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, foi estudado em sala um exemplo de um controlador que utiliza o SOAR de forma reativa para a tomada de decisões. Foram constatadas 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 ambiente de simulação e entra 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);
```

2. Criação de um novo conjunto de regras no arquivo planning.soar

2.1 Proposta 1: Implementação de look-ahead planning conforme o tutorial 5 do 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.
- Reutilização de proposta de operadores exemplo.Reutilizei 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.

Utilizando a estratégia acima, não obtive sucesso na execução do programa. Acredito que a falha ocorreu na implementação das rotinas de evaluation dos operadores. Considerei, no entanto, partir para uma nova proposta de implementação, utilizando planejamento combinado com uma proposta reativa.

2.2 Proposta 2: Implementação de planejamento utilizando proposta reativa.

Nesta segunda proposta procurei implementar regras para a movimentação da criatura procurando restringir as ações de acordo com o planejamento.

O planejamento a ser utilizado consiste em implementar regras para seguir o seguinte fluxo:

1. A criatura apenas está interessada em uma quantidade fixa de jóias de cada cor conforme especificado na estrutura do leaflet.
2. A criatura dispõe de 3 leaflets com quantidades de requisitos de jóias diferentes. A estratégia seguida será de coletar a soma dos requisitos dos 3 leaflets ao invés de coletar um leaflet por vez.
3. A criatura irá coletar as jóias percebidas pelo sistema visual, não será fornecida uma lista com conhecimento *a priori* antes da exploração do ambiente.

4. A criatura irá coletar uma jóia por vez e apenas irá reter um registro em memória do objetivo final.

Para tanto, implementei as seguintes propostas e aplicação de operadores:

- search-and-hold.soar

  Um conjunto de regras que procura no ambiente da criatura a jóia mais próxima que faça parte do conjunto de objetivos de jóias a serem coletadas de acordo com a informação do leaflet.

  Quando uma jóia com essa característica é encontrada, ela é adicionada como entidade da memória para posterior coleta. Note que o planejamento implementado aqui foi o de definir como objeto de busca apenas jóias que a criatura tenha interesse e coletar apenas uma jóia por vez.

  As condições explicadas acima são implementadas como regras conforme abaixo:

```
# First condition is, is there a jewel in the visual field of the
creature?
   (<creature> ^SENSOR.VISUAL.ENTITY <entity>)
   (<entity> ^TYPE <type> JEWEL)
   (<entity> ^COLOR <color>)
   (<entity> ^X <x>)
   (<entity> ^Y <y>)
   (<entity> ^NAME <name>)
# Second condition is, are there any entities in memory?
# The strategy is to only keep one entity in memory per round of search-
and-get.
   (<creature> ^MEMORY <memory>)
  -(<memory> ^ENTITY.NAME <name>)
# Third condition is, do we still need to get a jewel of that color?
   (<target>   ^<color> <tgtAmmount>)
   (<target>   ^<color> { <tgtAmmount> > 0 })
```

- go.soar

  Regras para movimentação da criatura até a posição da jóia registrada em memória. A única particularidade desta regra é que apenas uma jóia por vez será coletada. A especificação das propriedades da jóia-alvo é passada como elemento de memória:

```
   (<il> ^CREATURE.MEMORY <memory>)
   (<memory> ^ENTITY <entity>)
   (<entity> ^NAME <name>)
   (<entity> ^X <entityX>)
   (<entity> ^Y <entityY>)
```

- collect.soar

  Regras para a coleta da jóia. São utilizadas estruturas para registrar o total de jóias coletadas e o total de jóias que ainda devem ser coletadas. Quando uma coleta é realizada de forma bem sucedida, as estruturas de memória são atualizadas para realizar a entrega ao final.

```
  (<sack> ^<color> <sackQty>
            ^Red <sRedAmmount>
            ^Green <sGreenAmmount>
            ^Blue <sBlueAmmount>
            ^Yellow <sYellowAmmount>
            ^Magenta <sMagentaAmmount>
```

```
          ^White <sWhiteAmmount>)
   (<target> ^<color> <tgtQty>
             ^Red <tRedAmmount>
             ^Green <tGreenAmmount>
             ^Blue <tBlueAmmount>
             ^Yellow <tYellowAmmount>
             ^Magenta <tMagentaAmmount>
             ^White <tWhiteAmmount>)

...

   (<sack> ^<color> <sackQty> -) # Update knapsack quantity
   (<sack> ^<color> (+ <sackQty> 1))
   (<target> ^<color> <tgtQty> -)
   (<target> ^<color> (- <tgtQty> 1))
```

3. Resultados

A simulação do sistema proposto no visual debugger do SOAR mostra que o fluxo das operações é correto até o final da coleta das jóias. No entanto, encontrei um problema intermitente que não consegui resolver quando o sistema é testado com o WordServer3D.

Na IDE do Netbeans o crash ocorre conforme abaixo quando é realizada uma coleta de uma jóia pela criatura:

```
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
```

Notei que a passagem de dados para o comando de *GET* está correto, imagino que o problema acima seja causado por alguma incompatibilidade de biblioteca. O mesmo problema ocorre nos sistemas operacionais Linux e Windows.

4. 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.
- É possível implementar uma estratégia deliberativa utilizando regras, sem se utilizar o processo de look-ahead do SOAR. Para tanto, uma estratégia deve ser traçada desde o início do processo de coleta.
- Para uma criatura no mundo real, a estratégia de uso de regras me parece estar mais próximo de como a tomada de decisão ocorre para seres humanos: com informação incompleta e estimação de caminhos à medida que o processo de busca é realizado. Baseei nesta suposição a minha estratégia de implementação final.