JGOMAS

JADE Game Oriented MultiAgent System

Sistemas Inteligentes

FI, 2005

Toni Barella
tbarella@dsic.upv.es

Índice

- Introducción
- Descripción
- Especificación
- Conclusiones

Índice

Introducción (I)

- Plataforma de agentes sobre entornos 3D para simulaciones y videojuegos
- Básicamente…
  - Un puñado de agentes repartidos en dos bandos…
  - … con unos objetivos que cumplir…
  - … integrados en un entorno virtual

Introducción (II)

- Taxonomía de Agentes
- Componente Estratégico
- Integración en Entorno
  - Entre Agentes → cooperación
  - Con el Terreno → dificultad de movimiento
- Comunicación Estándar

Índice

Descripción

- Objetivo del juego
- Roles
- Estructura de la plataforma
- Ejecución

Objetivo (I)

- Disponemos de un número finito de agentes
- Cada agente pertenece a un bando
  - Aliados
  - Eje
- Los agentes aliados deben ir a la base del eje, capturar la bandera, y llevarla a su base
- Los agentes del eje deben defender la bandera y, en caso de ser capturada, devolverla a su base
- Hay un tiempo máximo para que los aliados lleven la bandera a su base

Objetivo (II)

- Comportamiento emergente como resultado de la actuación en grupo de los agentes
- Mejorar la inteligencia de los agentes en función de:
  - Especialización
  - Estrategia por objetivo
  - Generación de caminos
  - Etc…

Roles

- Hay definidos tres tipos de roles:
  - Soldier: acude a dar apoyo
  - Medic: acude a curar
  - FieldOps: acude a dar munición
- Un agente asume un único rol durante toda la partida
- Cada rol tiene unas características y ofrece unos determinados servicios

Estructura de la plataforma (I)

- A nivel de aplicación:
  - Sistema MultiAgente sobre JADE
  - Render Engine (Visualizador Gráfico)
- A nivel de Agentes:
  - Internos:
    - Manager: coordina todo el juego
    - Pack: paquetes de medicina, munición y objetivo
  - Externos:
    - Troop: agentes de usuario (roles)

Estructura de la plataforma (II)

- A nivel de Biblioteca:
  - Implementación:
    - Comportamientos
    - Métodos
    - Variables
  - Acceso:
    - Privados
    - Finales
    - Sobrecarga

Ejecución

- Línea de comandos:

```
java -classpath
lib\jade.jar;lib\jadeTools.jar;lib\Base64.jar;lib\http.jar;lib\iiop.j
ar;lib\beangenerator.jar;. jade.Boot -gui Manager:Classes.CManager(4
map_04 125 10) A1:Classes.CMedic(ALLIED) A2:Classes.CMedic(ALLIED)
E1:Classes.CMedic(AXIS) E2:Classes.CMedic(AXIS)
```

- Agente:

  `nombre:clase(parámetros)`

Manager:Classes.CManager(4 map_04 125 10)

Nombre Agente — Clase — Parámetros:

4: nº de agentes
map_04: nombre del mapa
125: refresco en ms.
10: duración (minutos)

A1:Classes.CMedic(ALLIED)

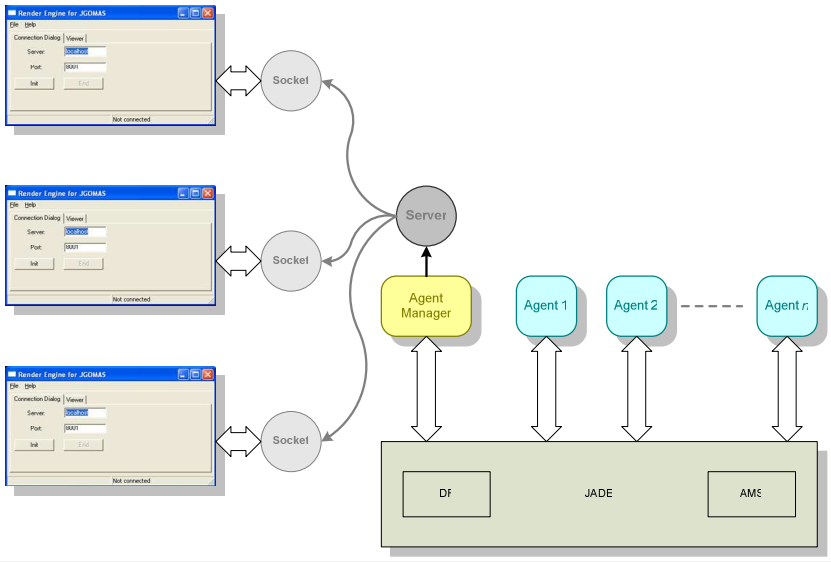
Nombre Agente — Clase — Parámetros: **BANDO**

