Pequeño tutorial de fútbol de robots en Squeak

1. Herramientas a utilizar

Las herramientas a utilizar serán el simulador RobotSoccer v1.5a que puede conseguirse en http://www.fira.net/soccer/simurosot/overview.html y el paquete de fútbol de robots desarrollado en Squeak que puede encontrarse en: http://www.squeaksource.com/RobotSoccer/.
El simulador RobotSoccer v1.5a no tiene ningún mecanismo intrínseco para permitir la comunicación con Squeak, sólo permite cargar equipos en forma de "dll" programadas en Lingo o C++. Dadas sus limitaciones, es algo muy común entre los desarrolladores de equipos de fútbol de robots la programación de una librería que se carga en el simulador como si fuera un equipo y que funciona como un intermediario entre el simulador y cualquier plataforma de software (comunicándose por sockets envía la información del ambiente y recibe la velocidad de los robots). Para trabajar con Squeak decidimos utilizar una librería desarrollada en C++ por la Universidad del Comahue, la cual modificamos ligeramente para poder configurar los parámetros de conexión. A esta librería la llamamos "proxy.dll".

2. Puesta en marcha del equipo de ejemplo

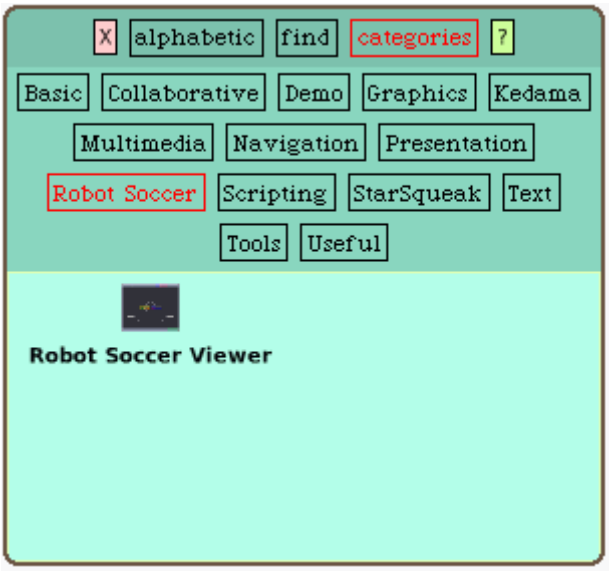
Una vez instalado el software del simulador y el paquete de fútbol de robots en Squeak, es necesario copiar el archivo "proxy.dll" que viene con el sotware de fútbol de robots en las carpetas que utiliza el simulador para guardar los equipos ("C:\Strategy\Blue" para el equipo azul y "C:\Strategy\Yellow" para el amarillo). Luego es necesario configurar la información de comunicación entre el simulador y Squeak. Coloque en las mismas carpetas que contienen el archivo "proxy.dll" un archivo de texto llamado "proxy.config" que contenga los datos siguientes como se muestra a continuación:

Ip de la máquina donde se encuentra Squeak
Número del servidor (puerto por el que se envía la información a Squeak)
Número del cliente (puerto por el que se recibe información desde Squeak)

La configuración por defecto para trabajar en forma local es la siguiente:

127.0.0.1
6364
6363

Con todo instalado y configurado, es momento de probar el equipo de ejemplo. Para esto, abra Squeak y saque un "Robot Soccer Viewer" del catálogo de objetos en la categoría "Robot Soccer".



Este objeto es un panel de control gráfico que sirve para conectar y desconectar equipos y para ver el partido en tiempo real. Nótese que los partidos podrían estar corriéndose en distintas máquinas, con una máquina para cada equipo y otra para el simulador. Esto hace que sea muy útil poder ver el partido desde Squeak en tiempo real.
Una vez creado el "Robot Soccer Viewer" debe seleccionar el equipo que jugará el partido (en este caso el equipo de ejemplo) y la forma de comunicación con el servidor de video y los robots.
El servidor de video (VideoServer) para comunicarse con el simulador debe ser "RSSimulatedVideoServer", y el puerto debe ser el mismo que el configurado en el archivo "proxy.config". El servidor de robots (RobotServer) para el simulador debe ser "RSSimulatedRobotServer", y el ip y puerto deben ser los mismos que los configurados en el archivo "proxy.config". El equipo de ejemplo que debemos conectar es "RSTeamExample".
Una vez que todo esté configurado correctamente debe hacer clic en "Start", el "RobotSoccerViewer" quedará a la espera de la conexión con el simulador.

