SIDANE: Sistema Inteligente para el Análisis del Comportamiento de Jugadores de Fútbol

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

SIDANE: Sistema Inteligente para el Análisis del Comportamiento de Jugadores de Fútbol

Gabriel Alises Cano

Febrero, 2013

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Tecnologías y Sistemas de Información

PROYECTO FIN DE CARRERA

SIDANE: Sistema Inteligente para el Análisis del Comportamiento de Jugadores de Fútbol

Autor: Gabriel Alises Cano
Director: David Vallejo Fernández

Febrero, 2013

Gabriel Alises Cano

Ciudad Real – Spain

E-mail: Gabriel.Alises@gmail.com

TRIBUNAL:

Presidente:

Vocal 1:

Vocal 2:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL 1	VOCAL 2	SECRETARIO
Fdo.:	Fdo.:	Fdo.:	Fdo.:

Resumen

El fútbol es uno de los deportes más populares, tal es su importancia que los últimos datos estiman que lo practican alrededor de 270 millones de personas en todo el mundo. El impacto social del fútbol es de tal magnitud que es capaz de movilizar a millones de personas de todos los continentes para el seguimiento de un partido importante o la celebración de un título. Si el impacto social es abrumador, el económico no se queda atrás. Según las últimas estimaciones, el fútbol genera alrededor de 500000 millones de dólares en todo el mundo. En todos los deportes de equipo, cuantos más títulos gana un equipo más importancia social y económica adquiere y en esto el fútbol no es una excepción. Al igual que tampoco lo es en la manera en que el rendimiento de un equipo depende del comportamiento de los jugadores del mismo durante un encuentro. Por tanto, será de gran utilidad construir sistemas que analicen, de forma automática, el comportamiento de los jugadores en un terreno de juego.

En la actualidad, existen diferentes herramientas que permiten definir comportamientos que los jugadores deben seguir durante el transcurso de un partido. Es tarea del cuerpo técnico de un equipo detectar si los jugadores cumplen realmente su rol en la táctica y estrategia de un equipo. Sin embargo, la idea de la detección de posibles errores en la ejecución de jugadas y su posterior análisis, de forma automática, es novedosa en el ámbito de los sistemas inteligentes aplicados al fútbol.

Sidane es un sistema que se apoya en una arquitectura adaptativa, extensible y modular, que hace sencilla la introducción y modificación tanto del conocimiento interno de la herramienta como de las funcionalidades que ofrece. Además, *Sidane* está creado empleando herramientas y estándares libres consiguiendo una fácil portabilidad.

Para probar la eficacia de la arquitectura, se utilizó *Sidane* para determinar y analizar el partido correspondiente a la final de la *RoboCup Soccer 2D*. Se estudiaron diversos comportamientos seguidos por los jugadores durante dicho partido, así como los datos obtenidos por la herramienta para dar explicación al resultado final del encuentro.

Abstract

Football is one of the most popular sports. Recent data estimate that is practiced around 270 million people in the world. The social impact of football is so great, that it is capable of mobilizing millions of people around the world to track a match or to celebrate a title. If the social impact is overwhelming, also the economic. Football generates about 500,000 million in the world, according to the latest estimates. In all team sports, the more titles a team wins, the more important social and economic gains and, in this football is no exception. Also, the team performance depends on the behavior of its components during a match. So, it will be useful to build systems that analyze, automatically, the player behavior in a field.

At present, there are different tools to define behaviors that players must follow during the course a game. The task of coaching team is to detect if players actually fulfilled their role in the tactics and strategy a team. However, the idea of detecting possible errors in the execution of moves and further analysis automatically is novel in the field of intelligent systems applied to football.

Sidane is a system based on an adaptive, extensible and modular architecture. Thus, the introduction and modification of the internal knowledge of the tool and the offered features is simple. Furthermore, *Sidane* is created using tools and open standards, getting easy portability.

To test the effectiveness of architecture, *Sidane* was used to determine and analyze the final match of the *RoboCup Soccer 2D*. We studied various behaviors followed by the players during the match and the data obtained by the tool to give the explanation of the final result of the match.

Índice general

Índice de tablas

Índice de figuras

Índice de listados

Agradecimientos

Dicen que *es de bien nacidos ser agradecidos*, a continuación y con riesgo de dejarme a alguien en el tintero, doy las gracias a las personas que de una manera u otra me han ayudado a llegar hasta aquí.

En primer lugar, a mi familia, a la que debo y siempre deberé todo. Ellos han hecho, siempre, todo lo ha estaba en su mano para ayudarme. Soy lo que soy gracias a ellos.

A Ana, ella ha aguantado estoicamente mis buenos y malos momentos, mis agobios y preocupaciones. Le doy las gracias porque me ha llenado de alegría y felicidad durante estos últimos años.

A Javi, Jorge, Monda, Rodri y Zaba por hacer mejor y más agradable mi paso por la universidad.

A mis amigos de toda la vida, porque como decía Richard Bach *Nuestra amistad no depende de cosas como el espacio y el tiempo.*

Por último, y no menos importante, a David Vallejo, mi director de proyecto. Él me ha ayudado a realizar este proyecto con su comprensión y paciencia infinita

Gabriel Alises Cano

A Rubén, siempre serás especial

Capítulo 1

Introducción

En los últimos años la industria futbolística se ha convertido en una de las industrias más poderosas a nivel mundial tal es así que, los últimos estudios estiman que el dinero generado por el fútbol es de unos 500000 millones de dólares anuales. Además, el fútbol es uno de los deportes cuyo impacto en la sociedad es enorme, puesto que es capaz de movilizar a millones de personas en todo el mundo para el seguimiento de un partido importante. Tanto la componente económica como social hacen del fútbol una disciplina que resulta interesante estudiar.

Actualmente, el fútbol es un deporte en el que se producen constantes cambios de tendencias. Muchos equipos pasan de la alegría de culminar una temporada brillante a la tristeza de encontrarse con una situación caótica poco tiempo después. En ocasiones, estos sucesos ocurren por causas económicas no controlables, como por ejemplo, la marcha de un jugador importante. Sin embargo, lo que lleva a un equipo a cuajar una mala temporada es, sin duda, una sucesión de malos resultados, siendo estos resultados la consecuencia de errores en la estrategia y táctica de un equipo. Por tanto, la existencia de un sistema que permita detectar errores, de forma automática, durante un partido para estudiarlos y que no se produzcan, repercutirá, a la postre, con la consecución de buenos resultados.

Para la detección de errores en la táctica y estrategia muchos equipos utilizan herramientas rudimentarias como pueden ser la grabación de partidos para su posterior reproducción o la toma apuntes en una libreta. Uno de los inconvenientes principales de estos métodos es que se pueden pasar por alto ciertos errores que hacen que el rival pueda obtener ventaja de los mismos. El principal obstáculo de la detección de errores es la cantidad de situaciones que componen un partido de fútbol puesto que, no solo hay que prestar atención a un jugador en concreto sino que hay que observar el comportamiento de todos los jugadores en el terreno de juego.

Con el objetivo de intentar remediar estas situaciones surge la necesidad de crear sistemas inteligentes que automaticen, en la medida de lo posible, el análisis de comportamientos de jugadores de fútbol. Esta idea resulta novedosa dentro del ámbito futbolístico puesto que, existe una carencia de sistemas que detecten y analicen automáticamente los comportamientos seguidos por los jugadores. Las ventajas que supondría utilizar un sistema de estas

características se corresponderían con la mejora del rendimiento de un equipo ya que, al detectarse comportamientos erróneos se realizarían más esfuerzos en preparar situaciones que otorgarían ventaja al rival para que no puedan ser aprovechadas.

