Ingeniería del conocimiento (06/07)

Práctica 1: Programación de un sistema de control para *SoccerBots* basado en reglas.

Luis de la Ossa

1 Introducción

El planteamiento de un problema de programación mediante un lenguaje de reglas difiere sustancialmente de los enfoques tradicionales basados en lenguajes imperativos u orientados a objetos.

El objetivo principal de esta primera práctica es la toma de contacto con este paradigma de programación mediante la implementación **de un sistema de control basado en reglas** para un robot software cuya misión es jugar al fútbol.

Para ello se utilizarán *TeamBots*, que es un entorno de simulación escrito en *Java,* y el motor de reglas *JESS,* también escrito en *Java* y diseñado expresamente para interaccionar con este lenguaje.

Además de describir el trabajo que ha de realizarse, en las siguientes secciones de esta memoria se proporciona una guía introductoria a los aspectos de estas dos herramientas con los que es necesario familiarizarse para el desarrollo de la práctica, y cuyo aprendizaje ha de ser completado mediante la revisión de los códigos fuente, *javadocs,* y documentación disponibles en las distribuciones e Internet.

El objetivo es que la mayor parte del esfuerzo se dedique al diseño del sistema de reglas.

2 Teambots y SoccerBots

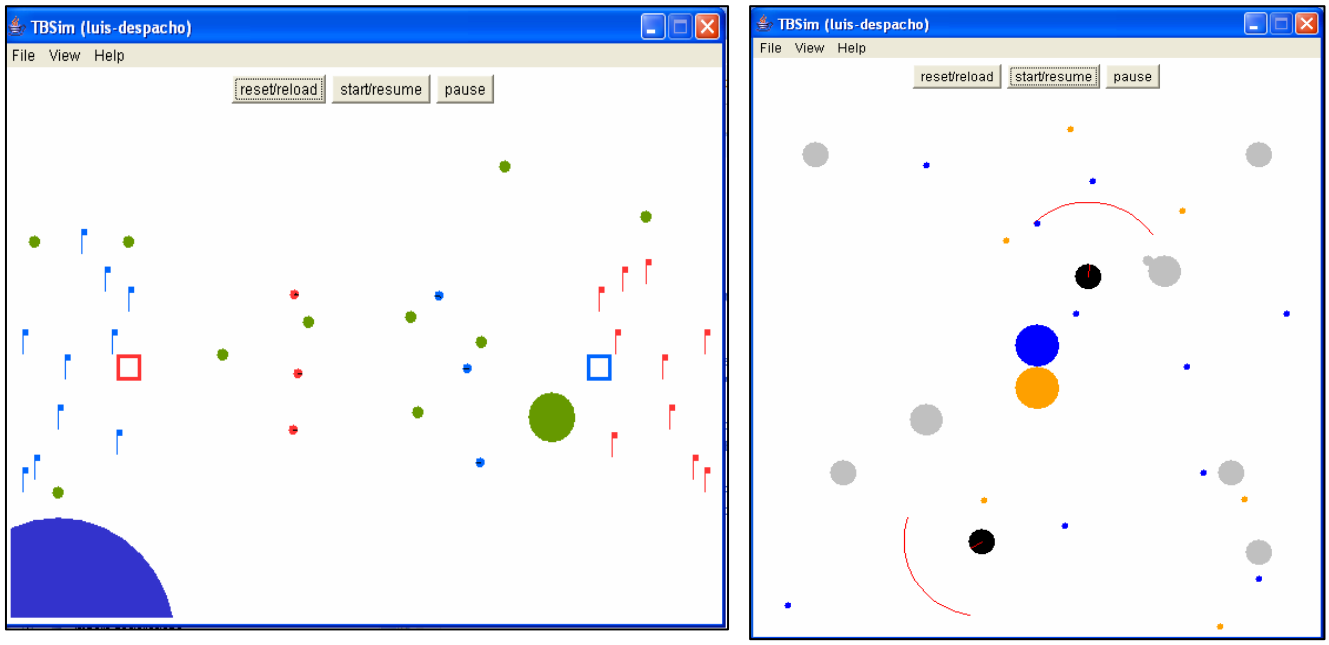
Un robot es un dispositivo hardware que se caracteriza por estar dotado de movilidad, y de elementos que le permiten interaccionar con el entorno en que se encuentra. Estos elementos se dividen en dos grupos:

- **Sensores**: Que son los que le suministran información sobre el entorno.
- **Actuadores**: Que son los que le permiten operar en él.