Ahora es tiempo de abrir el simulador RobotSoccer v1.5a y cargar aquí el equipo. Para eso se debe hacer clic en "Strategies", escribir el nombre de la estrategia (en nuestro caso debemos escribir "proxy" denotando el archivo "proxy.dll" que se encarga de la comunicación con Squeak) en uno de los cuadros de texto ("Blue" o "Yellow" dependiendo del equipo que queramos cargar). En el otro cuadro de texto no es necesario escribir nada. Es muy importante seleccionar el lenguaje C++ en vez de Lingo, de otra forma el equipo no funcionará. El simulador debería quedar más o menos así:

Una vez hecho esto, hacemos clic en “Send” y luego en “Start”. Si todo funcionó bien, los robots deberían empezar a moverse y desde Squeak deberíamos ser capaces de ver el partido así como hacer modificaciones.

3. Descripción general de la arquitectura

El software para fútbol de robots en Squeak se desarrolló considerando como requisito fundamental la posibilidad de utilizarlo para desarrollar variedades de equipos tanto para fútbol simulado como fútbol físico, por lo tanto, resulta imprescindible que la implementación refleje un buen diseño que permita independizar el desarrollo de equipos de la tecnología subyacente.

El modelo se basó en 4 conceptos esenciales: un environment, un equipo, un servidor de video y un servidor de robots. El environment simplemente representa el ambiente de juego y por lo tanto tiene todas las variables pertinentes. El equipo es el encargado de decidir las acciones (velocidad de las ruedas de cada robot) para un determinado environment. El servidor de video y el servidor de robots representan la capa de comunicación: son los encargados de comunicarse con el "mundo exterior" y actualizar las variables del environment y mandar la nueva velocidad a los robots. Estos 4 conceptos se implementaron en las siguientes clases: RSEnvironment, RSTeam, RSVideoServer y RSRobotServer. El equipo, el servidor de video y el servidor de robots son independientes uno de otro y permiten la utilización del mismo framework para el desarrollo de equipos tanto de fútbol físico como de simulado.

Por otro lado, la clase principal del modelo que vincula todos estos conceptos se denominó RSGame y representa un partido de fútbol de robots. Permite controlar el

estado del partido así como conectar y desconectar equipos, servidores de video y servidores de robots.
Finalmente, para representar las entidades inherentes al fútbol de robots (los robots, la pelota, etc.) se implementaron las clases adecuadas: RSRobot, RSOpponentRobot y RSBall.

A continuación se describirá más detalladamente cada clase del modelo:

RSGame

Representa un partido de fútbol de robots.

Variables de instancia:
- numberOfRobots – la cantidad de robots de cada equipo (se asume igual para ambos).
- videoServer – el servidor de video elegido y conectado por el usuario.
- robotServer – el servidor de robots elegido y conectado por el usuario.
- rsEnvironment – el environment actual.
- team: el equipo elegido y desarrollado por el usuario.
- process – el proceso encargado de recibir los datos, procesarlos y enviar una respuesta.

Métodos de instancia:
- #videoServer: – carga el valor de la variable videoServer.
- #robotServer: – carga el valor de la variable robotServer.
- #initializeEnvironment – carga el valor de la variable rsEnvironment dependiendo del team que se haya seleccionado.
- #team: – carga el valor de la variable team y luego se manda el mensaje #initializeEnvironment.
- #disconnectVideoServer – desconecta el servidor de video.
- #disconnectRobotServer – desconecta el servidor de robots.
- #disconnectTeam – desconecta el equipo.
- #disconnectAll – desconecta todo.
- #start – inicia el partido o lo continúa si estaba pausado.
- #pause – detiene el partido en el punto de ejecución actual permitiendo continuarlo más tarde.
- #stop – finaliza el partido.

RSVideoServer

Tiene la responsabilidad de conseguir la información del ambiente. Se decidió definirla como clase abstracta para que sus subclases implementen la forma de comunicación con distintos tipos de servidores de video (por ej. el proxy en caso de fútbol simulado, el software “Doraemon” en caso de fútbol físico, etc.).
La creación de una subclase es muy simple: sólo requiere que se sobrescriban dos métodos: #storeDataOn: que debe guardar los datos en un environment que se pasa como parámetro; y el método de clase #fromUser que debe responder una nueva

instancia basándose en el ingreso de datos del usuario (por ejemplo el puerto del que se espera recibir los datos del proxy en caso de fútbol simulado).