En el presente Proyecto Fin de Carrera se presenta un sistema cuyo objetivo principal es la **detección de situaciones y análisis automático del comportamiento** de los jugadores en dichas situaciones. La eficacia de la detección de estas situaciones estará ligada al conocimiento incluido en el sistema, el cuál será proporcionado por el usuario que utilice el sistema. Así pues, un usuario que utilice *Sidane* puede evaluar el rendimiento de los jugadores de un equipo, en base a los comportamientos efectuados durante la duración de un partido. Por ejemplo, utilizando *Sidane* se podría determinar cuando un jugador debe golpear a portería y además evaluar, de forma automática, la acción realizada por el propio jugador, es decir, analizar el comportamiento seguido por el jugador, identificándolo como erróneo si el jugador realiza cualquier acción que no sea disparar a portería y como correcto en caso contrario.

1.1. Fútbol

1.1.1. Historia

El juego tal y como lo conocemos ahora tuvo su origen en las Islas Británicas en 1863. En la actualidad, el fútbol lo practican unos 270 millones de personas en todo el mundo [1].

Aunque la cuna del fútbol tal y como lo conocemos ahora, fueran las Islas Británicas, ya existían competiciones rudimentarias con semejantes elementos y objetivos que los del fútbol. Antes de 1863, el fútbol no estaba regulado y eran frecuentes los enfrentamientos violentos entre los equipos. Tampoco había regulación en la limitación del número de jugadores por equipo. Por tanto, eran frecuentes los enfrentamientos entre pueblos enteros y pequeñas ciudades. Actualmente, se desconoce el origen de este tipo de fútbol rudimentario. En la página web [2] de la **FIFA** (Fédération Internationale de Football Association) se barajan varias hipótesis tales como, que el origen de este deporte masivo se encuentra en Francia y que fueron los normandos quiénes llevaron el deporte a Inglaterra o, que existen investigadores que sitúan dicho origen en pueblos paganos, ya que el balón esférico representaba el sol y debía conquistarse para lograr buenas cosechas.

A principios del siglo XIX, el fútbol fue ganando importancia en escuelas públicas, dónde se desarrollaba con las diferentes reglas de cada escuela. Las reglas divergían tanto entre sí, que en algunas escuelas se podía llevar el balón con la mano.

En 1983, en la Universidad de Cambridge se intentaron identificar algunas conductas que no serían legales a la hora de practicar el fútbol. Generalmente, las acciones que serían castigadas serían las que se basaran en la violencia, como zancadillear o patear a un rival. Tam-

[1] Según fuentes oficiales de la FIFA
[2] www.es.fifa.com

bién, comenzó a aceptarse que no se permitiera jugar el balón con la mano lo que supuso la separación del fútbol de la época en dos, por una parte el fútbol actual y por otra el rugby.

Las primeras reglas del fútbol que se conocen fueron firmadas en Londres el 26 de Octubre de 1863, fecha en la que once clubes y numerosas escuelas se reunieron en la taberna Freemasons. Y así nació la **Football Association**. En 1872 se celebra la primera competición organizada de mundo: la Copa Inglesa. También, en ese año se celebró el primer partido internacional fue Inglaterra contra Escocia. Poco después de ese enfrentamiento se fundaron la Asociación Escocesa de Fútbol en 1873, la Asociación de Gales en 1875 y la Asociación Irlandesa en 1880. El primer campeonato de liga, en cambio, vio la luz dieciséis años más tarde, en 1888.

En 1885, la Asociación de Fútbol legalizó la profesión de futbolista, a raíz de los múltiples casos de pagos por parte de clubes a futbolistas para que jugaran en su equipo. Poco después se fundaron diferentes asociaciones fuera de las islas como, las asociaciones de Holanda y Dinamarca en 1889, las de Nueva Zelanda en 1891, Argentina en 1893, Chile, Suiza y Bélgica en 1895, Italia en 1898, Alemania y Uruguay en 1900, Hungría en 1901, Noruega en 1902, Suecia en 1904, España en 1905, Paraguay en 1906 y Finlandia en 1907.

En mayo de 1904 nació la **FIFA**, que contó siete miembros fundadores: Francia, Bélgica, Dinamarca, Holanda, España, Suecia y Suiza. En 1912, la FIFA contaba ya con 21 asociaciones; en 1925 con 36; en 1930, año de la primera Copa Mundial, con 41.

Entre 1937 y 1938, las reglas del juego modernas fueron establecidas por Santley Rous, quien sería, más adelante, el Presidente de la FIFA. Rous tomó las reglas originales, creadas en 1886, y las ordenó de manera racional. Estas reglas serían revisadas por segunda vez en 1997.

Para el año 1950, cuando se volvió a competir por tercera vez (porque no hubo torneos durante la Segunda Guerra Mundial) por un titulo del mundo, la FIFA contaba ya con 73 asociaciones. Para el Congreso de la FIFA de2007, FIFA contaba con 208 asociaciones.

1.1.2. Posiciones básicas

Teniendo en cuenta la antigüedad de este deporte y la cantidad de personas que lo juegan, se pueden apreciar una serie de comportamientos básicos dentro del terreno de juego, como que hay principalmente 4 tipos de jugadores (**porteros, defensores, medio-campistas o delanteros**), o, por ejemplo, que la mayoría de los goles los anotan los delanteros porque son los que poseen una posición en el campo más adelantada que el resto de jugadores.

Las características básicas de cada tipo de jugador son las siguientes:

- **Portero**. El portero, también conocido como arquero, guardameta o cancerbero, es el encargado de evitar que el equipo contrario marque gol. El portero es el único jugador que puede tocar el balón con las manos durante un partido, aunque sólo en su propia

área. Está equipado con una vestimenta diferente del resto de sus compañeros y, suele llevar el número uno en su camiseta. Además, tanto al inicio como durante el transcurso del partido, debe haber un único portero por equipo. El arquero es el jugador más retrasado de un equipo.

- **Defensores**. Los defensores son los encargados de detener el ataque rival. Para denominar a los defensores o defensas existen multitud de términos en referencia a la distribución táctica del equipo. En el fútbol moderno, la defensa la componen cuatro jugadores. Los jugadores más cercanos a las bandas reciben el nombre de **laterales** y, dependiendo de la banda que ocupen serán laterales izquierdos o derechos. La principal función de un lateral es aparte de defender, proyectarse en ataque con el objetivo de sorprender al equipo rival. Por otro lado, los dos defensores restantes reciben el nombre de **centrales**, aunque también son conocidos como zagueros. Son los centrales los que deben tratar de contener los ataques frontales del rival. Los defensores se sitúan por delante del portero y por detrás de los medio-campistas.
- **Medio-campistas**. También conocidos como centro-campistas, tienen diferentes roles en el desarrollo de un partido. En defensa deben ayudar a los defensores a detener el ataque del contrario además de bascular su posición para que el equipo no quede roto. Se dice que un equipo está roto cuándo hay mucha separación entre los atacantes y los defensores. Por contra, en ataque, los medio-campistas deben mover el balón con el objetivo de conducirlo hacia el delantero. Al igual que ocurría con los defensores existen varios tipos de medio-campistas como pueden ser los siguientes:
  - Los **interiores o volantes** son los centro-campistas que se encuentran más escorados dentro del terreno de juego. Su principal función es la de conducir el balón por las diferentes bandas para, posteriormente pasar el balón a un compañero que se encuentre en una buena situación para lograr marcar un gol.
  - El **medio-centro**. Es el jugador que se encuentra inmediatamente después de los defensores. Este jugador tiene varias y distintas funciones, todas ellas son importantes para el rendimiento de un equipo. Una de ellas es la de dotar al equipo de equilibrio para evitar que el equipo se rompa. Otra función es la creación de juego, este jugador cuando tenga la posesión del balón, típicamente, se encargará de distribuirlo a las zonas dónde haya menos contrarios. Por último, otro objetivo de este jugador es el de arrebatar el control del balón al equipo rival antes de que la ofensiva llegue a posiciones comprometidas para los objetivos de un equipo.
  - El **enganche**. Es otra de las posiciones importantes en el desarrollo del juego. Un enganche deberá disponer de la habilidad necesaria para conducir el balón rápidamente entre los jugadores contrarios y entregárselo al delantero. Este jugador se sitúa, normalmente, por detrás de los delanteros y más adelantado que el medio-centro.

