UNICAMP
Universidade Estadual de Campinas

FEEC
Faculdade de Engenharia Elétrica e de Computação

Aluno: Mateus Neves Barreto
R.A.: 142358
Disciplina: IA006
Professor: Ricardo R. Gudwin

Relatório – Aula 10 e 11

1 Agente CLARION aplicado ao WorldServer 3D

O objetivo da aula 10 e 11 foi desenvolver um agente CLARION capaz de buscar *leaflets* do mundo WorldServer3D e cumprir da melhor forma a busca por joias. Cada *leaflet* possui uma recompensa e um número determinado de joias a ser buscado. Então a partir desta informação foi desenvolvido um agente CLARION que toma a decisão da movimentação da criatura no mundo WorldServer3D. Para realizar esta tarefa, o código ClarionDEMO foi usado como base.

A Figura 1 apresenta o trecho de código onde é possível verificar a preferência que o agente deve seguir. Ressaltando que a estrutura montada na orientação da linguagem foi a seguinte:

- Criatura: a criatura possui objetivos, Figura 2;
  - Objetivo: cada objetivo possui uma recompensa e *leaflets*, Figura 3;
    - *Leaflet:* cada *leaflet* é a joia a ser buscada do objetivo, que possui um número desejado e um número já atingido, Figura 4;

Observando o código da Figura 1, primeiramente é percorrido a lista de todos os objetivos, onde cada objetivo se verifica se o mesmo já foi cumprido. Na mesma iteração é selecionado o objetivo que retorna uma maior recompensa. Em seguida filtra-se os *leaflets* dos objetivos que ainda não foram cumpridos verificado pelo método: continuar procurando, Figura 4. Após seleciona os *leaflets* que a criatura ainda deve buscar, verifica-se a existência dessas joias no mundo WorldServer3D. Se as joias procuradas existirem no mundo, existem duas opções:

- Se a joia estiver próxima, a preferência é de pegá-la;
- Se a joia estiver longe, a preferência será buscá-la.

Porém se as joias buscadas pelo agente não existirem no mundo, o método “buscaTudo” é chamado, Figura 5.

```
} else if (creature != null && creature.cumpriuLeaflets()) {
int numLeaflets=creature.NumberOfLeaflets;
Objetivo obj=null;
int recompensa=int.MinValue;
int posicaoMaiorRecompensa=0;
for (int i = 0; i < numLeaflets; i++){
    obj=creature.Objetivos[i];
    if(!obj.objetivoCumprido() && recompensa<obj.recompensa){
        recompensa=obj.recompensa;
        posicaoMaiorRecompensa=i;
    }
}
obj=creature.Objetivos[posicaoMaiorRecompensa];
//List<Leaflet> procurados=new List<Leaflet>();
List<String> procurados=new List<string>();
foreach (Leaflet leaflet in obj.leaflets) {
    if(leaflet.continuarProcurando()){
        procurados.Add(leaflet.Color);
    }
}
List<Thing> jewelNoMundo=new List<Thing>();
foreach (Thing coisa in sensorialInformation) {
    if(coisa.CategoryId==Thing.CATEGORY_JEWEL
       && procurados.Contains(coisa.Material.Color)){
        jewelNoMundo.Add(coisa);
    }
}
if(jewelNoMundo.Count>0){
    Thing jewelMP=jewelNoMundo[0];
    foreach (Thing joia in jewelNoMundo) {
        if(jewelMP.DistanceToCreature >= joia.DistanceToCreature){
            jewelMP=joia;
        }
    }
    if(jewelMP.DistanceToCreature <=25.0){
        si.Add (InputTakejewel, 1.0);
        si.Add (InputEatFood, 0.0);
        si.Add (InputMove_food, 0.0);
        si.Add (InputMove_jewel, 0.0);
        si.Add (InputRotate, 0.0);
    }else{
        si.Add (InputTakejewel, 0.0);
        si.Add (InputEatFood, 0.0);
        si.Add (InputMove_food, 0.0);
        si.Add (InputMove_jewel, 1.0);
        si.Add (InputRotate, 0.0);
    }
}else{
    si= buscaTudo (sensorialInformation, si);
}
```