Una parte esencial en el funcionamiento de un robot es su sistema de control, que tiene como finalidad determinar qué acciones ha de emprender este en función de las entradas recibidas por los sensores.

Teambots es una librería de clases *Java* desarrollada en la *Carnegie Mellon University* y en el *Georgia Institute of Tecnology* que permite la prueba de sistemas de control para robots de diversas características.

Aunque está diseñada para el control de máquinas reales, cuenta también con un simulador `TBSim` que permite testear los sistemas de control en distintos tipos de escenarios virtuales. Estos se pueden construir a medida mediante la programación de los robots, sus controladores, y la definición del entorno en el que actuarán por medio de un fichero de configuración.



2.1 Instalación y ejecución de Teambots.

Ya que *TeamBots* es un software escrito en *Java*, es necesario tener instalada una máquina virtual cuya versión sea mayor o igual a la 1.2.

En la web www.teambots.org puede descargarse el archivo `TeamBots.zip` que contiene todas las clases, códigos fuente y ayuda que componen la librería. Supongamos que la ruta a la carpeta descomprimida será: `d:\TeamBots.`

Para que *Java* pueda ejecutar correctamente un programa, ha de acceder a todas las clases que este va a usar. Es necesario, por tanto, indicarle a la máquina *Java* la ubicación de las clases que componen *TeamBots*. Esto puede hacerse de dos modos:

1) Fijando la variable de entorno `CLASSPATH`:

```
CLASSPATH=d:\TeamBots\src;d:\Teambots\lib\collections.jar;
```

Con lo que ya se podría invocar el programa directamente:

```
java TBSim.TBSim [ruta]\fichero.dsc 511 300
```

2) Invocando directamente java con el argumento `-cp classpath`

```
java -cp d:\TeamBots\src;d:\Teambots\lib\collections.jar;
TBSim.TBSim [ruta]\fichero.dsc 511 300
```

Tal y como se verá más adelante, `fichero.dsc` contiene la descripción de la simulación que se va a lanzar. Por ejemplo:

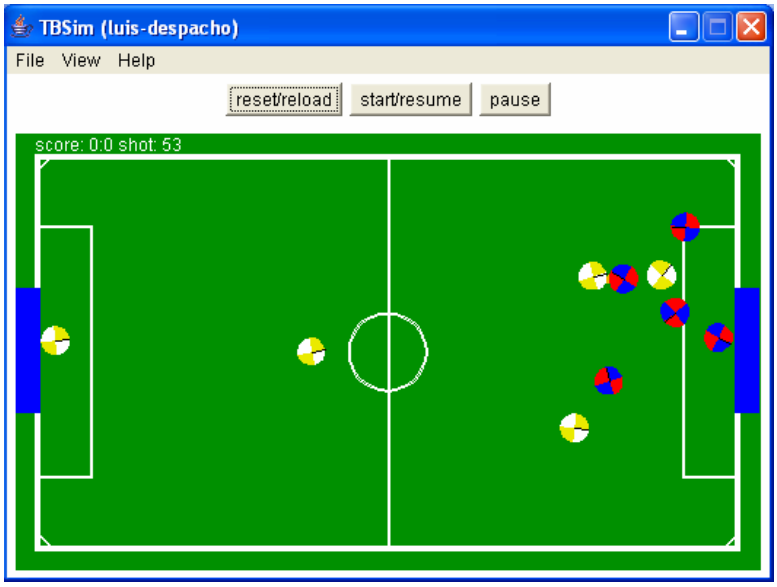
`d:\Teambots\Domains\CTF\basic.dsc`

Los dos últimos argumentos con que se invoca al simulación definen las dimensiones del entorno de simulación.

