Introducción a SoccerBots

Departamento de Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid

- El uso de entornos de simulación de fútbol ha recibido una gran importancia como aplicación en la que combinar distintas técnicas de IA.
  - RoboCup
    - 🕮 Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I. Osawa, E., Matsubara, H., *RoboCup: A Challenge Problem for AI and Robtics,* in "RoboCup-92: Robot Soccer World Cup I, Springer Verlag, 1998.
- Técnicas de IA que podéis usar
  - **Máquinas de estados para control de comportamientos**
  - **Sistemas expertos basados en reglas**
  - Redes neuronales: **aprendizaje** off-line / on-line
  - Algoritmos genéticos: **aprendizaje** off-line/ on-line
  - **Razonamiento basado en casos/ Planificación basada en casos**
  - **Agentes que se comunican/ con consenso .....**

RoboCup

- *Robot World Cup Initiative* (RoboCup)
  - RoboCup.org: http://www.robocup.org/02.html
  - Proyecto internacional para promover la IA, la robótica y los campos relacionados.
  - Proporcionar un problema estándar donde varias tecnologías puedan ser integradas y examinadas
- En 1997 se celebró el primer **Robot World Cup Soccer Games and Conferences** (en IJCAI-97) en Nagoya (Japón)

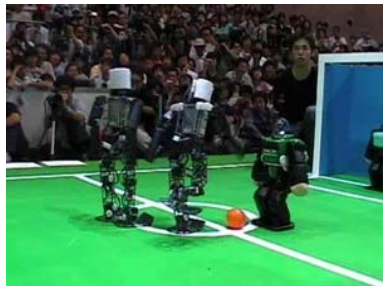
RoboCup

- Los robots compiten en distintas categorías
  - **Liga de simulación** software
  - Liga de robots reales
    - Pequeños, Medianos y ~~AIBOS~~
    - Humanoides que puedan competir contra un equipo humano
- Retos:
  - Trabajo en equipo entre los agentes
  - Modelado de cada agente
  - Aprendizaje personal y de equipo

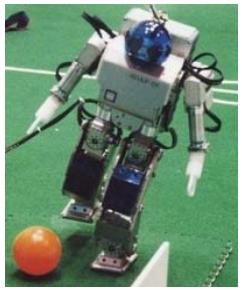
RoboCup 2006 Humanoid League website

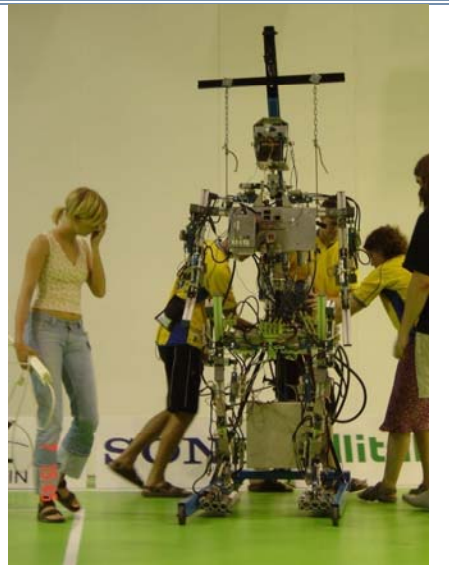
- http://www.humanoidsoccer.org/
- RoboCup is an international joint project to foster AI and intelligent robotics research by providing a standard problem. The ultimate goal of RoboCup is:

By 2050, develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer.