RSRobotServer

Es responsable de enviar la nueva velocidad de las ruedas a un conjunto de robots. Se definió como clase abstracta para que sus subclases implementen la forma de comunicación con distintos tipos de robots (por ej. robots que se comunican por radio para el caso de fútbol físico o simplemente el proxy para el caso del fútbol simulado).
La implementación de una subclase es muy simple: sólo requiere que se sobrescriban dos métodos: #sendMovesOf: que debe enviar la velocidad de las ruedas de una colección de robots que se pasa como parámetro; y el método de clase #fromUser que debe responder una nueva instancia basándose en el ingreso de datos del usuario (por ejemplo la ip y el puerto por el que se enviarán los datos al proxy en caso de fútbol simulado).

RSTeam

Representa un equipo y es el responsable de definir las acciones (velocidad de cada robot) para un environment determinado. Se decidió implementarla como clase abstracta para permitir que distintos equipos convivan en la misma imagen. La implementación de una subclase requiere sólo la sobreescritura del método #strategyFor: que recibe como parámetro un environment.

En la mayoría de los casos, las subclases necesitarán especializar el comportamiento de los robots o del environment (por ejemplo para añadir primitivas de navegación). Para solucionar esto se añadieron los métodos de clase #myBall, #myEnvironment, #myOpponent y #myRobot que deben devolver la clase de la que se deben crear las instancias de cada objeto.

RSEnvironment

Representa el ambiente de juego y es responsable de tener toda la información del juego disponible para que el equipo decida la estrategia.
La información necesaria incluye las siguientes variables:

- home – una colección de robots de nuestro equipo.
- opponent – una colección de robots del equipo contrario.
- currentBall – la pelota actual.
- fieldBounds – un rectángulo que representa los bordes de la cancha.
- goalBounds – un rectángulo que representa los bordes de los arcos.

Para facilitar la programación de equipos, el environment también se encarga de invertir la cancha si el equipo propio está del lado izquierdo. De esta forma sólo hay que programar equipos para el lado derecho (en caso del fútbol simulado para el equipo azul) y funcionan igual para ambos lados.

RSRobot

Representa un robot de nuestro equipo y tiene las siguientes variables de instancia:

- rsEnvironment – el environment al cual pertenece.
- rotation – el ángulo de rotación.
- position – un punto que indica su posición en la cancha.
- velocityLeft – la velocidad de la rueda izquierda.
- velocityRight – la velocidad de la rueda derecha.

RSOpponentRobot

Representa un robot del equipo contrario y tiene las siguientes variables de instancia:

- rotation – el ángulo de rotación.
- position – un punto que indica su posición en la cancha.

RSBall

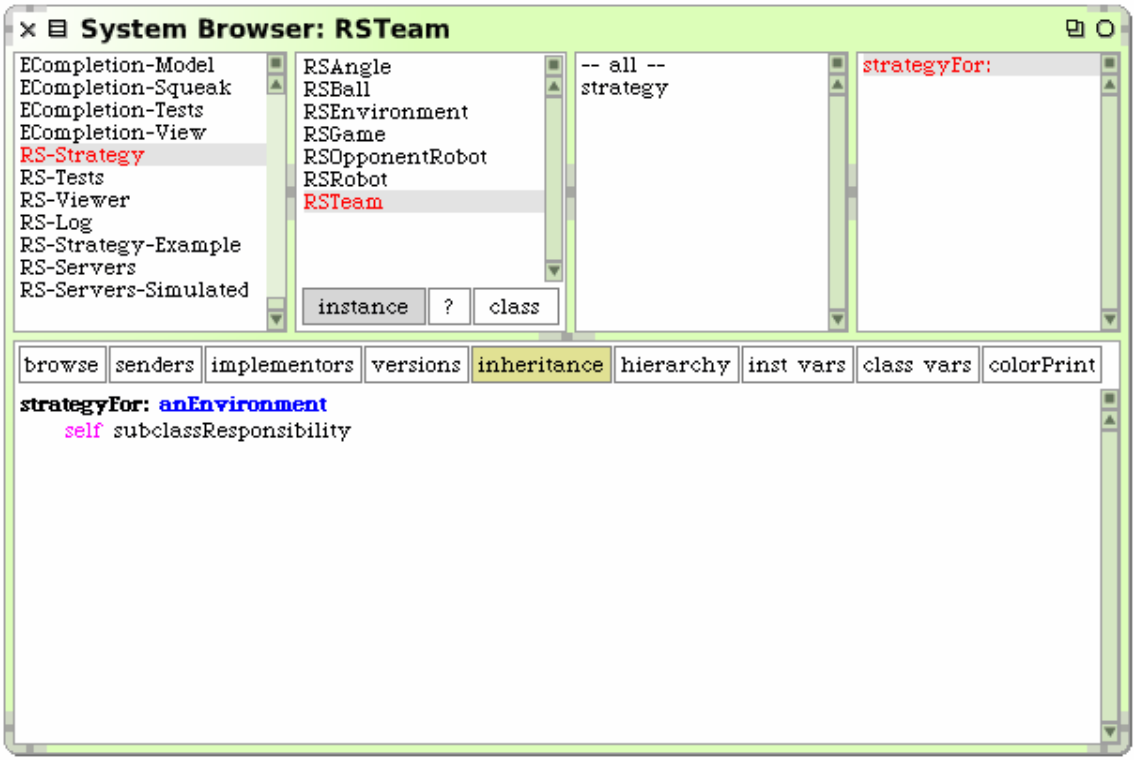
Representa una pelota y tiene las siguientes variables de instancia:

