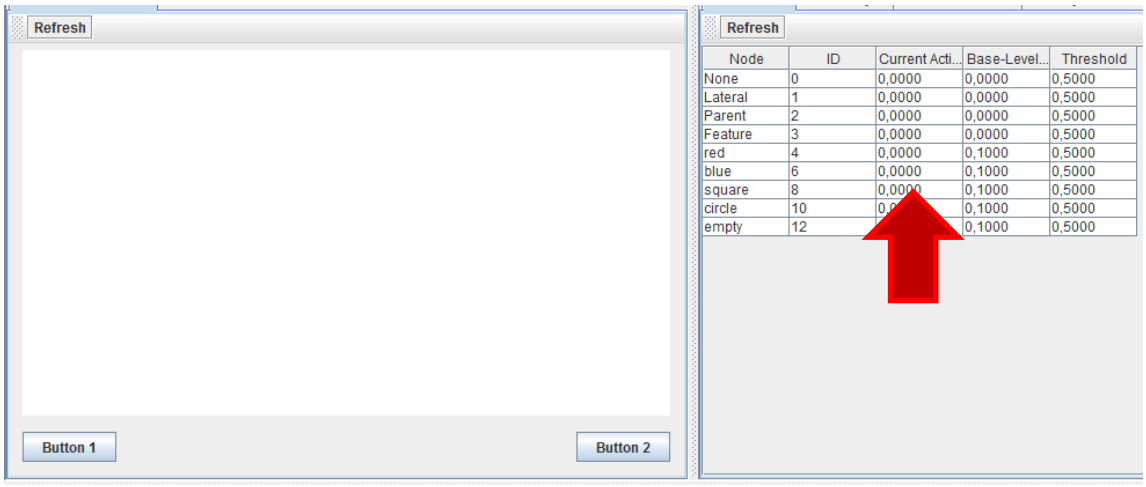
Um novo esquema foi incluído na memória procedural, para fazer com que a ação de soltar o botão seja executada se a tela estiver vazia

```
<module name="ProceduralMemory">
    <class>edu.memphis.ccrg.lida.proceduralmemory.ProceduralMemoryImpl</class>
    <param name="proceduralMemory.ticksPerStep" type="int"> 14 </param>
    <param name="scheme.1">if red square, press 1|(red,square)()|action.pressOne|()()|0.01</param>
    <param name="scheme.2">if blue circle, press 2|(blue,circle)()|action.pressTwo|()()|0.01</param>
    <param name="scheme.3">if empty, release press|(empty)()|action.releasePress|()()|0.01</param>
    <taskspawner>defaultTS</taskspawner>
    <initializerclass>edu.memphis.ccrg.lida.proceduralmemory.BasicProceduralMemoryInitializer</initializerclass>
        <!-- <initializerclass>myagent.initializers.ProceduralMemoryInitializer</initializerclass>-->
</module>
```

No módulo SensoryMotorMemory, o parâmetro smm.3 foi descomentado:

```
<module name="SensoryMotorMemory">
    <class>edu.memphis.ccrg.lida.sensorymotormemory.BasicSensoryMotorMemory</class>
    <associatedmodule>Environment</associatedmodule>
    <param name="smm.1">action.pressOne,algorithm.press1</param>
    <param name="smm.2">action.pressTwo,algorithm.press2</param>
    <param name="smm.3">action.releasePress,algorithm.releasePress</param>
    <taskspawner>defaultTS</taskspawner>
    <initializerclass>edu.memphis.ccrg.lida.sensorymotormemory.BasicSensoryMotorMemoryInitializer</initializerclass>
</module>
```

Ao executar o agente, o nó empty é detectado e ativado quando a área da imagem está vazia.



