

## Aula 10 – 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 item 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)).
- Quando a criatura estiver posicionada próxima ao delivery spot, executar a entrega das jóias.

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, GO\_TO\_DELIVER), 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,
    GO_TO_DELIVER,
    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";
private String DIMENSION_ALL_JEWELS_COLLECTED = "AllJewelsCollected";
```

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

```
private double CREATURE_CAN_DELIVER_ACT_VAL = 1.0;
private double ALL_JEWELS_COLLECTED_ACT_VAL = 0.9;
private double JEWEL_AHEAD_ACT_VAL = 0.8;
private double JEWEL_AWAY_ACT_VAL = 0.7;
private double FOOD_AHEAD_ACT_VAL = 0.6;
private double FOOD_AWAY_ACT_VAL = 0.5;
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;
private DimensionValuePair inputAllJewelsCollected;
private DimensionValuePair inputCreatureCanDeliver;
```

## 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;  
private ExternalActionChunk outputDoDelivery;
```

## 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, 2, 2, jewelToGoTo.X1, jewelToGoTo.Y1);  
    break;  
case CreatureActions.GO_TO_FOOD:  
    worldServer.SendSetAngle(creatureId, 0, 0, prad);  
    worldServer.SendSetGoTo(creatureId, 2, 2, foodToGoTo.X1, foodToGoTo.Y1);  
    break;  
case CreatureActions.WANDER:  
    worldServer.SendSetAngle(creatureId, 2, -2, 2);  
    break;  
case CreatureActions.GO_TO_DELIVER:  
    // Send creature to the delivery spot.  
    worldServer.SendSetAngle(creatureId, 0, 0, prad);  
    worldServer.SendSetGoTo(creatureId, 2, 2, deliverySpot.X1, deliverySpot.Y1);  
    break;  
case CreatureActions.DELIVER:  
    // Deliver jewels  
    worldServer.SendDeliver(creatureId, leaflet1Id);  
    worldServer.SendDeliver(creatureId, leaflet2Id);  
    worldServer.SendDeliver(creatureId, leaflet3Id);  
    worldServer.SendStopCreature(creatureId);  
    // Delete the Agent  
    Abort(true);  
    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 = null;  
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");
    if (n == 0) {
        leaflet1Id = l.leafletID.ToString();
    } else if (n == 1) {
        leaflet2Id = l.leafletID.ToString();
    } else {
        leaflet3Id = l.leafletID.ToString();
    }
    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;
}
Return sack;

```

- 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:
            case Thing.CATEGORY_NPFOOD:
            case Thing.CATEGORY_FOOD:
                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)
{
allJewelsCollectedActivationValue = ALL_JEWELS_COLLECTED_ACT_VAL;
Console.WriteLine("Go to delivery, all items collected");
}
}
```

## 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. Para execução nos ambientes Windows e Linux, é preciso manter pastas separadas com binários para cada sistema operacional.
- Instabilidade da comunicação com WS3D, provocando alguns crashes no ambiente Windows. Nota-se que os mesmos crashes não ocorrem no Linux.

## **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.