- – un punto que indica su posición en la cancha.

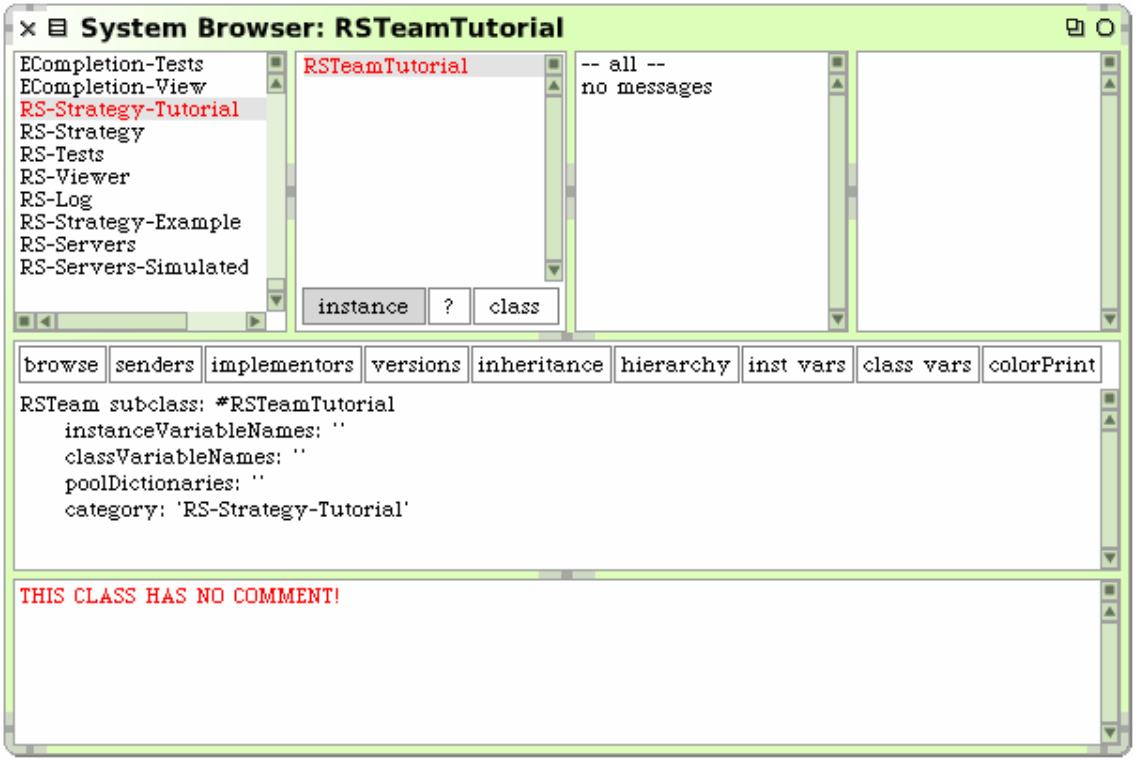
Para las competencias de fútbol de robots simulado se implementó una subclase de RSVideoServer que escucha la información enviada por el proxy. Esta clase se denominó RSSimulatedVideoServer.
De la misma forma, se desarrolló una subclase de RSRobotServer para las competencias de fútbol simulado que envía las velocidades de los robots al proxy. Esta clase se denominó RSSimulatedRobotServer.