Figura 1 – Buscando *Leaflets*.

```
namespace WorldServerLibrary.Model
{
    public enum MotorSystemType
    {
        CAR,
        TWO_WHEEL
    }

    public class Creature:Thing
    {
        public Int32 index { get; set; }
        public MotorSystemType MotorSystem { get; set; }
        public double Wheel { get; set; }
        public double Speed { get; set; }
        public double Fuel { get; set; }
        public Boolean HasLeaflet { get; set; }
        public Int32 NumberOfLeaflets { get; set; }
        public List<Objetivo> Objetivos { get; set; }

        public Creature ()
        {
            Objetivos=new List<Objetivo>();
            CategoryId=0;
        }
        public Boolean cumpriuLeaflets ()
        {
            foreach (Objetivo obj in Objetivos) {
                if(!obj.objetivoCumprido()){
                    return false;
                }
            }
            return true;
        }
    }
}
```

Figura 2 – Criatura.

```
namespace WorldServerLibrary.Model
{
    public class Objetivo
    {   public Int32 numObj { get; set; }
        public Int32 recompensa { get; set; }
        public List<Leaflet> leaflets { get; set; }

        public Objetivo ()
        {
            leaflets=new List<Leaflet> () ;
        }
        public Boolean objetivoCumprido ()
        {
            foreach (Leaflet item in leaflets) {
                if(item.numDesejado-item.numDesejado!=0){
                    return false;
                }
            }
            return true;
        }
    }
}
```

Figura 3 – Objetivo.

```
namespace WorldServerLibrary.Model
{
    public class Leaflet
    {
        public String Color { get; set; }
        public int numDesejado { get; set; }
        public int numCapturado { get; set; }

        public Leaflet ()
        {
        }
        public Boolean continuarProcurando ()
        {
            if(numDesejado-numCapturado!=0){
                return true;
            }
            return false;
        }
    }
}
```

Figura 4 – Leaflet.

```
SensoryInformation buscaTudo (IList<Thing> sensorialInformation, SensoryInformation si)
{
    if (sensorialInformation.Where (item => ((item.CategoryId == Thing.CATEGORY_JEWEL)
                                             && item.DistanceToCreature <= 25.0)).Any ()) {
        si.Add (InputTakejewel, 1.0);
        si.Add (InputEatFood, 0.0);
        si.Add (InputMove_food, 0.0);
        si.Add (InputMove_jewel, 0.0);
        si.Add (InputRotate, 0.0);
    }
    else {
        si.Add (InputTakejewel, 0.0);
        if (sensorialInformation.Where (item => ((item.CategoryId == Thing.CATEGORY_FOOD
                                                  || item.CategoryId == Thing.CATEGORY_NPFOOD
                                                  || item.CategoryId == Thing.categoryPFOOD)
                                                 && item.DistanceToCreature <= 25.0)).Any ()) {
            si.Add (InputEatFood, 1.0);
            si.Add (InputMove_food, 0.0);
            si.Add (InputMove_jewel, 0.0);
            si.Add (InputRotate, 0.0);
        }
        else {
            si.Add (InputEatFood, 0.0);
            if (sensorialInformation.Where (item => ((item.CategoryId == Thing.CATEGORY_JEWEL)
                                                     && item.DistanceToCreature > 25.0)).Any ()) {
                si.Add (InputMove_food, 0.0);
                si.Add (InputMove_jewel, 1.0);
                si.Add (InputRotate, 0.0);
            }
            else {
                si.Add (InputMove_jewel, 0.0);
                if (sensorialInformation.Where (item => ((item.CategoryId == Thing.CATEGORY_FOOD
                                                          || item.CategoryId == Thing.CATEGORY_NPFOOD
                                                          || item.CategoryId == Thing.categoryPFOOD)
                                                         && item.DistanceToCreature > 25.0)).Any ()) {
                    si.Add (InputMove_food, 1.0);
                    si.Add (InputRotate, 0.0);
                }
                else {
                    si.Add (InputMove_food, 0.0);
                    si.Add (InputRotate, 1.0);
                }
            }
        }
    }
    return si;
}
```

Figura 5 –Método: *buscaTudo()*.

O método apresentado na Figura 5 é utilizado quando as joias buscadas pelo agente não existem no mundo. Assim, a criatura deve começar a pegar e comer qualquer joia e comida que esteja em sua visão. Mas mesmo assim ainda é mantida a preferência por buscar joias antes que comida.
O código que antecede o código da Figura 1 possui a função de evitar obstáculos como joias e comidas que estão no caminho. Deste modo comidas e joias que estiverem próximos são capturados pela criatura, Figura 6.

```
if (sensorialInformation.Where (item => ((item.CategoryId == Thing.CATEGORY_JEWEL)
                                         && item.DistanceToCreature <= 30.0)).Any ()) {
    si.Add (InputTakejewel, 1.0);
    si.Add (InputEatFood, 0.0);
    si.Add (InputMove_food, 0.0);
    si.Add (InputMove_jewel, 0.0);
    si.Add (InputRotate, 0.0);
} else if (sensorialInformation.Where (item => ((item.CategoryId == Thing.CATEGORY_FOOD
                                                 || item.CategoryId == Thing.CATEGORY_NPFOOD
                                                 || item.CategoryId == Thing.categoryPFOOD)
                                                && item.DistanceToCreature <= 30.0)).Any ()) {
        si.Add (InputTakejewel, 0.0);
        si.Add (InputEatFood, 1.0);
        si.Add (InputMove_food, 0.0);
        si.Add (InputMove_jewel, 0.0);
        si.Add (InputRotate, 0.0);
    } else if (creature != null && creature.cumpriuLeaflets()) {
```

