Diseño de equipos para SoccerBots

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

Estrategias de equipos

- Estudio de estrategias de algunos equipos de la distribución
- Diseño de equipos con máquinas de estados
- CBR y Soccer

Ejemplo: Dteam (David H. Johnson)

- Aproximación **heterogénea** basada en Basic Team
- Miembros del equipo ➔ agentes **reflexivos**
- Agentes que **no aprenden**. Se basan en heurísticas predefinidas.
- Identifica roles de juego y cada miembro del equipo juega un rol único (fijo según su ID)
  - Goalie
  - Offside
  - Designated Driver
  - Backup
  - Center

Dteam: tipos de jugadores

```
/*--- Goalie ---*/
if( id== 0)  {        play_goalie();        }
/*--- Backup ---*/
else if( id == 1)  {play_backup();        }
/*--- Offside ---*/
else if( id == 2)     {           play_offside();       }
/*--- Designated Driver ---*/
else if( id == 3)    { drive_ball();          }
/*--- Center ---*/
else        play_center();
```

- **Goalie**
  - Mantenerse entre el balón y la portería en cualquier momento.
  - Desviar el balón si está muy cerca de la portería.
  - Si se encuentra muy lejos de la portería o fuera de sus límites vuelve al centro lo antes posible.
  - Si el balón está detrás del portero, corre hacia la pelota e intenta enviarla fuera
  - Todo el código de control está en la función *play_goalie()*

- **Offside**
  - Este jugador tiene como objetivo el oponente que esté más cerca de la portería contraria (es el que identifica como goalie)
  - El primer paso es identificar su objetivo (varía con las posiciones de jugadores)
  - Dirige su movimiento hacia el punto que está detrás del oponente
  - En el camino evita cualquier colisión con otros jugadores
  - Este jugador sirve para bloquear al jugador objetivo

- **Designated Driver**
  - Intenta continuamente llevar la pelota. Para ello utiliza la función drive_ball().
  - void drive_ball()
    - **Si** el robot está detrás de la pelota con respecto a la portería (es decir, se puede dibujar una línea desde el robot a la portería pasando "**por**" la pelota) y la pelota está **cerca entonces** se mueve hacia la portería.
    - **Si** el robot se puede mover hacia la portería y está cerca de la misma **entonces** intenta disparar el balón.
    - **Si** no estás detrás del balón **entonces** dirígete hacia la posición detrás del balón evitando colisiones en el camino.

- Backup
  - Si "Backup" es el robot más cercano a la pelota se encarga de dirigirla.
  - Este robot sirve de reserva (backup) para el jugador "designated driver"
  - Este robot calcula un círculo concéntrico cuyo radio es 3 veces el de un robot y centrado en la posición de la pelota y se coloca "detrás" (respecto a la portería contraria)
- Center
  - Se coloca detrás del centro del campo (respecto a la portería contraria)
  - Si es robot más cercano a la pelota trata de dirigirla.
  - El objetivo principal de este robot es esperar su oportunidad de intervenir en el centro del campo
  - Este es un robot reflexivo y ¿tonto?

Dteam. Comportamiento y posibles mejoras

- Ganaba a todos los equipos dados (stock teams)
- Esto lo cumplen casi todos los equipos nuevos ya que usan los equipos dados para su entrenamiento.
- Curiosidad → equipo Team-Brick
- Mejoras:
  - get_behind() usa demasiados cálculos
  - Sería más efectivo usar un “goalie” y un "offside" junto con jugadores del tipo SchemaDemo (or BrianTeam)
  - La opción de disparar es mejor que dirigir la pelota porque avanza más rápido → necesitas comunicaciones para estar seguro de no pasar al goalie o al “offside” → problemas de eficiencia

Otros equipos

- **SchemaDemo:** Utiliza un sistema de pesos y aprendizaje (con entrenamiento)
- **BrianTeam.**

Implementa una estrategia basada en las siguientes **condiciones**

- havetheball: am "behind" the ball and very near it
- clear: within kicking distance and no robots lie along path
- open: no opponent robot within x distance
- defense/offense: which side has ball, if unclear, say defense

- Y **acciones**
  - Shoot
  - dribble towards goal
  - go to point
  - Move away point