4. Creación de un nuevo equipo

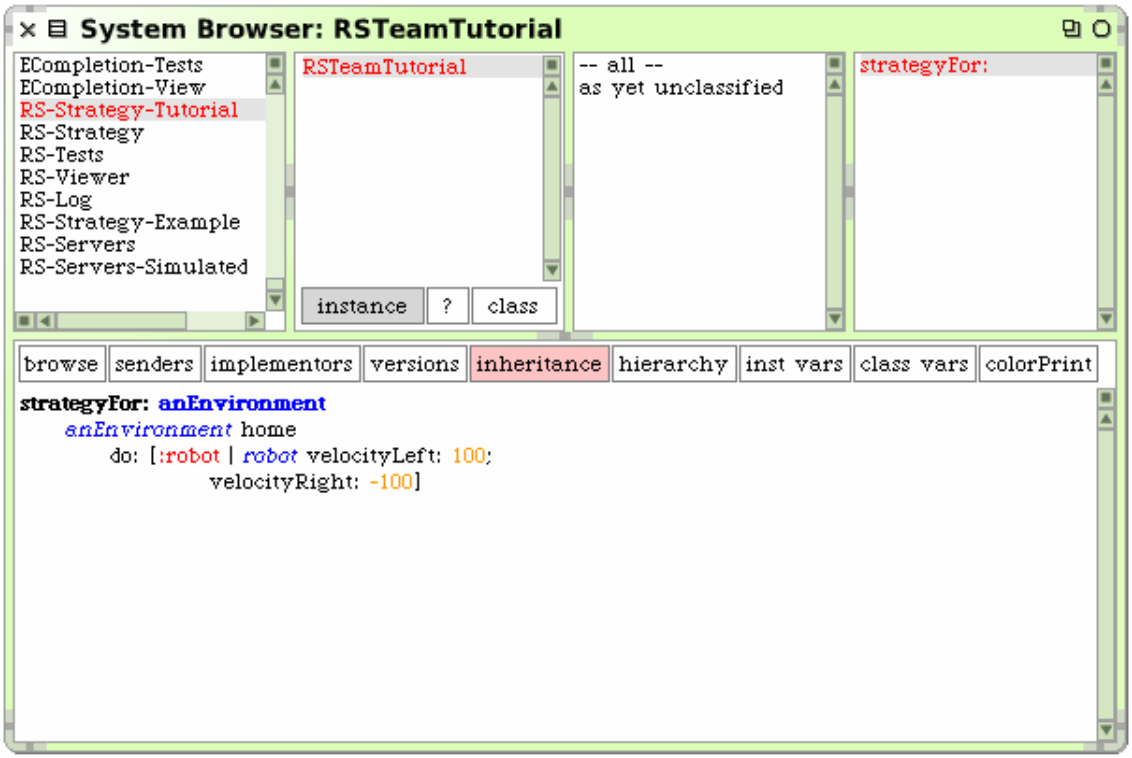
Para crear un equipo propio tenemos varias formas: una es modificar las clases que vienen por defecto (RSTeam, RSEnvironment, RSRobot, RSOpponentRobot, RSBall) para agregar el comportamiento deseado. El problema con esta solución es que nos limita a tener sólo una estrategia en la imagen.
La solución recomendada es subclasificar las clases genéricas, y modificar el comportamiento de las subclases. De esta forma, podemos tener tantas estrategias como queramos sin tener que cambiar de imagen.
El primer paso es crear una subclase de RSTeam. Veremos que RSTeam es una clase extremadamente simple, sólo contiene 1 método de instancia y 4 métodos de clase.
Lo que nos interesa por ahora es el método de instancia #strategyFor:. Éste método se ejecuta en un loop durante todo el partido y se espera que defina la velocidad de las ruedas de cada robot para el RSEnvironment (el ambiente de juego actual) que recibe como parámetro. Como notarán, RSTeam no define ninguna implementación concreta porque se espera que las subclases lo hagan.



Lo que haremos entonces será crear una subclase de RSTeam llamada RSTeamTutorial.