Índice

Especificación

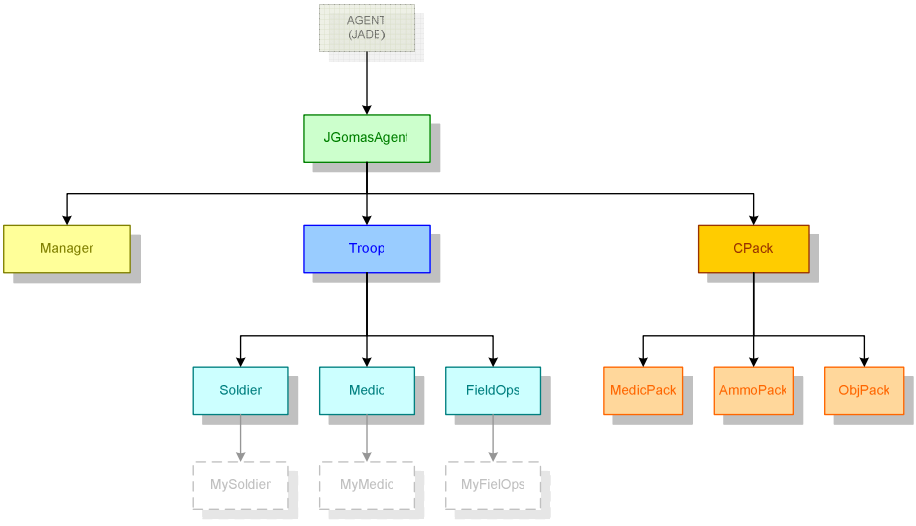
- Arquitectura
- Taxonomía de Agentes
- Servicios
- Bucle de Ejecución
- Tareas
- Comunicaciones
- Interfaz (API)

Arquitectura (I)



Arquitectura (II)

Taxonomía de Agentes



Servicios

- Registro
- Solicitud
- Respuesta

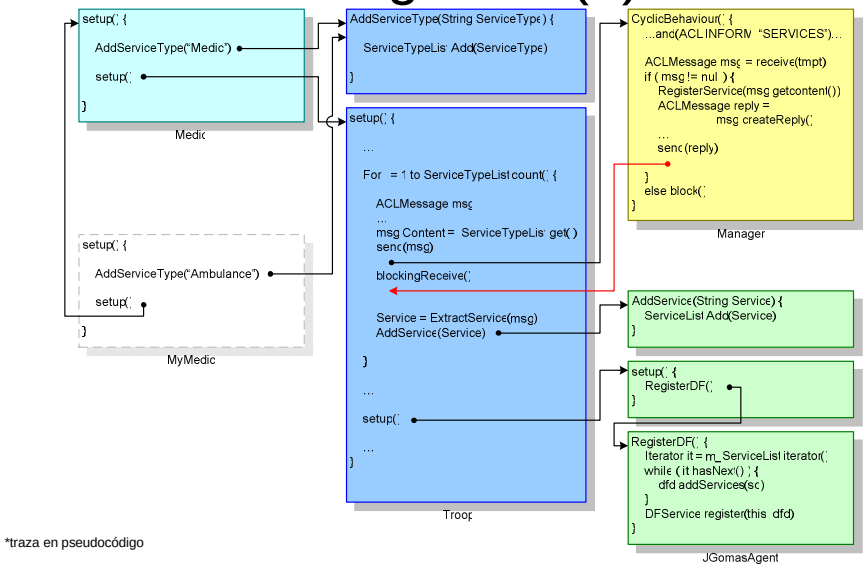
Servicios: Registro (I)

