Taller de Programación de Agentes con JADE

Gonzalo A. Aranda Corral

Dpto. Tecnologías de la Información

Escuela Politécnica Superior “La Rábida”

Universidad de Huelva

Todos buscais la fama...

AGENTES

pero los agentes cuestan,

y aquí es donde vais a empezar a pagar

con Jade

Taller de

Programación de Agentes

¿ INTELIGENTES ?

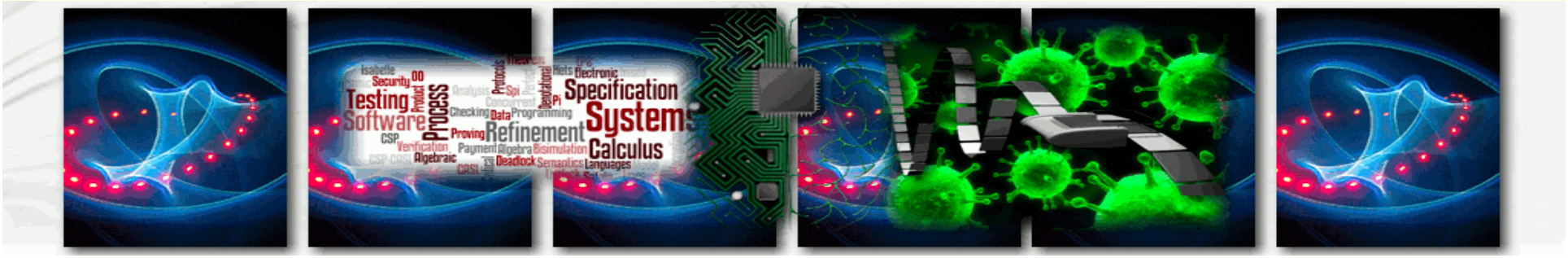
Taller de

Programación de Agentes

Publireportaje

- Para Programación de Agentes INTELIGENTES:
  - Dpto. Ciencias de la Computación e Inteligencia Artificial
  - Programa de Master Oficial y Doctorado !

http://master.cs.us.es



¿Qué es JADE?

- Jade es básicamente dos cosas:
  - Una plataforma: que permite “VIVIR” y “CONVIVIR” a los agentes dentro de ella.
  - Un conjunto de herramientas para el desarrollo de agentes y sistemas de forma rápida.
- **J**ava **A**gent **DE**velopment framework

¿Qué es JADE?

- Totalmente **realizado en Java**. (Portabilidad y Movilidad)
- Software libre distribuido por TILAB en código fuente bajo LPGL

http://jade.tilab.com/

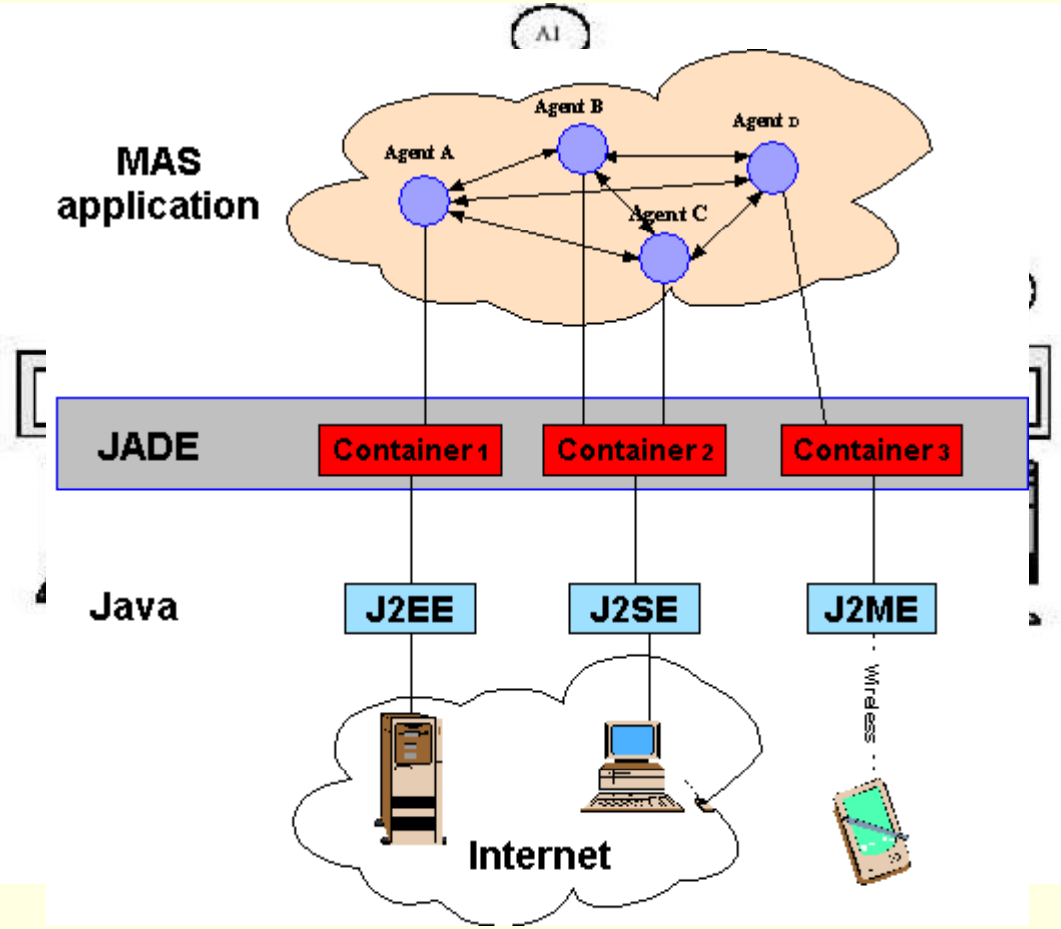
Estándares

- ***Foundation for Intelligent Physical Agents (FIPA)***
  - **Arquitectura**: Integración de diferentes aplicaciones, incluso con plataformas de diferentes propietarios.
  - **Lenguaje** de comunicación empleado FIPA-ACL.
  - **Servicios** de agentes: ciclo de vida, páginas blancas, páginas amarillas, transporte de mensajes,...
  - Conjunto de **herramientas** gráficas que soportan la depuración y ejecución de agentes (RMA, sniffer, ...)

- http://www.fipa.org

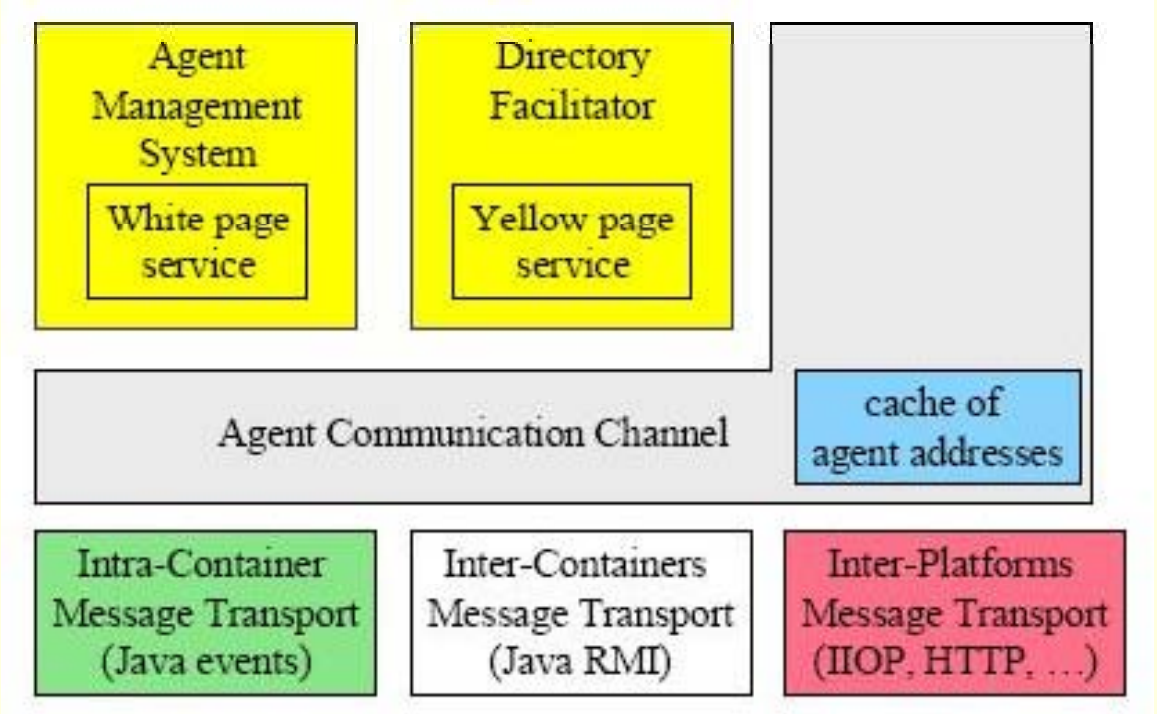
FIPA: Arquitectura básica

- Plataforma distribuida
- Comunicación entre plataformas
- Protocolos estándares
  - Internet



FIPA: Plataforma

- Especificación FIPA
- Agentes y servicios
- Comunicación



FIPA: Arquitectura Agentes

- **AMS . Agent Management System**:
  - Garantiza que cada agente en la plataforma tenga un único nombre.
  - Encargado de proporcionar los servicios de páginas blancas y ciclo de vida, y de mantener el directorio de los identificadores de agentes (AID: Agent Identifier) y su estado.
  - Cada agente debe registrarse con el AMS para obtener un AID válido

- **DF . Directory Facilitator**:
  - Agente que proporciona el servicio de páginas amarillas.
  - Un agente puede encontrar otros agentes que proporcionan los servicios que requiere para cumplir sus objetivos

- **ACC . Agent Communication Channel**:
  - Software que controla el intercambio de mensajes

JADE

- FIPA - Compliant
  - Plataforma
  - Arquitectura
  - Agentes

Cómo obtener Jade

- Dirección: http://jade.tilab.com
- Para la mayoría de las acciones es necesario registrarse, y aceptar los requisitos de la licencia LGPL
- La versión actual de Jade es la 4.0(Abril 2010), aunque nosotros vamos a realizar este seminario sobre la versión 3.7 (Julio 2009)