Teniendo una subclase de RSTeam lo que debemos hacer ahora es sobrescribir el método #strategyFor: para nuestro nuevo equipo. Para este ejemplo sencillo vamos a crear un equipo en el cual todos los robots giren sobre sí mismos (no tiene mucho sentido si queremos competir pero va a servir para explicar el mecanismo de creación de una nueva estrategia). El código que debemos escribir es el siguiente:



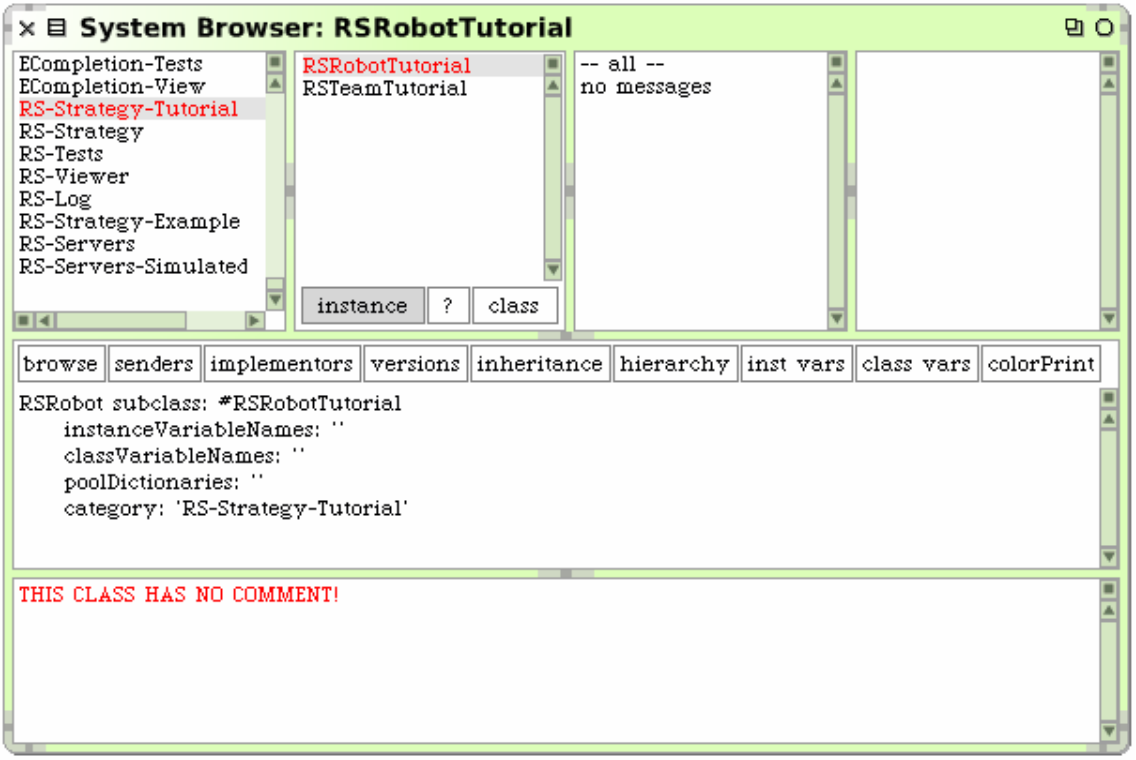

```
strategyFor: anEnvironment
        anEnvironment home
                do: [:robot | robot velocityLeft: 100;
                                velocityRight: -100]
```

Este código simplemente recorre la colección de robots de nuestro equipo poniendo valores contrarios en las velocidades izquierda y derecha de cada robot, haciendo así que giren sobre sí mismos.
Este cambio es suficiente como para tener un equipo andando.