- El **delantero**. La principal responsabilidad de este jugador es la anotar los goles de un equipo. Este tipo de jugador es el más adelantado de un equipo. Su función en ataque está definida pero, en defensa debe encargarse de dirigir la presión de un equipo hacia el rival.

1.1.3. Impacto social y económico

El fútbol es uno de los deportes que más capacidad de movilización de la sociedad posee. El fútbol es un deporte en el que los equipos más poderosos son capaces de llenar estadios de más de 80000 personas, prácticamente cada fin de semana. Tanto es así, que existen datos que estiman que la final del mundial del 2010 fue vista por unos 700 millones de personas en todo el mundo. En comparación con otros deportes de alto seguimiento social, como son el fútbol americano o el tenis, este dato resulta imponente, puesto que la final de la *SuperBowl* del 2012 tuvo una audiencia de 111 millones de personas, mientras que la final de *Roland Garros* del 2012 fue seguida por 46 millones de personas. Esto pone de manifiesto la importancia del fútbol en la sociedad actual.

En el fútbol, los mejores jugadores son reconocidos mundialmente, esto hace que se conviertan en ídolos para una parte de la sociedad. Este reconocimiento se convierte en expectación cuando uno de estos jugadores disputa un partido. Dicha expectación se transforma en dinero, el cuál va a parar a las arcas del club que posee los derechos de dicho jugador. Así pues, existe gente que ve el fútbol como un negocio.

Tal es la importancia de fútbol a nivel económico que los últimos datos estiman que este deporte es capaz de generar alrededor de unos 500.000 millones de dólares anualmente. Este dato no resulta tan impactante si se tiene en cuenta que el coste medio diario para club *UEFA*[3] por jugador internacional es de 27.000 euros y que el coste medio diario para clubes de las cinco mejores ligas es de 86000 euros. La producción económica del fútbol está por encima de las estimaciones realizadas sobre otras industrias como, por ejemplo, la de los videojuegos que genera unos 105.000 millones de dólares anuales.

En España, el fútbol supone unos ingresos al estado de 821 millones de euros anuales. El impacto total en la producción española es de 8.066 millones de euros, lo que supone 1.7 % del PIB[4] y la generación de alrededor de 66000 puestos de trabajo.

1.2. Problemática

En la actualidad, la mayoría de sistemas existentes dentro del ámbito futbolístico están relacionados con la visualización de vídeos que permiten indicar la secuencia de movimientos que deben seguir los jugadores en un terreno de juego. En estos sistemas es posible definir la

[3] Union of European Football Associations
[4] Producto Interior Bruto

disposición táctica de los jugadores en ciertas situaciones del partido, como *corners*, saques de faltas o contraataques, entre otros. La principal utilidad de estos sistemas se basa en la preparación previa a un partido, de manera que se utilizan para indicar a cada jugador del equipo su rol en cada una de las circunstancias que se den durante el desarrollo del partido.

Por otro lado, existen sistemas de gestión que proporcionan los medios necesarios para controlar la información relativa a un equipo, como el número de goles de cada jugador o el número de minutos jugados. Estos sistemas se utilizan para la construcción de estadísticas de un equipo.

Aunque estos sistemas mejoren en parte la preparación de un partido y, en parte el análisis de las estadísticas de un equipo, no sirven para evaluar el rendimiento de un equipo durante un partido, que es lo que resulta determinante para la consecución de buenos resultados.

En la actualidad, existe una carencia de sistemas que, de forma automática, detecten situaciones complejas y analicen el comportamiento seguido por los jugadores. Por tanto, el análisis del comportamiento de los jugadores de un equipo en un partido de fútbol, sigue siendo manual. Los principales motivos que dificultan el desarrollo de este tipo de sistemas son los siguientes:

- **El análisis automático del comportamiento**. Analizar el comportamiento de diferentes entidades resulta complejo por varias razones, como son las siguientes:
  - **La cantidad de posibles acciones a realizar**. Por ejemplo, en el contexto del fútbol, un jugador que se encuentre escorado en una banda podrá, idealmente, pasar la pelota a un compañero que se encuentre en el centro del campo, adelantado o atrasado. Pero no sólo podrá pasar el balón, sino que también podrá avanzar con el o regatear a su marca, entre otras acciones. Cuál de estas acciones es la que debe realizar el jugador, la respuesta es que depende de la situación en que se encuentre. Por tanto, un sistema que analice el comportamiento automáticamente deberá identificar la situación en que se encuentren las entidades.
  - **Determinar la acción realizada**. Una vez que se ha realizado un acción, el sistema deberá determinar si dicha acción es la adecuada o no. Identificar la acción llevada a cabo es, normalmente complejo, puesto que hay que tener en cuenta diferentes variables. Por ejemplo, en el fútbol, el comportamiento que debe seguir un jugador cuando pasa el balón es, normalmente, desmarcarse. Así pues, el sistema deberá evaluar si el jugador ha realizado algún movimiento y además determinar si se ha librado del marcaje del rival. Este es un ejemplo concreto para una situación concreta. Para diferentes situaciones el sistema deberá evaluar diferentes variables, lo cuál supone un aumento de la complejidad.

- **La obtención de la totalidad de la información de un partido**. Para realizar un análisis automático de los comportamientos de los jugadores, no basta con tener los datos de los goleadores del encuentro o de la posesión de un equipo y de otro, sino que es necesaria información exhaustiva de cada uno de los jugadores, puesto que se debe evaluar cada situación y determinar el comportamiento seguido por los jugadores en dicha situación. Por ejemplo, sería necesario disponer de la posición de los jugadores en cada segundo, determinar el jugador que tiene el control del balón o el nivel de presión que tiene un jugador cuando está rodeado, entre otras.

  A pesar de que se celebren centenares de partidos cada fin de semana, resulta complicado extraer información que se adecúe a las necesidades de un sistema de este tipo, puesto que, la mayoría de información se encuentra disponible en un formato audiovisual difícil de procesar.

  Actualmente, mucha información útil para sistemas de este tipo, se encuentra en manos de organizaciones privadas que, imponen costes muy elevados y restricciones, lo que dificulta aún más la obtención de la misma.

1.3. Propuesta de Proyecto Fin de Carrera

El desarrollo de un sistema inteligente de análisis automático de comportamientos en un dominio concreto es una tarea ardua, debido a la complejidad vista en la sección anterior. En este Proyecto Fin de Carrera se propone un sistema que permita al usuario identificar situaciones relevantes situaciones que se dan durante el desarrollo de un partido de fútbol, así como analizar el comportamiento de los jugadores en dichas situaciones.

Como se ha comentado anteriormente en la problemática discutida, existe una carencia de sistemas que analicen de forma automática los comportamientos de los jugadores durante un partido. Actualmente, son los expertos en fútbol los que determinan si el comportamiento de los jugadores es correcto o erróneo. Por tanto, existe la necesidad de desarrollar sistemas que analicen automáticamente el comportamiento de los componentes de un equipo durante un partido.

El sistema a desarrollar deberá ser fácilmente extensible y modular de manera que la inclusión de nuevas funcionalidades sea lo más sencilla posible. Con una arquitectura modular cada módulo es independiente de los demás y tiene asignada una responsabilidad específica. Así pues, si hubiese que incluir nueva funcionalidad, bastaría con identificar el módulo responsable del cumplimiento de dicha funcionalidad sin tener que modificar el resto de módulos.

Otra característica del sistema a desarrollar será la adaptabilidad. El sistema ofrecerá la posibilidad al usuario de aportar su propio conocimiento acerca del fútbol. De esta manera, cualquier usuario de *Sidane* podrá definir el conocimiento necesario para la identificación de

situaciones que requieran ser analizadas automáticamente.