- Un rol debe registrar un servicio para que el resto de roles puedan solicitarlo:

```
void setup() {
    …
    AddServiceType("Medic");
    super.setup();
    …
}
```

- Existen definidos tres tipos de servicios básicos:
  - m_sMedicService;
  - m_sAmmoService;
  - m_sBackupService;

Servicios: Registro* (II)



*traza en pseudocódigo

Servicios: Solicitud (I)

```
protected void CallForMedic() {
      try {
```

Solicitamos al DF (páginas amarillas) un listado de aquellos agentes que proporcionen el servicio "`m_sMedicService`"

Si hay alguno en la partida..

Creamos un mensaje ACL tipo REQUEST
Añadimos todos los agentes del listado recibido del DF como receptores del mensaje

ConversationID = "CFM"
Contenido = "(x y z) (vida)"

Enviamos el mensaje

Sino sabemos que no hay medicos

```
      } catch (FIPAException fe)
                  fe.printStackTrace();
}
```

Servicios: Solicitud (II)

```
protected void CallForMedic() {
      try {
                  DFAgentDescription dfd = new DFAgentDescription();
                  ServiceDescription sd = new ServiceDescription();
                  sd.setType(m_sMedicService);
                  dfd.addServices(sd);
                  DFAgentDescription[] result = DFService.search(this, dfd);

                  if ( result.length > 0 ) {
                              m_iMedicsCount = result.length;
                              // Fill the REQUEST message
                              ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
                              for ( int i = 0; i < result.length; i++ ) {
                                          DFAgentDescription dfdMedic = result[i];
                                          AID Medic = dfdMedic.getName();
                                          if ( ! Medic.equals(getName()) )
                                                      msg.addReceiver(dfdMedic.getName());
                                          else
                                                      m_iMedicsCount--;
                              }
                              msg.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
                              msg.setConversationId("CFM");
                              msg.setContent(" ( " + m_Movement.m_Position.x + " , "
                                          + m_Movement.m_Position.y + " , "
                                          + m_Movement.m_Position.z + " ) ( "
                                          + m_iHealth + " ) ");
                              send(msg);
                              System.out.println(getLocalName()+ ": Need a Medic!");
                  }
                  else
                              m_iMedicsCount = 0;
      } catch (FIPAException fe)
                  fe.printStackTrace();
}
```


Servicios: Respuesta (I)

```
private void Launch_CFM_ResponderBehaviour() {
      // Behaviour to handle a Call For Medic request
      addBehaviour(new CyclicBehaviour() {
                  public void action() {
```

Filtrado de Mensajes: (Tipo == REQUEST) and (ConversationID == CFM)

Recibimos una petición y ..

si decidimos aceptar la petición
crear tarea TASK_GIVE_MEDICPACKS
performativa = AGREE
sino
performativa = REFUSE