Figura 6 –Evitando obstáculos.

Foram criados métodos *delagates* que auxiliam nas a decisão das regras inseridas no agente, especificamente na estrutura ACS, Figura 7.

```
SupportCalculator moveF = FixedRuleDelegateToMoveFood;
FixedRule ruleAvoidMove =
    AgentInitializer.InitializeActionRule(CurrentAgent, FixedRule.Factory, OutputMove_food, moveF);
// Commit this rule to Agent (in the ACS)
CurrentAgent.Commit(ruleAvoidMove);

SupportCalculator moveJ = FixedRuleDelegateToMoveJewel;
FixedRule ruleAvoidMoveJewel =
    AgentInitializer.InitializeActionRule(CurrentAgent, FixedRule.Factory, OutputMove_jewel, moveJ);
// Commit this rule to Agent (in the ACS)
CurrentAgent.Commit(ruleAvoidMoveJewel);

SupportCalculator eatfoodSupportCalculator = FixedRuleDelegateToEatFood;
FixedRule ruleAvoidEatfood =
    AgentInitializer.InitializeActionRule(CurrentAgent, FixedRule.Factory, OutputEatfood, eatfoodSupportCalculator);
// Commit this rule to Agent (in the ACS)
CurrentAgent.Commit(ruleAvoidEatfood);

SupportCalculator takeJewelSupportCalculator = FixedRuleDelegateToTakeJewel;
FixedRule ruleAvoidTakeJewel =
    AgentInitializer.InitializeActionRule(CurrentAgent, FixedRule.Factory, OutputTakejewel, takeJewelSupportCalculator);
// Commit this rule to Agent (in the ACS)
CurrentAgent.Commit(ruleAvoidTakeJewel);

SupportCalculator rotateSupportCalculator = FixedRuleDelegateToRotate;
FixedRule ruleRotate =
    AgentInitializer.InitializeActionRule(CurrentAgent, FixedRule.Factory, OutputRotate, rotateSupportCalculator);
CurrentAgent.Commit(ruleRotate);
```

Figura 7 –Regras ACS.

Cada *delagate* criado, Figura 8, apresenta a decisão de uma regra a partir de condições do agente corrente.

```
private double FixedRuleDelegateToEatFood(ActivationCollection currentInput, Rule target)
{
    if((currentInput.Contains(InputEatFood, CurrentAgent.Parameters.MAX_ACTIVATION))){
        return 1.0;
    }else{
        return 0.0;
    }
}
private double FixedRuleDelegateToTakeJewel(ActivationCollection currentInput, Rule target)
{
    if((currentInput.Contains(InputTakejewel, CurrentAgent.Parameters.MAX_ACTIVATION))){
        return 1.0;
    }else{
        return 0.0;
    }
}

private double FixedRuleDelegateToMoveFood(ActivationCollection currentInput, Rule target)
{
    return ((currentInput.Contains(InputMove_jewel, CurrentAgent.Parameters.MIN_ACTIVATION))
            &&(currentInput.Contains(InputMove_food, CurrentAgent.Parameters.MAX_ACTIVATION))) ? 1.0 : 0.0;
}

private double FixedRuleDelegateToMoveJewel(ActivationCollection currentInput, Rule target)
{
    return ((currentInput.Contains(InputMove_jewel, CurrentAgent.Parameters.MAX_ACTIVATION))) ? 1.0 : 0.0;
}

private double FixedRuleDelegateToRotate(ActivationCollection currentInput, Rule target)
{
    if(currentInput.Contains(InputEatFood, CurrentAgent.Parameters.MIN_ACTIVATION)
       && (currentInput.Contains(InputTakejewel, CurrentAgent.Parameters.MIN_ACTIVATION))
       && (currentInput.Contains(InputMove_food, CurrentAgent.Parameters.MIN_ACTIVATION))
        && (currentInput.Contains(InputMove_jewel, CurrentAgent.Parameters.MIN_ACTIVATION)))
       {
        return 1.0;
    }else{
        return 0.0;
    }
}
```

Figura 8 –*Delegates*.