¿Qué os parece esta forma de diseñar equipos?

```
if( i have the ball ) then
   if( clear shot ) then  shoot
   else if( i am open ) then dribble towards goal
   else if( closest teammate is open && clear pass ) then  pass to him
   endif
else   if( i am closest to ball ) then  go to "behind" ball (avoid bumping
    it backwards)
   else   if( i am closest to ourgoal ) then
         go to point between ball and ourgoal nearest ourgoal
      else if( on defense )  mark a man (go to nearest open opponent)
      else  get open (move away from nearest opponent)
      endif
   endif
endif
```


Otros equipos

- BrianTeam. Estados
  - " is going to shoot!",
  - " is dribbiling toward the goal.",
  - " is running toward the ball.",
  - " is defending the goal.",
  - " is trying to get open, away from opponent.",
  - " is trying to get open, away from teammate.",
  - " is trying to get open, toward the ball.",
  - " is trying to clear the ball.",
- JunTeamHeteroG

```
public int takeStep() {
      updateSensors();      selectStrategy();      updateActuators();
      }
```

Otros equipos

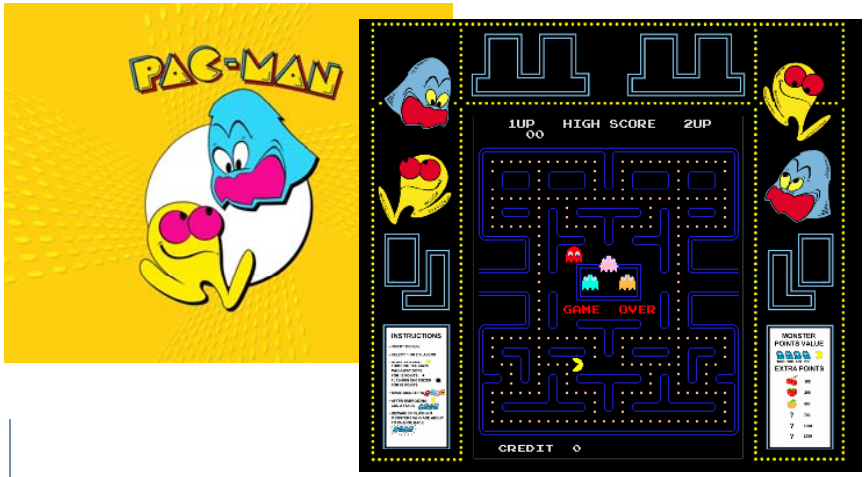
- JunTeamHeteroG

```
/** Selects a strategy to execute. Ideally, would select strategy based
   on opponent's history and formation. Didn't have time  to finish.    */
  private void selectStrategy() {
        // homogeneous();
        // heterogeneous();
        // kechze();
        // schema();
        // basicteam();
        dteam();
        // Wall();
        // Horde();
    }
```

Máquinas de estados de forma intuitiva

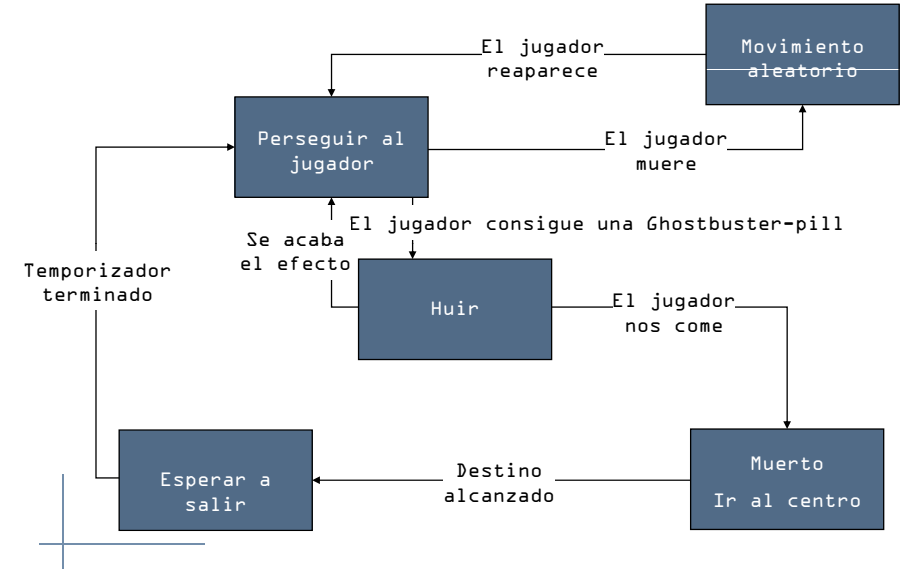
- Las máquinas de estados son un modo natural de dividir la conducta de un agente en varios comportamientos.
- En un momento dado, el personaje está en un “estado” de todos los posibles.
- El comportamiento en cada momento depende del estado actual.
- Se producen cambios de estado ante sucesos.
- Es una estructura “con memoria”.
- Son deterministas.