Contestamos al agente que solicita la petición

Bloqueamos el Comportamiento del Agente

```
                  }
      });
}
```

Función de Decisión

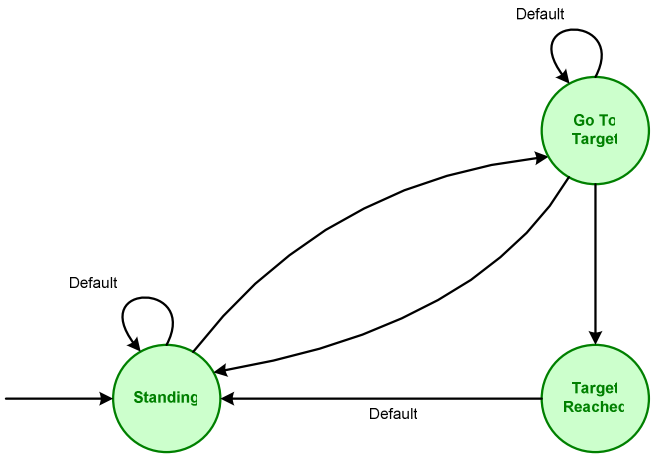
Servicios: Respuesta (II)

```
private void Launch_CFM_ResponderBehaviour() {
      // Behaviour to handle a Call For Medic request
      addBehaviour(new CyclicBehaviour() {
                  public void action() {
                        MessageTemplate template = MessageTemplate.and(
                        MessageTemplate.MatchPerformative(ACLMessage.REQUEST),
                        MessageTemplate.MatchConversationId("CFM"));

                        int iPerformative;
                        ACLMessage msgCFM = receive(template);
                        if ( msgCFM != null ) {

                              AID owner = msgCFM.getSender();
                              String sContent = msgCFM.getContent();

                              if ( checkMedicAction(sContent) ) {
                                    AddTask(TASK_GIVE_MEDICPAKS, owner, sContent);
                                    iPerformative = ACLMessage.AGREE;
                              }
                              else {
                                    iPerformative = ACLMessage.REFUSE;
                              }
                              ACLMessage reply = msgCFM.createReply();
                              reply.setContent(sContent);
                              reply.setPerformative(iPerformative);
                              send(reply);
                        }
                        else block();
                  }
      });
}

protected boolean checkMedicAction(String _sContent) { return ( true ); }
```

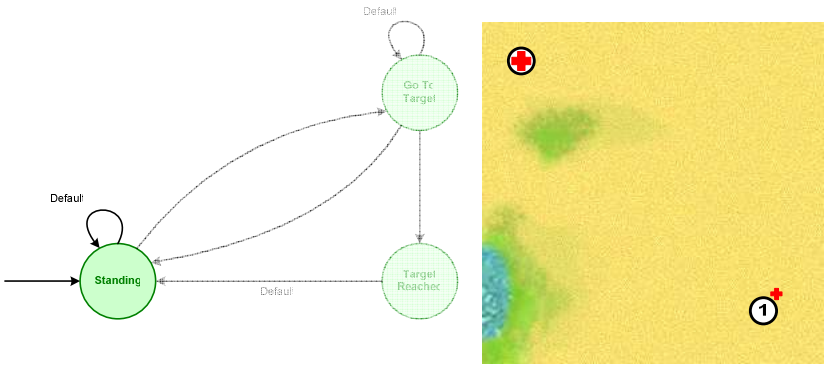

Bucle de Ejecución (I)

- Cada agente ejecuta una FSM:
  - STATE_STANDING
  - STATE_GOTO_TARGET
  - STATE_TARGET_REACHED
- FSM se utiliza para realizar tareas:
  - Inicio (Lanzamiento)
  - Desarrollo (Ejecución)
  - Final (Acción y Destrucción)
- Se lanza siempre la tarea de prioridad más alta

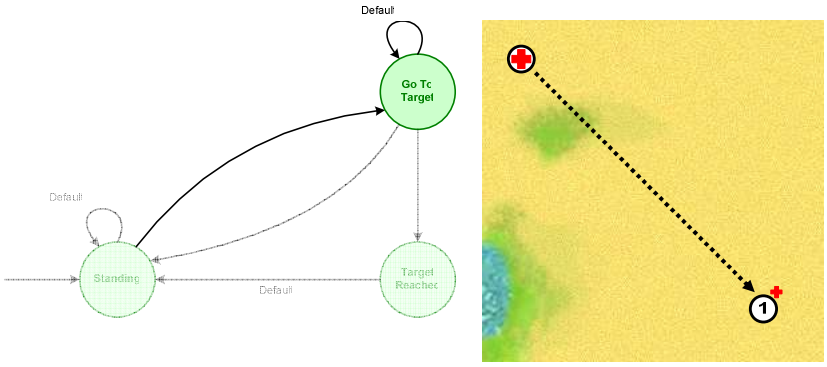
Bucle de Ejecución (II)