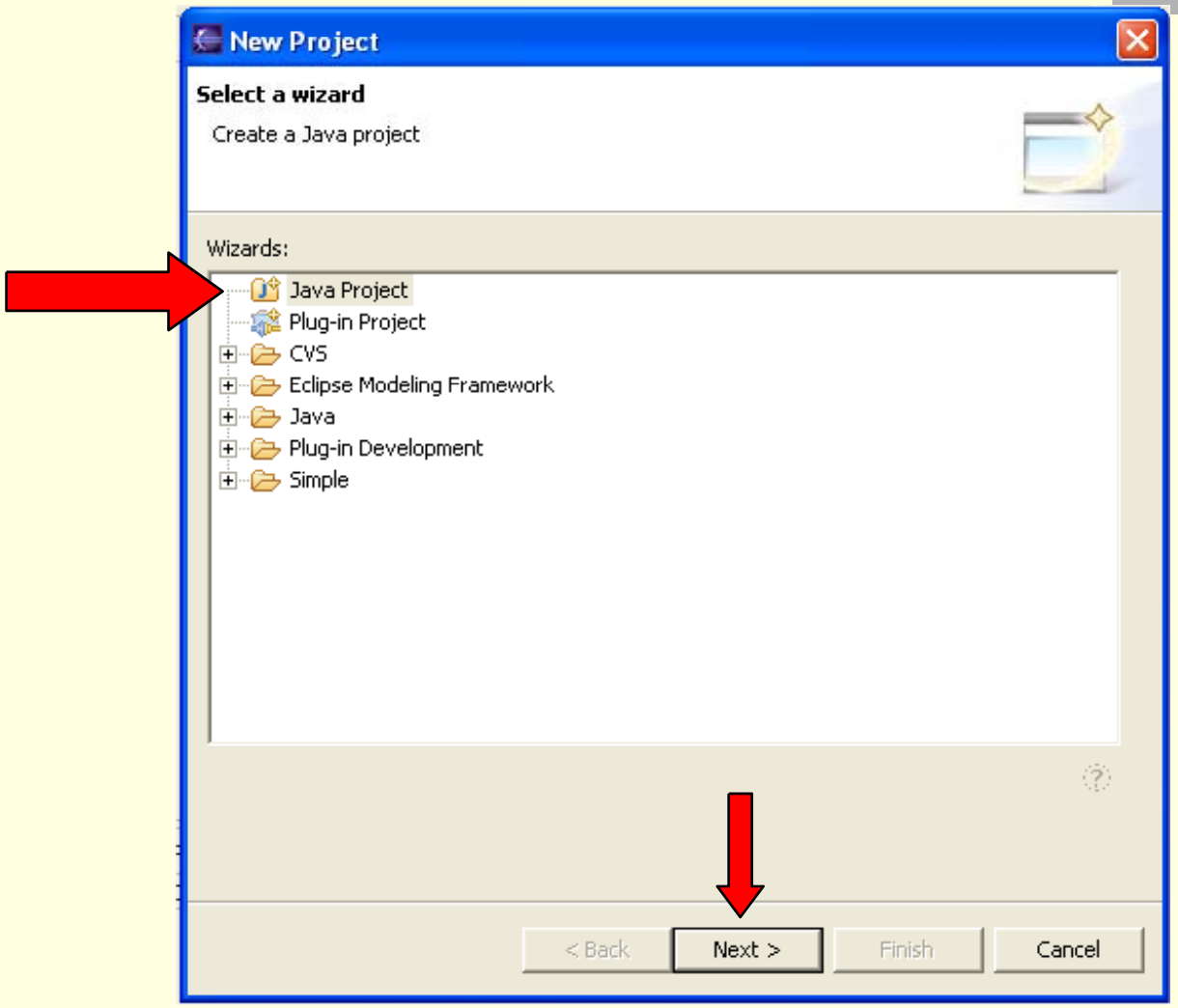
Instalación

- Descargamos el fichero (**JADE-all-3.7.zip**) de la versión correspondiente.
- Descomprimimos el fichero y nos salen cuatro nuevos ficheros:
  - **JADE-doc-3.7.zip**: la documentacion javadoc, el manual del administrador, el del programador y un tutorial.
  - **JADE-src-3.7.zip**: el codigo fuente sin compilar.
  - **JADE-bin-3.7.zip**: el codigo ya compilado y listo para ser invocado.
  - **JADE-examples-3.7.zip**: ejemplos de uso de la plataforma.

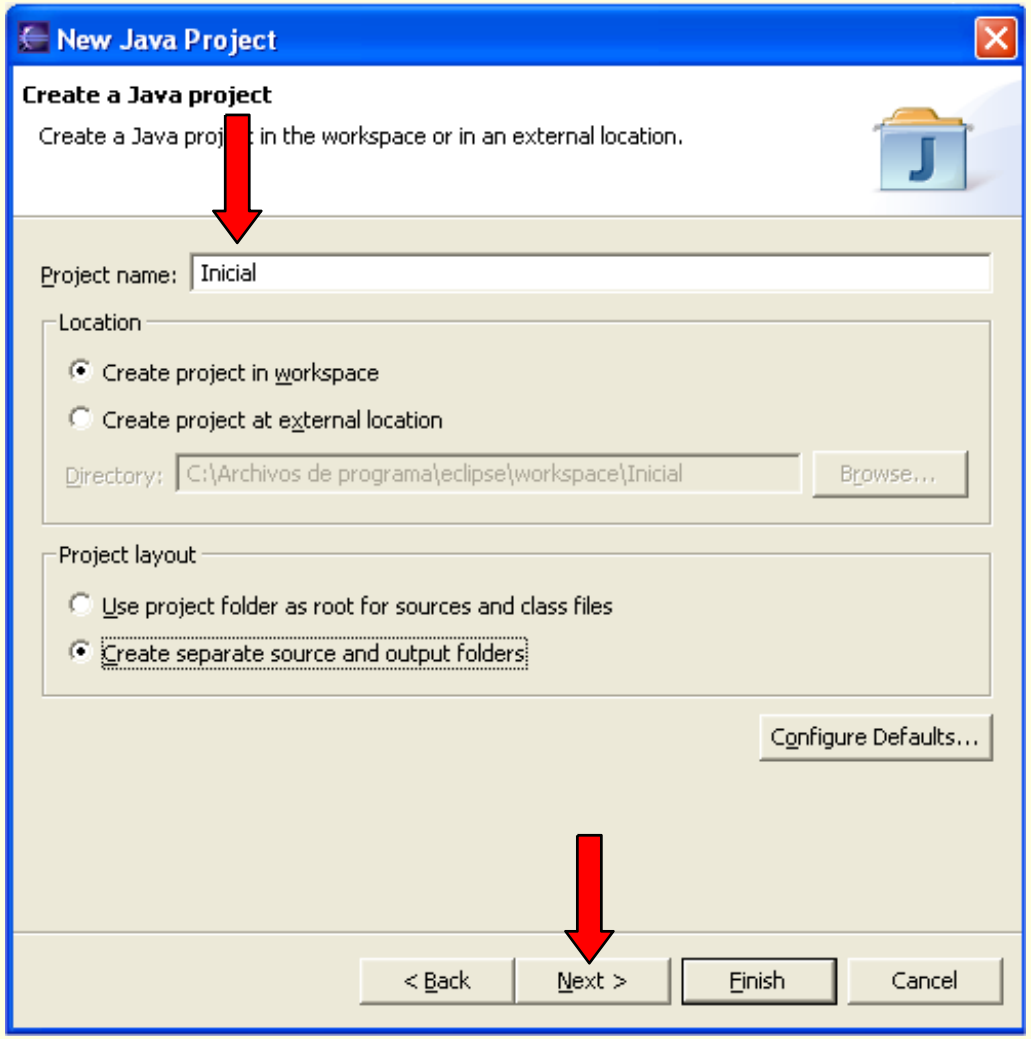
Instalación

- Se creará un directorio **lib/** debajo del cual estarán las librerías necesarias para la ejecución de Jade
- NECESITA, al menos, JAVA 1.4.2
  - (Aunque usaremos la 1.6)

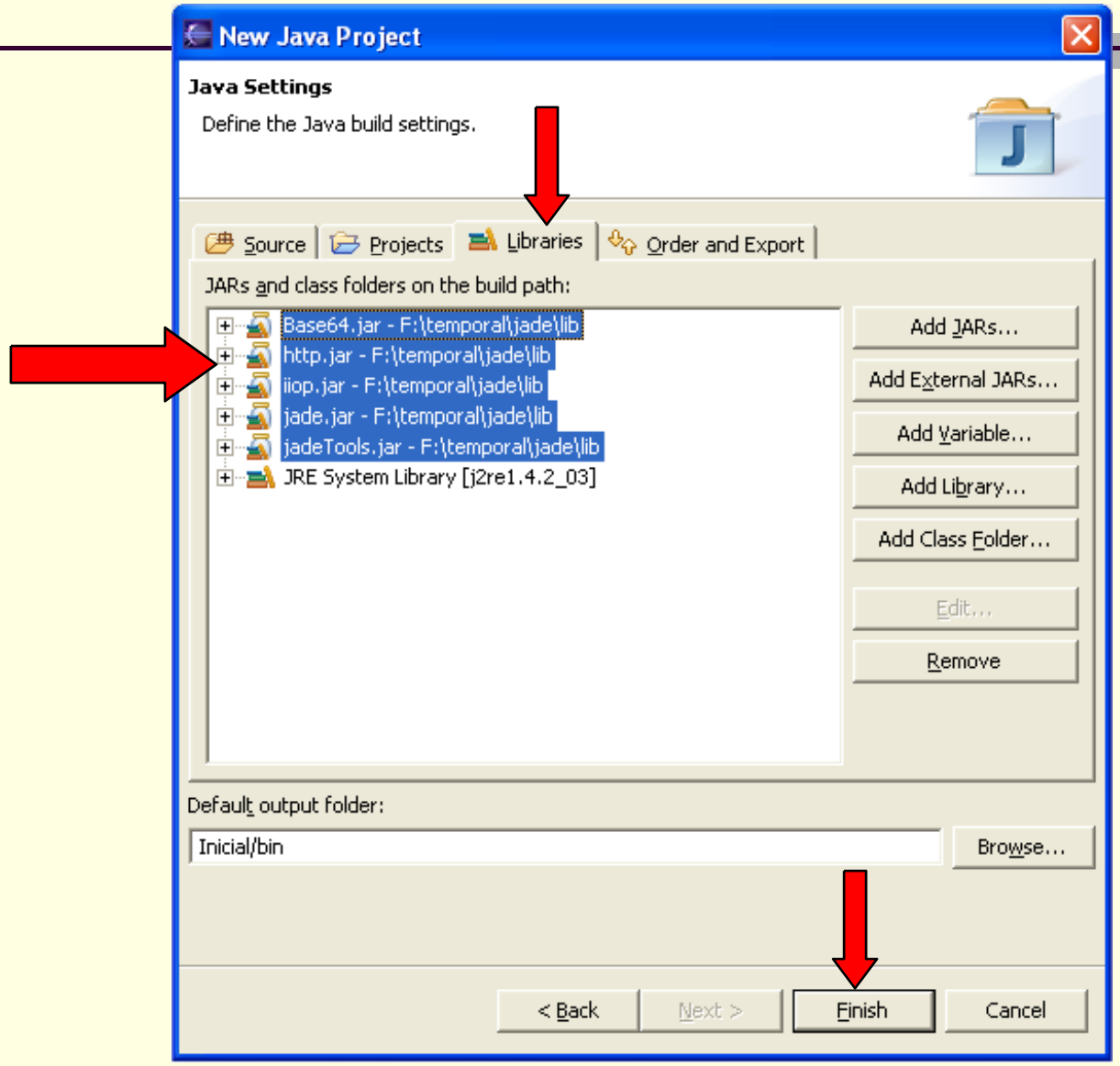
Eclipse: Creación del proyecto



Eclipse: Creación del proyecto



Eclipse: Creación del proyecto



La clase Agent

- Implementar agentes(heredar de): **jade.core.Agent**
- **NUNCA** SOBREESCRIBIR EL CONSTRUCTOR
- Inicialización: **setup()**.
- Morir: **doDelete()**.
  - Sobreescribir: **takeDown()**.
- Argumentos: **getArguments()**
  - Object[] que proporciona los argumentos que se le han pasado al agente.

Identificadores de agentes

- Descripción del agente: AID (Agent Identier)

jade.core.AID

- La clase Agent incorpora el método getAID() que permite recuperar el nombre del agente.
- El nombre del agente, un identificador único globalmente(normativa FIPA), va a tener la estructura

<nickname>@<nombre-plataforma>:<puerto>/JADE

Ejercicio 1.