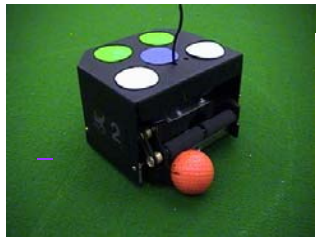
RoboCup: Humanoides





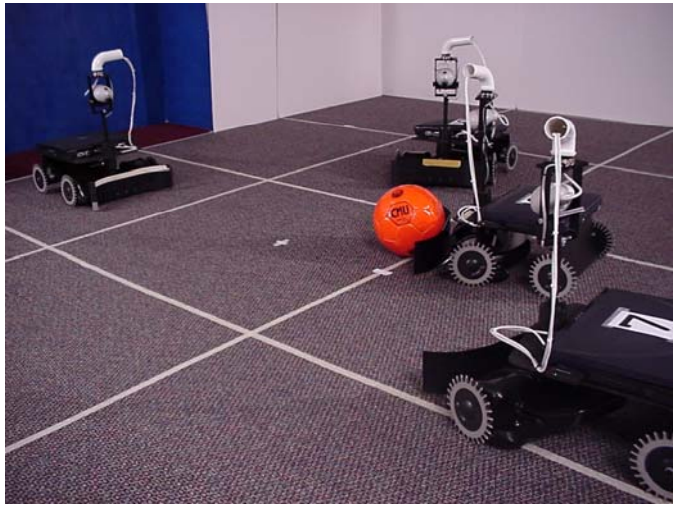
RoboCup: Sony Four-legged Robot League

RoboCup: Small Size

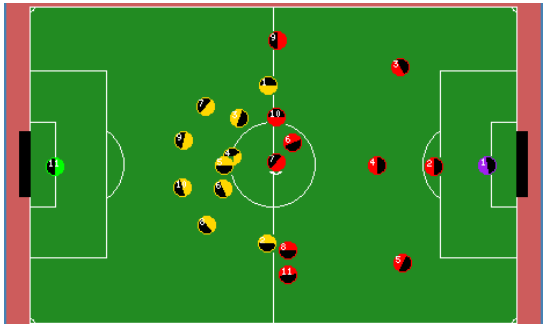


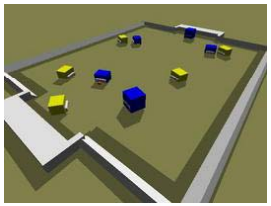
RoboCup: Medium Size

mid-size robots HammerHeads



RoboCup: Simulator





SoccerBots: Introducción y motivación

- SoccerServer (Itsuki Noda) es el simulador oficial de RoboCup: http://sserver.sourceforge.net/
- SoccerBots es una versión reducida implementada en Java del simulador oficial *SoccerServer*
  - **JavaSoccer,** Balch, T., *RoboCup-97: Robot Soccer World Cup I*, Springer-Verlag, 1998

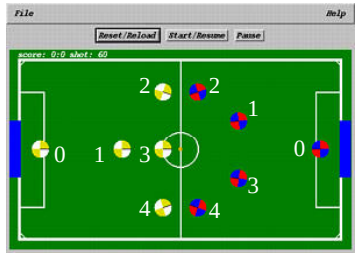
  Desarrollado en el Georgia Tech Intelligent Systems and Robotics Group por Tucker Balch.
  - Permite implementar en Java *comportamientos* básicos de robots.
- Entorno de simulación TeamBots: no es específico de soccer
  - C:\TeamBots\Domains
  - C:\TeamBots\Domains\SoccerBots → nos centraremos en la programación de *comportamientos* para equipos de soccer

SoccerBots: Introducción y motivación

- Limitaciones:
  - Los robots no puedan controlar la dirección del disparo. Lo único que pueden hacer es disparar en la misma dirección hacia la que esté orientado el cuerpo del robot (otra dirección → girar + disparar).
  - Los jugadores no saben levantar la pelota por encima del portero.
  - Los porteros no pueden agarrar la pelota ...