Es importante que el sistema disponga de una interfaz gráfica de usuario que permita visualizar y analizar los comportamientos detectados. Así, un usuario podrá, de una forma sencilla, identificar las situaciones detectadas por el sistema y determinar el comportamiento analizado por el sistema. La interfaz gráfica permitirá al usuario interactuar con el sistema de una forma cómoda.

En resumen, *Sidane* tratará de detectar situaciones que se produzcan en el transcurso de un partido, tales como la detección la línea mal trazada por parte de la defensa de un equipo, el cambio de juego o el disparo a portería, entre otras. También, analizará automáticamente el comportamiento seguido por los jugadores cada una de estas situaciones, evaluando y determinando si se realiza la acción correcta, según el conocimiento introducido en *Sidane*.

1.4. Estructura del documento

La estructura que se sigue en el presente documento se ajusta la normativa vigente para la realización del Proyecto Fin de Carrera en la titulación de Ingeniería Superior Informática por la Escuela Superior de Informática de Ciudad Real, aprobada en Junta de Centro el 8 de noviembre de 2007.

Capítulo 1: Introducción

Se expone una breve introducción del área en la que se enmarca el presente Proyecto Fin de Carrera. Además, se detalla la problemática que supone el desarrollo de un sistema inteligente para el análisis automático de comportamientos, así como la propuesta de Proyecto Fin de Carrera.

Capítulo 2: Objetivos

Se describen los objetivos planteados para el presente Proyecto Fin de Carrera. En este capítulo se describe el objetivo general del sistema, así como, los objetivos específicos que deberá resolver la herramienta desarrollada.

Capítulo 3: Antecedentes

Se realiza un estudio de las áreas de conocimiento necesarias para entender y desarrollar este proyecto.

Capítulo 4: Método de trabajo

Se recoge y expone la metodología utilizada, así como el software y el hardware necesario para la construcción de este proyecto.

Capítulo 5: Arquitectura

se describe la arquitectura del sistema, analizando la funcionalidad de cada uno de los submódulos en los que se divide la arquitectura.

Capítulo 6: Evolución, Resultados y Costes

Se detalla la evolución del proyecto, defiendo las iteraciones necesaria para el desarrollo del mismo. También, se exponen y contrastan los resultados obtenidos por el sistema. Por último, se detalla una estimación del coste del proyecto.

Capítulo 7: Conclusiones y Propuestas

En este capítulo se realiza una valoración general tanto del desarrollo del sistema, como de los resultados obtenidos. Se exponen las posibles líneas de trabajo futuro en la herramienta. El capítulo concluye con una valoración personal acerca de lo que ha supuesto el desarrollo del proyecto.

Capítulo 2

Objetivos

La motivación del presente proyecto, titulado *Sidane*, viene dada por la necesidad de proporcionar una herramienta que ayude a la identificación, análisis automático de comportamientos y visualización de determinadas situaciones que ocurren en partidos de fútbol. La herramienta debe ser extensible y configurable dando la posibilidad de definir variedad de situaciones que analizar.

2.1. Objetivo general

El objetivo principal de *Sidane* es el desarrollo de un **sistema inteligente** para el **análisis automático** del comportamiento de jugadores de fútbol, ofreciendo una herramienta que facilite dicho análisis de una manera interactiva y sencilla para el usuario. La aplicación deberá detectar situaciones con el objetivo de facilitar la identificación de las mismas en un partido de fútbol.

A menudo, en cualquier área se utiliza una información específica para expresar ciertas situaciones. En el deporte en general, y en el fútbol en particular, es frecuente escuchar expresiones cuya información es, frecuentemente, ambigua o vaga. Por ejemplo, dentro del ámbito futbolístico es común la expresión "el lateral sube *mucho* por la banda rival" o "se han replegado *pocos* defensores". Pero, ¿cuánto es *subir mucho* o cuánto son *pocos defensores*?. Teniendo en cuenta esto, la tarea de diseñar e implementar software que evalúe información con incertidumbre es, a menudo, compleja. El manejo de la incertidumbre es uno de los principales problemas a solventar, puesto que, la presencia de la misma cambia radicalmente la manera en que el software toma decisiones. El sistema a desarrollar deberá solucionar estas situaciones para conseguir un análisis del comportamiento de los jugadores efectivo y real. Para manejar el tratamiento de la incertidumbre se emplearán diferentes técnicas. La mayoría de estas técnicas se basan en diferentes conceptos y algoritmos de la Inteligencia Artificial.

La herramienta deberá analizar los comportamientos que siguen los jugadores en el terreno de juego, determinando para cada caso, si el jugador hace lo que se espera o no. Teniendo en cuenta esto, es necesario que la aplicación posea mecanismos para que un usuario sea capaz

de definir nuevos comportamientos a detectar y analizar. Por ejemplo, una situación a analizar puede ser determinar lo que debe hacer un jugador cuando se encuentra presionado por varios oponentes. Así, para que un usuario tenga la capacidad de introducir nuevo conocimiento a la herramienta, será necesario que la misma disponga de mecanismos de obtención de información de diferentes variables. Algunas de estas variables puede ser determinar el jugador que posee el balón en un instante determinado, el número de oponentes que se encuentran cercanos, la posición en los ejes cartesianos de cada uno de los jugadores, así como del balón o la resistencia restante de un jugador.

El sistema utilizará información almacenada en una base de datos. De manera que se podrá adaptar cualquier información procedente de diferentes medios para que sea almacenada y, posteriormente utilizada para el análisis de comportamientos. La intención de la herramienta es que cualquier entrada de información de partidos pueda ser traducida a un formato estándar que la herramienta maneje. De esta manera, se podrá analizar el comportamiento de los jugadores en cualquier partido, tanto profesional como *amateur*.

Desde el punto de vista de la visualización, el usuario debe poder manejar el partido a su antojo, de manera que, si desea ver cómo ha sido el primer gol pueda, de alguna manera, navegar hacia el mismo y observarlo. De la misma manera ocurrirá con los comportamientos que la herramienta analice. Además, mediante el uso de la aplicación el usuario será capaz de generar información, de forma numérica o gráfica, que permita examinar y sacar conclusiones y estadísticas acerca de las distintas fases de un encuentro. También, el usuario podrá interactuar con la herramienta de una manera cómoda y sencilla. Además, la aplicación permitirá cierta personalización, como por ejemplo, cambiar el balón o las equipaciones de cada uno de los equipos.

En resumen, la herramienta perseguirá la construcción y desarrollo de un sistema inteligente para el análisis automático de comportamientos de jugadores de fútbol. La interacción con dicho sistema será sencilla, siendo posible la introducción de diferentes comportamientos a analizar.

2.2. Objetivos específicos

A partir de este objetivo general surge una serie de objetivos específicos que detallan el alcance del presente proyecto:

- **Análisis automático de comportamientos**: El sistema deberá analizar de forma automática el comportamiento seguido por los jugadores en diferentes acciones que componen los partidos de fútbol. El sistema indicará cuando un jugador realiza una acción correcta y cuando realiza una acción incorrecta en un acción determinada. Por ejemplo, una situación posible a analizar es qué hace un jugador cuando se encuentra desmarcado y con la posesión del balón dentro del área rival. En esta situación,

el comportamiento correcto sería disparar a portería. Por tanto, el sistema analizará el comportamiento del jugador, determinando si dicho jugador dispara a portería o no.