O código presente na Figura 9 possibilita a recepção dos *leaflets* do mundo real e em seguida inseri-los na lista de "Thing" que é repassada ao agente CLARION. Este código precisou ser completado a partir do exemplo ClarionDEMO, Figura 9.

```
SetAttribute(creature, "HasLeaflet", enumerator);
SetAttribute(creature, "NumberOfLeaflets", enumerator);

//  It is missing the parser of leaflet! ...
Leaflet leaf;
Objetivo obj;
if(creature.HasLeaflet){

    int numLeaflets=creature.NumberOfLeaflets;
    for (int i = 0; i < numLeaflets; i++){
        enumerator.MoveNext();
        obj=new Objetivo();
        SetAttribute(obj, "numObj", enumerator);
        for (int j = 0; j < obj.numObj; j++){
            leaf=new Leaflet();
            SetAttribute(leaf, "Color", enumerator);
            SetAttribute(leaf, "numDesejado", enumerator);
            SetAttribute(leaf, "numCapturado", enumerator);
            obj.leaflets.Add(leaf);
        }
        SetAttribute(obj, "recompensa", enumerator);
        creature.Objetivos.Add(obj);
    }
}
```

Figura 9 –Recebendo *leaflets*.

Para que o mundo WordServer3D fique sorteando joias e comidas durante um determinado intervalo de tempo, como na atividade do Agente SOAR. Foi aproveitado e adaptado o código desta atividade, Figura 10.

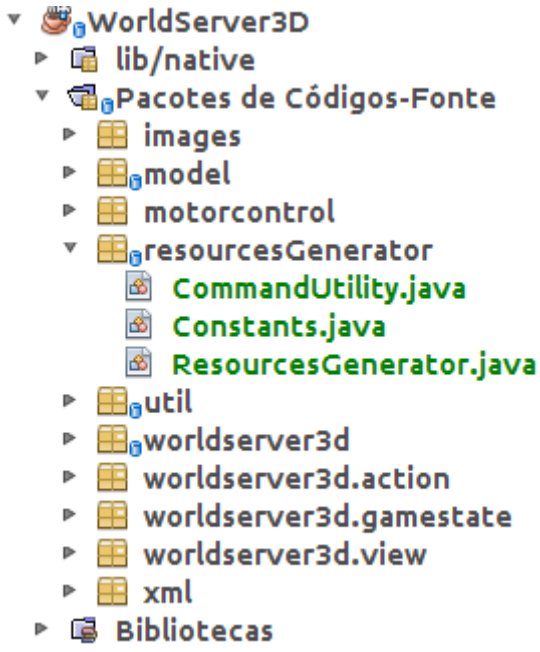

Figura 10 – Criando joias e comidas aleatórios.

O resultado desta adaptação pode ser observado no mundo WordServer3D em execução, Figura 11.