Entorno

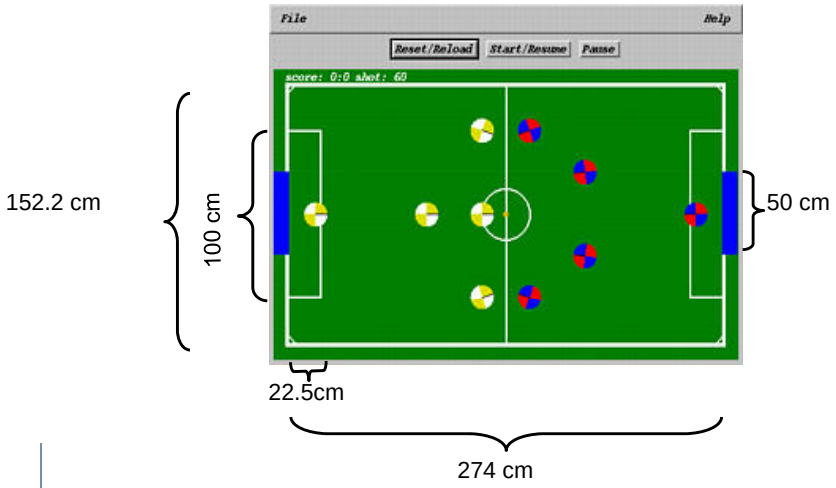
- 2 equipos
  - Este – derecha
  - Oeste – izquierda
  - El equipo oeste hace el saque inicial. El resto los hace el equipo que recibe un gol.
- Máximo 5 robots por equipo
- Configuración de salida de los robots en el campo (RoboCup)
- Se pueden manipular las posiciones iniciales de los jugadores, los colores de los equipos, ...
- También se permite (menú view) visualizar el número del jugador, la traza de movimientos y la acción que está realizando.



Datos técnicos

- **Dimensiones del campo de fútbol** 152.2 cm x 274 cm
- **Medida de cada portería** (anchura) 50 cm
- **Zona de defensa** (centrada en la portería) 100 cm x 22.5cm
- **Diámetro de la pelota** 40 mm
- **Diámetro de Robots** 12 cm
- **Sistema de coordenadas**
  - El centro del campo es el (0,0)
  - +x hacia la derecha (este)
  - +y hacia el norte

Datos técnicos



Datos técnicos

- Velocidad
  - Robots que se pueden mover a una velocidad lineal de 0.3 m/s y girar 360 grados por segundo.
  - Pelota
    - Velocidad máxima 0.5 m/s (cuando es chutada)
    - Desacelera a 0.1 m/s2 .
- Existe una pared alrededor de todo el campo salvo en las porterías
- Las colisiones de la pelota con los robots y con las paredes son perfectamente elásticas
- Duración del partido no limitada → ajustar a los 10 minutos de RoboCup
- Los robots puede caminar pegados a la pared siempre que la dirección de avance no "choque" con la pared
- **No hay faltas**
- Reloj de disparo: 60 segundos → si no se marca un gol en este tiempo el balón vuelve automáticamente al centro del campo

Funcionamiento de SoccerBots

- **Cada objeto de la simulación incluye dos componentes**
  1) Método de dibujar
  2) Método de simulación de la dinámica
- **El núcleo de la simulación ejecuta un bucle que en cada paso llama a los dos métodos de cada objeto de la simulación (los jugadores)**
- **Sistemas de control de los robots**
  - Los robots tienen un sistema de control (su "cerebro") que considera las entradas (*sensores*) y actúa en consecuencia (*actuadores*)
  - Este sistema de control es lo que hay que modificar para experimentar con distintas "estrategias futbolísticas"
  - El sistema de control no sabe si se está ejecutando en una simulación o con robots HW reales

La interfaz

- **Ejemplo de sensores** (para cada robot)
  - Detectar si el robot está en situación de disparar (*suficientemente* cerca de la pelota)
  - Obtener vectores que apuntan a:
    - la pelota
    - La portería de tu equipo y del contrario
  - Obtener un array de vectores que apunten a todos los otros jugadores
  - Obtener un array de vectores que apunten a los jugadores de nuestro equipo
  - Obtener un array de vectores que apunten a los jugadores del equipo oponente
  - Obtener el número del jugador en el equipo (el 0 es el portero)
  - Obtener la posición del jugador en el campo
  - Obtener la orientación (dirección de avance) del jugador.
  - Obtener el tiempo en milisegundos desde que el partido comenzó

La interfaz

- **Ejemplo de actuadores**
  - Chutar la pelota a una velocidad de 0.5 metros/sec.
  - Empujar el balón avanzando sobre él.
  - Cambiar la orientación. El robot puede girar a 360 grados/sec.
  - Cambiar la velocidad (máximo 0.3 metros/sec)

3 novedades respecto a la versión original ...