2.2 SoccerBots

La *RoboCup* es un evento de carácter científico-académico en el que equipos de robots de distintas categorías compiten en ligas de fútbol.

Uno de los entornos definido en *TeamBots, SoccerBots*, simula la dinámica y dimensiones que se utilizan en la liga de robots pequeños de la *RoboCup*. En ella, equipos de 5 robots compiten en una mesa de ping-pong golpeando una pelota de golf naranja.



Con el fin de hacer una primera aproximación al funcionamiento de *SoccerBots*, se describirán el fichero que describe el entorno de simulación y, posteriormente, los detalles relevantes de los robots y controladores que se usan en él.

`Robocup.dsc`

Como se ha comentado anteriormente, los diferentes escenarios de simulación se definen mediante ficheros de configuración cuyas líneas sirven para establecer tanto las propiedades generales del entorno, como los objetos y robots que forman parte de él.

`Robocup.dsc` es el fichero usado para la simulación de *SoccerBots*, y va a servir como referencia para definir los distintos escenarios y configuraciones. A continuación, se describen las distintas líneas de este fichero.

```
// bounds left right bottom top
   bounds -1.47 1.47 -.8625 .8625
```

Son los límites del área donde se mueven los robots.

```
// seed number
   seed 3
```

Es la semilla de números aleatorios.

```
// time accel_rate
   time 1.0
```

Velocidad de la simulación con respecto al tiempo real. Si se fija a 2.0 la simulación iría dos veces más rápido. Sin embargo, es preferible dejarlo a 1.0 o incluso a 0.5, ya que los cálculos para determinar las acciones de cada robot pueden resultar lentos dependiendo de lo complejos que sean (que podría ser el caso de *Jess*).

```
// timeout time
   timeout 60000
```

Indica la duración de la simulación en milisegundos.

```
// background color
   background x009000
```

Define el color del área donde se va a llevar a cabo la simulación.

Una vez definidos los parámetros generales para el entorno de simulación, es necesario especificar qué tipo de objetos formarán parte del entorno. Para cada uno de ellos encontraremos una línea del tipo:

```
// object objecttype x y theta forecolor backcolor visionclass
```

Donde `objecttype` es el tipo de objeto, `x , y,theta,` fijan la posición inicial y orientación en el área, `forecolor` y `backcolor` son los colores y `visionclass` sirve para ubicar cada objeto en una clase perceptual. De ese modo, cuando los sensores de robots simulados busquen objetos u otros robots, los pueden clasificar.

A continuación se enumeran los objetos que definen el entorno de simulación en *SoccerBots*:

- El campo de fútbol, que está definido por el objeto `SocFieldSmallSim`:

```
object EDU.gatech.cc.is.simulation.SocFieldSmallSim 0 0 0 0 x009000
x000000 0
```

- Los objetos que representan los ángulos de los corners(que son objetos circulares invisibles de un metro cuyo centro está fuera del campo)

```
object EDU.gatech.cc.is.simulation.ObstacleInvisibleSim 2.047 1.4396 0 1.0
x000000 x000000 0

object EDU.gatech.cc.is.simulation.ObstacleInvisibleSim -2.047 1.4396 0
1.0 x000000 x000000 0

object EDU.gatech.cc.is.simulation.ObstacleInvisibleSim 2.047 -1.4396 0
1.0 x000000 x000000 0

object EDU.gatech.cc.is.simulation.ObstacleInvisibleSim -2.047 -1.4396 0
1.0 x000000 x000000 0
```

- La pelota, que como puede verse, tiene 0.02 metros de diámetro (con los robots hardware es una pelota de golf).

```
object EDU.gatech.cc.is.simulation.GolfBallNoiseSim 0 0 0 0.02
xF0B000 x000000 3
```

Por ultimo, hay que definir los robots. Para cada uno de ellos, encontraremos una línea

```
// robot robottype controlsystem x y theta forecolor backcolor visionclass
```

Donde `robottype` es la clase de robot y `controlssytem` la clase que va a implementar el control para ese tipo de robot. En el caso de *SoccerBots,* los robots están definidos por la clase *SocSmalSim*, y con respecto a sus sistemas de control, existen varios en la libraría.