Nuestro primer agente:

- Debemos de crear un agente cuya misión sea imprimir un “Hola Mundo”

```
package ej01;
import jade.core.Agent;
public class Ejercicio01 extends Agent {
protected void setup() {
        System.out.println(“Hola Mundo.”);
}
}
```

Ejecución

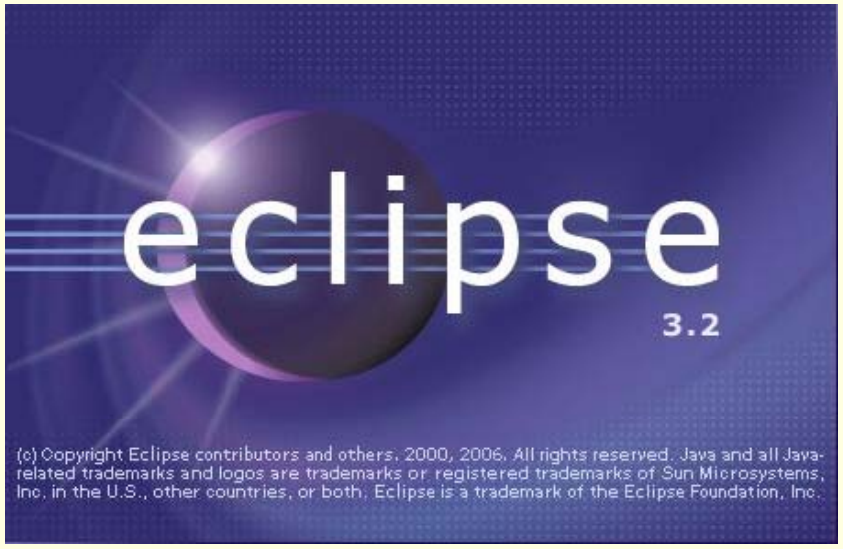
- Desde línea de comandos:

java <classpath> [opciones] jade.Boot [agente:ClaseAgente]

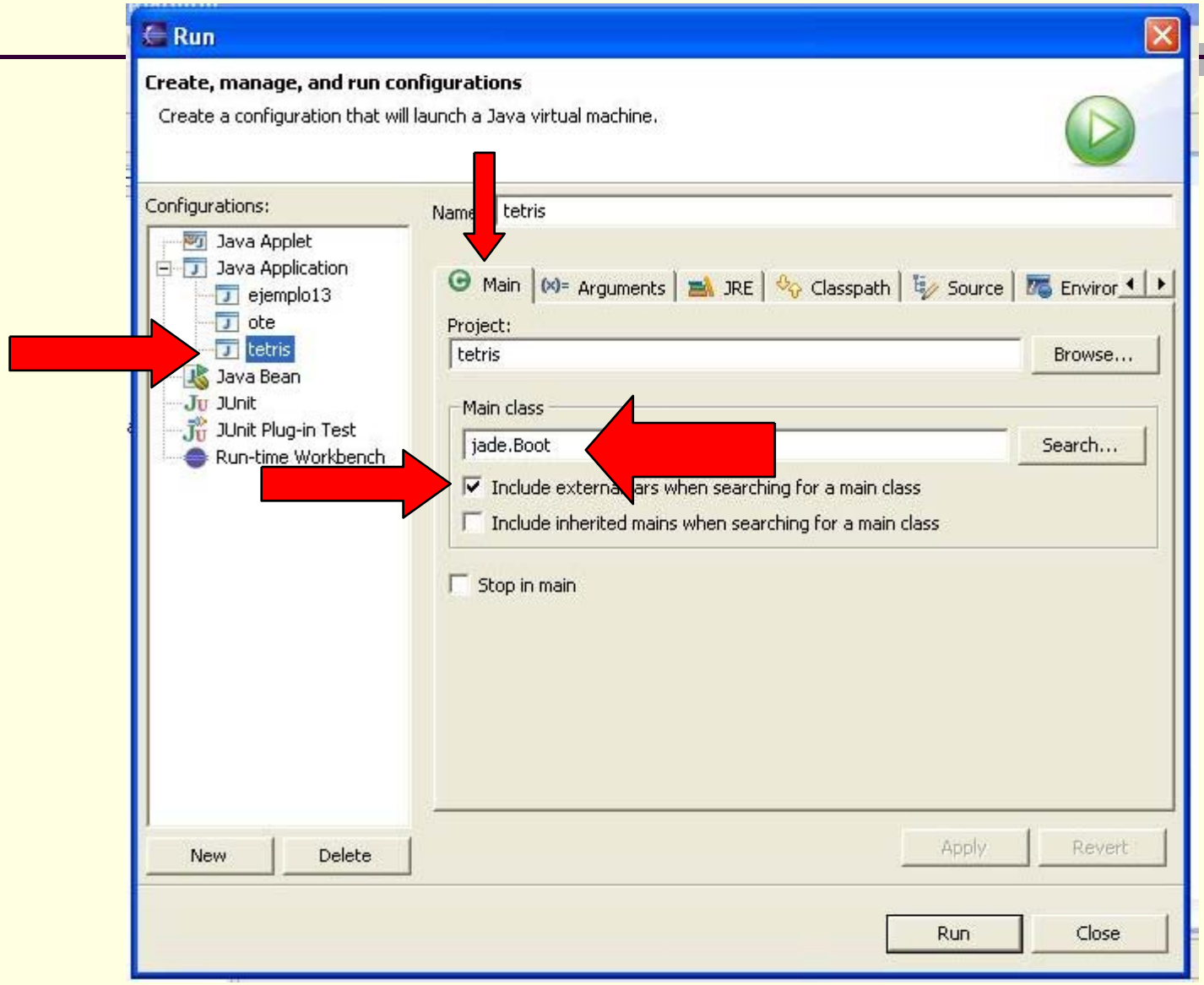
  - **classpath**: direccion y nombre de los *.jar de Jade
  - **opciones**: en principio, usaremos la opcion “-gui”
  - **jade.Boot**: es la clase de arranque de la plataforma
  - **agente:ClaseAgente**: Nombre y clases(incluyendo paquetes) de nuestro agente

Ejecución

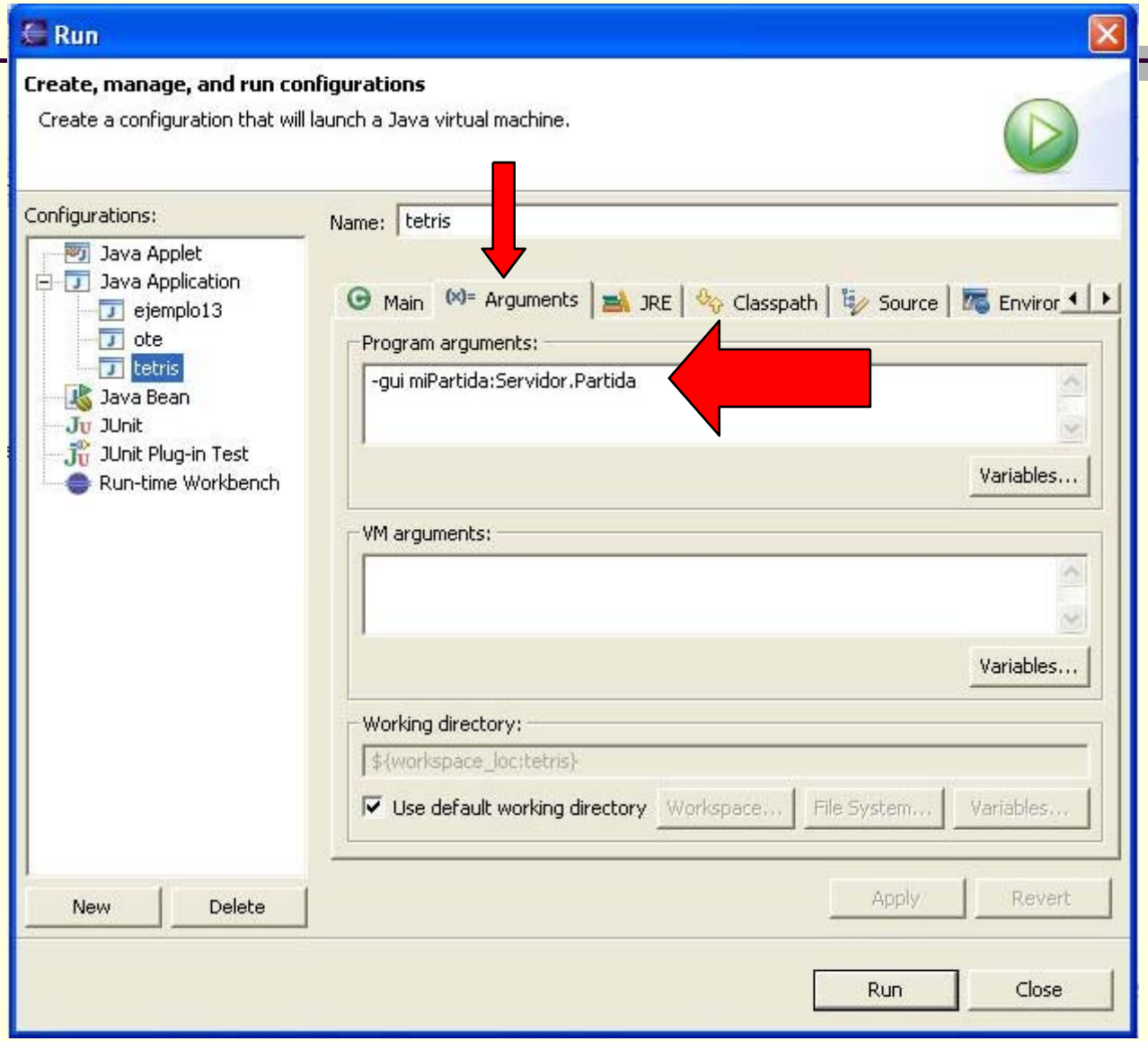
- Desde Eclipse:
  - Creamos una configuración de ejecución.
  - Configuramos los siguiente parámetros
    - (Ver pag siguiente)



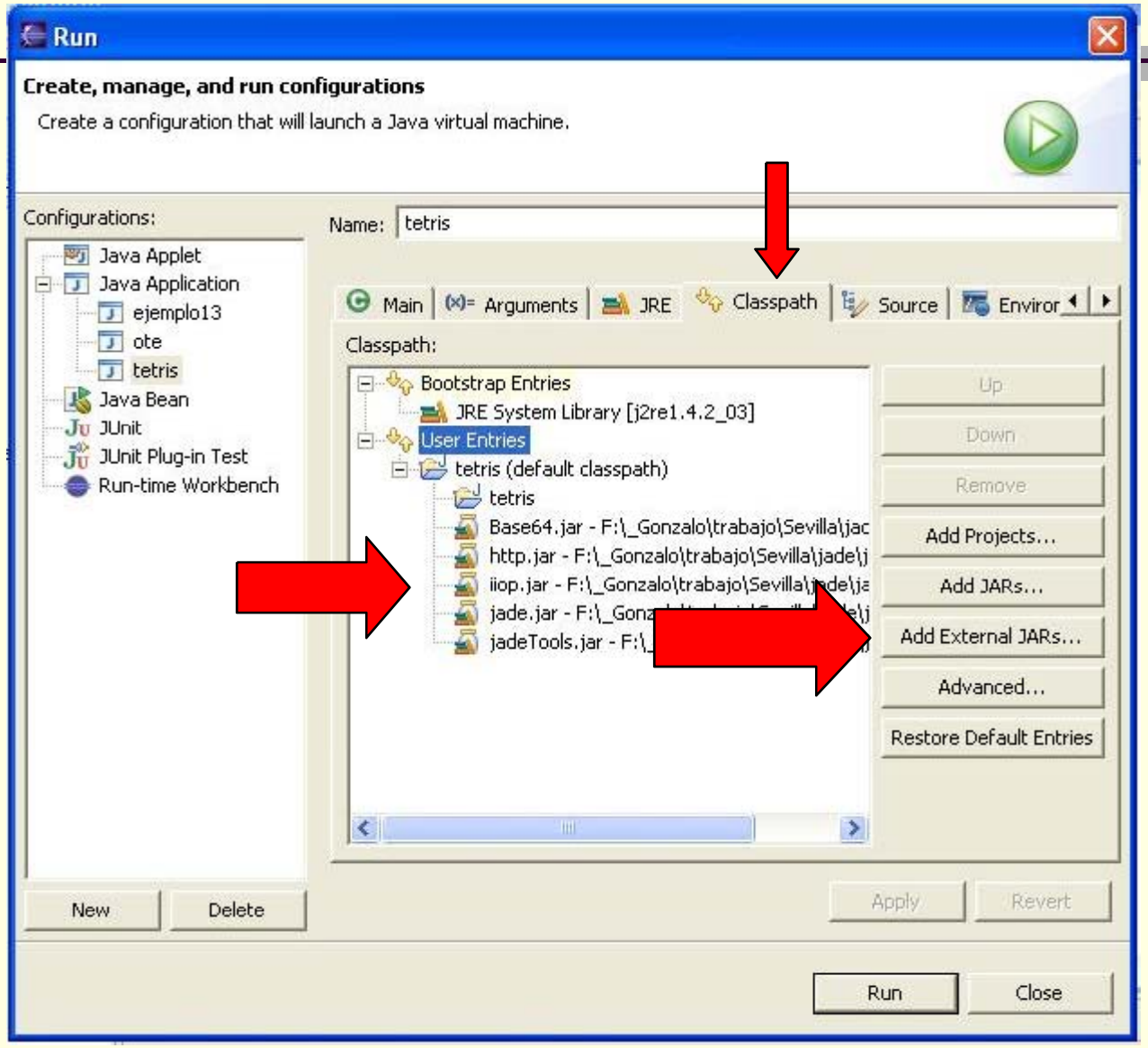
Ejecución


Run
Create, manage, and run configurations
Create a configuration that will launch a Java virtual machine.
Configurations:
Java Applet
Java Application
ejemplo13
ote
tetris
Java Bean
JUnit
JUnit Plug-in Test
Run-time Workbench
Name: tetris
Main
Arguments
JRE
Classpath
Source
Enviror
Project:
tetris
Browse...
Main class
jade.Boot
Search...
Include external jars when searching for a main class
Include inherited mains when searching for a main class
Stop in main
New
Delete
Apply
Revert
Run
Close