- Software para los torneos (SBTournament)
- Sobre los sensores/actuadores originales hemos añadido una capa (API) que tendréis que utilizar.
  - Básicamente son llamadas a los mismos métodos + funcionalidad adicional
  - Evita que podamos manejar distintas versiones del simulador
- Aplicación (ECO) para diseñar los comportamientos de forma gráfica dibujando máquinas de estados → genera código

Para la práctica...

- **Objetivo:** crear varios equipos "inteligentes"
  - Diseño de comportamientos de jugadores
  - Diseño de estrategias de equipo
- **Presentación de prototipos y competición entre los equipos**

- **Diseño de comportamientos para los agentes.**
  - **Toma de decisiones**
    - ¿qué hago ahora? – depende de la situación a mi alrededor.
    - Técnicas de IA
      - Reglas → Máquinas de estados
      - CBR. Casos
      - Aprendizaje
        - Agentes que no aprenden → inteligencia cableada por medio de heurísticas
        - Agentes que aprenden de su experiencia
  - **Estrategias de equipo**
    - Cambio de comportamiento de los jugadores según la situación del partido
    - Comunicación entre agentes

http://gaia.fdi.ucm.es/research/sbtournament

Reglas del juego

- La duración de los partidos será aproximadamente 1 o 2 minutos
- No se puede:
  - Poner más de dos jugadores de portero en ningún momento del juego. Sí se pueden poner dos.
  - No se puede tocar ninguna de las variables de configuración del simulador.
  - Los bloqueos de otros jugadores no están prohibidos (por ejemplo, el DTeam bloquea el portero). Los equipos pueden reaccionar a los bloqueos.
  - Después de cada gol la pelota saldrá de nuevo al campo en una zona cercana al centro pero no tiene que ser exactamente el punto central.

- SBTournament es un entorno que permite la simulación de partidos SoccerBots, la realización de torneos así como el entrenamiento automático de equipos y generación de trazas junto con su visualización.
- En el laboratorio se os proporcionará un archivo zip con un proyecto Eclipse configurado y listo para utilizar.
  - Los equipos deben estar en el directorio *robots*
- Cada grupo deberá generar las clases dentro de un paquete *grupoXX, donde XX es el número de* grupo. Cualquier librería a utilizar deberá copiarse al directorio *lib y entregarla junto al equipo. A* su vez, los ficheros temporales o cualquier recurso creado y/o reutilizado entre partidos deberá guardarse en el directorio *robots|grupoXX*

Planificación

- **Primera sesión de laboratorio (viernes 20 de abril):**
  - Tutorial guiado de la API para desarrollar un equipo sencillo.
- **Segunda sesión de laboratorio (viernes 27 de abril):**
  - Tutorial de la herramienta eCo para diseñar comportamientos.
- Durante esta semana os llevareis como “deberes” pensar y diseñar sobre el papel un comportamiento más complejo. Comportamientos para los jugadores del equipo. Entregar versión en papel.
- **Tercera sesión de laboratorio (viernes 4 de mayo):**
  - Experimento de uso de la herramienta eCo + Encuesta
- **Sesiones de laboratorio para trabajo en los equipos**
  - **Viernes 11 y 18 de mayo**
  - Todos los Jueves a partir de ... que acabemos la teoría
- **Campeonato: 25 de mayo y 1 de junio.**

Desarrollo de equipos

- 2 opciones:
  - Equipo **homogéneo** (ejemplo SchemaDemo team),
    - Cada miembro del equipo intenta hacer lo mismo pero desde una perspectiva distinta (ya que tiene distintas lecturas de los sensores)
    - Todos los jugadores tendrán asignado el mismo comportamiento
  - Equipo **heterogéneo** (ejemplo BasicTeam)
    - Cada jugador realiza una tarea específica
    - Cada jugador tiene asignado un comportamiento distinto
- ¿El comportamiento de un jugador puede cambiar durante el partido?
  - Existe la figura del **entrenador** (TeamManager)