- **`robot EDU.gatech.cc.is.abstractrobot.SocSmallSim AIKHomoG -1.2 0 0 xEAEA00 xFFFFFF 1`**

- **`robot EDU.gatech.cc.is.abstractrobot.SocSmallSim SchemaDemo 1.2 0 0 xFF0000 x0000FF 2`**

A cada robot se le asigna un número en función del orden de aparición en el fichero.

`SocSmallSim`

La interfaz *SocSmall* permite interactuar con robots cuyas especificaciones son las establecidas en la liga de robots pequeños en la *RoboCup*.

Como se ha comentado anteriormente, existe una clase *SocSmallSim* que implementa las características de estos (y por tanto la interface *SocSmall*), en el entorno de simulación. Estas características son:

Sensores

- Detectar cuando el robot está en posición de golpear la pelota.
- Obtener un vector apuntando a la pelota.
- Obtener vectores apuntando a la propia portería y a la contraria.
- Obtener un array de vectores apuntando a los compañeros.
- Obtener un array de vectores apuntando a los oponentes.
- Obtener la posición propia.
- Obtener la orientación.
- Obtener el tiempo en milisegundos desde que el partido empezó.

Actuadores:

- Golpear la bola a 0.5 metros/seg.
- Conducir la bola.
- Girar hasta 360 grados/seg.
- Establecer la velocidad deseada hasta 3 metros/seg.

En la clase `EDU.gatech.cc.is.abstractrobot.SocSmall.java` y las interfaces `SimpleInterface`, `KinSensor`, `KickActuator`, `GoalSensor`, `BallSensor`, y `Transceiver` pueden consultarse las variables y funciones que determinan y dan acceso a estas funcionalidades.

ConstrolSystemSS

ControlSystemSS es la clase que implementa un sistema de control para un robot *SocSmall*, por lo que, cualquier clase que defina el comportamiento de un jugador ha de extender esta.

Básicamente, son dos los métodos que han de implementarse:

- **`public void`** `Configure()`: Que sirve para inicializar los controles de los distintos jugadores.
- **`public void`** `TakeStep()`: La simulación consiste en la actualización del estado de los robots en intervalos discretos de tiempo, y es en cada uno de estos instantes cuando el sistema de control recoge la información procedente de los sensores del robot y provoca unas acciones. Esta tarea ha de implementarse dentro de esta función.

Por tanto, en la función *TakeStep*, han de llevarse a cabo tres acciones: recoger la información, procesarla, y producir unas acciones. Siendo la fase de proceso lo que distingue a unos controladores de otros.

El siguiente cuadro muestra el código que obtiene la información de los sensores y la almacena (en muchos casos en vectores de tamaño dos).

```
long curr_time = abstract_robot.getTime();
Vec2 ball = abstract_robot.getBall(curr_time);
Vec2 ourgoal = abstract_robot.getOurGoal(curr_time);
Vec2 theirgoal = abstract_robot.getOpponentsGoal(curr_time);
Vec2[] teammates = abstract_robot.getTeammates(curr_time);
Vec2[] oponents = abstract_robot.getOpponents(curr_time);
int mynum = abstract_robot.getPlayerNumber(curr_time);
```

Finalmente, ha de llevarse a cabo una acción con los valores que se hayan obtenido como fruto del proceso.

```
// set the heading
abstract_robot.setSteerHeading(curr_time, grados);

// set speed at maximum
abstract_robot.setSpeed(curr_time, velocidad);

// kick it if we can
if (abstract_robot.canKick(curr_time))
      abstract_robot.kick(curr_time);

// tell the parent we're OK
return(CSSTAT_OK);
}
```

3 Jess

JESS es un motor de reglas que usa una sintaxis similar a la de *CLIPS,* por lo que los ficheros que describen los sistemas pueden ejecutarse en ambos entornos.