- **Escalabilidad**. Se pretende conseguir que el sistema sea extensible, es decir, que permita al usuario definir nuevas situaciones a analizar, sin que suponga un cambio en la arquitectura. El usuario debe poder introducir diferentes comportamientos de jugadores para que sean detectados y analizados por la herramienta. También, la herramienta debe ser extensible de manera que la inclusión de nuevas funcionalidades o la modificación de algunas de ellas se lleve a cabo de manera sencilla.
- **Adaptabilidad**. Diseño de una arquitectura adaptativa basada en el uso de patrones de diseño. Mediante el almacenamiento de la información en base de datos se permitirá que cualquier entrada de datos, en un formato concreto, pueda ser reconocida por la herramienta, para su posterior análisis. De esta manera, se podría obtener información de partidos reales, en cualquier formato (vídeo, audio, texto,...) para, posteriormente, realizar un análisis automático de comportamientos.
- **Modularidad**: La arquitectura estará formada por componentes independientes con el objetivo de obtener un acoplamiento bajo y una cohesión alta. Para facilitar esto, las responsabilidades de cada módulo estarán bien definidas. Aparte de obtener buenos niveles de acoplamiento y cohesión se pretende que la arquitectura sea escalable para añadir nuevos módulos en un futuro.
- **Robustez**: El sistema deberá ser tolerante a fallos de manera que se recupere de errores y pueda continuar operando como si no se hubiesen producido.
- **Facilidad de uso**: Se pretende que la herramienta pueda ser utilizada por usuarios con conocimientos futbolísticos, no informáticos, de manera que, la aplicación sea tan sencilla como sea posible para que, un usuario con escasos conocimiento en informática pueda interactuar con la misma fácilmente.
- **Portabilidad**: El desarrollo de proyecto se realizará siguiendo estándares y tecnologías libres, multiplataforma y consolidados, con el objetivo de que pueda ser portado al mayor número de plataformas posibles.
- **Interpretabilidad**: Se generarán estadísticas de comportamiento de los jugadores. Los resultados serán relativos a cada uno de los jugadores de un equipo en distintos intervalos de tiempo y cómo varían a lo largo del partido. Dando la posibilidad de comparar los comportamientos en diferentes partidos.
- **Interfaz interactiva**: Se desarrollará una interfaz gráfica de usuario interactiva para visualizar simulaciones, así como avanzar o retroceder en ellas de una manera sencilla y atractiva al usuario.

Capítulo 3

Antecedentes

En este capítulo se realiza un estudio del estado del arte de aquellas áreas de trabajo vinculadas directamente con la realización del presente Proyecto Fin de Carrera. Además, se estudian diversas estrategias y tácticas utilizadas en el fútbol moderno, útiles para identificar correctamente situaciones que se den en partidos de fútbol. También, se proponen técnicas de Inteligencia Artificial y sistemas artificiales aplicados al fútbol con el objetivo identificar conceptos que serán de utilidad en el desarrollo del software.

3.1. Estrategias y tácticas en el fútbol profesional

Dado que el fútbol es un deporte esencialmente físico, existen diferentes cualidades que es recomendable que un jugador de fútbol posea:

- **Velocidad**. La velocidad en el fútbol se emplea en el fútbol para, por ejemplo, anticiparse a los defensores del equipo rival a la hora de obtener el balón.

- **Técnica**. Dominar el control del cuerpo y del balón es esencial para conservar la posesión del balón, especialmente en situaciones desfavorables.

- **Resistencia**. Esta cualidad es imprescindible para un jugador de fútbol, ya que cuanto mayor sea la resistencia mayor tiempo podrá aguantar dentro del terreno de juego.

Existen otras cualidades no técnicas que también son importantes para un jugador fútbol, como el **compañerismo**, el **sacrificio** o la **entrega**. Estas cualidades no técnicas se repiten en cada deporte en el que varios jugadores comparten un equipo.

Relegando a un segundo plano la habilidad técnica de cada jugador, la mayoría de los partidos se decantan de un lado o de otro gracias a la preparación previa de un encuentro. Dicha preparación se lleva a cabo por medio del **entrenador** y del **cuerpo técnico del equipo**. Por ejemplo, la mayoría de entrenadores saben que, si se enfrentan contra un equipo ofensivo, entonces tendrán más posibilidades de ganar si el equipo contrario tiene menos tiempo la posesión del balón. Por tanto, la figura del entrenador y de todo el cuerpo técnico es una de las principales bazas de un equipo a la hora de preparar un partido.

En este apartado se estudian diversas tácticas que ayuden a aprovechar cada uno de los medios disponibles para obtener ventaja con respecto al equipo rival. El objetivo del mismo consiste en identificar nuevos conceptos que sean aplicables a cualquier equipo dentro de un terreno de juego.

3.1.1. Terreno de juego

El **terreno de juego** es el lugar dónde se práctica el fútbol. Se trata de un rectángulo compuesto de distintos elementos (ver figura 3.1):

- Dos porterías, que poseen una anchura de 7.32 por 2.44 de altura, en las se intenta introducir el balón para anotar un gol. En dichas porterías se coloca el **portero**, que es el único jugador de fútbol que puede utilizar los brazos para interactuar con el balón.
- Dos áreas, denominada **área grande** en la jerga futbolística, principalmente porque contienen otro área más reducida a la que se refiere como **área pequeña**. A su vez, dentro del área grande se encuentra un pequeño circulo situado a 11 metros de la portería que se denomina **punto de penalti**. Es en este pequeño círculo donde se colocará el balón en caso de que se produzca una infracción dentro del área grande.
- Cuatro esquinas, denominadas **córners**, desde las que se pone el balón en juego cuando éste traspase la línea anterior a la portería.
- Un círculo en el centro del campo, desde el cuál se pone el balón en movimiento al inicio de cada parte del partido y cuando un equipo haya anotado un gol.

3.1.2. Acciones a balón parado

La planificación, en cualquier contexto, es determinante para la obtención de buenos resultados. Así pues, el deporte no es una excepción, puesto que se trata de un conjunto de personas que pretende lograr un objetivo.

En el fútbol existen diferentes situaciones en el que se hace necesario un plan que permita obtener algún beneficio. Dicho plan debe estar determinado desde el inicio del encuentro y todos los jugadores deben saber cuál será su rol en él. Las situaciones en las que se hace necesario un planteamiento específico son, en su mayoría, **acciones a balón parado**, es decir, cuando un integrante de un equipo debe poner el balón en movimiento, como por ejemplo:

- **Saque de centro**: El balón se encuentra en el centro del campo esperando a ser puesto en movimiento. El saque de centro es un momento ideal para observar el planteamiento de cada uno de los equipos. En este momento, los equipos se encontrarán en alguna disposición táctica que permita obtener alguna ventaja con respecto al otro equipo. A las disposiciones de los equipos se les denomina **alineaciones**, y todas tienen sus ventajas y desventajas. El apartado 3.1.3 discute algunas de las alineaciones más utilizadas

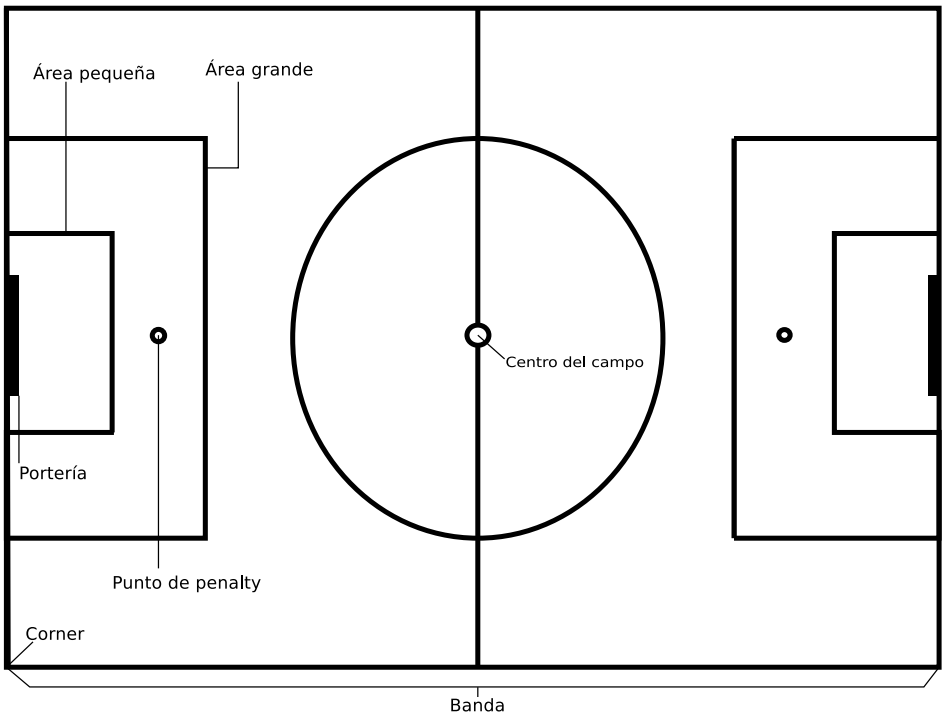


Figura 3.1: Terreno de juego

en el fútbol moderno.

