Relatório 05 - CST: Controlando o WorldServer3D

Leonardo de Oliveira Ramos RA171941

Junho 2018

1 Entrega

Nas pastas WS3DLeaflet e WS3DOrigin se encontram os códigos fontes da criatura para a tarefa de cumprir os leaflets e buscar comida e de desviar de parede para ir a um destino, respectivamente.

Para executar apenas abra a pasta bin e digitar make run ou make runl para rodar o WS3DLeaflet e make runo para o WS3DOrigin, make stop para matar o processo do WS3D e make restart pra reiniciar o programa.

2 WS3DLeaflet

Este código faz a criatura ter o comportamento de ir até as jóias de seu leaflet ou ir atrás de uma maçã se sua energia estiver abaixo de 40%, e pegar qualquer jóia, ou comendo uma comida que obstruir sua passagem.

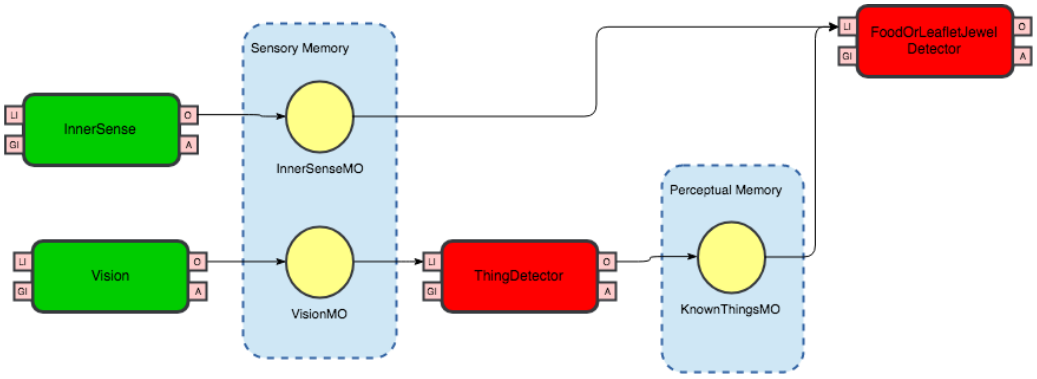

Figure 1: Primeira parte do ciclo cognitivo do WS3DLeaflet.

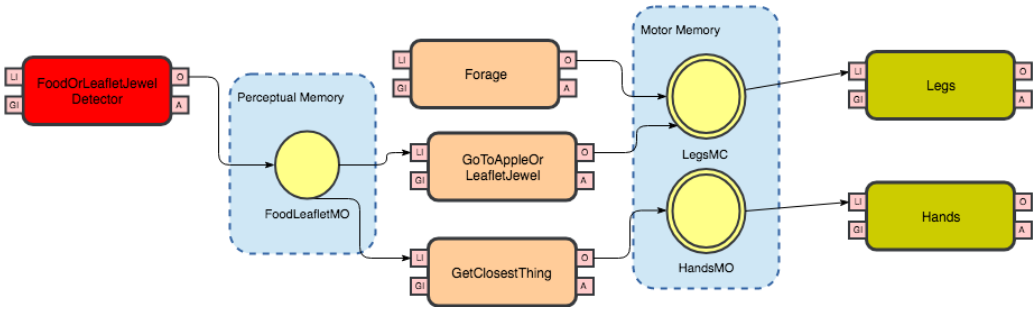

Figure 2: Segunda parte do ciclo cognitivo do WS3DLeaflet.

2.1 FoodOrLeafletDetector

Este codelet é responsável por detectar o que ele deve pegar, jóia do leaflet, ou comida caso sua energia esteja abaixo de 40%.

2.2 Forage

O comportamento do Forage foi alterado para sempre mandar o comportamento de forage, assim só toma outra ação se ela tiver mais prioridade.

2.3 GoToAppleOrLeafletJewel

Envia a ação de ir até a coisa mais próxima da lista enviada por FoodOrLeafletDetector.

2.4 GetClosestThing

Pega (ou come) qualquer coisa muito próxima ou obstruindo sua passagem.

3 WS3DOrigin

Este código implementa o comportamento de ir até a origem do mapa (0,0), desviando de qualquer parede em seu caminho.

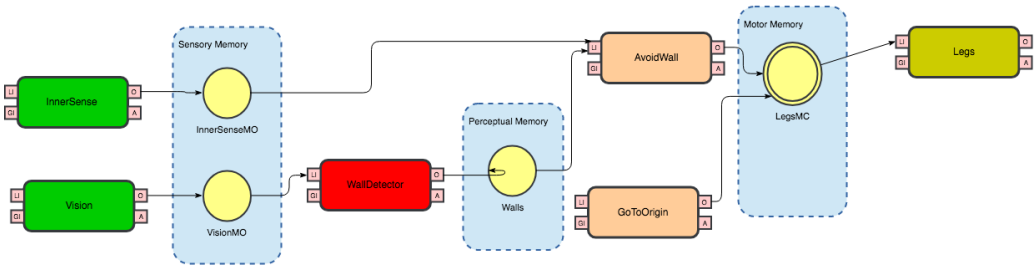

Figure 3: Ciclo cognitivo do WS3DOrigin.

3.1 WallDetector

Para isso esta classe detecta e guarda as paredes que já foram vistas pela criatura.

3.2 AvoidWall

Este comportamento seleciona a parede que tem que ser desviada e propõe a ação a ser tomada para desviar da parede (ir até a quina da parede depois atravessar pelo lado).

3.3 GoToOrigin

Sempre envia a ação de ir até a origem.

4 Considerações Finais

Esta foi a arquitetura que deu mais liberdade de programação e que faz mais sentido desenvolver um tipo qualquer de comportamento de uma criatura. A única comparação razoável seria com o SOAR, que tem uma forma diferente de desenvolvimento, facilitando decisões por comparação, mas não é a maioria dos casos que o SOAR facilitaria. Mesmo assim esta arquitetura cabe a utilização do SOAR internamente para tomada de decisões, na parte de behavior, possivelmente. Já em comparação com Lida e Clarion, acredito que cst seja a melhor arquitetura.

A única crítica envolvida (apesar de não ser conceitual, e neste cenário não fez diferença) é no framework, sobre uma perspectiva de desempenho, deve haver alguma maneira que não faça comparação de String, ou nenhuma comparação, para a passagem dos MemoryObjects.