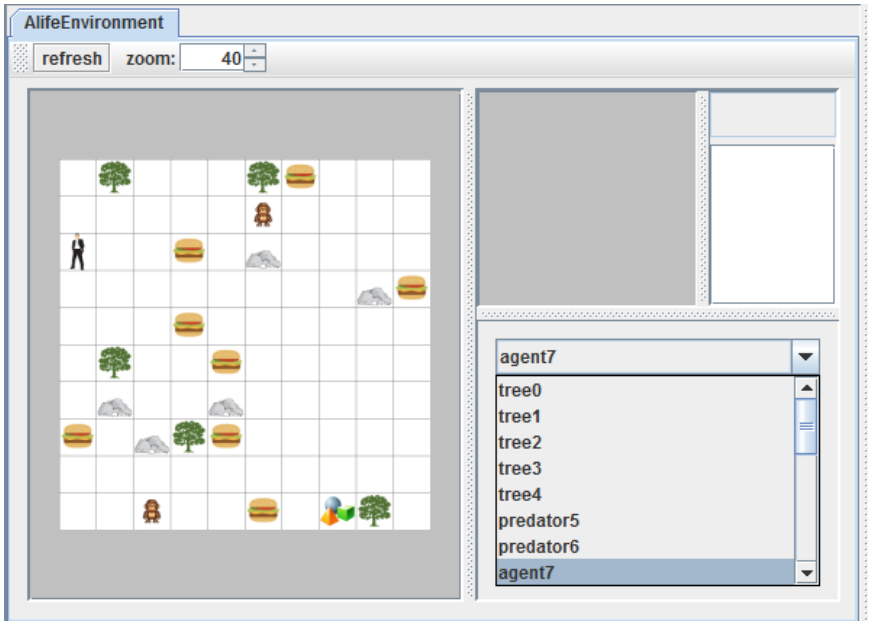
Atividade 2 - Tutorial Project II

Agent Exercise 1

Neste exercício executamos outro agente que simula um ambiente com predadores, obstáculos e alimento.

Tarefa 1

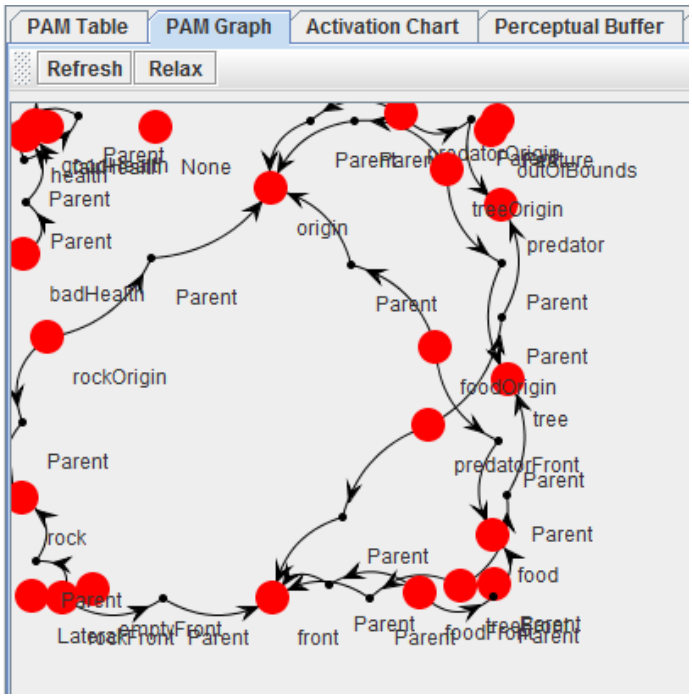
Podemos clicar sobre cada célula para visualizar os dados de cada elemento, ou podemos seleciona-los na lista à direita.



Questão 2.1: A interface do alifeAgent é mais complexa que a do basicAgent. Nesta interface pode-se identificar todos os elementos do ambiente e seus parâmetros.

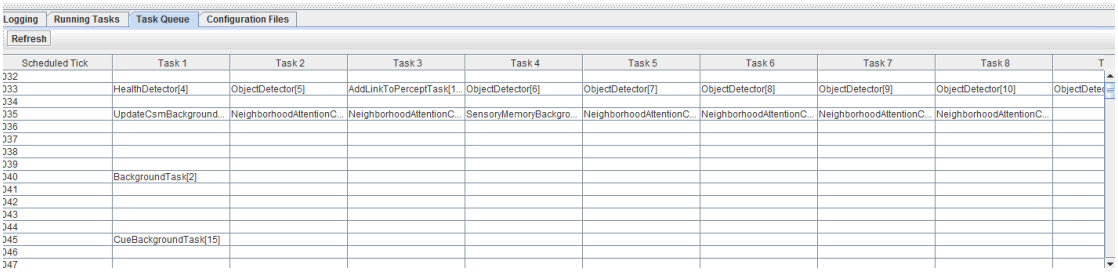
Tarefa 2

Nesta tarefa vimos o painel de gráfico do PAM. Neste gráfico podemos ver todos os nós e suas ligações. Quando o número de nós é muito grande, a visualização fica prejudicada. Para isto podemos aumentar o zoom para inspecionar um nó, ou então clicar no botão relax, para que sejam distribuídos de uma forma melhor.

