Nome: Fabio Grassiotto

RA: 890441

Disciplina: IA941A, 1º S 2018

Aula 6 – SOAR: Controlando o WorldServer3D

**Objetivo**

Utilizar o Soar para controlar uma aplicação externa por meio da interface SML.

**Atividade 1**

Na atividade 1, estudamos em sala um exemplo de um controlador que utiliza o SOAR como um controlador reativo para a tomada de decisões. Notei as seguintes características relativas ao funcionamento do código Java do DemoJSOAR:

* Funcionamento do loop principal em Main.java:

O loop principal efetua inicialmente a leitura de um arquivo de regras do Soar.

A partir daí inicializa o environment de simulação e entra logo depois em um loop infinito que executa as regras lidas do arquivo soar passo a passo através da chamada soarBridge.step().

* Acesso ao WorldServer3D através do Proxy:

O método step() executa os seguintes passos:

* + Prepara o input link, criando o ambiente no WS3D
  + Executa as regras do Soar
  + Processa o output link, criando uma lista de comandos.
  + Envia os comandos para o WS3D utilizando o método processCommands() com a lista de comandos retornados anteriormente.

O método mstep é utilizado de forma similar, mas quebrando os passos nas fases de execução do Soar (micro-step).

* Leitura do Estado do WS3D: A leitura do estado do WS3D é realizada através dos métodos do SoarBridge:
  + prepareInputLink() - cria elementos de memória de trabalho WMEs relacionadas ao estado do ambiente.
  + processOutputLink() - envia comandos de saída do output link do Soar para controlar o WS3D.
* Como os dados do Soar são utilizados para controlar a criatura: através da execução do método processCommands().
* Arquivo soar-rules.soar:

São propostas regras com operadores distintos para cada passo da criatura. Regras de preferência são utilizadas para selecionar cada um dos operadores.

**Atividade 2**

Nesta atividade, é proposto o desenvolvimento de um conjunto de regras no SOAR para implementar uma estratégia deliberativa de comportamento para o controle da criatura. Esta estratégia deverá deliberar todas as ações intermediárias que são necessárias para que o objetivo seja atingido.

Os seguintes passos foram implementados nesta atividade:

1. Alterações no código Java do DemoJSOAR:

O código do método SoarBridge::PrepareInputLink foi alterado para adicionar ao input link uma estrutura contendo um somatório dos objetivos dos leaflets da criatura, adicionando o código abaixo:

// Create the creature leaflets in the input link.

List<Leaflet> leafletList = c.getLeaflets();

Identifier leaflet = CreateIdWME(creature, "LEAFLET");

int targetRed = 0, targetGreen = 0, targetBlue = 0, targetYellow = 0,

targetMagenta = 0, targetWhite = 0;

for (Leaflet l: leafletList)

{

// Get what to collect from leaflet.

HashMap<String, Integer> h = l.getWhatToCollect();

for (String key: h.keySet())

{

// Count all jewel ocurrences in the leaflets.

if (key.equals(COLOR\_RED))

{

targetRed++;

} else if (key.equals(COLOR\_GREEN)) {

targetGreen++;

} else if (key.equals(COLOR\_BLUE)) {

targetBlue++;

} else if (key.equals(COLOR\_YELLOW)) {

targetYellow++;

} else if (key.equals(COLOR\_MAGENTA)) {

targetMagenta++;

} else {

targetWhite++;

}

}

}

// Create target in the inputlink. All three leaflets are summed up

// as a single list of target colors.

CreateFloatWME(leaflet, COLOR\_RED, targetRed);

CreateFloatWME(leaflet, COLOR\_GREEN, targetGreen);

CreateFloatWME(leaflet, COLOR\_BLUE, targetBlue);

CreateFloatWME(leaflet, COLOR\_YELLOW, targetYellow);

CreateFloatWME(leaflet, COLOR\_MAGENTA, targetMagenta);

CreateFloatWME(leaflet, COLOR\_WHITE, targetWhite);

1. Criação de um novo conjunto de regras no arquivo planning.soar:

Inicialmente o conjunto de regras foi alterado para integrar as regras *default,* copiando a pasta do mesmo nome dos exemplos do tutorial e adicionando uma regra para o carregamento do arquivo selection.soar. O software VisualSoar foi utilizado para edição das regras para possibilitar a divisão das regras em arquivos separados, simplificando o fluxo do trabalho.

Seguindo o tutorial 5 do Soar, os seguintes passos foram seguidos para implementar a estratégia deliberativa para solução do problema:

* Criação de um estado inicial (executado no arquivo initialize-planning.soar).
* Criação de condições para sucesso, definido como o momento em que a quantidade de jóias no knapsack se torna a mesma quantidade de jóias especificadas como target dos 3 leaflets.
* Criação de condição de falha. Utilizei para tanto a repetição de estados já presentes na pilha de estados.
* Re-utilização de proposta de operadores exemplo.Re-utilizei no caso os operadores *wander*, os dois operadores de memorização (*see entity)* e os operadores de movimentação e obtenção de jóias (*move e get jewel)*.
* Remoção de priorização de operadores, para provocar impasses e utilizar a simulação de operadores em etapas de simulação para resolver o problema.

Executando a simulação obtive alguns problemas de crash do sistema, quando a criatura encontrou com uma jóia no ambiente e iria colocar no knapsack:

s3dproxy.CommandExecException: @@@ Thing to grasp is missing

ws3dproxy.CommandExecException: @@@ Thing to grasp is missing

at ws3dproxy.CommandUtility.checkIfErrorMessage(CommandUtility.java:1268)

at ws3dproxy.CommandUtility.sendCmdAndGetResponse(CommandUtility.java:1276)

at ws3dproxy.CommandUtility.sendPutInSack(CommandUtility.java:265)

at ws3dproxy.model.Creature.putInSack(Creature.java:598)

at SoarBridge.SoarBridge.processGetCommand(SoarBridge.java:504)

at SoarBridge.SoarBridge.processCommands(SoarBridge.java:457)

at SoarBridge.SoarBridge.step(SoarBridge.java:408)

at Simulation.Main.<init>(Main.java:47)

at Simulation.Main.main(Main.java:65)

Apr 25, 2018 10:47:16 PM Simulation.Main <init>

SEVERE: Unknown errorws3dproxy.CommandExecException: @@@ Thing to grasp is missing

Apesar da resolução incompleta do problema, pude verificar na simulação o procedimento utilizado no look-ahead do processo de planning do Soar.

**Conclusões**

Ao longo do desenvolvimento deste trabalho cheguei às seguintes conclusões relativas à solução da Atividade 2 utilizando uma estratégia deliberativa:

* A documentação do Tutorial 5 do Soar é deficiente. Seriam necessários exemplos mais claros de como implementar o processo de planning no Soar.
* O sistema oferecido para simulação é mais intuitivo que o debugger do Soar. No entanto, existem alguns bugs que precisam ser evitados para seu uso.
* A estratégia deliberativa não é facilmente implementada no Soar. Para tanto, é necessário se utilizar de criação de estados de simulação internos.