Además de la portabilidad, la principal ventaja que presenta *JESS* es que está diseñado expresamente para interaccionar con programas escritos en *Java*.

La librería puede descargarse en http://herzberg.ca.sandia.gov/jess/.

Como podrá verse, hay dos versiones para descargar, una libre y una gratuita. La gratuita contiene simplemente las clases (sin fuentes) pero es suficiente para realizar esta práctica por que la documentación está disponible.

Para utilizar las clases de *JESS* es necesario, tal y como ocurría con TeamBots, indicar donde están estas. Se puede hacer también de las dos maneras:

1) Fijando la variable de entorno `CLASSPATH`:

```
CLASSPATH=d:\Jess\lib\jess.jar
```

Con lo que ya se podría invocar el programa directamente:

```
Java jess.Main
```

2) Invocando directamente java con el argumento –cp classpath

```
Java –cp d:\Jess\lib\jess.jar jess.Main
```

Como se puede comprobar, `Main` ejecuta *JESS* en modo consola. Para ejecutarlo en una ventana es necesario invocar la clase `jess.Console`.

4 Desarrollo de la práctica

La práctica consiste en diseñar un sistema de control basado en reglas para robots del tipo `SocSmallSim.`

Para ello, se ha de utilizar una clase, `jugador.java`, que implementa un sistema de control genérico para estos robots, es decir, extiende `ControlSystemSS`, y que se encarga de leer un fichero con extensión .clp que contendrá las reglas, y de ejecutarlas en cada paso de la simulación.

Por tanto, el trabajo consiste fundamentalmente en escribir este sistema de reglas y, si procede, modificar el código de la clase jugador.java

Además, han de tenerse en cuenta dos restricciones en la mecánica del juego.

1) Los equipos estarán formados solamente por 3 robots.
2) El sistema de reglas controlará el funcionamiento de los 3 robots de manera indistinta y actuará en función del estado de cada uno.

5 Evaluación

En la evaluación de la práctica se considerarán dos aspectos fundamentales:

1) El diseño de la estrategia.
2) Su implementación.

Se valorarán, la complejidad del sistema, el número de niveles y, por supuesto, la corrección en la implementación.

Ha de elaborarse una memoria que incluya una introducción y planteamiento del problema, la descripción completa de la estrategia (sin código) y una descripción del código utilizado.

Asimismo, han de entregarse los ficheros `jugador.java` y `jugador.clp` correspondientes a la implementación de los controladores.

La fecha tope de entrega es el 20 de abril.

Una vez recogidas todas las prácticas, se hará un torneo en cada grupo, y se multiplicarán las notas del primero, segundo y tercero por 1.3, 1.2 y 1.1 respectivamente.

Anexo A: Programación de TeamBots y Jess usando eclipse.

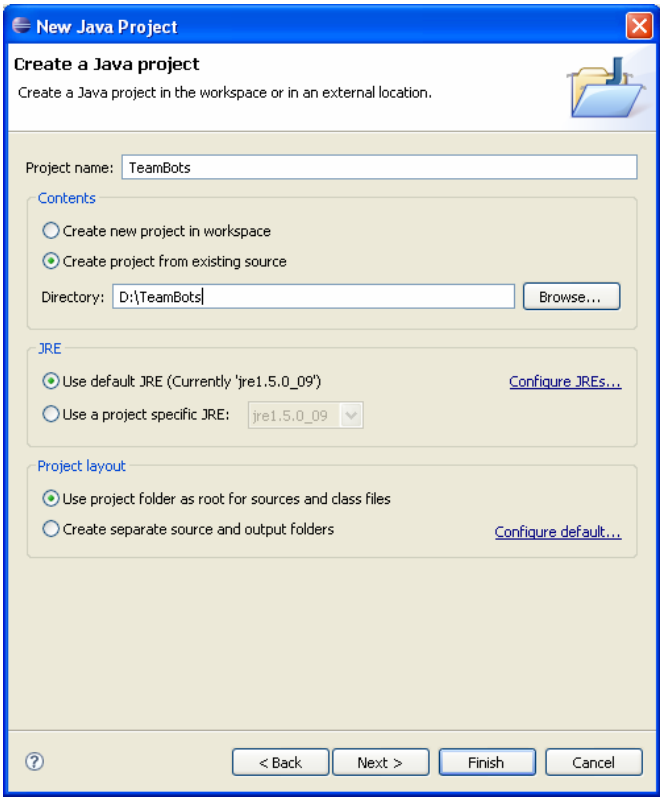
Tanto TeamBots como Jess están escritos en Java standard, por lo que no es necesario utilizar ninguna herramienta concreta en el desarrollo de la práctica salvo una versión de Java igual o superior a la 1.2.