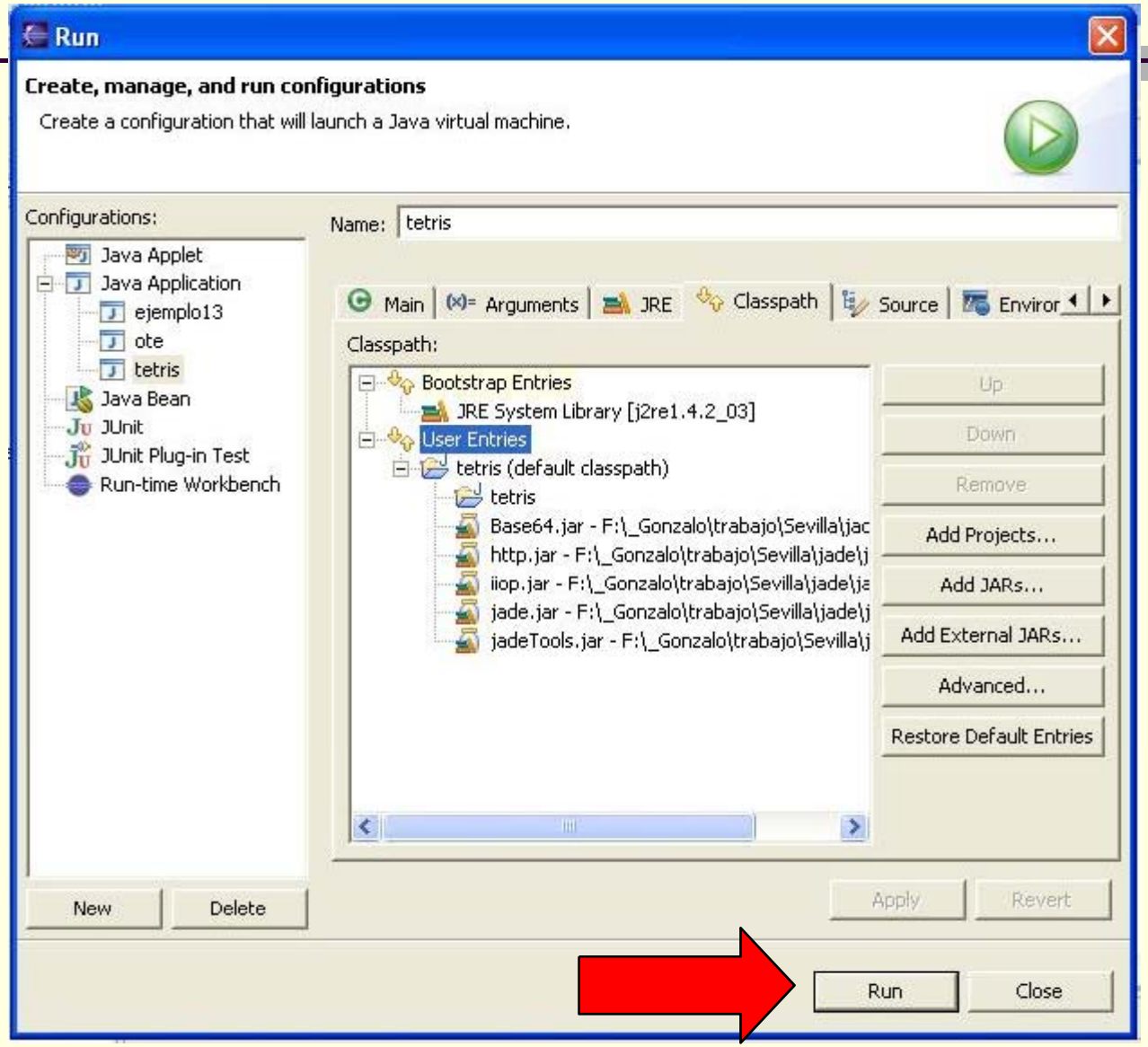
Ejecución


Run
Create, manage, and run configurations
Create a configuration that will launch a Java virtual machine.
Configurations:
Java Applet
Java Application
ejemplo13
ote
tetris
Java Bean
JUnit
JUnit Plug-in Test
Run-time Workbench
New
Delete
Name: tetris
Main
Arguments
JRE
Classpath
Source
Enviror
Program arguments:
-gui miPartida:Servidor.Partida
Variables...
VM arguments:
Variables...
Working directory:
${workspace_loc:tetris}
Use default working directory
Workspace...
File System...
Variables...
Apply
Revert
Run
Close

Ejecución


Run
Create, manage, and run configurations
Create a configuration that will launch a Java virtual machine.
Configurations:
Java Applet
Java Application
ejemplo13
ote
tetris
Java Bean
JUnit
JUnit Plug-in Test
Run-time Workbench
New
Delete
Name: tetris
Main
Arguments
JRE
Classpath
Source
Enviror
Classpath:
Bootstrap Entries
JRE System Library [j2re1.4.2_03]
User Entries
tetris (default classpath)
tetris
Base64.jar - F:_Gonzalo\trabajo\Sevilla\jac
http.jar - F:_Gonzalo\trabajo\Sevilla\jade\j
iiop.jar - F:_Gonzalo\trabajo\Sevilla\jade\ja
jade.jar - F:_Gona
jadeTools.jar - F:\
Up
Down
Remove
Add Projects...
Add JARs...
Add External JARs...
Advanced...
Restore Default Entries
Apply
Revert
Run
Close

Ejecución

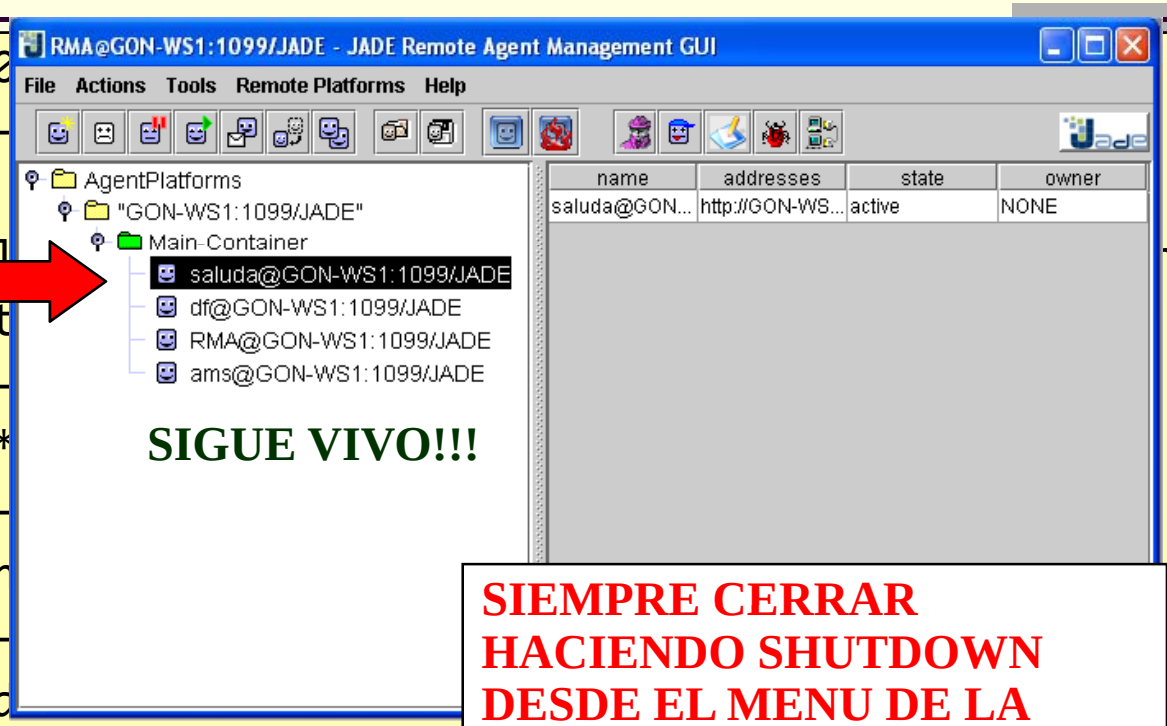

Run
Create, manage, and run configurations
Create a configuration that will launch a Java virtual machine.
Configurations:
Java Applet
Java Application
ejemplo13
ote
tetris
Java Bean
JUnit
JUnit Plug-in Test
Run-time Workbench
New
Delete
Name: tetris
Main
Arguments
JRE
Classpath
Source
Enviror
Classpath:
Bootstrap Entries
JRE System Library [j2re1.4.2_03]
User Entries
tetris (default classpath)
tetris
Base64.jar - F:_Gonzalo\trabajo\Sevilla\jac
http.jar - F:_Gonzalo\trabajo\Sevilla\jade\j
iiop.jar - F:_Gonzalo\trabajo\Sevilla\jade\ja
jade.jar - F:_Gonzalo\trabajo\Sevilla\jade\j
jadeTools.jar - F:_Gonzalo\trabajo\Sevilla\j
Up
Down
Remove
Add Projects...
Add JARs...
Add External JARs...
Advanced...
Restore Default Entries
Apply
Revert
Run
Close

Resultado

```
25-may-2006 1:46:38 jade.core.Runtime beginContainer
INFO: ----------------------------------
    This is JADE 3.3 - 2005/03/02 16:11:05
    downloaded in Open Source, under LGPL restrictions,
    at http://jade.cselt.it/
------------------------------------------
**********************MAS INFORMACION
INFO: ----------------------------------------
Agent container Main-Container@JADE-IMTP://GON-WS1 is ready.
------------------------------------------------
Hola Mundo.
```