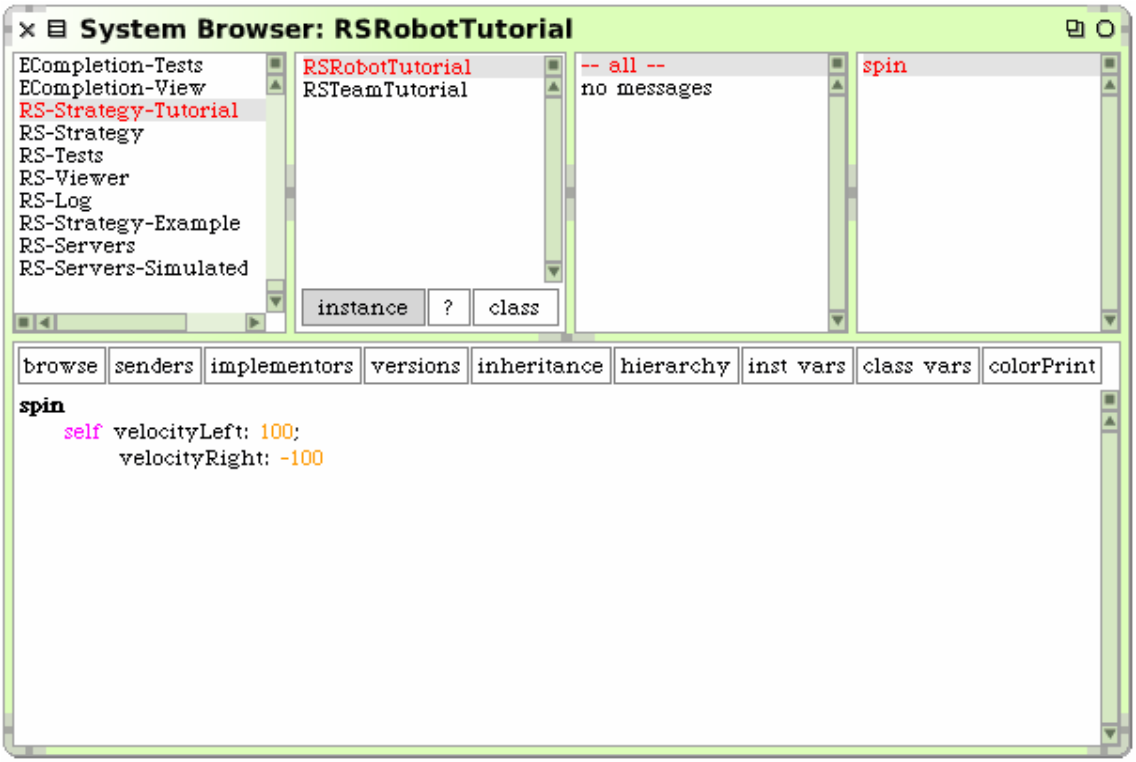
Ahora bien, cuando tengamos comportamientos más complejos es probable que queramos encapsular el comportamiento de los robots en métodos. Por ejemplo hacer un método #spin para la acción que acabamos de programar.

Para esto es necesario subclasificar la clase RSRobot con el método deseado y luego modificar en RSTeam el método de clase #myRobot para que devuelva la clase nueva. A continuación lo haremos paso a paso.

Primero creamos una subclase de RSRobot llamada RSRobotTutorial.



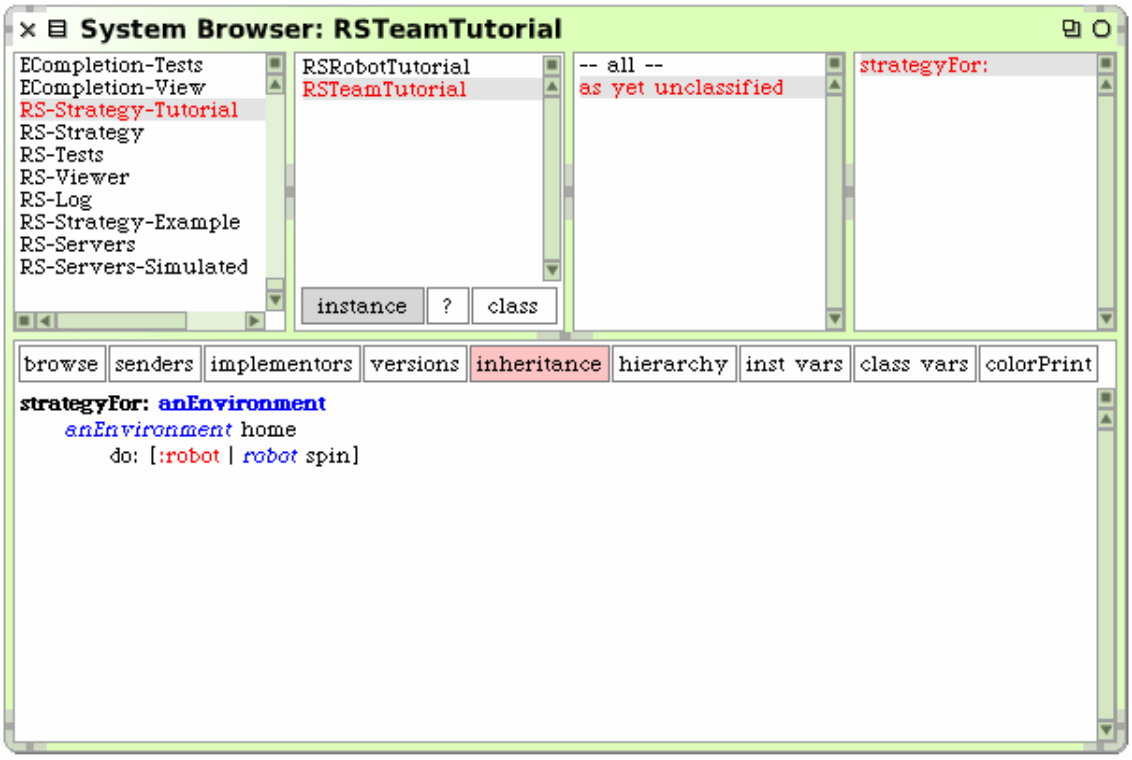
Luego creamos el método #spin.