Ejemplo: PACMAN. Comportamiento de los fantasmas

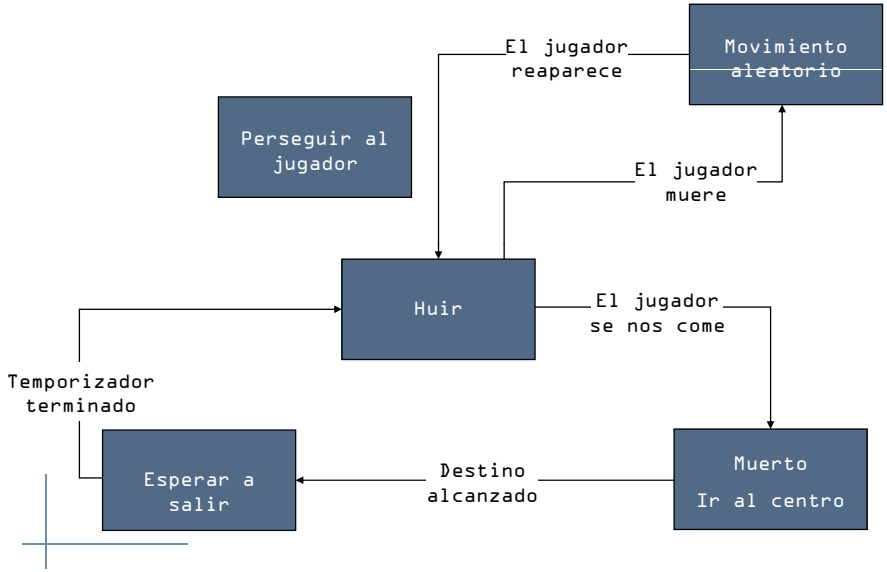


- Diferentes comportamientos, roles,.. “personalidades” de fantasmas:
  - Asustadizo: siempre huye del jugador.
  - Indeciso: va a por el jugador a veces sí, y a veces no.
  - Tonto: siempre va a por el jugador, incluso cuando se ha tomado una “Ghostbuster-pill”.
  - “Alelao”: pulula de forma aleatoria
  - ...

Máquina de estados del fantasma listo



Máquina de estados del fantasma asustadizo



Máquinas de estados finitas

- Un conjunto finito (no vacío) de estados + estado inicial
- Un conjunto de eventos de entrada
- Transiciones entre los estados
- Acciones (función de salida)

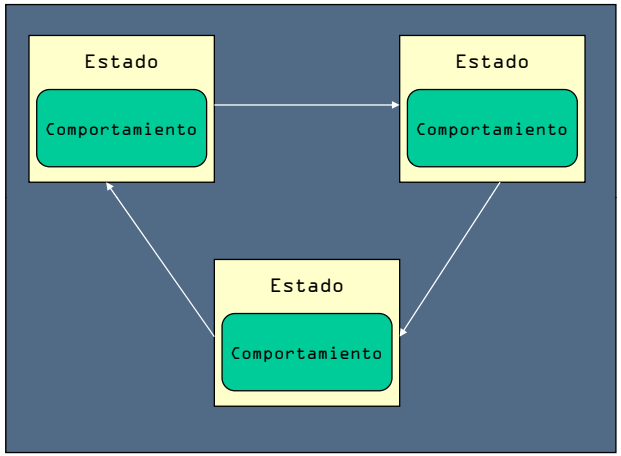
- Cualquier secuencia de eventos puede ser una entrada. La única restricción es que sea una del conjunto de entradas finitas identificado.
- Los eventos de entrada se procesan de forma síncrona
  - El siguiente evento se procesa únicamente después de que el evento actual haya sido procesado y la transición (si hay alguna) se haya ejecutado.
- Las acciones
  - Generan salidas, o se comunican con otros procesos.
  - **Las acciones no generan eventos de entrada, es decir, las máquinas de estados consumen entradas externas.**
  - Las acciones no tienen memoria.
  - Se asocian con las transiciones (Mealy) o con estados (Moore)

