

Práctica 2 - Simulación del Juego de las Palas de Playa (Campeonato)

Nombre estudiante: *Alberto Mur López*

Curso: *Sistemas inteligentes*
Docente: *Gregorio de Miguel*
Fecha de entrega: *December 10, 2023*

INTRODUCCIÓN

En esta práctica utilizaremos el framework JADE (Java Agent DEvelopment Framework), que es una plataforma software para el desarrollo de agentes. El objetivo de esta práctica será crear un simulador de campeonatos de palas de playa.

Para reutilizaremos parte del código generado en la práctica anterior. El objetivo de este desarrollo es utilizar un interfaz gráfico para el manejo de la aplicación. Para ello se ha recurrido al *framework* de Java Swing. La aplicación permite:

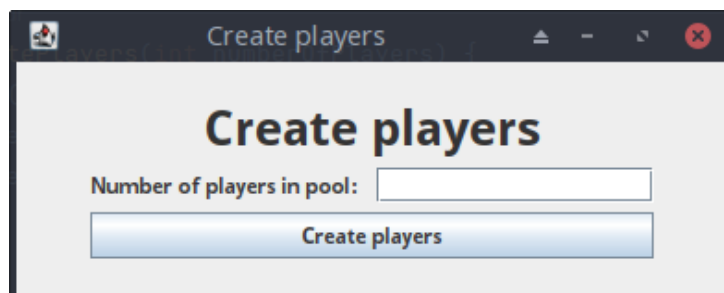
1. Crear un número arbitrario de jugadores.
2. Crear un campeonato con un número arbitrario de jugadores y jugar las diferentes rondas.

Aunque ha aumentado en parte su complejidad, se ha creado una ontología para el intercambio de mensajes. Con la ontología se han podido intercambiar estructuras de datos más complejas, facilitando su procesado y consistencia.

CREACIÓN DE JUGADORES Y JUGADORES

En el paquete generator se ha programado un agente que, a través de un formulario permite crear un número de agentes de tipo Player (Figura 1).

Figure 1: Creación de jugadores



The image shows a Java Swing window titled "Create players". Inside the window, there is a heading "Create players" in a large, bold, black font. Below this heading, there is a label "Number of players in pool:" followed by a text input field. At the bottom of the window, there is a button labeled "Create players". The window has a standard title bar with a small icon on the left and window control buttons (minimize, maximize, close) on the right.

Cuando se crean estos agentes se registran en el directorio de agentes como agentes con servicio *Player*. Una vez registrados inician el comportamiento *Call4PlayersResponder*, que es un respondedor para el protocolo de interacción *FIPA_CONTRACT_NET* y mensajes de tipo *Call4Players*.

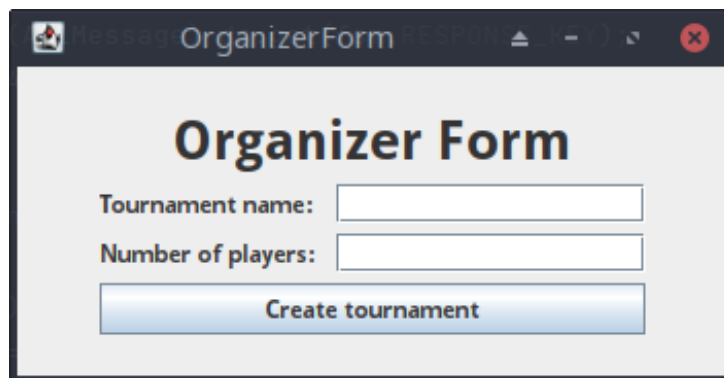
Este respondedor devuelve, con probabilidad 0.8 la propuesta del jugador de registrarse en el campeonato, enviando un mensaje con un contenido tipo *RegisterAction*. Si la propuesta es aceptada por parte del organizador, el comportamiento *Call4PlayersResponder* termina y se programa la ejecución de comportamiento *TournamentBehaviour* en el agente.

Este comportamiento, es un comportamiento paralelo. Esta pensando para ejecutar dos comportamientos en paralelo, el que recibe las ordenes de jugar un partido y el de finalización. En el momento de escribir este informe, sólo está implementado el comportamiento de jugar partidos *FixtureActionResponder*, implementado como un *AchieveREResponder* al que se le ha modificado el estado de *Prepare-result-notification*, para que sea un comportamiento de tipo *FixtureActionBehaviour*. Este comportamiento simula la partida de palas poniendo el resultado del partido como contenido del mensaje de respuesta.