- **Saque de banda**: El balón se encuentra en una de las bandas. A diferencia del saque de centro, el saque de banda lo realiza algún integrante del equipo con las manos. El saque de banda es, junto con el de centro, el menos peligroso de las acciones a balón parado puesto que, normalmente, el recorrido del balón es menor al ser impulsado por los brazos.
- **Saque de esquina**: Los saques de esquina han cobrado especial importancia en el fútbol moderno. De hecho, existen multitud de estrategias que permiten a los equipos marcar gol más fácilmente. El saque de esquina consiste en poner el balón en movimiento desde una esquina del terreno de juego. Esta acción da lugar a varias posibilidades como, por ejemplo, impulsar el balón al área o jugar el balón con un compañero que esté al lado de la esquina. El apartado 3.1.4 propone ejemplos de estrategias en saques de esquina.
- **Saque de falta**: Es una situación en la que un jugador pone el balón en juego con el pie desde el punto donde se ha cometido una infracción. Estas situaciones son más ventajosas cuanto más cerca del área rival se haya cometido la infracción.

En todas las jugadas a balón parado es necesaria una perfecta sincronización por parte de cada uno de los jugadores que intervienen en la estrategia. Aparte de dicha sincronización, es determinante una buena precisión a la hora de realizar los movimientos y el desplazamiento del balón. Es decir, el balón no debe desplazarse a la zona donde lo espera un jugador antes

o después de que dicho jugador se encuentre en esa zona, puesto que no llegaría a recibirlo correctamente. El lanzador tampoco debería enviar el balón a un lugar donde no lo esperen los jugadores implicados en la estrategia.

Para llevar a cabo las jugadas a balón parado es necesaria una buena preparación, la cual se consigue mediante la repetición de las estrategias en los entrenamientos.

Existen, sin embargo, otras estrategias distintas a las de balón parado que se corresponden con la especulación. Por ejemplo, es común cambiar el planteamiento del partido si en el minuto 75 el resultado es desfavorable o introducir un defensor más si el resultado es favorable.

3.1.3. Alineaciones

La **alineación** de un equipo se corresponde con la disposición de los jugadores sobre el terreno de juego [Fun99, Sau10]. Una alineación se denota con una serie de números separados normalmente con guiones. Cada uno de estos números se corresponde con la cantidad de jugadores que ocupan esa línea del campo, entendiéndose ésta como una recta imaginaria que cruza verticalmente el terreno de juego. Algunas de las alineaciones más comunes en el fútbol moderno son las siguientes:

4-4-2

Esta alineación (ver figura 3.2) es la más utilizada en todo el mundo [Has09]. Se compone de cuatro defensores (dorsales: 2, 3, 4, 5), cuatro centrocampistas (dorsales: 7, 8, 6, 11) y dos delanteros (dorsales: 9 y 10).

En un principio se ideó como táctica defensiva, pero con el paso del tiempo se ha planteado como una estrategia que puede ser utilizada tanto ofensiva como defensivamente. Esto depende de las características individuales de los centrocampistas.

Con esta estrategia se puede pasar al ataque, adelantando los centrocampistas y los delanteros, obteniendo una alineación claramente ofensiva como es el 4-1-3-2 (ver figura 3.3).

Esta alineación también permite cambiar fácilmente a un modo defensivo. Simplemente bastaría con retrasar cada una de las líneas, dejando que los delanteros se encarguen de presionar la línea de centrocampistas del equipo contrario (ver figura 3.4).

4-3-3

Esta alineación (ver figura 3.5) se centra en la táctica ofensiva, modificándose a una alineación del tipo 4-4-2 para la defensa [Has09]. Se compone de cuatro defensores (dorsales: 2, 3, 4, 5), tres centrocampistas (dorsales: 6, 8, 19) y tres delanteros (dorsales: 7, 9, 11).

Con esta alineación, el papel de los delanteros cobra especial importancia porque son los

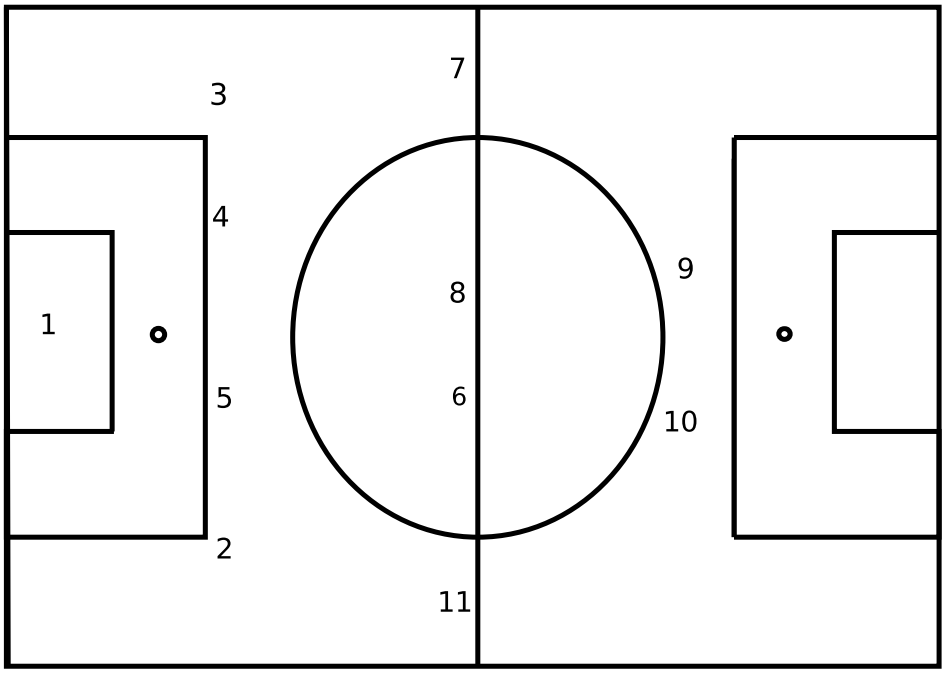


Figura 3.2: Alineación 4-4-2

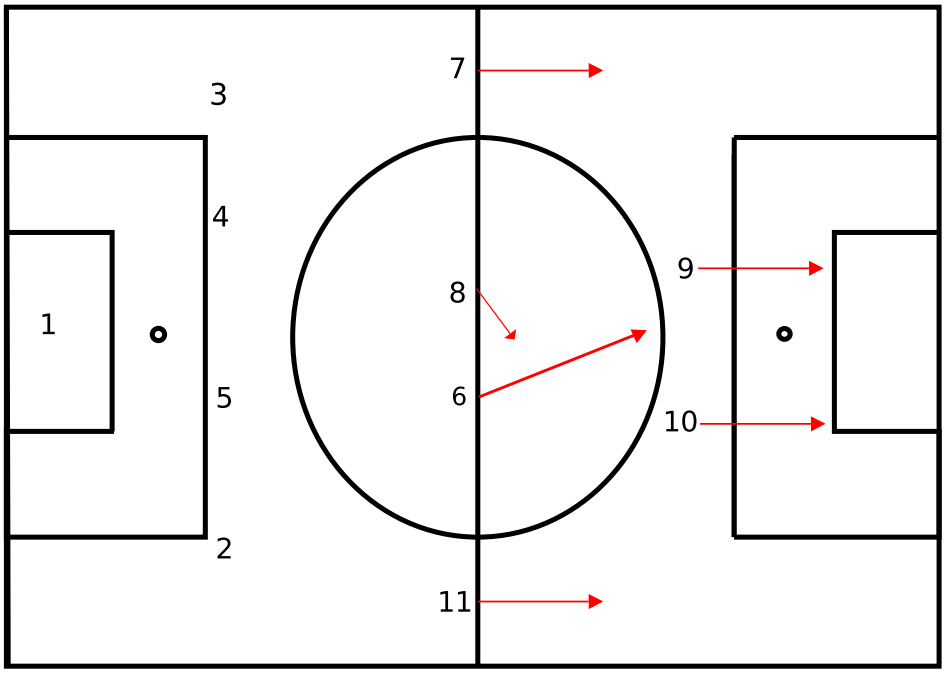


Figura 3.3: Alineación 4-1-3-2

jugadores que más posibilidad de movimientos poseen:

- **Diagonal interior**, (ver figura 3.6) en la que el delantero-centro retrasa su posición al centro del campo, arrastrando a los defensas del equipo contrario y creando un espacio en el centro el área rival que puede ser aprovechado por alguno de los delanteros de las bandas.

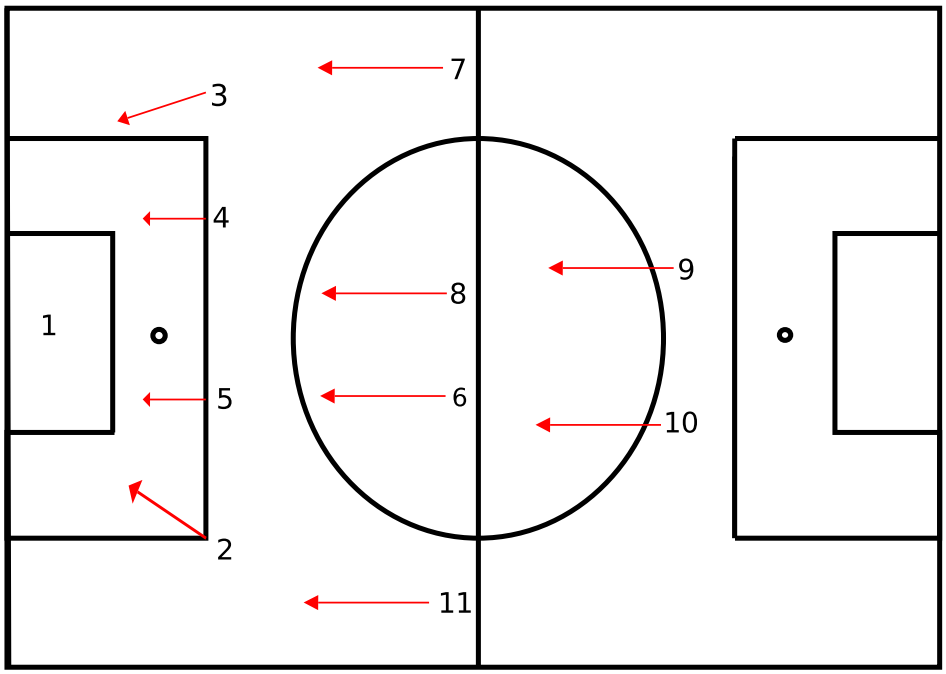


Figura 3.4: Alineación 4-2-2 defendiendo

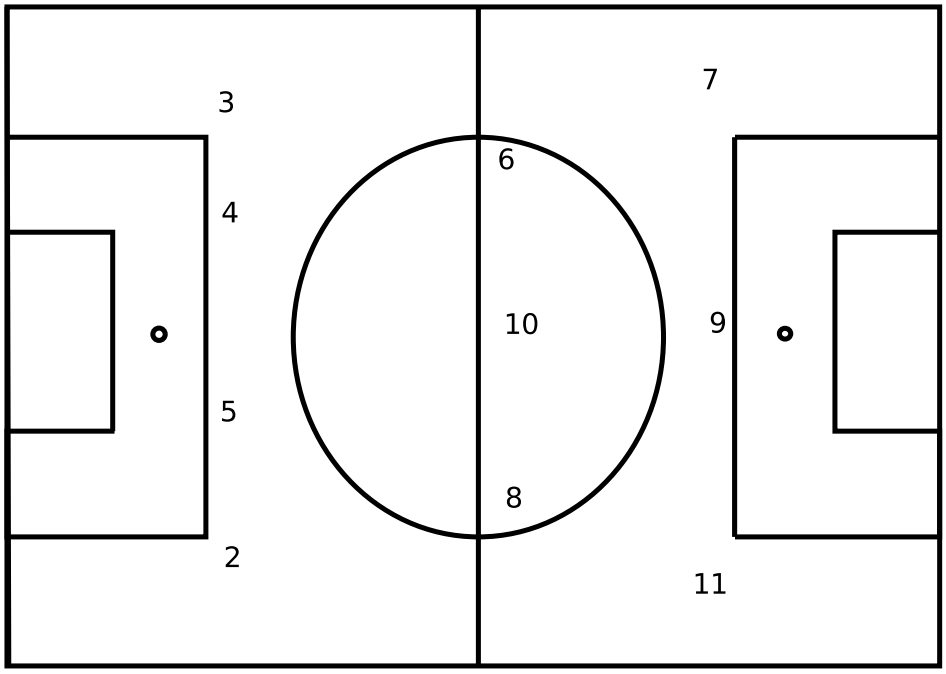


Figura 3.5: Alineación 4-3-3

- **Diagonal exterior**, (ver figura 3.7) la que, a diferencia del anterior, el delantero de la banda avanza acercándose a los límites del terreno de juego con objeto de provocar un uno contra uno con el defensa que se encarga de su marcaje. Por otro lado, el delantero-centro avanza su posición hacia el área rival con el objetivo de recibir el balón por parte del delantero de la banda que ha hecho el movimiento de diagonal exterior.