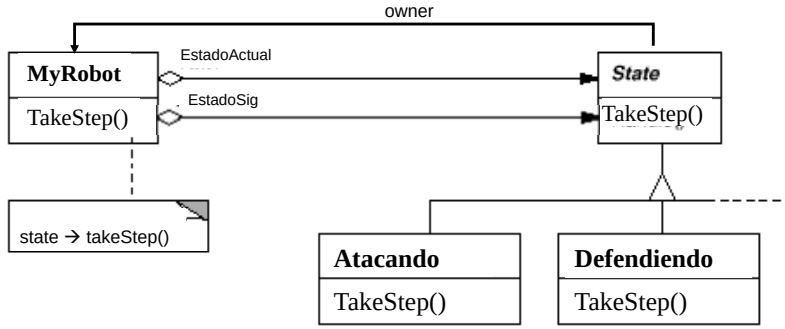
Máquinas de estados finitas

- Las máquinas de estados finitos se representan normalmente por medio de diagramas de transición de estados
- Grafos dirigidos donde cada vértice es un estado y cada arista es una transición.
- También se pueden representar mediante tablas de transición de estados: cada fila es un estado, cada columna es un evento de entrada, y las celdas dan los estados siguientes (o las acciones asociadas a cada transición).

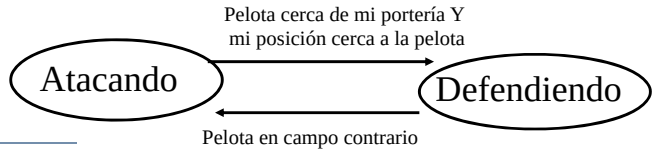
arraySiguiente	Entrada 1	Entrada 2	...	Entrada n
Estado 1				
Estado 2	Estado Siguiente			
...				
Estado m				

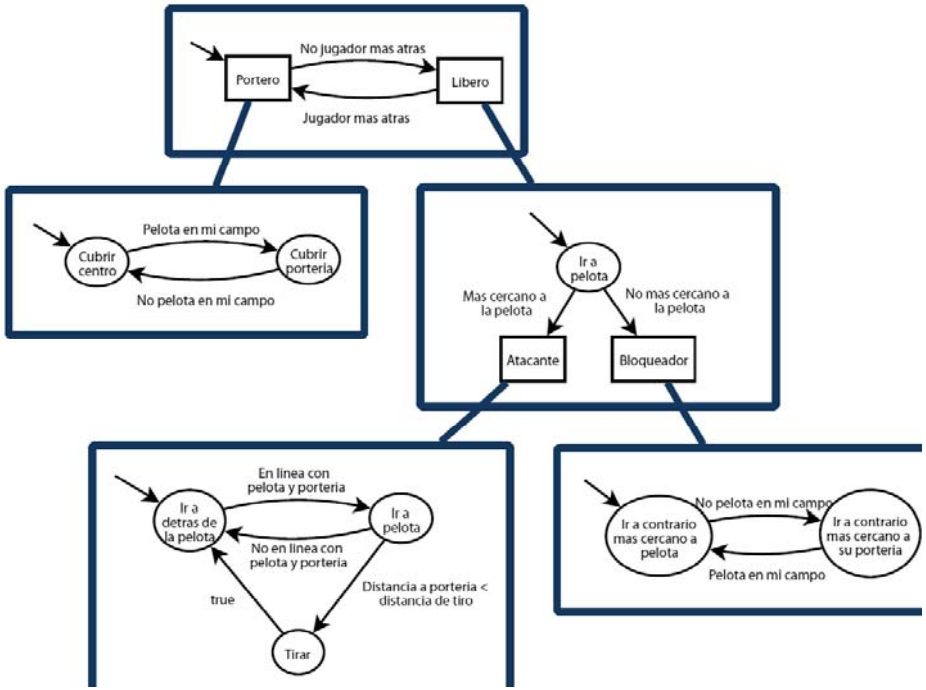
También puede haber máquinas de estados jerárquicas