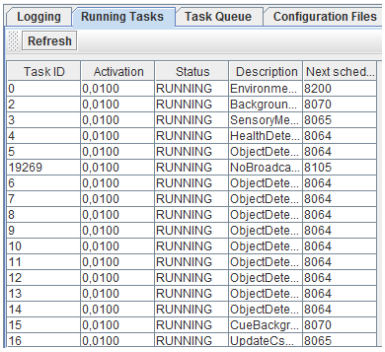


Tarefa 3

No painel Task queue podemos ver as tarefas que estão enfileiradas para serem executadas em cada tick.



No painel Running Tasks podemos ver as tarefas que estão sendo executadas no momento:

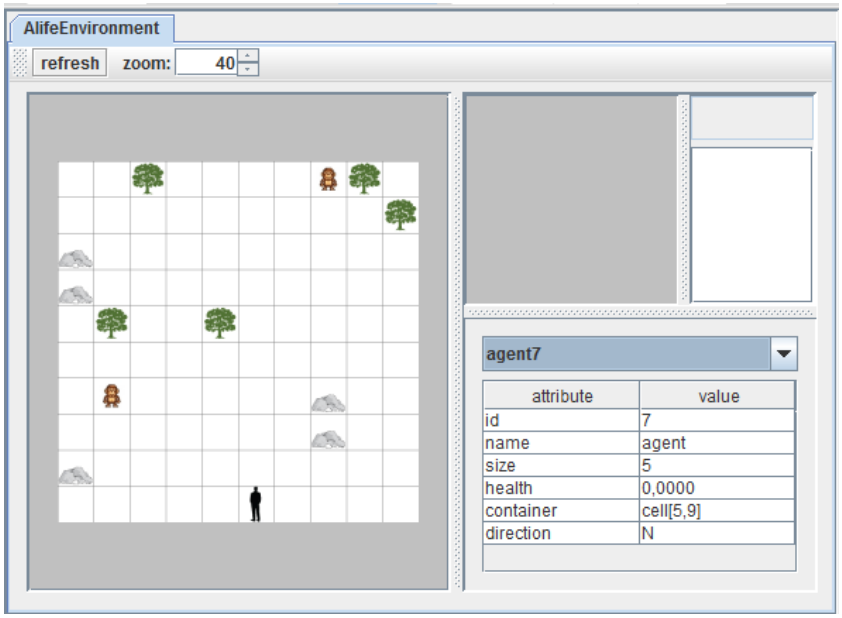


ALIfe Agent Exercise 2

Tarefa 0

Nesta tarefa, modificamos os parâmetros para não haver mais comida no ambiente virtual. Desta forma, o agente não consegue mais recuperar sua energia.

Percebe-se também que quando a energia cai para 0,33 o agente não se movimenta mais.



Tarefa 1

Nesta tarefa criamos uma classe para detectar quando a energia do agente está baixa, ou seja, inferior a 0,33

```
8   public class BadHealthDetector extends BasicDetectionAlgorithm {
9
10      private final String modality = "";
        private Map<String, Object> detectorParams = new HashMap<String, Object>();
12
13      @Override
        public void init() {
15          super.init();
16          detectorParams.put("mode", "health");
17      }
18
19      @Override
        public double detect() {
21          double healthValue = (Double) sensoryMemory.getSensoryContent(modality, detectorParams);
22          double activation = 0.0;
23          if (healthValue <= 0.33) {
24              activation = 1.0;
25          }
26          return activation;
27      }
28  }
```

Tarefa 2

Nesta tarefa incluímos um detector para baixa energia no arquivo factoryData.xml

```
<!-- INSERT YOUR CODE HERE ************************ -->
<task name="BadHealthDetector">
    <class>alifeagent.featuredetectors.BadHealthDetector</class>
    <ticksperrun>3</ticksperrun>
    <associatedmodule>SensoryMemory</associatedmodule>
    <associatedmodule>PerceptualAssociativeMemory</associatedmodule>
</task>
```

Tarefa 3

Criamos uma nova tarefa dentro do módulo PerceptualAssociativeMemory para que um novo nó de baixa energia seja criado

```
<!-- TASK 3 INSERT YOUR CODE HERE ************************ -->
<task name="BadHealthDetector">
    <tasktype>BadHealthDetector</tasktype>
    <ticksperrun>3</ticksperrun>
    <param name="node" type="string">badHealth</param>
</task>
```

Tarefa 4

Adicionalmente criamos um novo detector e nó para quando o predador estiver em frente ao agente.

```
<!-- TASK 4 INSERT YOUR CODE HERE ************************ -->
<task name="predatorFrontDetector">
    <tasktype>ObjectDetector</tasktype>
    <ticksperrun>3</ticksperrun>
    <param name="node" type="string">predatorFront</param>
    <param name="object" type="string">predator</param>
    <param name="position" type="int">1</param>
</task>
```

Executando o agente, ele passa a se mover quando a energia estiver baixa e também foge do predador quando este estiver à sua frente.