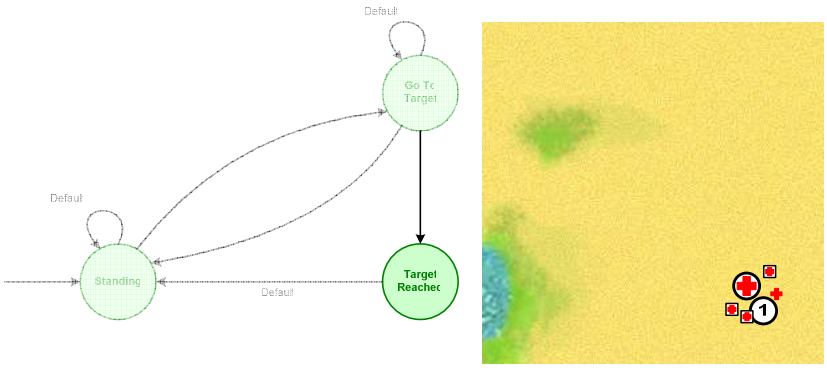
Ejecución: Caso 1 (I)



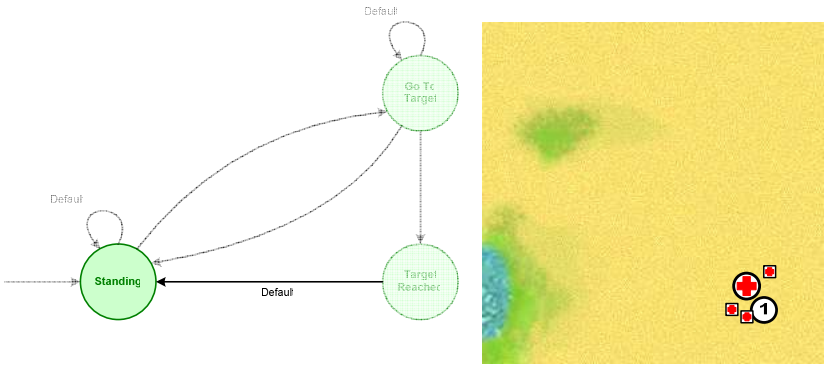
Ejecución: Caso 1 (II)



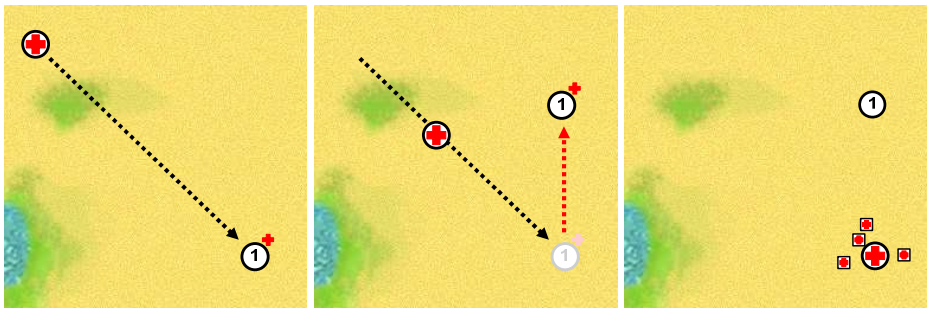
Ejecución: Caso 1 (III)



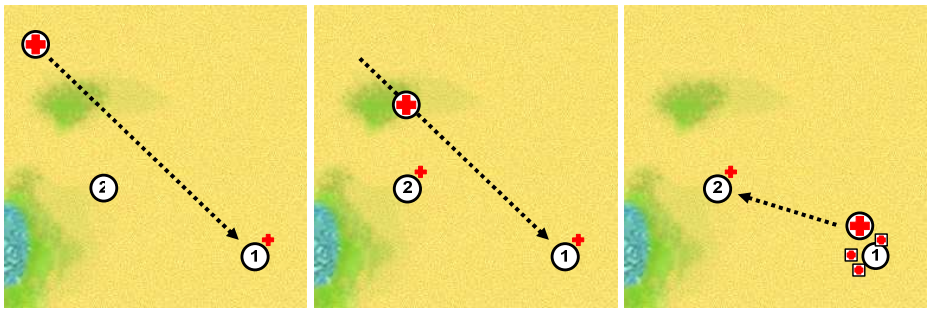
Ejecución: Caso 1 (IV)