Resultado

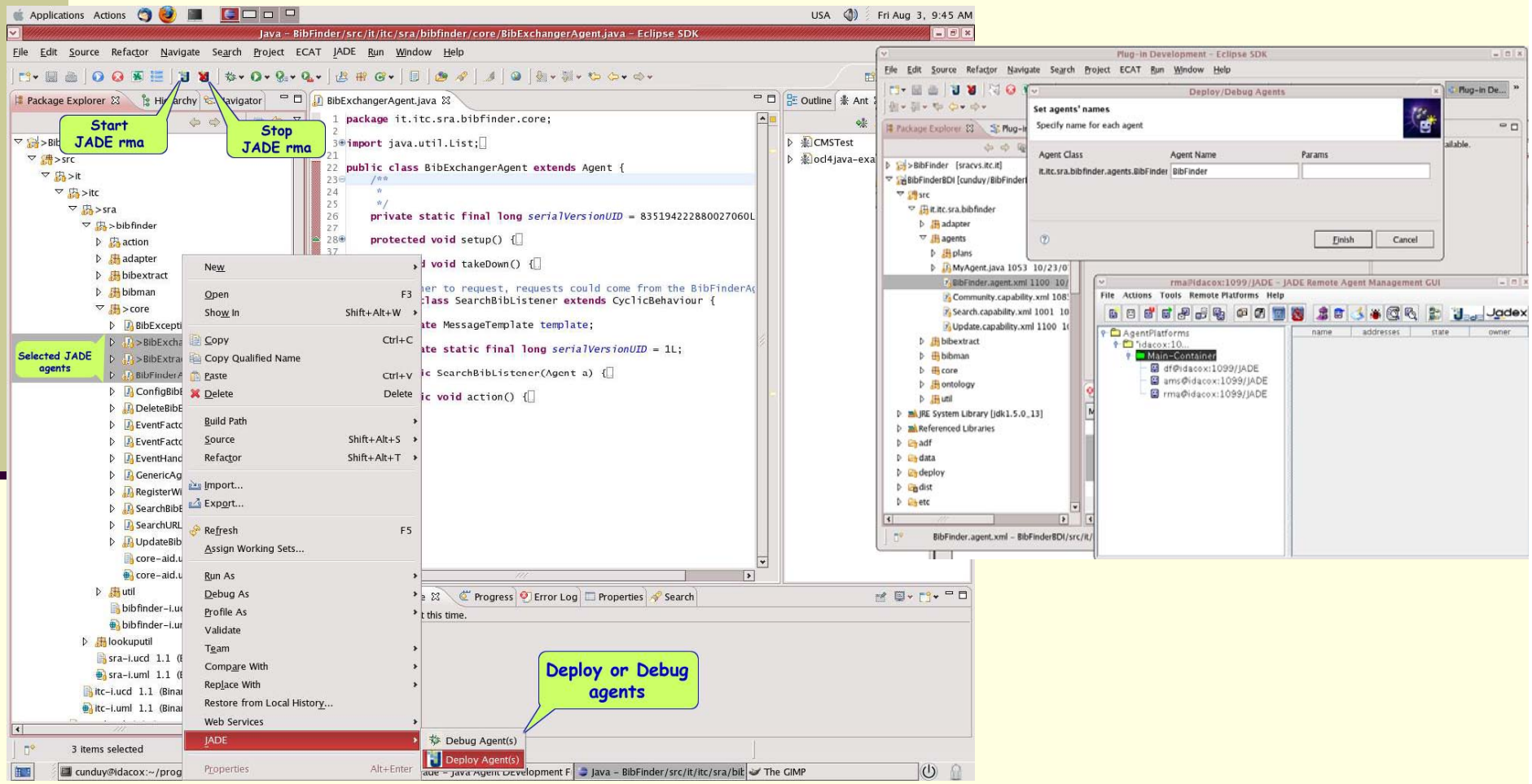
```
25-may-20
INFO: ---
     This
     downl                                                    ns,
     at ht
---------
*********
INFO: ---
Agent cor                                                     ready.
---------
Hola Mund
```



SIEMPRE CERRAR HACIENDO SHUTDOWN DESDE EL MENU DE LA PLATAFORMA... SI NO, SE QUEDA ABIERTA!

Plugin para Eclipse

- http://dit.unitn.it/~dnguyen/ejade/



Comportamientos

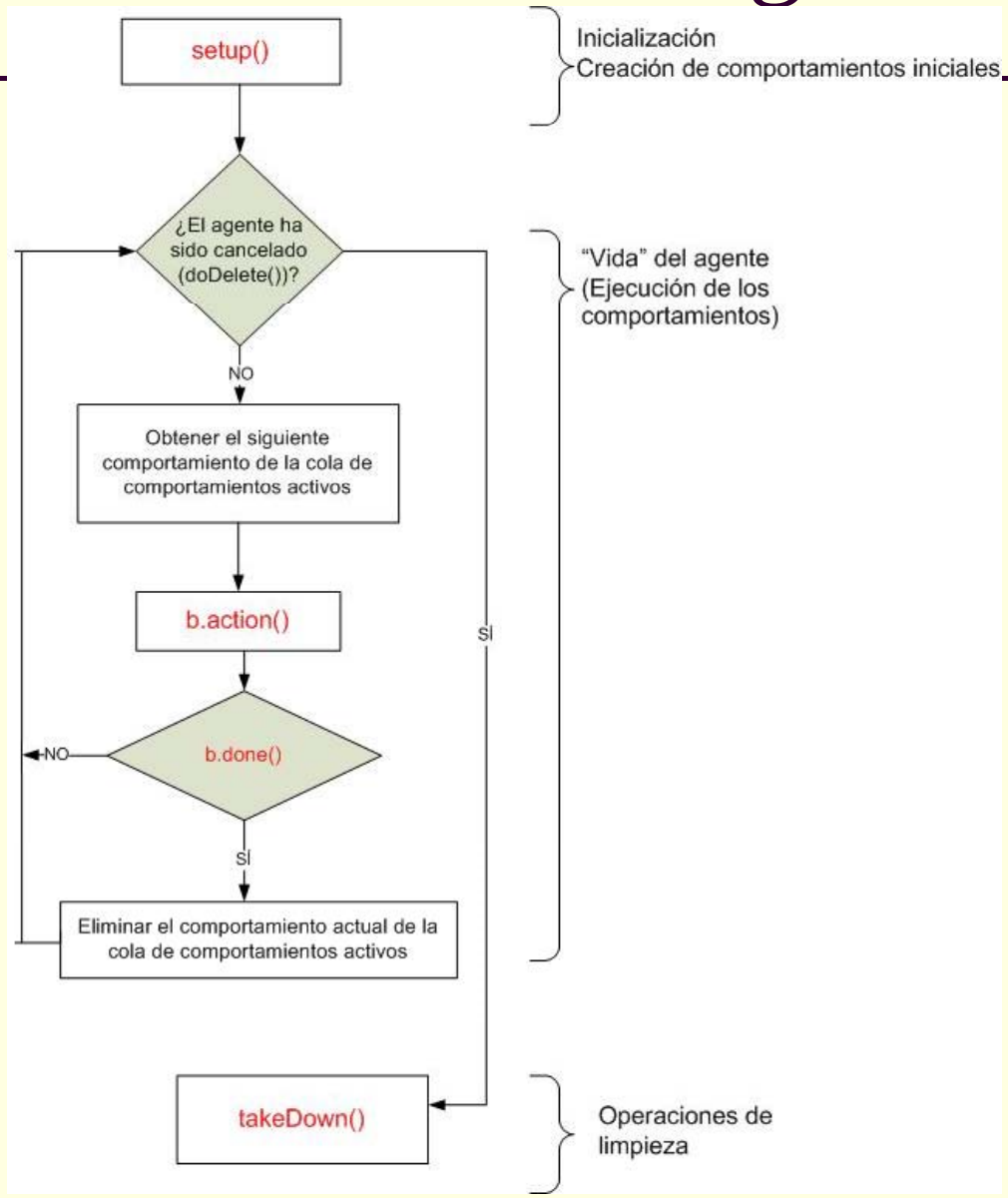
Comportamientos

- Acciones y Percepciones:
  - “**Comportamientos**”.

Comportamientos

- El funcionamiento de los comportamientos está implementado a 2 niveles:
  - Una cola circular de los comportamientos activos
  - Una cola con los comportamiento bloqueados.
- Los comportamientos se desbloquean al recibir el agente un mensaje.

Ciclo de vida de un agente



Comportamientos

- La clase principal es:

  jade.core.behaviours.Behaviour

- Tiene 2 métodos principales:
  - **action()**: que es el método que se ejecuta cada vez que se invoca el método
  - **done()**: hace la comprobación de si el comportamiento ha terminado o no.
- Aquí se puede sobreescribir el constructor (aunque no es aconsejable)

Comportamientos

Ejercicio 2: Crear un agente que implemente un comportamiento que salude al mundo.

```
package ej02;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;

public class Ejercicio02 extends Agent {
   protected void setup() {
    this.addBehaviour(new Behaviour() {
        public void action() {
            System.out.println("Hola Mundo. Ejercicio2");
        }
        public boolean done() { return true; }
    }); } }
```

Comportamientos

Ejercicio 3: Repetir el agente anterior, con comportamiento en clase separada.

```
public class B_Ej03 extends Behaviour{
   public void action() {
        System.out.println("Hola Mundo. Ejercicio3");
   }
   public boolean done() {
        return true;
   }
}
```

```
protected void setup() {
    B_Ej03 ej3 = new B_Ej03();
    addBehaviour(ej3);
}
```

Comportamiento

- Ejercicio 4: Implementar un comportamiento que cuente desde 1 hasta 10

```
public class B_Ej04 extends Behaviour{
   int contador = 1;
   public void action() {
        System.out.println(contador);
        contador++;
   }
   public boolean done() {
        //  CONDICION DE FINAL DE COMPORTAMIENTO
        return contador > 10;
   }
}
```

Comportamiento

- Variable: **myAgent**
- Ciertas acciones las realiza el agente
- Reutilización de los comportamientos

Tipos de comportamientos

- Comportamientos estándar:
  - **Behaviour**: Comportamiento genérico.
  - **OneShotBehaviour**: done() siempre devuelve “true”.
  - **CyclicBehaviour**: done() siempre devuelve “false”.
  - **TickerBehaviour**: se ejecuta periódicamente (dado en el constructor).
  - **FSMBehaviour**: máquina finita de estados(Finite State Machine).
  - .....

Ejemplo

- Ejercicio 5: Crear un comportamiento cíclico que lea un sensor constantemente

```
public class B_Ej05 extends CyclicBehaviour{
   private int estado = 0;
   public void action() {
        int medida = Sensores.getBumper();
        switch(estado) {
        case 0:
            // No ha tocado
        case 1:
            // Ha tocado
        }
   }
}
```

Tipos de comportamientos

- Los comportamientos además se **pueden componer** y formar comportamientos más complejos.
- Una implementación interesante sería el comportamiento **BDI** dentro de Jade.

Comportamientos Compuestos

- **FSMBehaviour** es un comportamiento compuesto.
  - De manera secuencial, y definiendo transiciones entre estados, se implementa la acción que debe realizar el comportamiento.
- **FSMBehaviour carece** de método **action()**

Comportamientos Compuestos

- Para la devolución del estado se sobreescribe el método **onEnd()**, que devuelve el entero que nos decidirá la transición de estado.