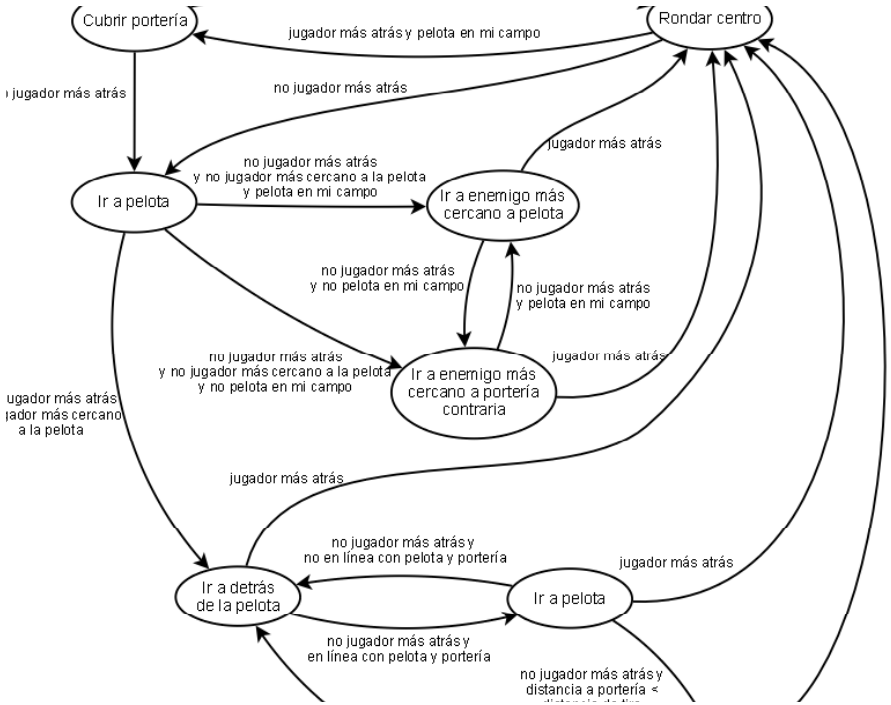


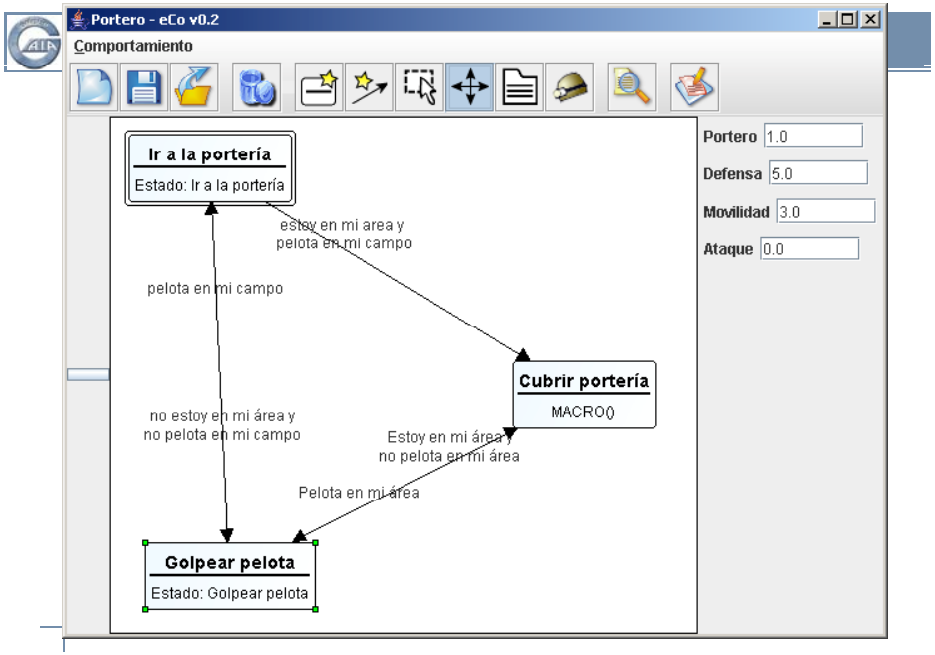


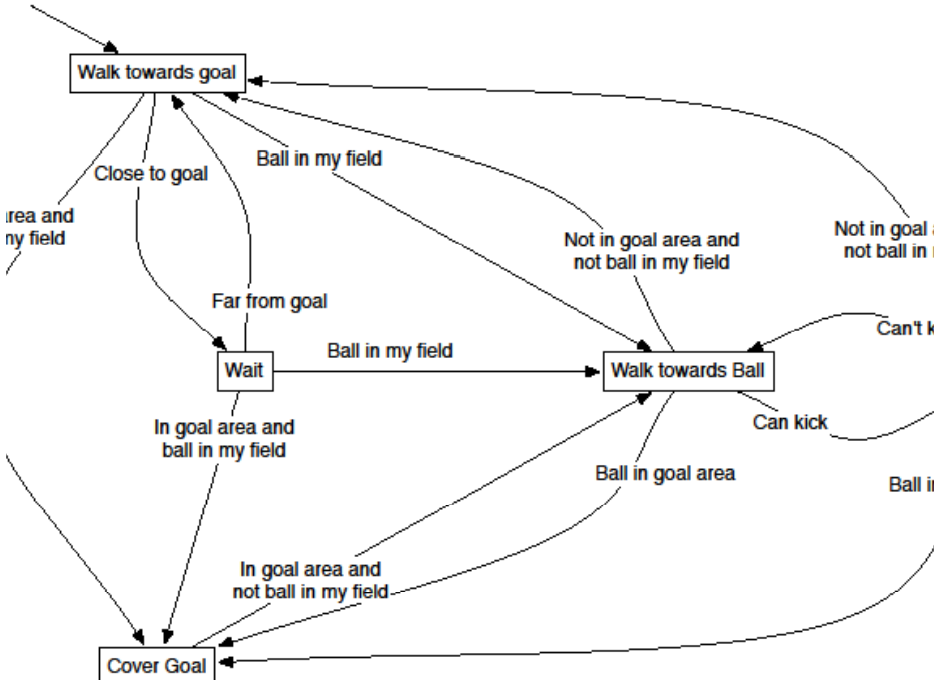
Ejemplo sencillo con dos estados











Diseño orientado a roles pero.. roles a distinto nivel de detalle

- Máquina de estados = Personalidad
- Roles = Comportamientos (tanto los de alto como los de bajo nivel)
- ¿A qué nivel de detalle tenemos que definir un comportamiento?
  - A alto nivel → Portero
  - A bajo nivel → Ir a por el balón / Ir a la portería
- Cada rol hay que verlo como una pieza que luego se va a poder componer para hacer otros roles.
  - Hacer una batería de comportamientos básicos (unas 10 líneas)
- Cualquier diseño es válido.
  - Comportamiento del portero codificado directamente en JAVA con estructuras IF-THEN.... El rol PORTERO es difícilmente reutilizable.
  - Comportamiento del portero diseñado como máquina de estados y transiciones. Cada estado del diseño es un rol que se puede reutilizar en otras máquinas de estados o en un diseño CBR.

Bibliografía

- **AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors.** Alex J. Campandard, New Riders, 2003
- **AI Game Engine Programming.** Brian Schwav, Charles River Media, 2004
- Rabin, Steve. "Implementing a State Machine Language", *AI Game Programming Wisdom*, Charles River Media, 2002
- **The Ultimate Guide to FSMs in Games _ Ryan Houlette, Dan Fu (Stottler Henke Associates, Inc.)AI Game Programming Wisdom 2, 2003.**
- http://dis.cs.umass.edu/research/fsm/fsmdummies.html
- http://sakharov.net/fsm.html
- **http://www.gamasutra.com/features/20041118/gill_01.shtml**
- **http://ai-depot.com/FiniteStateMachines/**