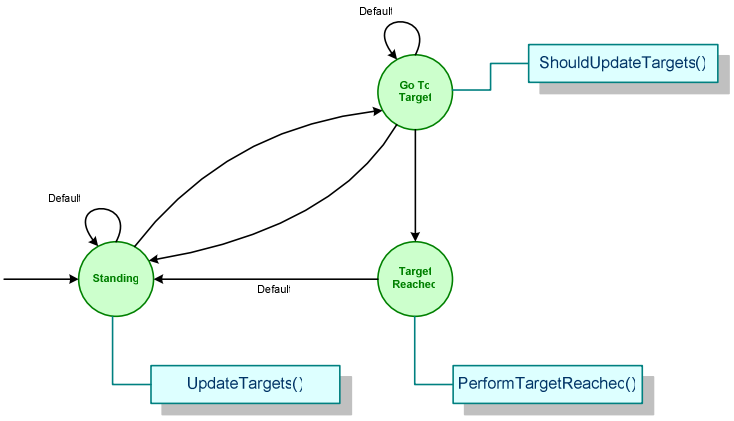
Ejecución: Caso 2



Ejecución: Caso 3



Bucle de Ejecución (III)



Tareas (I)

- Definición

```
class CTask {
        AID             m_AID;
        int             m_iType;
        int             m_iPriority;
        Vector3D        m_Position;
        long            m_StampTime;
        int             m_iPacksDelivered;
        boolean         m_bErasable;
}
```

- **`m_AID`** es el identificador del agente que *provoca* la creación de la tarea

JGOMAS Especificación

Tareas (II)

- **`m_iType`** es el tipo de tarea
- Algunos de los tipos son:
  - TASK_GIVE_MEDICPAKS
  - TASK_GIVE_AMMOPACKS
  - TASK_GIVE_BACKUP
  - TASK_GET_OBJECTIVE
  - TASK_GOTO_POSITION
  - …

Tareas (III)

- `m_iPriority` indica la prioridad **actual**
- Se lanza siempre la tarea de prioridad más alta
- Es posible redefinir la prioridad de cada tipo de tarea:

```
protected void SetUpPriorities() {
        …
        m_TaskPriority[TASK_GIVE_MEDICPAKS] = 2000;
        m_TaskPriority[TASK_GIVE_AMMOPACKS] = 0;
        m_TaskPriority[TASK_GIVE_BACKUP] = 0;
        m_TaskPriority[TASK_GET_OBJECTIVE] = 1000;
        m_TaskPriority[TASK_GOTO_POSITION] = 750;
        …
}
```


Tareas (IV)

- Las tareas las crea el sistema, no el usuario
- Declaración:
  -
```
AddTask(int _tTypeOfTask,
          AID _owner,
          String  _sContent)
```
  -
```
AddTask(int _tTypeOfTask,
          AID _owner,
          String _sContent,
          int _iPriority)
```
- Ejemplo de uso:

```
String sNewPosition = " ( " + x + " , " + y + " , " + z + " ) ";
AddTask(TASK_GOTO_POSITION, getAID(), sNewPosition);
```

Comunicaciones

- Estándar FIPA
  - Partículas ilocutivas
- Formato de los mensajes
  - StringTokenizer
- Uso de templates y conversationID

Comunicaciones (I)

- Estándar FIPA
  - Protocolos establecidos
    - Soportados por JADE
  - Partículas ilocutivas
    - INFORM
    - REQUEST
    - AGREE
    - REFUSE
    - CANCEL

Comunicaciones (II)

- Formato de los mensajes

  - StringTokenizer

```
StringTokenizer tokens = new StringTokenizer(sContent);

tokens.nextToken(); // Get "ID:"
Integer id = Integer.parseInt(tokens.nextToken());

tokens.nextToken(); // Get "("
double x = Double.parseDouble(tokens.nextToken());
tokens.nextToken(); // Get ","
double y = Double.parseDouble(tokens.nextToken());
tokens.nextToken(); // Get ","
double z = Double.parseDouble(tokens.nextToken());
```


Comunicaciones (III)

- Uso de templates y conversationID

```
MessageTemplate template = MessageTemplate.and(
        MessageTemplate.MatchPerformative(ACLMessage.INFORM),
        MessageTemplate.MatchConversationId("SHOT"));

ACLMessage msg = receive(template);

if ( msg != null ) {
        .
        .
        Agent does some actions
        .
        .
}
else
        block();
```