Questão 2.2: O comportamento do agente muda apenas colocando os detectores pois já existem schemas na memória procedural para executar ações quando estes novos nós forem ativados.

ALIfe Agent Exercise 3

Neste exercício modificamos a configuração do agente para usar as declarações do arquivo *configs/alifeAgent_ex3.xm*

Tarefa 0

Executamos a simulação e percebemos que o agente não reage ao predador.

Questão 2.3: O agente não reage ao predador pois não há codelet de ateção para tratar a aproximação do predador.

Tarefa 1

Nesta tarefa incluímos o codelet de atenção para tratar os nós do tipo “predator”.

```
<!-- TASK 1 INSERT YOUR CODE HERE ************************ -->
<task name="PredatorAttentionCodelet">
    <tasktype>NeighborhoodAttentionCodelet</tasktype>
    <ticksperrun>5</ticksperrun>
    <param name="nodes" type="string">predator</param>
    <param name="refractoryPeriod" type="int">50</param>
    <param name="initialActivation" type="double">1.0</param>
</task>
<!-- ******************************************************* -->
```

Questão 2.4: A partir de agora, o nó predador pode ser levado à consciência e ser tratado adequadamente.

Tarefa 2

Nesta tarefa modificamos o codelet de atenção para comida e reduzimos a ativação inicial para 0.01.

```
<task name="FoodAttentionCodelet">
    <tasktype>NeighborhoodAttentionCodelet</tasktype>
    <ticksperrun>5</ticksperrun>
    <param name="nodes" type="string">food</param>
    <param name="refractoryPeriod" type="int">50</param>
    <param name="initialActivation" type="double">0.01</param>
</task>
```

Ao executarmos a simulação percebemos que o agente não reage à comida que está próxima.

Questão 2.5: O Agente passa a ficar menos interessado na comida.

Tarefa 3

Modificamos agora o codelet de atenção para GoodHealth. Alteramos o parâmetro refractoryPeriod para 10.

```
<task name="GoodHealthAttentionCodelet">
    <tasktype>NeighborhoodAttentionCodelet</tasktype>
    <ticksperrun>5</ticksperrun>
    <param name="nodes" type="string">goodHealth</param>
    <param name="refractoryPeriod" type="int">10</param>
    <param name="initialActivation" type="double">0.10</param>
</task>
```

Após a alteração, executamos a simulação e percebemos que a frequência com que coalizões envolvendo GoodHealth aparecem no Workspace Global aumentou.

ALIfe Agent Exercise 4

Neste exercício alteramos a configuração da arquitetura para utilizer o xml de agente configs/alifeAgent_ex4.xml

Tarefa 0

Executamos a simulação e percebemos que o agente só se move se sua saúde estiver abaixo de 0.66 ou se algum predador se aproximar.

Tarefa 1

Modificamos o schema 10.b na memória procedural para que o agente passe a se mover ao encontrar espaço vazio à sua frente.

```
<!-- MODIFY THE FOLLOWING scheme -->
<!--    ____scheme_name_____ scheme_label_____|_context_____|__action_name____|result|baseLabelActiv. -->
<param name="scheme.10b">if emptyFront move agent|(emptyFront)()|action.moveAgent|()()  |0.1</param>
```

Executamos a simulação e percebemos que o a gente se move mesmo quando está com boa saúde.

Tarefa 2

Nesta tarefa modificamos o incializador do PAM para utilizar uma classe customizada. Este inicializador é responsável por criar nós e links no PAM.