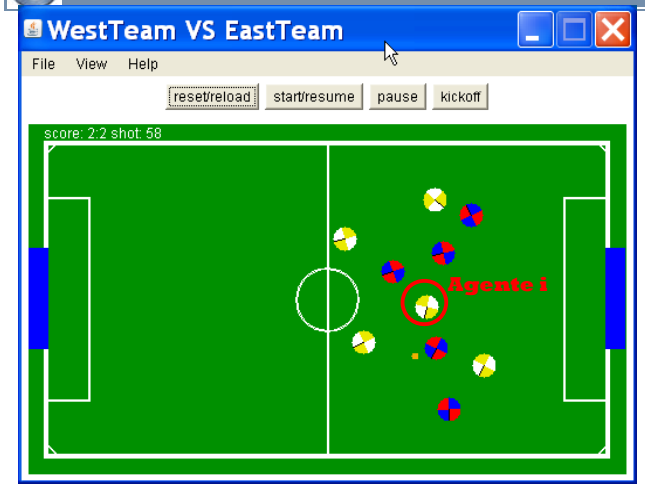
CBR y Soccer

- CBR
  - Los problemas tienden a recurrir
  - Reutilizar las soluciones anteriores cuando una situación se repita
  - La situación (o caso) no tiene que ser idéntica: similitud + adaptación de la solución
- Situación → Recupero un caso similar y reutilizo su solución
- ¿Qué es un caso? ¿Cuáles son los casos iniciales?
  - Base de casos semilla con la que empezar
    - Casos con jugadas típicas que metería un experto
  - Aprendizaje
    - Guardar casos obtenidos de logs de equipos que jueguen bien
- Vamos a ver dos opciones
  - Opción 1. Bajo nivel. Basada en trazas, posiciones y acciones. Casos para cada jugador.
  - Opción 2. Alto nivel. Basada en roles. Casos para el equipo completo.

Opción 1: casos para para cada agente

- Toma de decisión local para cada agente
- La base de casos puede ser **local o compartida**
- Agentei tiene que tomar una decisión, se define una consulta para la situación actual y recupera un caso:
  - Descripción (por ejemplo)
    - Valores de posiciones de las entidades (jugadores, pelota)
    - Resultado del partido
    - Tiempo
  - Solución
    - Acción concreta (atacar, mover, girar)

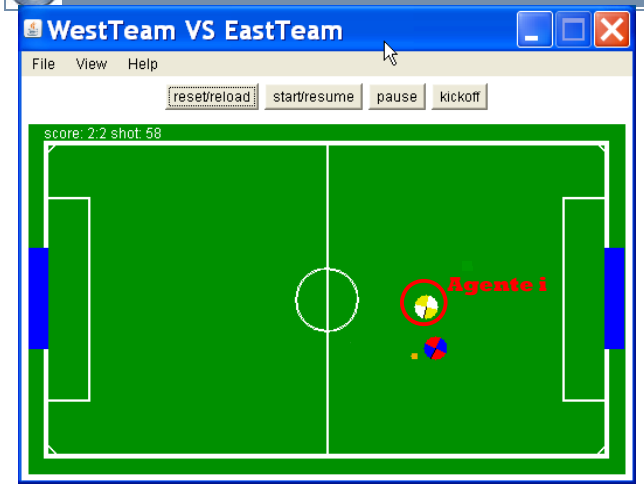
Situación



Los casos (consultas) pueden describir situaciones completas

Score 2:2
Tiempo restante
Posiciones (figura)

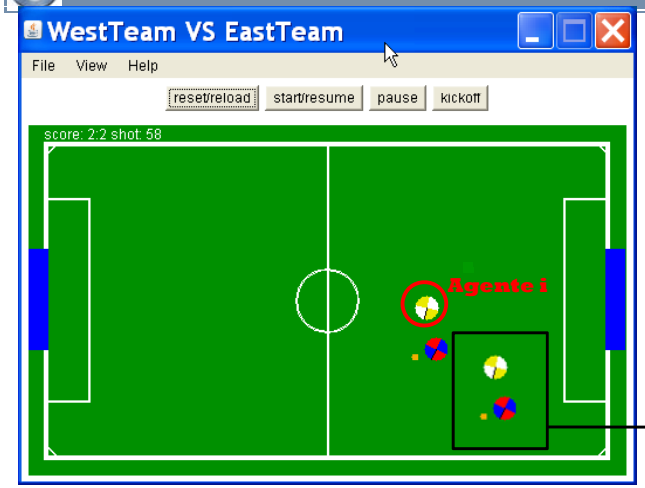
O situación simplificada



Los casos (consultas) pueden describir situaciones simplificadas

Score 2:2
Tiempo restante
Posiciones (figura)

Caso recuperado



Acciones sugeridas: Avanzar, kick,..