Interfaz

- CTroop
  - Métodos finales
  - Métodos sobrecargables
  - Atributos
- CMedic
- CFieldOps
- CSoldier

CTroop (I): Métodos finales

```
final int  GetHealth ()
final int  GetAmmo ()
final int  GetStamina ()
final void  UseStamina ()
final int  GetPower ()
final void  UsePower ()
final void  AddServiceType (String _sServiceType)
final boolean  CheckStaticPosition ()
final boolean  CheckStaticPosition (double _x, double _z)
final void  AddTask (int _tTypeOfTask, AID _Owner, String
_sContent)
final void  AddTask (int _tTypeOfTask, AID _Owner, String
_sContent, int _iPriority)
final void  Look ()
final boolean  Shot (int _iShotNum)
final void  PerformAimAction ()
final boolean  HaveAgentToShot ()
```

CTroop (II): M. sobrecargables

- `void CallForMedic ()`
- `void CallForAmmo ()`
- `void CallForBackup ()`
- `void UpdateTargets ()`
- `boolean ShouldUpdateTargets ()`
- `void ObjectivePackTaken ()`
- `void SetUpPriorities ()`
- `void PerformNoAmmoAction ()`
- `void PerformTargetReached (CTask _CurrentTask)`
- `void GenerateEscapePosition ()`
- `boolean GeneratePath ()`
- `void CreateControlPoints ()`
- `void PerformEscapeAction ()`
- `boolean GetAgentToAim ()`
- `void PerformLookAction ()`

CTroop (III): Atributos (1)

- `AID m_Manager = null`
- `Hashtable m_TaskList`
- `CTask m_CurrentTask`
- `boolean m_bObjectiveCarried`
- `int m_TaskPriority []`
- `Vector3D m_ControlPoints []`
- `int m_iControlPointsIndex`
- `Vector3D m_AStarPath []`
- `int m_iAStarPathIndex`
- `ArrayList m_FOVObjects`
- `CSight m_AimedAgent`
- `int m_eClass`

CTroop (III): Atributos (2)

- `boolean m_bFighting`
- `boolean m_bEscaping`
- `CThreshold m_Threshold`
- `CMobile m_Movement`
- `CTerrainMap m_Map`
- `String m_sMedicService`
- `String m_sAmmoService`
- `String m_sBackupService`

- `int m_iSoldiersCount, m_iMedicsCount`
- `int m_iEngineersCount, m_iFieldOpsCount`
- `int m_iTeamCount`

CMedic

- Métodos finales
  - `final int CreateMedicPack ()`
- Métodos sobrecargables
  - `void SetUpPriorities ()`
  - `boolean checkMedicAction (String _sContent)`

CFieldOps

- Métodos finales
  - `final int CreateAmmoPack ()`
- Métodos sobrecargables
  - `void SetUpPriorities ()`
  - `boolean checkAmmoAction (String _sContent)`

CSoldier

- Métodos finales
  - -
- Métodos sobrecargables
  - `void SetUpPriorities ()`
  - `boolean checkBackupAction (String _sContent)`

Índice

- Introducción
- Descripción
- Especificación
- ► **Conclusiones**

Conclusiones

- Se dispone de una plataforma para simulación de agentes en entornos 3D
- Completamente estándar:
  - JADE: Mecanismos de comportamiento, sincronización, etc., ya implementados
  - FIPA: Protocolos establecidos

Trabajo a Realizar (I)

- Objetivo:
  - Diseñar e implementar un equipo de 10 agentes con la distribución de tipos que deseéis (médicos, soldados e intendencia) para jugar a **capturar la bandera** en un mapa cualquiera como **atacante y como defensor**, de manera que **ganen en cualquier situación a los equipos suministrados**.

Trabajo a Realizar (II)

- Opcionalmente, se puede mejorar el rendimiento del equipo:
  - Añadiendo roles: **vigía, explorador, escolta,…**
  - Cuando un agente necesite un servicio que él mismo suministra, no deba solicitarlo fuera, sino que se lo pueda suministrar él mismo.