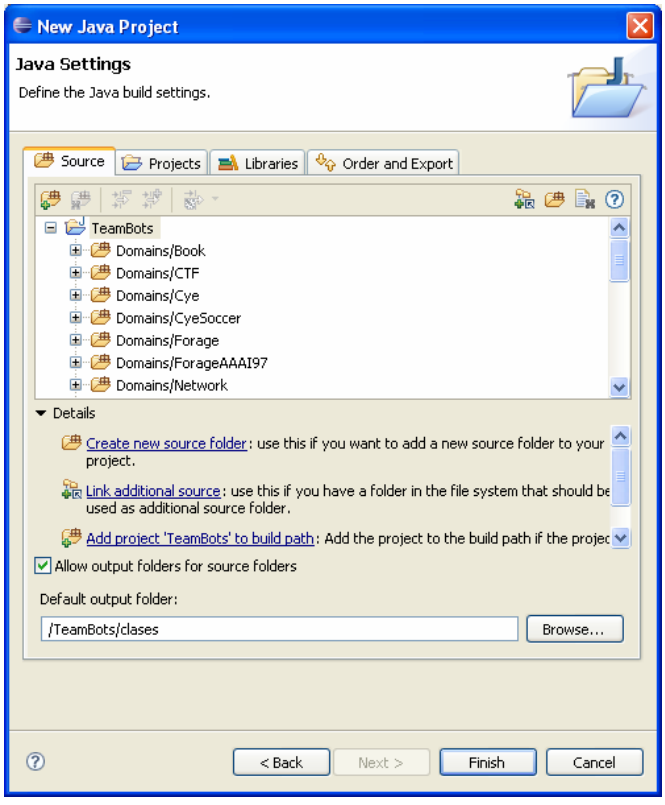
Con el fin de agilizar el proceso de instalación y configuración del código, se explicará detalladamente como hacerlo utilizando eclipse (www.eclipse.org) que es un entorno de programación libre que ha cobrado mucha importancia en los últimos años debido a su versatilidad.

En la primera ejecución (aunque luego es configurable), eclipse te pide un directorio trabajo (*workspace*) que será el que use para guardar el código de los proyectos, las configuraciones, parámetros de ejecución, etc.

El primer paso que hay que dar cuando se trabaja con eclipse es crear un proyecto. Para esto, iremos al menú *File→New Project*, y una vez allí, se nos abrirá una ventana en la que hemos de elegir el tipo de proyecto. En este caso *Java Project*.

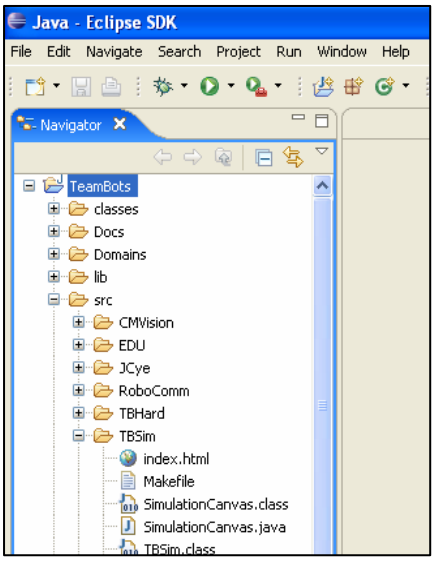
Una vez elegido elegida esa opción nos aparecerá una ventana:



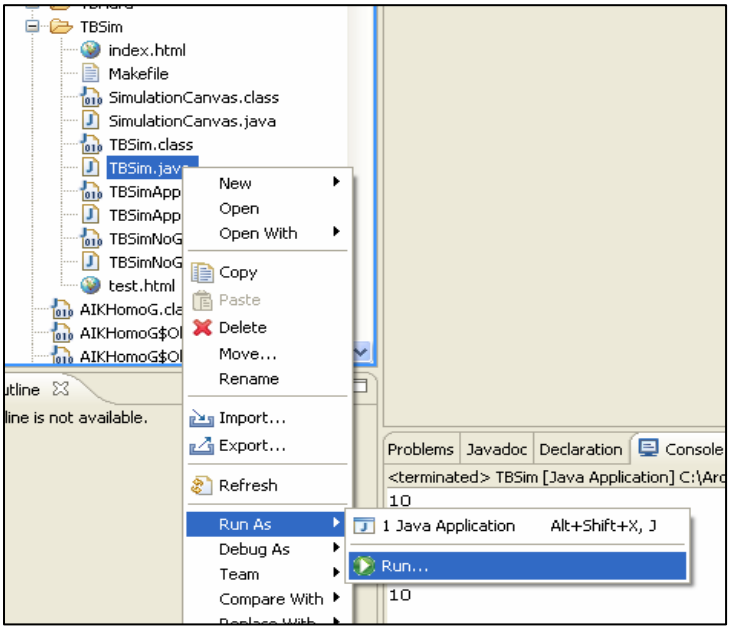


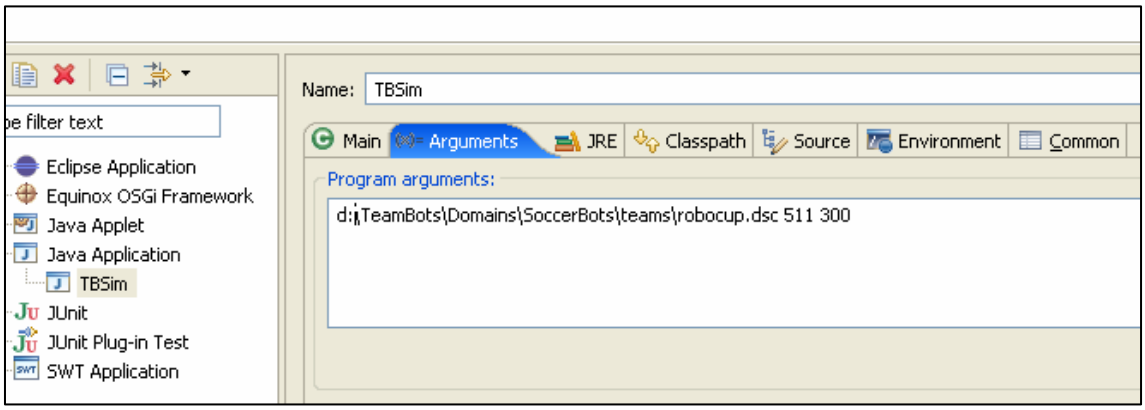
Como puede verse, además de nombrar el proyecto, aparecen dos opciones: Crear un nuevo proyecto y trabajar con código existente. En el primer caso, se creará una carpeta con el nombre del proyecto en el espacio de trabajo y habrá que importar el código desde `d:\TeamBots`, mientras que en el segundo caso, se trabajará directamente con el código ubicado en esa carpeta. Vamos a elegir la segunda de las opciones. Hemos de tener en cuenta que **el código existente no puede estar en la carpeta de trabajo que usa eclipse.**

Una vez hecho este paso, ya tendremos el código disponible para trabajar.