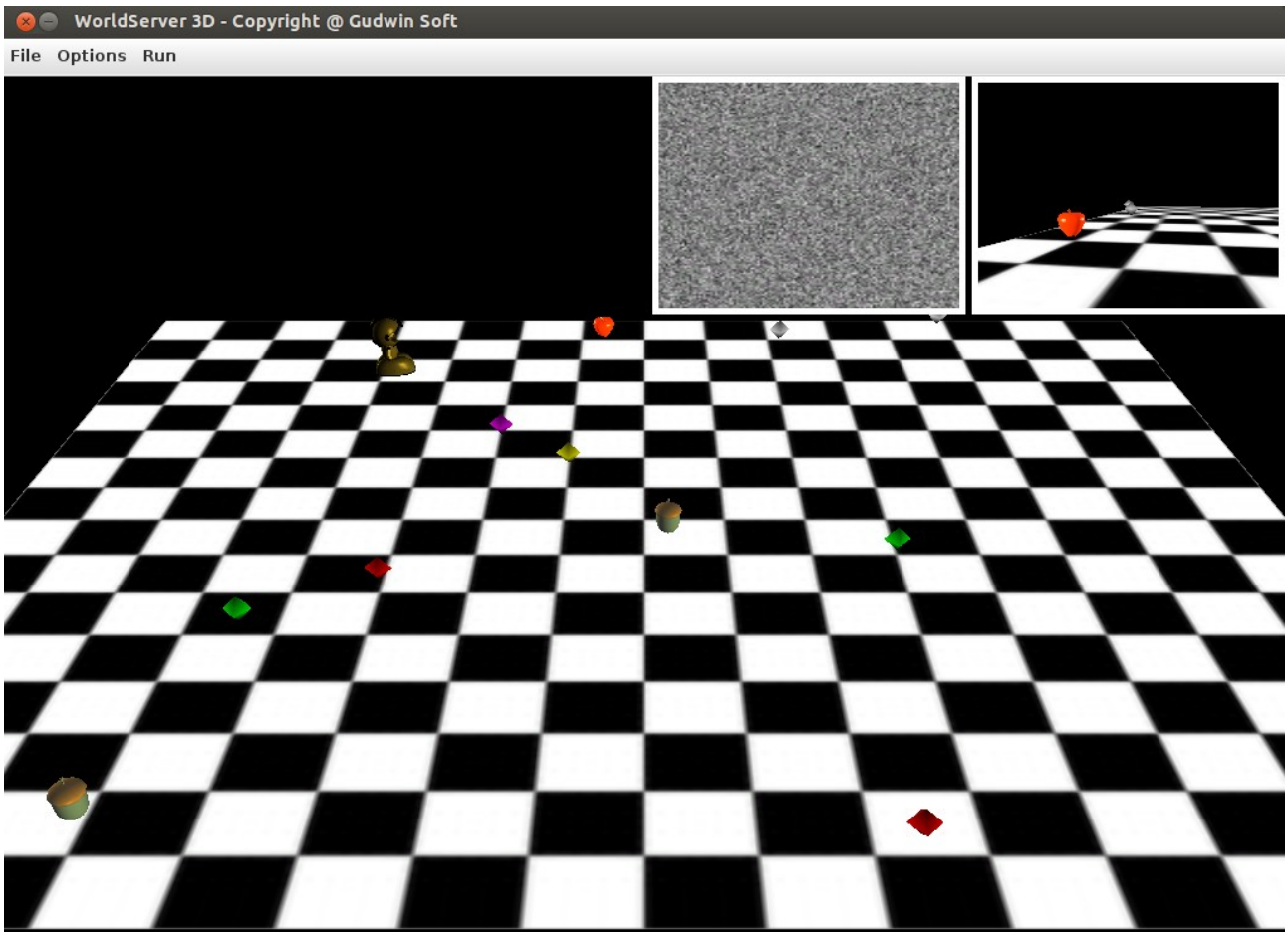

Figura 11 – Execução do mundo com adaptação.

Para usar esta adaptação, foi necessário umas modificações na classe Main, Figura 12 e 13.

```
public Main() {
    if (main == null) {
        main = this;
    }
    Logger.getLogger("com.jme").setLevel(Level.OFF);
    NativeUtils.setLibraryPath(".");
    NativeUtils.prepareNativeLibs();
    i = new WorldFrame();
    sf = new SimulationFrame(this);
    clientsConnected = new ArrayList<ServerThread>();
    try {
        ServerSocket ss = new ServerSocket(Constants.PORT);
        main.grow((int)1);
        while (true) {
            clientsConnected.add(new ServerThread(ss.accept()));
        }
    } catch (Exception e) {
        System.out.println("Error while trying to connect! " + e.toString());
    }
}
```

Figura 12 – Main – I.

```
public synchronized void grow(int time) {

    if (time <= 0) {
        time = Constants.TIMEFRAME;
    }
    getDimensionAndDeliverySpot();
    ResourcesGenerator rg = new ResourcesGenerator(time, environmentWidth, environmentHeight, xds, xds);
    rg.start();

}
public static synchronized void getDimensionAndDeliverySpot()  {
    StringTokenizer st = CommandUtility.sendGetSimulationParameters();

    if (st.hasMoreTokens()) {
        environmentWidth = Integer.parseInt(st.nextToken());
    } else {
        return;
    }
    if (st.hasMoreTokens()) {
        environmentHeight = Integer.parseInt(st.nextToken());
    } else {
        return;
    }
    if (st.hasMoreTokens()) {
        xds = Double.parseDouble(st.nextToken());
    } else {
        return;
    }
    if (st.hasMoreTokens()) {
        yds = Double.parseDouble(st.nextToken());
    } else {
        return;
    }
}
```

Figura 13 – Main - II.

Note que o método “grow(int)” é chamado após a inicialização da comunicação com o mundo WordServer3D através de socket, Figura 12. A Figura 13 apresenta uma adaptação dos métodos tirados do projeto “WS3DProxy”.

O resultado das implementações deste projeto podem ser visto no vídeo disponibilizado no ambiente de entrega de tarefas (este vídeo possui áudio).

A conclusão tirada deste trabalho foi que a estrutura CLARION é uma estrutura mais flexível que a a estrutura cognitiva SOAR.