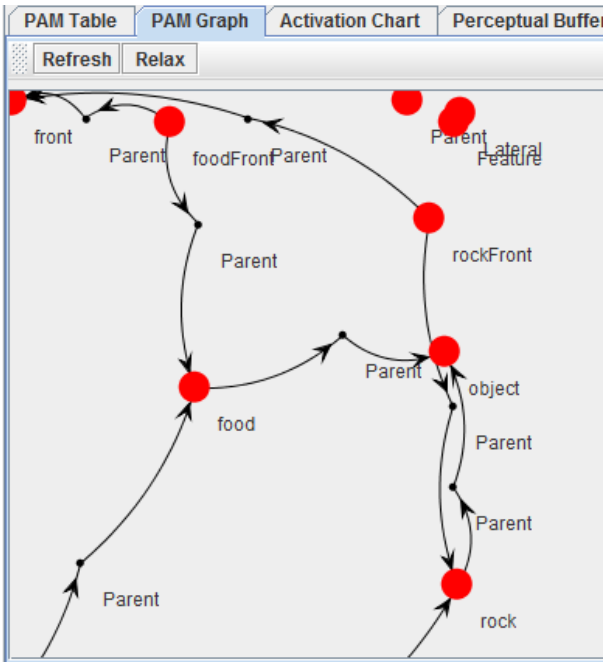
```
<initializerclass>alifeagent.initializers.CustomPamInitializer</initializerclass>
```

Em seguida modificamos a classe customizada para criar um link entre o nó food e object:

```
// Task 2: INSERT YOUR CODE HERE *************************
// Obtain the 'food' node from PAM
child = pam.getNode("food");

//  Add a Link in PAM from 'food' to 'object' with link category PARENT
pam.addDefaultLink(factory.getLink(child, objectNode, PerceptualAssociativeMemoryImpl.PARENT));
```

Executamos o agente e através do PAM Graph podemos ver que á um link entre rock e object, e também food e object.



Tarefa 3

Nesta tarefa ajustamos o tempo de decaimento do nó Object. Para isto utilizamos a estratégia de decaimento “slowDecay” definida no factoryData.xml

```
// Task 3: INSERT YOUR CODE HERE **************************
DecayStrategy decayStrategy = factory.getDecayStrategy("slowDecay");
objectNode.setDecayStrategy(decayStrategy);
```

Após a alteração, executamos a simulação e percebemos que a ativação do nó object permanece mais tempo em 1.0 e demora mais a começar a cair, com relação aos outros nós.

Advanced Exercise 1

Na simulação atual, o agente não tem nenhuma ação ao se deparar com uma árvore. Porém é interessante que ele se mova até ela para que se proteja dos predadores.

Para isto, criamos um codelet de atenção para detectar quando há uma árvore na frente do agente.

```
<task name="TreeFrontAttentionCodelet">
    <tasktype>NeighborhoodAttentionCodelet</tasktype>
    <ticksperrun>5</ticksperrun>
    <param name="nodes" type="string">treeFront</param>
    <param name="refractoryPeriod" type="int">50</param>
    <param name="initialActivation" type="double">1.0</param>
</task>
```

Depois incluímos um schema na memória procedural para selecionar a ação de mover o agente quando houver uma árvore na frente.

```
<param name="scheme.11">move forward if treeFront |(treeFront)()|action.moveAgent|()()  |0.1</param>
```

Advanced Exercise 2

Neste exercício criaremos uma nova classe para seleção de ação. Esta nova classe deverá herdar da classe *FrameworkModuleImpl* e implementar as interfaces *ActionSelection* e *BroadcastListener*.

Primeiro modificamos o arquivo de configuração do agente e incluimos um listener onde o *ActionSelection* é um *BroadcastListener* para o *GlobalWorkspace:*

```
<!--<listener>
    <listenertype>edu.memphis.ccrg.lida.proceduralmemory.ProceduralMemoryListener</listenertype>
    <modulename>ProceduralMemory</modulename>
    <listenername>ActionSelection</listenername>
</listener>-->
<listener>
    <listenertype>edu.memphis.ccrg.lida.globalworkspace.BroadcastListener</listenertype>
    <modulename>GlobalWorkspace</modulename>
    <listenername>ActionSelection</listenername>
</listener>
```

No módulo ActionSelection, modificamos para que pudesse utilizar a nova classe criada.

```
<!--<module name="ActionSelection">
    <class>edu.memphis.ccrg.lida.actionselection.BasicActionSelection</class>
    <param name="actionSelection.ticksPerStep" type="int"> 10</param>
    <taskspawner>defaultTS</taskspawner>
</module>-->
<module name="ActionSelection">
    <class>alifeagent.modules.CustomActionSelection</class>
    <param name="actionSelection.ticksPerStep" type="int">10</param>
    <taskspawner>defaultTS</taskspawner>
</module>
```

Não foi possível ainda implementar a classe “CustomActionSelection”