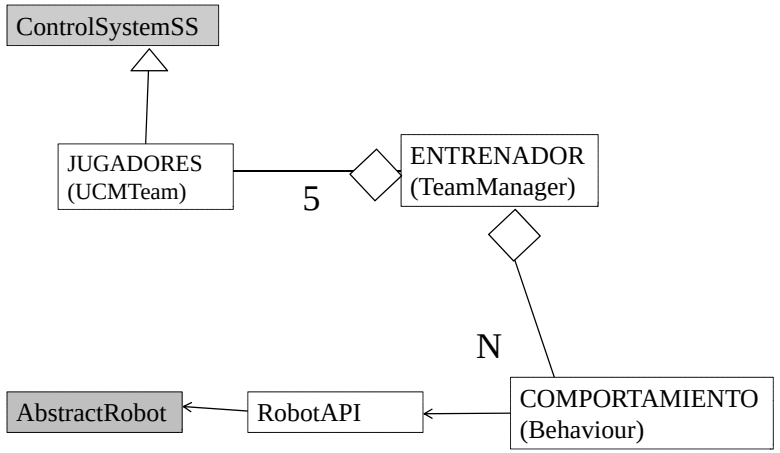
UCMTeam

- UCMTeam es un paquete de clases que permite construir robots que pueden ser utilizados en Soccerbots
- Los robots de UCMTeam se basan en comportamientos intercambiables durante la ejecución del partido.
- Un equipo UCMTeam tiene un TeamManager, responsable de crear todos los comportamientos disponibles durante la ejecución de un partido, así como de asignar dichos comportamientos a los distintos robots del equipo durante el partido.

Package teams.ucmTeam

Class Summary	
Behaviour	The interface that represents a robot behaviour.
Message	A generic message that can be sent by the UCMTeam robots.
RobotAPI	This is the robot's interface with the world.
TeamManager	
UCMTeam	UCMTeam is a Control System that represents a team whose robots are controlled based on predefined behaviours.

Arquitectura UCMTeam



RobotAPI: Sensores

- Obtener información sobre el robot
  - ID: Único para la simulación (no hay dos robots en ninguno de los dos equipos con el mismo número)
  - playerNumber: único para el equipo (no hay dos robots en el equipo con el mismo número pero este número se puede repetir entre equipos)
  - Position, Heading, canKick
  - fieldSide: En qué lado del campo juega el robot

Sensores

- Obtener información sobre los elementos que hay en el campo
  - Opponents, teammates, ourGoal, opponentGoal, ball,
- Sobre el partido
  - Total time, Remaining time, timestamp (current time), myScore, opponentScore,
    justScored: Parecido a un evento, devuelve un valor si se acaba de marcar un gol Actuadores (de bajo nivel)
  - speed (0, 1)
  - heading (en radianes)
  - Cankick
  - displayString: lo que presenta el robot por pantalla Sensores alto nivel
-

Sensores

- **Obtener información sobre bloqueos**
  - blocked: saber si estoy bloqueado (sin necesidad de saber quién es)
  - opponentBlocking, teamMateBlocking, goalKeeperBlocked, isBlocking: hay dos, una para saber si un jugador concreto bloquea a otro; y otra para saber si un jugador concreto nos bloquea a nosotros
- **Posición propia y de elementos del campo**
  - goalkeeper, closestTeammate, sortedteammates, closestOpponent, sortedopponent, closestToBall, closestTo, behindEverybody, aligned to ball and goal, Opponents have goalkeeper
-

RobotAPI:actuadores

- Actuadores
  - Avanzar, girar, disparar..
- Actuadores (de alto nivel)
  - Avoid collision
  - Set behind ball
  - Block goalkeeper
  - block forward
  - block closest
  - surround point
  - passball (experimental)

Métodos adicionales

- Métodos para obtener las dimensiones del campo y del robot
- Método para crear/cargar ficheros: createFile
- Método para acceder al TeamManager del equipo al que pertenece el robot
- Métodos para cambiar de coordenadas egocéntricas a no egocéntricas y viceversa:
  - toFieldCoordinates y toEgocentricalCoordinates

