Nome: Fabio Grassiotto

RA: 890441

Disciplina: IA941A, 1º S 2018

Aula 9 – Clarion: Controlando o WorldServer3D

**Objetivo**

Utilizar a arquitetura cognitiva Clarion para controlar uma criatura artificial no ambiente virtual WordServer3D (WS3D) através de implementação de um aplicativo utilizando linguagem de programação C# para execução em ambiente .Net ou Mono.

Serão implementados dois testes:

* Teste 1: Através da Clarion, será criada de uma criatura no WS3D que irá executar a coleta de jóias no ambiente de simulação de acordo com uma especificação prévia de quantidades e cores. Ao final do processo de coleta a criatura será deslocada para um posição determinada de entrega das jóias (Delivery Spot), convencionada na origem do ambiente (0,0).
* Teste 2: Será executada uma competição entre a criatura descrita acima e uma segunda criatura controlada pela arquitetura cognitiva Soar.

Para execução dos dois testes, serão providos scripts nos sistemas operacionais Linux e Windows:

run-test1.sh / run-test1.bat

run-test2.sh / run-test2.bat

**Descrição da implementação**

Esta seção do relatório visa descrever as principais alterações necessárias ao código fonte do DemoClarion.

A estratégia para implementação seguiu o modelo do ClarionDemo. Regras condicionais no top level do ACS foram definidas, com o uso de níveis de ativação para os inputs do ambiente.

O algoritmo empregado para a busca das jóias implementa as seguintes regras:

* Computar inicialmente a quantidade de jóias-objetivo de acordo com a entrada do leaflet menos a quantidade de jóias no knapsack para posterior processamento.
* Examinar o ambiente próximo à posição da criatura. Se for percebida a presença de jóias ou comidas próximas à criatura (distância menor de 31 unidades), executar a ação de coleta.
* Examinar o ambiente distante da criatura. Caso jóias distantes façam parte do objetivo, direcionar a criatura para a jóia.
* Caso a criatura esteja com a quantidade de combustível abaixo de 40 unidades e um ítem de comida esteja presente no campo visual distante, direcionar a criatura para a comida.
* Caso nenhuma das situações acima ocorra, instruir a criatura a permanecer parada girando no próprio eixo até que apareçam novas entidades em seu campo visual.
* Quando o objetivo final de jóias coletadas for atingido, direcionar a criatura para as coordenadas do delivery spot (por convenção em (0,0)).

Foram assim realizadas as seguintes alterações no código da classe ClarionAgent, no arquivo ClarionAgent.cs:

**1. Adição de novas ações para a criatura:**

Foram adicionadas as ações de direcionamento até entidades (GO\_TO), coleta (GET), busca no ambiente (WANDER) e entrega das jóias (DELIVER).

public enum CreatureActions

{

DO\_NOTHING,

ROTATE\_CLOCKWISE,

**GO\_TO\_JEWEL,**

**GO\_TO\_FOOD,**

**GET\_JEWEL,**

**GET\_FOOD,**

**WANDER,**

**DELIVER**

}

**2. Adição de novas constantes para valores nos pares de dimensão-valor:**

private String DIMENSION\_JEWEL\_AHEAD = "JewelAhead";

private String DIMENSION\_JEWEL\_AWAY = "JewelAway";

private String DIMENSION\_FOOD\_AHEAD = "FoodAhead";

private String DIMENSION\_FOOD\_AWAY = "FoodAway";

**3. Adição de constantes para os níveis de ativação das entradas**

private double JEWEL\_AHEAD\_ACT\_VAL = 1.0;

private double JEWEL\_AWAY\_ACT\_VAL = 0.9;

private double FOOD\_AHEAD\_ACT\_VAL = 0.8;

private double FOOD\_AWAY\_ACT\_VAL = 0.6;

private double WALL\_AHEAD\_ACT\_VAL = 0.3;

private double MIN\_ACT\_VAL = 0.0;

**4. Adição de pares de dimensão-valor para descrever os objetos do ambiente**

Para a simulação do WS3D, os pares relevantes são a presença de jóias e comidas na proximidade ou à distância da criatura.

private DimensionValuePair inputJewelAhead;

private DimensionValuePair inputFoodAhead;

private DimensionValuePair inputJewelAway;

private DimensionValuePair inputFoodAway;

**5. Adição de ações sobre o ambiente a serem executados pela criatura**

private ExternalActionChunk outputGetJewel;

private ExternalActionChunk outputGetFood;

private ExternalActionChunk outputGoToJewel;

private ExternalActionChunk outputGoToFood;

private ExternalActionChunk outputWander;

private ExternalActionChunk outputGoToDeliverySpot;

**6. Adição de interfaces com o WS3D para execução de ações externas no método** **processSelectedActions()**

case CreatureActions.GET\_JEWEL:

worldServer.SendSackIt(creatureId, jewelToGet.Name);

break;

case CreatureActions.GET\_FOOD:

worldServer.SendEatIt(creatureId, foodToGet.Name);

break;

case CreatureActions.GO\_TO\_JEWEL:

worldServer.SendSetAngle(creatureId, 0, 0, prad);

worldServer.SendSetGoTo(creatureId, 1, 1, jewelToGoTo.X1, jewelToGoTo.Y1);

reak;

case CreatureActions.GO\_TO\_FOOD:

worldServer.SendSetAngle(creatureId, 0, 0, prad);

worldServer.SendSetGoTo(creatureId, 1, 1, foodToGoTo.X1, foodToGoTo.Y1);

break;

case CreatureActions.WANDER:

worldServer.SendSetAngle(creatureId, 2, -2, 2);

break;

case CreatureActions.DELIVER:

// Send creature to the delivery spot.

worldServer.SendSetAngle(creatureId, 0, 0, prad);

worldServer.SendSetGoTo(creatureId, 1, 1, deliverySpot.X1, deliverySpot.Y1);

break;

**7. Alterações no método prepareSensoryInformation()**

Este é o método principal para a implementação da lógica de níveis de ativação. As seguintes alterações foram implementadas:

* Implementação do método updateSackAndTarget, que agrega as quantidades de jóias por cor que ainda precisam ser coletadas pela criatura.

Sack sack;

targetRed = 0;

targetGreen = 0;

targetBlue = 0;

targetYellow = 0;

targetMagenta = 0;

targetWhite = 0;

// Update leaflets

int n = 0;

foreach (Leaflet l in c.getLeaflets())

{

targetRed += l.getRequired("Red");

targetGreen += l.getRequired("Green");

targetBlue += l.getRequired("Blue");

targetYellow += l.getRequired("Yellow");

targetMagenta += l.getRequired("Magenta");

targetWhite += l.getRequired("White");

mind.updateLeaflet(n, l);

n++;

}

if (worldServer != null && worldServer.IsConnected)

{

sack = worldServer.SendGetSack("0");

targetRed -= sack.red\_crystal;

targetGreen -= sack.green\_crystal;

targetBlue -= sack.blue\_crystal;

targetYellow -= sack.yellow\_crystal;

targetMagenta -= sack.magenta\_crystal;

targetWhite -= sack.white\_crystal;

}

* Implementação de detecção de entidades próximas à criatura, configurando níveis de ativação:

// Loop through the list of things in the environment.

// First, handle close objects.

foreach (Thing thing in listOfThings)

{

// Thing is close to the creature, so change activation values accordingly.

int categoryId = thing.CategoryId;

if (thing.DistanceToCreature <= 50 && categoryId != Thing.CATEGORY\_CREATURE)

{

switch (categoryId)

{

case Thing.CATEGORY\_BRICK:

wallAheadActivationValue = WALL\_AHEAD\_ACT\_VAL;

break;

case Thing.CATEGORY\_JEWEL:

jewelAheadActivationValue = JEWEL\_AHEAD\_ACT\_VAL;

jewelToGet = thing;

break;

case Thing.categoryPFOOD:

foodAheadActivationValue = FOOD\_AHEAD\_ACT\_VAL;

foodToGet = thing;

break;

default:

break;

}

}

}

* Implementação de lógica para detecção de jóias no ambiente para completar a coleta especificada na leaflet. Uma lógica similar foi implementada para coleta de comida.

// Look now for the closest jewel to go to.

IEnumerable<Thing> jewels = listOfThings.Where(item => (item.CategoryId == Thing.CATEGORY\_JEWEL && item.DistanceToCreature > 50));

if (jewels.Any())

{

IEnumerable<Thing> orderedJewels = jewels.OrderBy(item => item.DistanceToCreature);

Boolean foundJewel = false;

foreach (Thing jewel in orderedJewels)

{

// Check if the jewel is required, otherwise skip.

if ((jewel.Material.Color.Equals("Red") && targetRed > 0) ||

(jewel.Material.Color.Equals("Green") && targetGreen > 0) ||

(jewel.Material.Color.Equals("Blue") && targetBlue > 0) ||

(jewel.Material.Color.Equals("Yellow") && targetYellow > 0) ||

(jewel.Material.Color.Equals("Magenta") && targetMagenta > 0) ||

(jewel.Material.Color.Equals("White") && targetWhite > 0))

{

jewelAwayActivationValue = JEWEL\_AWAY\_ACT\_VAL;

jewelToGoTo = jewel;

// Found one jewel as target, no need to keep looking.

Console.WriteLine("Jewel to go to: " + jewel.Name);

Console.WriteLine("Jewel color: " + jewel.Material.Color);

break;

}

}

* Implementação de lógica para detectar que a coleção de jóias foi finalizada:

// Verify if all collection of jewels is done.

if (targetRed <= 0 && targetGreen <= 0 && targetBlue <= 0 &&

targetYellow <= 0 && targetMagenta <= 0 && targetWhite <= 0)

{

AllJewelsCollected = true;

}

**Resultados**

Foi possível simular a coleta das jóias com sucesso pela criatura controlada pela Clarion, assim como executar uma competição entre a Clarion e o SOAR.

**Dificuldades Encontradas**

Algumas dificuldades foram encontradas ao longo da implementação do código:

* Execução no ambiente Windows não é trivial, necessitando alterar segmentos da classe principal para carregar DLLs em caminhos apropriados.
* Instabilidade da comunicação com WS3D, provocando alguns crashes no ambiente Windows.

**Futuras Melhorias**

Algumas melhorias na implementação poderiam ser realizadas, tais como:

* Detecção de proximidade de outras criaturas, para as situações de competição.
* A criatura ocasionalmente não consegue detectar entidades próximas provavelmente devido ao ângulo de visão, criando situações de travamento de movimentação. Para resolver este problema, seria necessário ajustar a distância máxima de coleta.