- También existen otros como: **SequentialBehaviour**, **ParallelBehaviour**, …

Ejemplo FSM

- Constantes

```
public class B_Ej06 extends FSMBehaviour {
    // estados FSM
    private static final String ESTADO_0  = "cero";
    private static final String ESTADO_1  = "uno";
    private static final String ESTADO_2  = "dos";
    private static final String ESTADO_3  = "tres";
    private static final String ESTADO_ERR  = "error";

    // Valores devueltos
    private final int CERO = 0;
    private final int CINCO = 5;
    private final int DIEZ = 10;
    private final int QUINCE = 15;
```

Ejemplo FSM

- Constructor

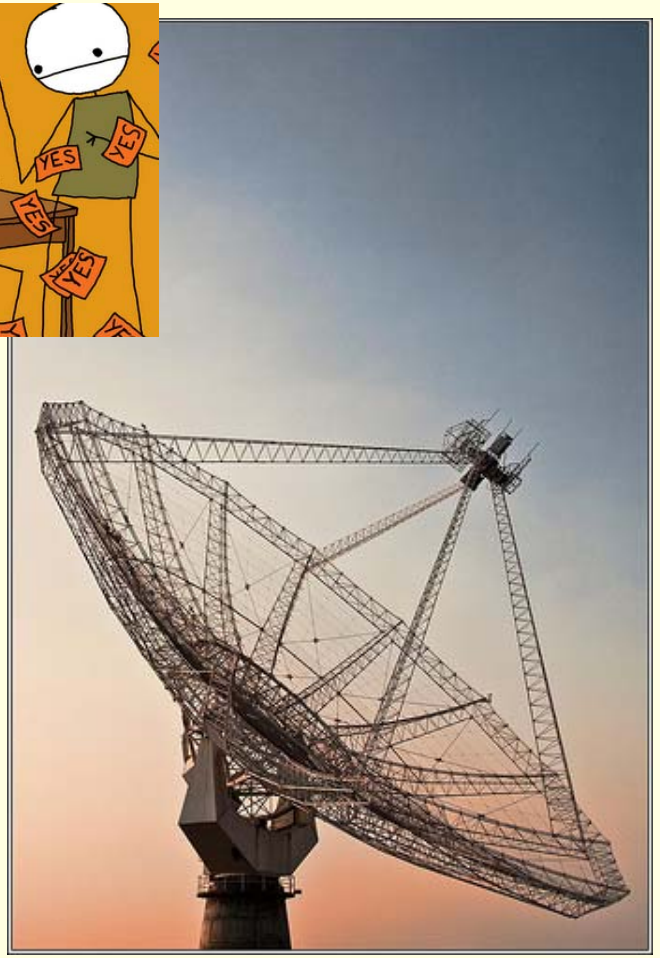
```
public B_Ej06(Agent a) {
        super(a);
        // Registrar los estados
        registerFirstState(new ZeroBehaviour(myAgent), ESTADO_0);
        registerState(new FiveBehaviour(myAgent), ESTADO_1);
        registerState(new TenBehaviour(myAgent), ESTADO_2);
        registerLastState(new TwentyFiveBehaviour(myAgent), ESTADO_3);
        // transiciones
        registerTransition(ESTADO_0, ESTADO_0, CERO);
        registerTransition(ESTADO_0, ESTADO_1, CINCO);
        registerDefaultTransition(ESTADO_0, ESTADO_ERR);

        scheduleFirst();
    }
```

Ejemplo FSM

- Comportamientos internos

```
 class ZeroBehaviour extends OneShotBehaviour {
        int transition = CERO;
        public void action() {
        // Cosas que hacer
        }

// Se ejecuta al final y es el encargado de devolver el valor.
        public int onEnd() {
            return transition;
        }
    }
```

Comunicación



Comunicación

- Comunicación TRANSPARENTE
- 3 niveles
  - Máquina: Eventos
  - Entre Máquinas misma red: RMI
  - Distinta red: CORBA

Comunicación

- La capacidad de comunicación = envío de mensajes ACL.

- En 1 mensaje podemos distinguir 4 partes *principales*:
  - Receptor y Emisor
  - Contenido
  - Directiva
  - Lenguaje y Ontologías

Comunicación

- El mensaje es un objeto de la clase

  jade.lang.acl.ACLMessage

- Intención del mensaje: “PERFORMATIVA”

- Además, posee otros parámetros: ReplyWith, ConversationId, etc

Comunicación

- Creación de un mensaje

```
ACLMessage mensaje = new
    ACLMessage(ACLMessage.REQUEST);
AID agente = new AID("agente2",AID.ISLOCALNAME);
mensaje.addReceiver(agente);
mensaje.setContent("Quiero Jugar");
myAgent.send(mensaje);
```

Comunicación

- **Todas las acciones de un agente, deben de estar implementadas como comportamientos.**
- La **comunicación** es una **acción**.
- Estos comportamientos podrián ser OneShot*.
- Los comportamientos cíclicos se usan para la recepción contínua de mensajes.

Parámetros de ejecución

- Si creamos la plataforma y ejecutamos varios agentes:
  - (sin parámetros especiales)
- Unir un agente a una plataforma creada (en la misma máquina=desde 2 aplic)
  - Parámetro: -container
- Unir un agente a una plataforma creada OTRA máquina
  - Parámetro: -host xxx.xxx.xxx.xxx -container
  - Parámetro: -host localhost -container

Comunicación

- Ejercicio 7: Crear un comportamiento de un disparo que envíe un mensaje a la plataforma del servidor.
  - Intención: “Inform”
  - Nombre del agente servidor: “servidor”
  - Ip de la plataforma:

Comunicación

```
AID    Destino; String Mensaje; int    Intencion;
public B_Ej07(String _destino, String _mensaje, int
   _intencion) {
   super();
   Destino = new AID(_destino,AID.ISLOCALNAME);
   Mensaje = _mensaje;
   Intencion = _intencion;
}
public void action() {
   ACLMessage mensaje = new ACLMessage(Intencion);
   mensaje.addReceiver(Destino);
   mensaje.setContent(Mensaje);
   myAgent.send(mensaje);
}
```

Comunicación

- Recepción de mensajes.
- Quien recibe el mensaje es el “agente”
  - myAgent.receive();
- Si no recibe mensaje, el comportamiento se debe de bloquear
  - block();
- receive() .vs. blockingReceive()

Comunicación

- Ejercicio 8: Crear un comportamiento cíclico que reciba mensajes y los muestre por pantalla.

```
public void action() {
   ACLMessage envio = myAgent.receive();
   if (envio != null) {
        System.out.println(envio.getSender().getLocalName()
                + ": " + envio.getContent());
   } else
        block();
}
```

Comunicación

- Existen plantillas de mensajes (MessageTemplate)
- Se usan para filtrar la recepción de mensajes
- Las plantillas se pueden componer.

```
ACLMessage a = myAgent.receive(
        MessageTemplate.and(
  MessageTemplate.MatchPerformative(ACLMessage.PROPOSE),
   MessageTemplate.MatchConversationId("12221")
                                ));
```

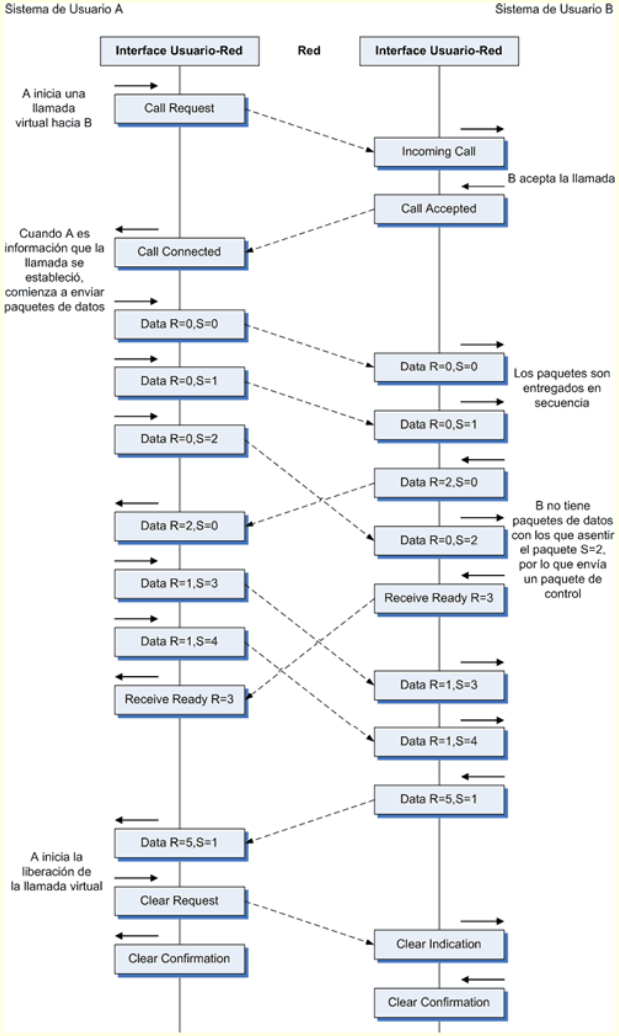
Ping-Pong

Ejercicio extra:

- Crear dos agentes, donde uno envíe un Ping a otro y este le conteste con el Pong correspondiente.
  - A1 envía un mensaje a A2
  - A2 lo escribe y RESPONDE
  - A1 lo escribe y se auto-finaliza

Protocolos


¿ El móvil?.
¿Se coloca a la izquierda
o a la derecha?


Sistema de Usuario A
Sistema de Usuario B
Interface Usuario-Red
Red
Interface Usuario-Red
A inicia una llamada virtual hacia B
Call Request
Incoming Call
B acepta la llamada
Call Accepted
Cuando A es información que la llamada se estableció, comienza a enviar paquetes de datos
Call Connected
Data R=0,S=0
Data R=0,S=0
Data R=0,S=1
Los paquetes son entregados en secuencia
Data R=0,S=1
Data R=0,S=2
Data R=2,S=0
Data R=2,S=0
B no tiene paquetes de datos con los que asentir el paquete S=2, por lo que envía un paquete de control
Data R=0,S=2
Data R=1,S=3
Receive Ready R=3
Data R=1,S=4
Data R=1,S=3
Receive Ready R=3
Data R=1,S=4
Data R=5,S=1
Data R=5,S=1
A inicia la liberación de la llamada virtual
Clear Request
Clear Indication
Clear Confirmation
Clear Confirmation

Protocolos

- **FIPA** establece unos protocolos estándares.
- Están basados en el uso de **directivas** (performativas).
- Algunos implementados en JADE
  - **FIPA Request** Interaction Protocol Specification
  - **FIPA Query** Interaction Protocol Specification
  - **FIPA Request When** Interaction Protocol Specification
  - **FIPA Contract Net** Interaction Protocol Specification
  - ...

Protocolos

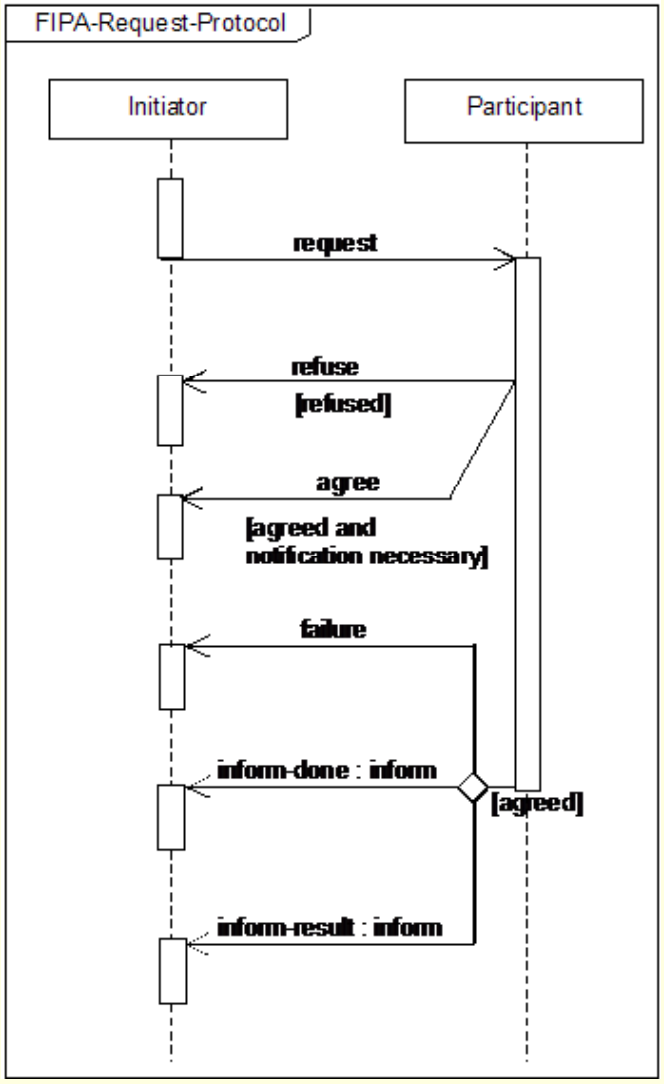
- JADE da libertad de lenguaje.
- JADE trae soporte para lenguaje SL y LEAP(binario)
- FIPA **recomienda** el lenguaje SL
- También se puede hacer uso de ontologías

Protocolos

- Ejercicio 8: Implementación del protocolo FIPA-REQUEST para conexión a una plataforma.
  - Nombre del servidor: “entorno”
  - Equipos: 7 → Azul, 8 → Rojo
  - Clave: “a”
  - Clase a sobreescribir:
    - SimpleAchieveREInitiator