Documentación de RobotAPI

Method Summary

boolean	`alignedToBallandGoal()` Checks if the robot is behind the ball and aligned with the opponent's goal
void	`avoidCollisions()` Tries to avoid a collision with the closest player
boolean	`behindEverybody()` Checks if the robot is the closest to our goal
void	`blockClosest()` Tries to block the closest opponent
boolean	`blocked()` Returns if the robot is blocked
void	`blockForward()` Tries to block opponent's forward (opponent closest to our goal)
void	`blockGoalKeeper()` Tries to block opponent's goalkeeper (opponent closest to their goal)
boolean	`canKick()` Reveals whether or not the ball can be kicked.
EDU.gatech.cc.is.util.Vec2	`closestTo(EDU.gatech.cc.is.util.Vec2[] objects, EDU.gatech.cc.is.util.Vec2 point)` Returns the position of the closest object to a given point
boolean	`closestToBall()` Checks if the robot is the closest one to the ball
static java.io.File	`createFile(java.lang.Class<?> classRequester, java.lang.String filena` Gets a file in a unique folder according to the class that requests the file.
EDU.gatech.cc.is.util.Vec2	`getBall()` Get a Vec2 that points egocentrically from the center of the robot to the ball.
EDU.gatech.cc.is.util.Vec2	`getClosestMate()` Gets a Vec2 that points egocentrically from the center of the robot to the closest teammate

Ejemplos y el uso de vectores

- Vec2 **getClosestOpponent**()
  - Gets a Vec2 that points egocentrically from the center of the robot to the closest opponent
- public boolean **closestToBall**()
  - Checks if the robot is the closest one to the ball
- Vec2 **closestTo**(Vec2[] objects, Vec2 point)
  - Returns the position of the closest object to a given point
  - **Parameters:**
    - objects - An array with the objects to compare
    - point - the reference point
  - **Returns:**The position of the closest object to the given point

La clase Vect2

- La mayoría de los parámetros que se utilizan en los métodos sensores y actuadores (de la interfaz SocSmall) pertenecen a la clase Vect2
- La classe Vec2 se utiliza para manipular vectores de dos dimensiones
  - Coordenadas cartesianas **(x , y),** en metros
  - Coordenadas polares **(t, r)** , t es el ángulo en radianes y r es el radio en metros
- Coordenadas +x (positivas) a la derecha del centro del campo y +y hacia arriba.
- t es 0 en la dirección de +x y PI en la de –x y avanza en sentido antihorario (tanto para los jugadores del este como para los del oeste)
- Podemos cambiar los valores usando lós métodos setx, sety, sett y setr

La clase Vect2

- **Constructores**
  - **Vec2 ():**

    x = 1; y = 0;

    t = 0; r =1;
  - **Vec2 (double x0, double y0)**
    - x = x0; y = y0;
    - t = Math.atan2(y,x); r = Math.sqrt(x*x + y*y);
  - **Vec2 (Vec2 v)**
    - x = v.x; y = v.y;
    - t = v.t; r = v.r;

La clase Vect2

- **Métodos**
  - **setx (double newx):** cambia el valor de la x y actualiza el radio y el ángulo en consecuencia
  - **sety (double newy):** ídem para y.
  - **sett (double newt):** cambia el ángulo y actualiza x e y adecuadamente.
  - **setr (double newr):** ídem para el radio r.
  - **rotate (double t1):** suma al valor del ángulo el ángulo de rotación dado (t1) y recalcula x e y. Hace lo mismo que sett (t + t1).
  - **sub (Vec2 v2):** resta el vector v2 a él mismo (this = this – v2).
  - **normalize (double r1):** hace lo mismo que setr (r1)
  - **add(Vec2 v2):** suma el vector v2 a él mismo (this = this + v2).
  - **octant ():** devuelve un valor entre 0 y 7 y sirve para saber en qué octante está apuntando el vector. El valor 0 indica 0 radianes y los siguientes (± PI/8) en sentido anti-horario.
  - **quadrant ():** ídem para cuadrantes
  - **toString():** genera una cadena que contiene el valor del vector. "(x, y) (r, t)"