Trabajo a Realizar (III)

- Reglas Básicas:
  - No se puede consultar/solicitar información del sistema sobre el bando contrario que no sea suministrada por el entorno.
  - No puede existir comunicación entre agentes que no sea usando FIPA ACL y de acuerdo a la especificación proporcionada.

Trabajo a Realizar (IV)

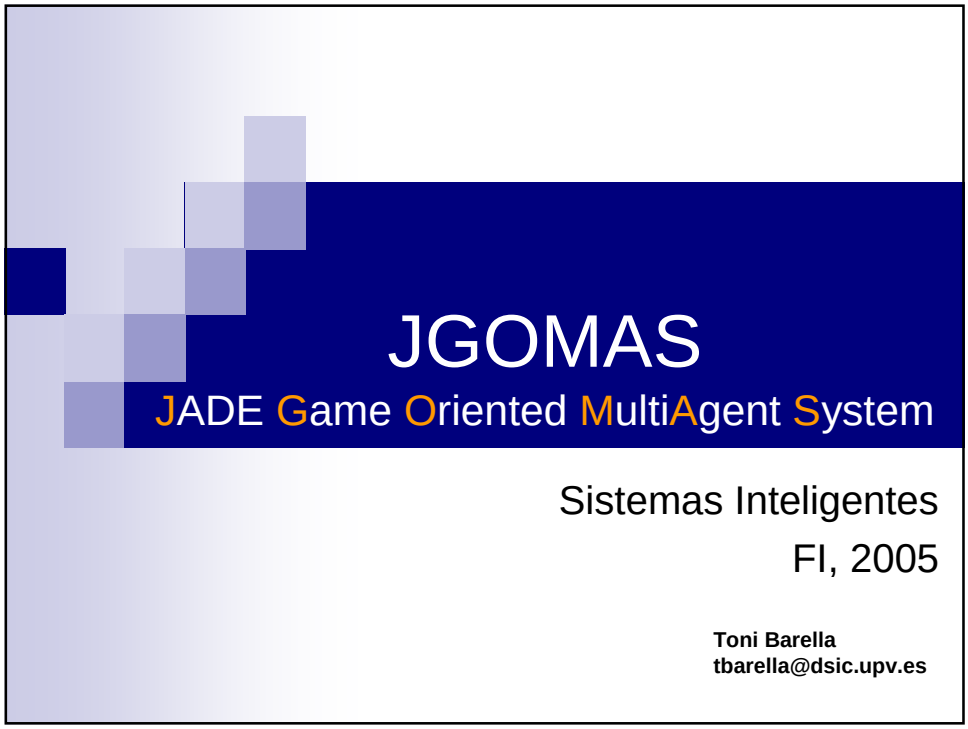
- Documentación a entregar:
  - Pequeña memoria, indicando las principales ideas de mejora aplicadas al equipo, así como unas breves conclusiones sobre los resultados obtenidos.
  - El código, comentado y documentado, indicando qué partes se han añadido y/o modificado.
  - Este código debe seguir unas mínimas normas de estilo: tabulado, comentado, y usando notación húngara.

Trabajo a Realizar (V)

- Notación Húngara: Identificadores
  - CString m_sNameFirst;
    Prefijo Calificador
    - Prefijo:
      - Siempre en minúsculas
      - Indica el tipo del dato
    - Calificador:
      - Resto del nombre, indica el uso de la var.
      - Empieza por mayúscula, para delimitarlo del prefijo
      - Si contienen más de una palabra, la inicial de cada una va en mayúscula. Ej.: int nEstoEsUnIdMuyLargo;

Trabajo a Realizar (VI)

- Notación Húngara: Prefijos comunes
  - **s** ó **str**: *String*
  - **c**: Carácter
  - **i** ó **n**: Entero
  - **f**: *Float*
  - **d**: Doble
  - **b**: *Boolean*
  - **l**: *Long*
  - **C**: Clase
  - **m_**: Identificador de un miembro de una clase
  - **v**: *Void*, sin tipo


JGOMAS
JADE Game Oriented MultiAgent System
Sistemas Inteligentes
FI, 2005
Toni Barella
tbarella@dsic.upv.es