Protocolos

- FIPA-REQUEST
  - Según FIPA



Protocolos

- Implementacion del protocolo

```
public B_Protocolo(Agent a, ACLMessage msg) {
    super(a, msg); }

public void   handleAgree(ACLMessage msg) {  }
public void   handleRefuse(ACLMessage msg) { }
public void   handleInform(ACLMessage msg) { }
public void   handleNotUnderstood(ACLMessage msg) { }
public void   handleOutOfSequence(ACLMessage msg) { }
```

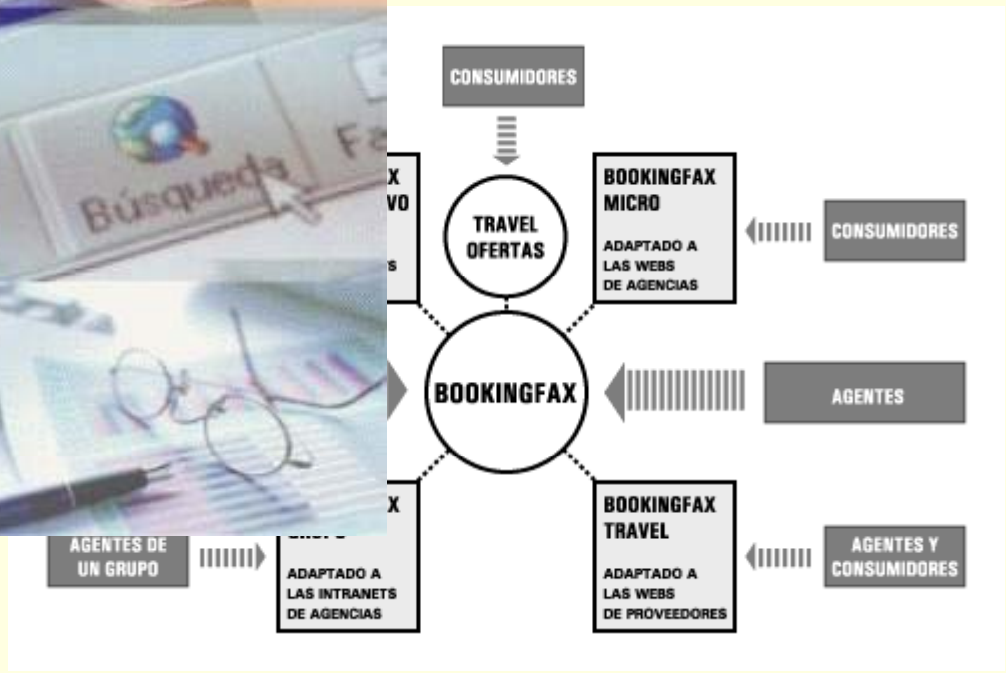
Protocolos

- Inicialización
  - Creación de un mensaje y
  - Activación del comportamiento.

```
import jade.domain.FIPANames.InteractionProtocol;
…

ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);

msg.addReceiver(new AID("entorno",AID.ISLOCALNAME));
msg.setContent(Equipo + Clave);
msg.setProtocol(InteractionProtocol.FIPA_REQUEST);

this.addBehaviour(new B_Protocolo(this,msg));
```

Servicios


WWW
Búsqueda


CONSUMIDORES
TRAVEL OFERTAS
BOOKINGFAX MICRO
ADAPTADO A LAS WEBS DE AGENCIAS
CONSUMIDORES
BOOKINGFAX
AGENTES
AGENTES DE UN GRUPO
ADAPTADO A LAS INTRANETS DE AGENCIAS
BOOKINGFAX TRAVEL
ADAPTADO A LAS WEBS DE PROVEEDORES
AGENTES Y CONSUMIDORES

Servicios

- 2 servicios fundamentales:
  - AMS → Páginas Blancas
  - DF → Páginas Amarillas

Servicios: Registro y búsqueda

- El registro dentro del servicio de ***Páginas Blancas*** es obligatorio para pertenecer a la plataforma.
- El registro en las ***Páginas Amarillas*** es opcional. En él se registran los servicios que ofrece cada agente, pudiéndose dar más de una entrada por agente.

Servicios: Registro y búsqueda

- “No tenemos por que saber como se llama el agente que posee nuestra partida”
- “El secreto compartido”, es decir, tenemos que saber ALGO de lo que queremos buscar.
- Los agentes se pueden registrar en las plataformas para poder ser buscados (*)

Registro en Páginas Blancas

- Se realiza automáticamente al unirse a la plataforma.
- Se puede realizar “a mano” si se desea,
- Si extendemos de la clase Agent, esta lo tiene implementado.

Registro en Páginas Amarillas

```
//-------------------------------------------------------------//
// Registrar Servicio en el DF
//-------------------------------------------------------------//
   DFAgentDescription dfd = new DFAgentDescription();
   dfd.setName(getAID());
   ServiceDescription sd = new ServiceDescription();
   sd.setType(“Servidor");
   sd.setName(“Tetris");
   dfd.addServices(sd);
   try {
     DFService.register(this,dfd);
     System.out.println(“Servidor:\t\tRegistrado");
   } catch(FIPAException fe) {
     fe.printStackTrace();
   }
```

Búsqueda en Páginas Amarillas

```
DFAgentDescription template = new DFAgentDescription();
sd = new ServiceDescription();
sd.setType(“Servidor");
template.addServices(sd);
AID[ ] creadores = null;
try {
     DFAgentDescription[ ] result = DFService.search(this,template);
     creadores = new AID[result.length];
     for (int i=0; i< result.length; i++) {
       creadores[i] = result[i].getName();
     }
} catch (FIPAException fe) {
      creadores = null;  fe.printStackTrace();
}
if (creadores == null) {
     this.doDelete();
}
```

De-Registro

- El borrado de las Páginas Blancas ES AUTOMÁTICO
- El borrado de las Páginas Amarillas ES MANUAL
  - Debemos de hacerlo cuando el agente muere, dentro del método **takeDown()**

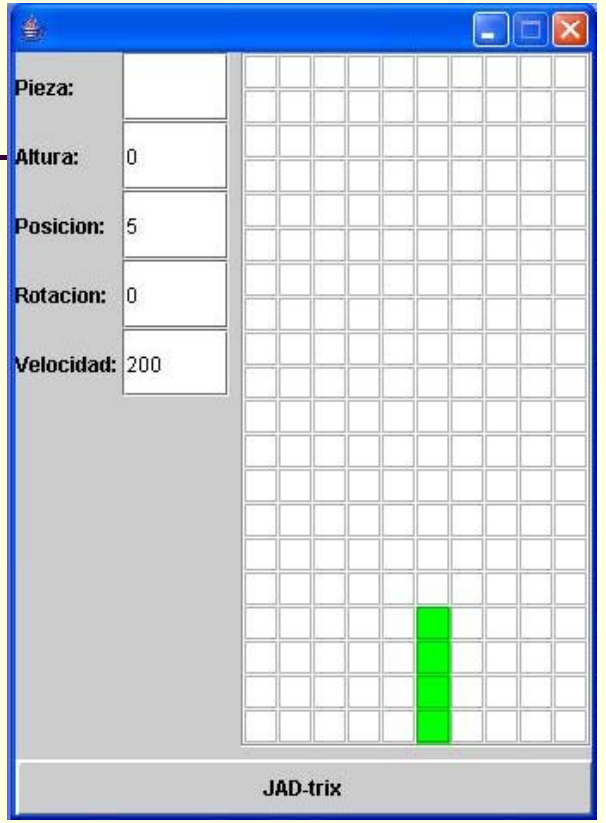
Unos casos prácticos

Tetris: Introducción.

- Sistema Multi-agente (2) que van a interactuar por medio de mensajes ACL para implementar el juego del Tetris

- Jade nos ofrece la facilidad de implementación rápida de comportamientos y la exportabilidad, casi inmediata, a un funcionamiento en red.

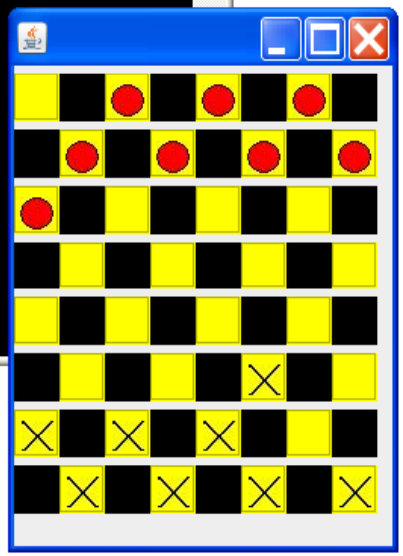
Agente Partida



Agente Jugador

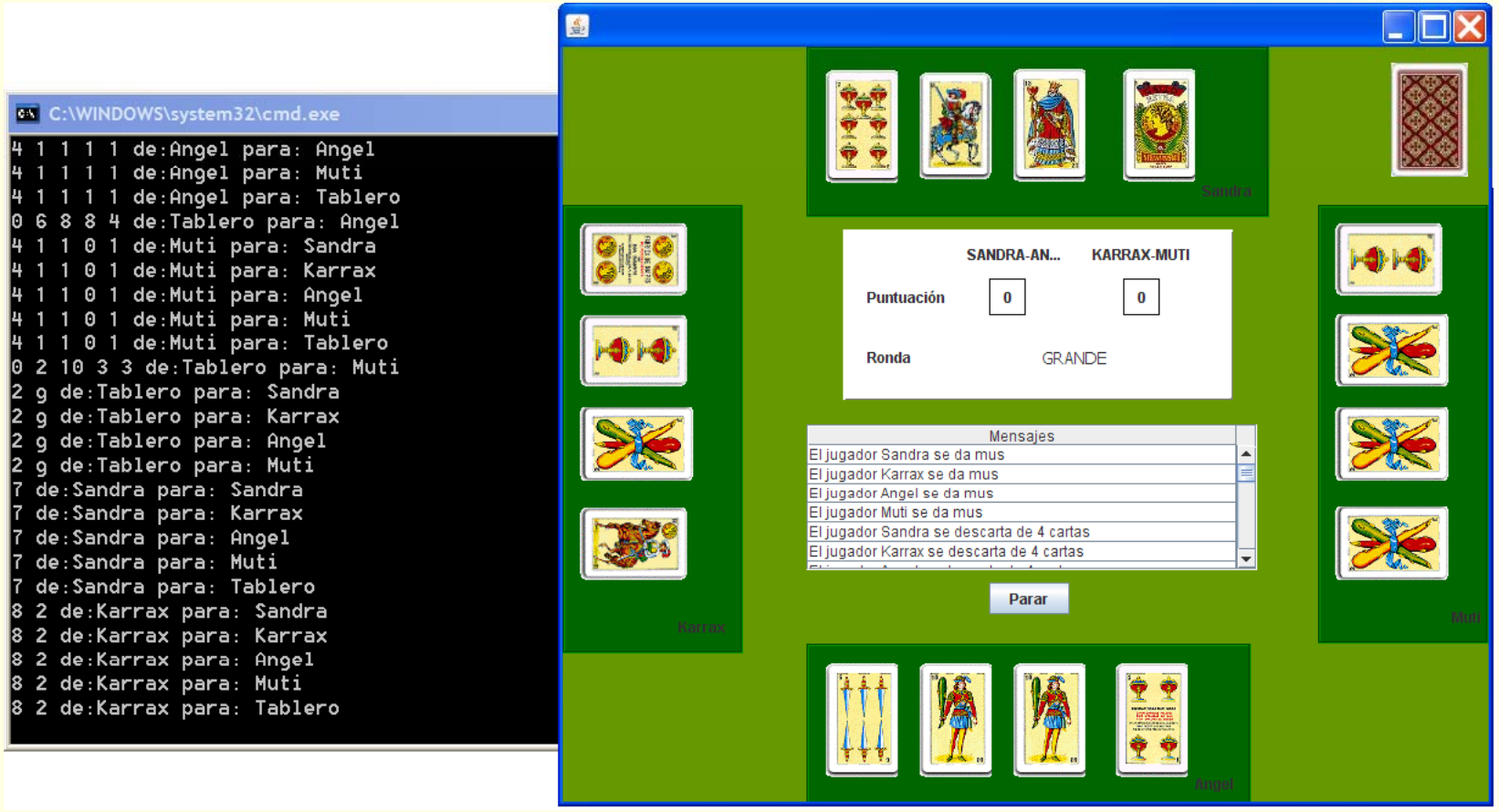
Juego de las damas.

```
arbitro- envio a judagor1 el tablero

jugador1- recibo tablero--->Tablero:o-B
jugador1- envio movimiento--->Jugador:1 Inicial:0,0 Destino:1,1


arbitro- recibo movimiento de-->jugador1@SANFERNAN2:1099/JADE
arbitro- el movimiento es -->Jugador:1 Inicial:0,0 Destino:1,1


NBoBoBoB
BoBoBoBo
oBNBNBNB
BNBNBNBN
NBNBNBNB
BNBNBxBN
xBxBxBNB
BxBxBxBx
```