Jugadas complejas

Opción 2: caso para el equipo completo

- Descripción
  - Resultado parcial
  - Tiempo
  - ..... Posiciones ?
  - Estado actual de los jugadores
  - Cualquier característica obtenida de forma directa o no: forma parte del diseño
    - ¿cuántos contrarios hay en mi campo? ¿qué equipo tiene la pelota?
- Solución
  - 5 roles para asignar a los 5 jugadores

Este: rojo, defiende portería derecha
Oeste: amarillo, defiende portería izquierda

Consulta

Score 2:0

Tiempo restante: 10 seg

¿Quién tiene la pelota?

¿En qué campo está la pelota?

¿Cuántos de mi equipo están en el mismo campo?

Acción sugerida:

Este.

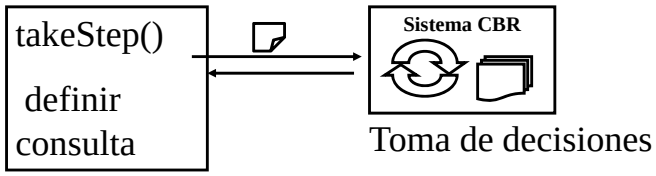
(portero, ataque, ataque, ataque, bloqueador

Oeste

(portero, portero, defensa, defensa, bloqueador)

CBR y Soccer

- Procesos CBR
  - Definir y programar la similitud entre casos
  - Adaptaciones simples
  - Aprendizaje (logs)
- Implementar la lógica de decisión de cada jugador utilizando un sistema CBR
  - Sistema externo más o menos genérico
  - ¿Cada jugador tiene su memoria de casos? ¿Todos los jugadores comparten la misma memoria de casos?



CBR y Soccer

- Idea de entrenador o asesor CBR o capitán del equipo (externo)
  - Decisiones a nivel de acción: un jugador le pregunta en cada takeStep ¿qué hago?
  - Secuencia de acciones
  - Decisiones a nivel de rol (o estado)
    - Identificar y programar distintos roles de jugadores (defensa, ataque, portero, offside, backup, estorbar, ...)
    - El entrenador decide la "alineación", es decir, le indica a cada jugador que "rol" debe jugar ➔ tiene más sentido en equipos de 11 jugadores
  - Le puede preguntar siempre (en cada decisión) o cada ciertos "ciclos" o si la situación ha cambiado mucho desde la última consulta, o .....

Base de casos

- ¿Existe una base de casos inicial?
  - Tiene sentido: la hace un experto. Casos semilla.
    - Base de casos de jugadas
    - Base de casos de equipos
      - Equipo de ataque (roles + posiciones de los jugadores)
      - Equipo de defensa
      - Equipo de bloqueo
      - .....
      - MUCHO QUE VER CON EL DISEÑO BASADO EN ESTADOS
  - Si no hay una base de casos inicial, el equipo empieza sin saber jugar, haciendo jugadas aleatorias y guardandolas.
  - Problema: en partidos de 1 minuto (traer equipos ya entrenados)

- ¿CBR para entrenamiento previo a los partidos o durante el partido?
  - Aprende a jugar contra un equipo en concreto... no hay mucho tiempo
  - Podéis hacer pruebas con equipos de otros grupos o con alguno de la distribución
    - Resultados antes del entrenamiento CBR
    - Resultados después del entrenamiento CBR
- Para aprender casos que jueguen como un cierto equipo se pueden usar los logfiles sobre un partido jugado.
- Es un archivo XML con información sobre las posiciones y la velocidad de la pelota y de los jugadores cada N milisegundos.

¿Cuándo decidir? ¿cuándo aprender?

- Cuando se pasa del campo de ataque al campo de defensa (y viceversa)
- Cada n segundos
- Después de un gol

- Aprender: memorizar casos nuevos + pesos
  - Difícil: determinación de responsabilidades/culpas

- Aprender en función de si el gol es a favor o en contra

Evaluación de la actividad docente

- DOCENTIA. Encuesta de calidad docente y evaluación del profesorado.
- http://vrdcd.ucm.es/encuestacalidad
- **Estará activa hasta el 31 de mayo.**
- **Los alumnos/as deberán cumplimentar la encuesta de evaluación de la actividad docente entrando en dicha URL y utilizando su dirección de correo institucional (correo estumail) de la Universidad .**