spin

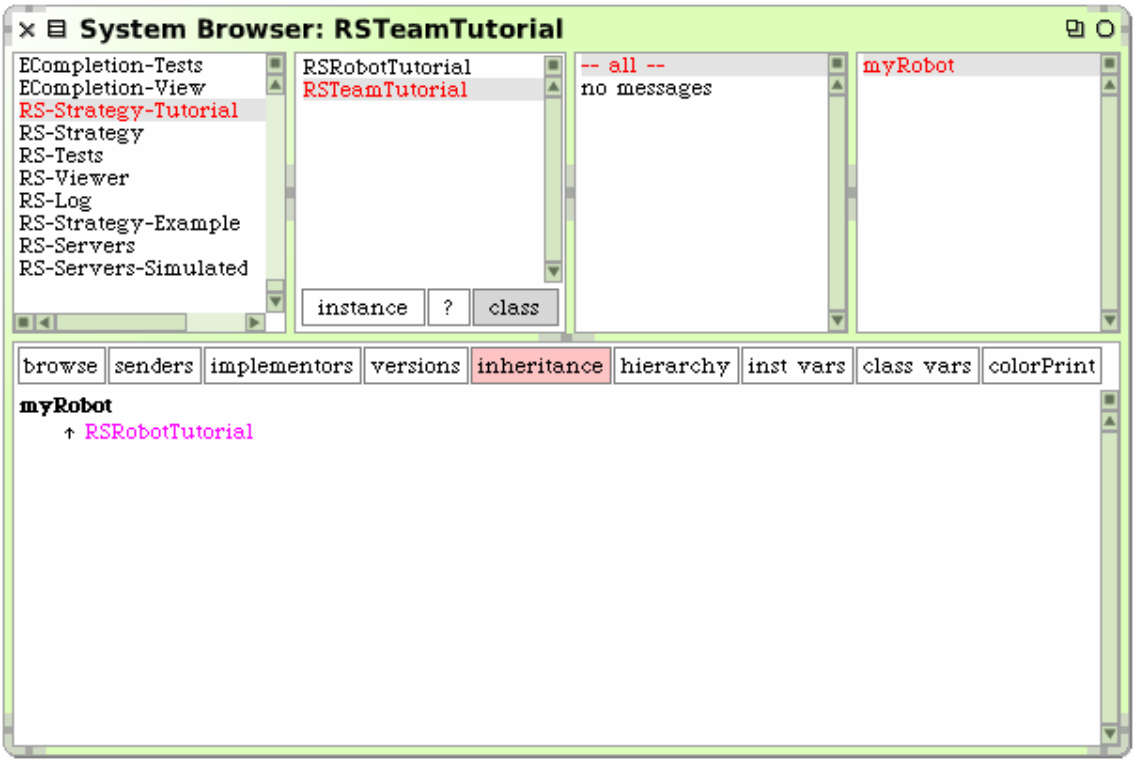
```
self velocityLeft: 100;
        velocityRight: -100
```

Modificamos luego el método #strategyFor: para mandar el mensaje #spin.



strategyFor: anEnvironment
anEnvironment home
do: [:robot | robot spin]

Y, por último, sobrescribimos el método de clase #myRobot para que devuelva la clase RSRobotTutorial.



Ahora podemos probar nuestro equipo con el “Robot Soccer Viewer” y veremos cómo se comporta nuestro nuevo equipo.