Mus

- Colaboración entre agentes por un objetivo común: ganar

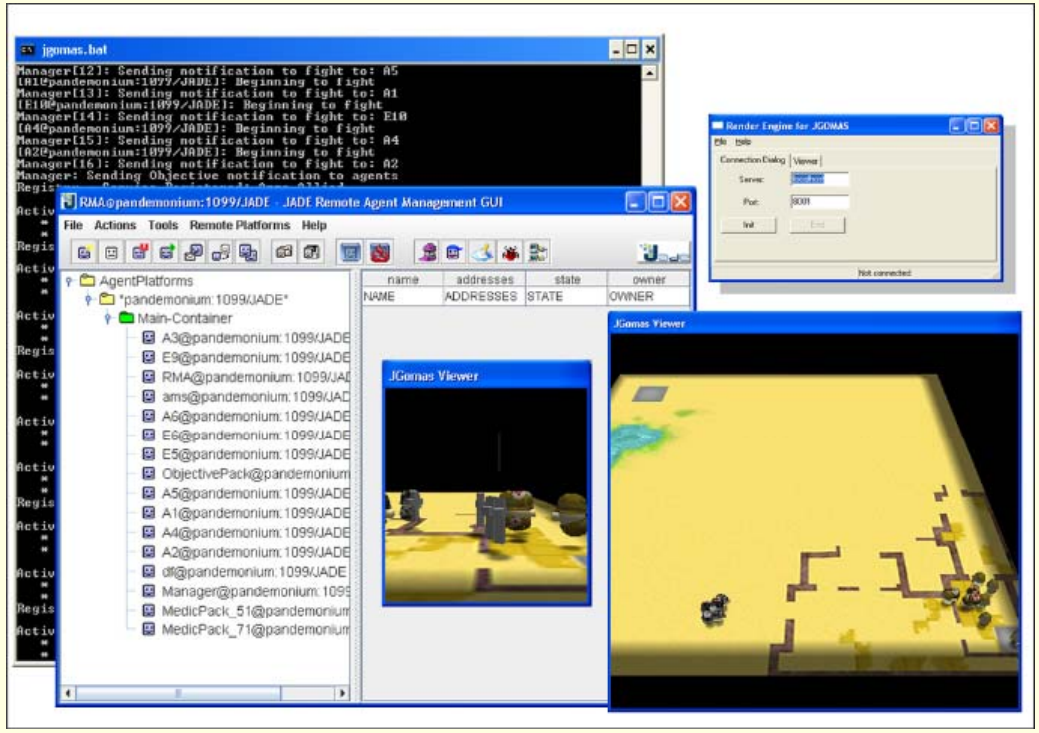


Mini-WarCraft

- Implementación del entorno como un agente, que es el que se encarga de la “física” del sistema.

JGomas

- **JGOMAS: Game Oriented Multi Agent System**
  - **basado en Jade**



UNIVERSIDAD POLITECNICA DE VALENCIA

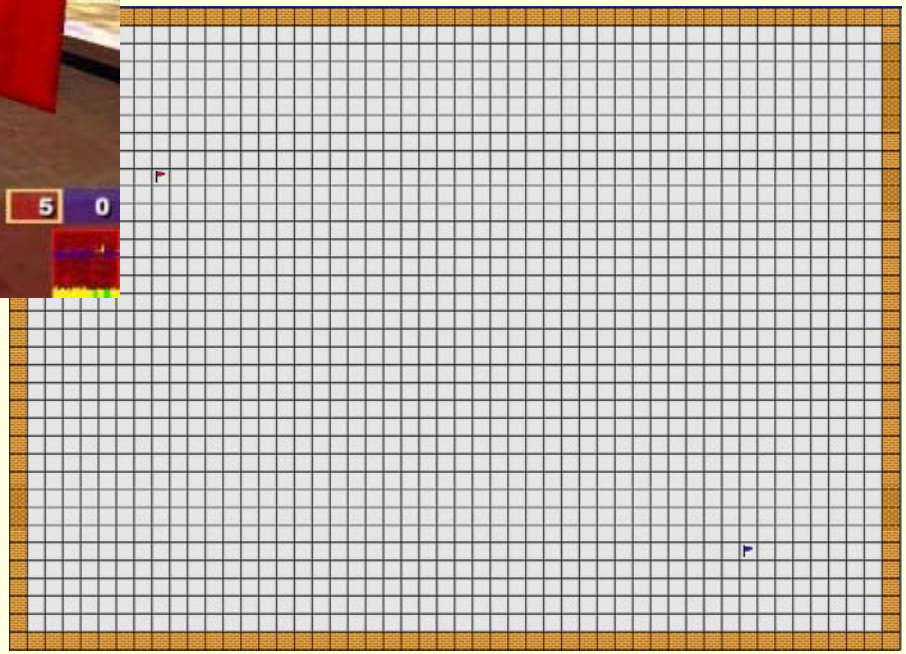
Grandes Retos…

Robocup Rescue Simul

Robocup (Soccer) Simulation

Y más…

CAPTURAR LA BANDERA

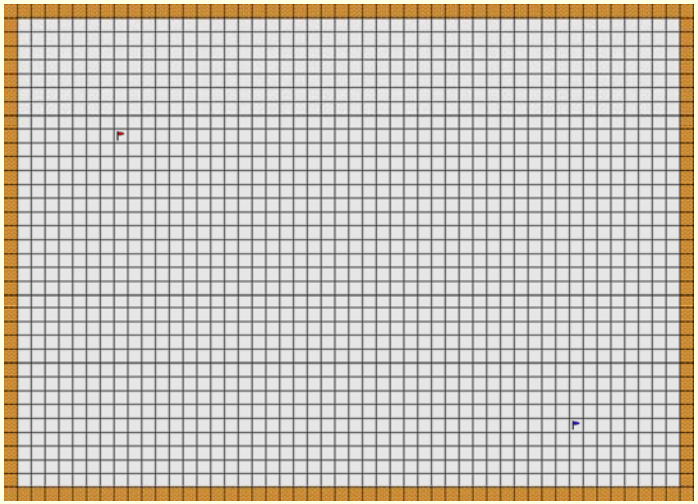


CAPTURAR LA BANDERA

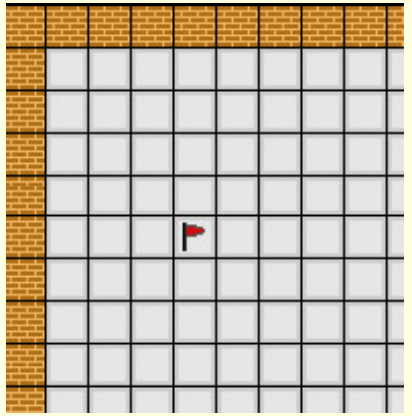
- Objetivo del juego:
  - Capturar la bandera del equipo contrario y llevarla a su base.
  - Evitar que los contrarios capturen su bandera.

CAPTURAR LA BANDERA

- Descripción:
  - Tablero de dimensión: n x m
  - Las celdas pueden
    - Estar libres: “ “
    - Pared: “H”
  - Por equipos (Rojo/Azul)
    - Bandera: “A” / ”B”
    - Base*: “C” / “D”
    - Jugadores: “1” / “2”
    - Jug con Bandera: “3” / “4”
  - (*) Sólo si no está la bandera



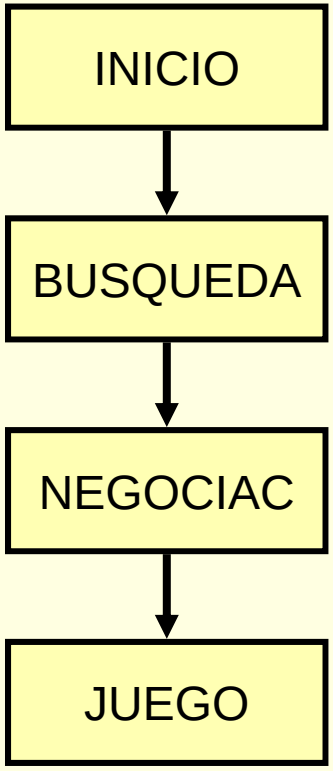
CAPTURAR LA BANDERA



- Descripción
  - Código de equipos
    - Rojo: “8”
    - Azul: “7”
  - Servicio de la partida
    - Nombre del servicio: "SERVER"
    - Tipo del servicio: "SERVIDOR_PARTIDA“
  - Protocolo de conexión:
    - FIPA-REQUEST
    - jade.proto.SimpleAchieveREInitiator

CAPTURAR LA BANDERA

- Flujo:



CAPTURAR LA BANDERA

- INICIO
  - Inicialización de la representación de datos internos (Cerebro).
  - Lanzamiento del comportamiento de búsqueda

CAPTURAR LA BANDERA

- Búsqueda
  - Búsqueda del servicio dentro del DF
  - Obtención del nombre del servidor y almacenarlo.

CAPTURAR LA BANDERA

- Negociación
  - Implementación del FIPA-REQUEST
  - Cuando se acepta,
    - En el inform se manda un mensaje con la información del tablero
      [ANCHO_MAPA],[ALTO_MAPA],
      [ANCHO_VENTANA],[ALTO_VENTANA],
      [POS_X],[POS_Y],
      [MAPA]
    - No envía la posición de los jugadores contrarios

CAPTURAR LA BANDERA

- Juego
  - El servidor envía cada x ms(1 ciclo) un mensaje con la ventana de visión del jugador.
  - Sólo acepta 1 acción por ciclo.
    - Si se envían varias, elige una al azar.
  - Si se realizan acciones NO permitidas, el agente es sancionado.

CAPTURAR LA BANDERA

- Movimientos posibles(Acciones):
  - Derecha: "1"
  - Izquierda: "2"
  - Arriba: "3"
  - Abajo: "4“
  - TableroCompleto: "10" -> Envia un string con el mismo formato del tablero parcial, pero con todo lo que contiene el mapa, jugadores del mismo equipo y rivales, banderas...
    - Penaliza con varios ciclos

CAPTURAR LA BANDERA

- Otras percepciones
  - "6" -> el jugador ha sido expulsado de la plataforma
  - Game Over: "9" -> Cuando el jugador gana la partida, se envia este mensaje

CAPTURAR LA BANDERA

- **Ejercicio final:**
  - Crear un jugador que, mediante un comportamiento basado en una máquina de estados finitos, vaya a por la bandera contraria y vuelva.

Enlaces interesantes

- http://programacionjade.wikispaces.com/
- http://dukechile.blogspot.com/search/label/JADE
- http://www.exa.unicen.edu.ar/catedras/tmultiag/apuntes.html
- http://jade.tilab.com
- …. GOOGLE!!!

Gracias por su asistencia!

Gonzalo A. Aranda Corral

Dpto. Tecnologías de la Información

Escuela Politécnica Superior “La Rábida”

Universidad de Huelva