ORGANIZADOR

El agente organizador muestra un formulario (Figura 2) al inicio para crear el campeonato en el que hay que indicar el nombre del campeonato y el número de jugadores.

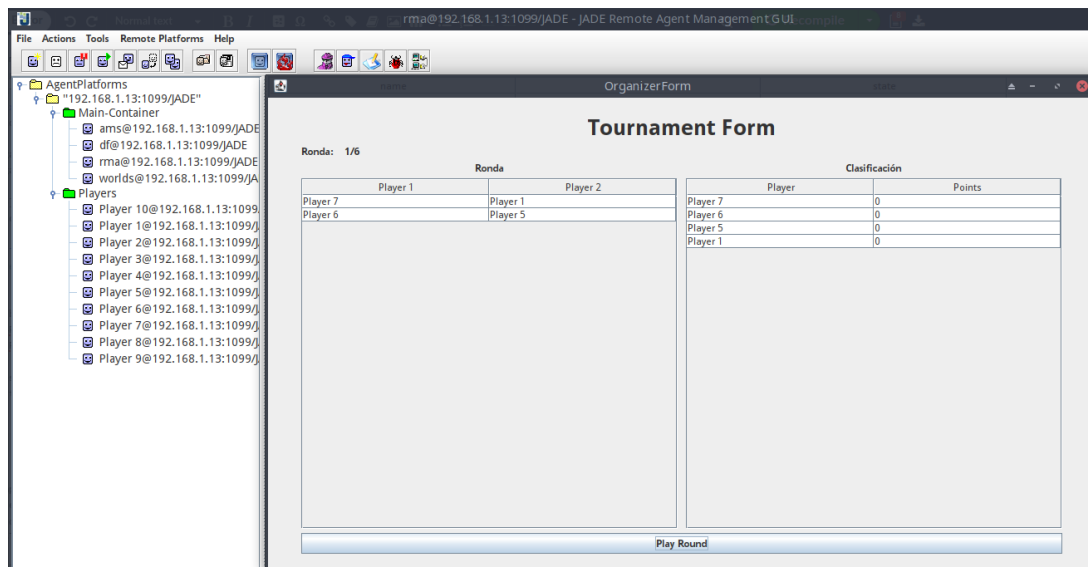
Figure 2: Creación de torneo

The image shows a window titled "OrganizerForm" with a standard Windows-style title bar. Inside the window, the title "Organizer Form" is displayed in a large, bold, black font. Below the title, there are two labels: "Tournament name:" and "Number of players:", each followed by a text input field. At the bottom of the form, there is a blue button with the text "Create tournament" in white.

Al pulsar el botón, el organizador se registra en el directorio. Y acto seguido inicia un comportamiento *Call4PlayersBehaviour* que inicia un protocolo *FIPA_CONTRACT_NET* contra los agentes que ofrezcan un servicio de tipo *Player* con el contenido *Call4Players*.

En este proceso puede haber algunos jugadores que rechacen participar o que lleguen tarde al registro, a los cuales se les rechaza. En el siguiente estado del protocolo, a los agentes que responden con un acción *RegisterAction* se les añade al listado de jugadores del torneo. Cuando finaliza este comportamiento se inicia el torneo (Figura 3).

Figure 3: Torneo



En esta imagen se muestra la información de la ronda actual y de la clasificación. Cuando se pulsa el botón de *Play round*, se inicia un comportamiento *RoundMapBehaviour* que es un comportamiento paralelo, donde para cada *Fixture* inicia un protocolo *FIPA_REQUEST* mediante el comportamiento *FixtureInitiator* cuyos destinatarios son los jugadores del partido. Cada *FixtureInitiator* comprueba en los resultados recibidos que ambas partes devuelvan el mismo resultado, como comprobante de que ninguno miente. Tras ello se actualiza el torneo con el ganador sumando un punto y se actualiza la tabla de clasificación. El comportamiento *RoundMapBehaviour* finaliza cuando todos los comportamientos han terminado. Cuando finaliza se prepara la siguiente ronda. Si ya no hay más rondas el torneo a terminado y se muestra la clasificación (Figura 4).

CONCLUSIÓN

En esta práctica se ha desarrollado un sistema multi-agente utilizando diferentes protocolos de interacción y ajustándolos según la necesidad. Además se ha empleado una ontología que añade una capa de adicional formalidad en las interacciones entre agentes. Con esta práctica se ha podido comprobar la complejidad que supone organizar y diseñar un sistema multi-agente

Figure 4: Torneo