Para ejecutar los TeamBots, hay que invocar al simulador general TBSim con los parámetros correspondientes. Esto se hace así:





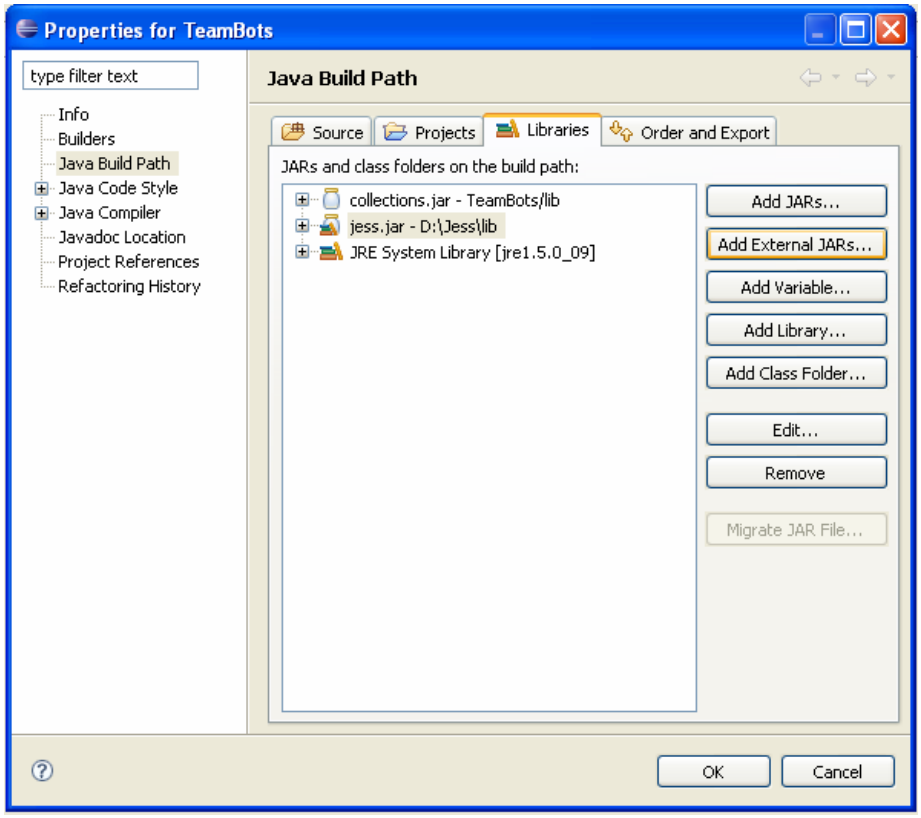
Como puede verse, se ha pasado al simulador el archivo `robocup.dsc` que es el que define el entorno `SoccerBots`.

Ejemplos de robots programados para el futbol pueden ser encontrados en el directorio `Domains\SoccerBots\teams.`

Por último, para que el sistema puede compilar el robot que se ha programado en la práctica, es necesario importar la libería Jess, que vamos a suponer que se ubica en `D:\Jess.`

Para ello nos situamos en la carpeta principal del proyecto y abrimos sus propiedades: *botón derecho del ratón → Properties.* Una vez en la ventana de propiedades, elegimos en el menú de la derecha *Java Build Path* y abrimos la pestaña *Libraries.*

En uno de los botones, se da la opción de añadir un archivo *jar* externo. Buscamos `jess.jar`, que se ubica en `D:\Jess\lib`, y la añadimos.



Por último, apuntar que para el trabajo en eclipse no es necesario haber modificado la variable de entorno CLASSPATH, aunque como se ha comentado anteriormente, sí que lo es para ejecutar los programas en línea de comandos.

Plug-ins de Jess para eclipse.

En la distribución 7.0 de *JESS* existen unos plug-ins para eclipse. La instalación de estos posibilita el uso de eclipse para trazar y depurar archivos escritos en *JESS*, que por defecto han de tener, al igual que los de CLIPS, extensión .clp.