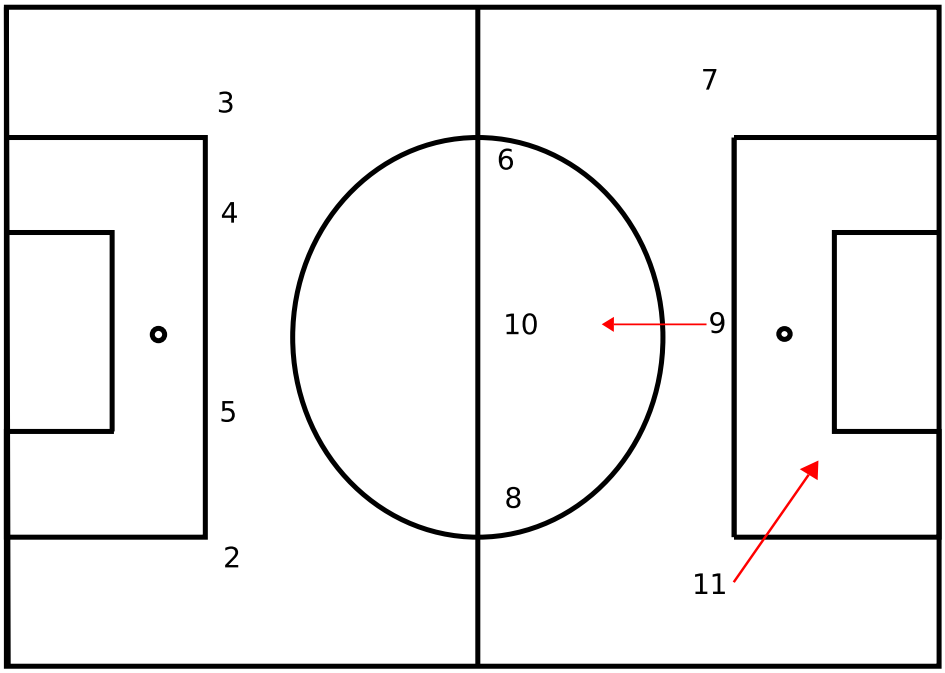


Figura 3.6: Diagonal interior

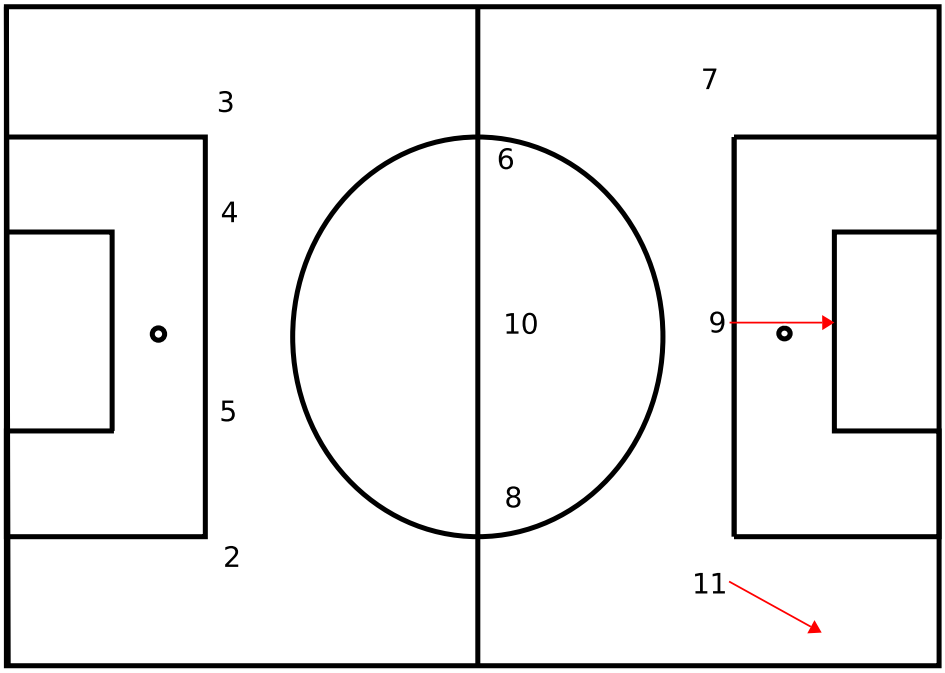


Figura 3.7: Diagonal exterior

- Los delanteros también pueden **retrasar su posición** para la creación de espacios (ver figura 3.8) que pueden ocupar otros jugadores, rompiendo la dinámica de la táctica defensiva del equipo contrario.

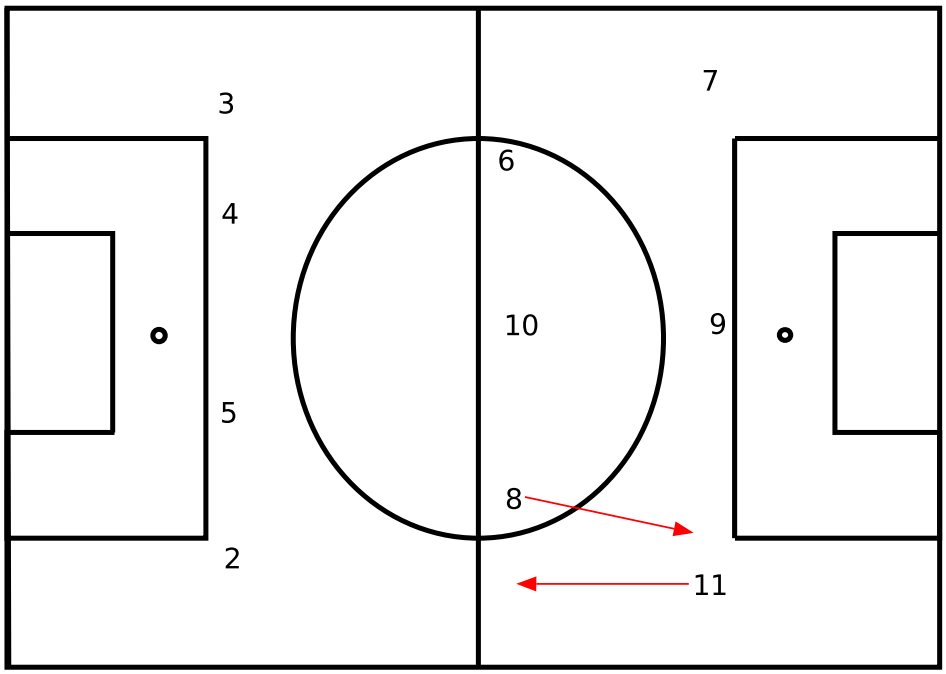


Figura 3.8: Creación de espacios mediante permuta de posiciones

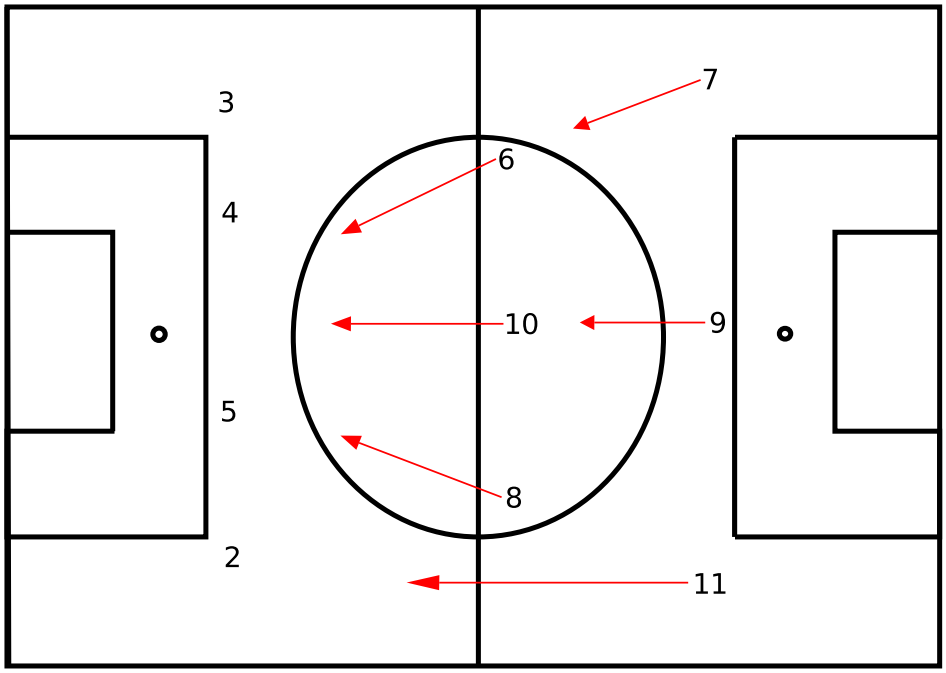


Figura 3.9: 4-3-3 → 4-4-2

Para cambiar a una modalidad más defensiva, simplemente bastaría con retrasar la posición de un delantero al centro del campo, obteniéndose así una alineación 4-4-2 (ver figura 3.9).

A diferencia de la alineación 4-4-2, que basa la presión al centro del campo del equipo rival, esta táctica centra su presión en la línea defensiva del contrario, haciendo que la salida controlada del balón sea difícil.

3.1.4. Saque de esquina

Un **saque de esquina** (o córner) es una excelente situación para anotar un gol [Fou08, Fun99]. El equipo que dispone de un córner tiene una ocasión para realizar una jugada de estrategia en la que puede obtener ventaja sobre el rival y así poder marcar gol. Tanta es la importancia de los saques de esquina que incluso en algunos países se ha llegado a denominarse como medio gol.

Antes de estudiar alguna jugada de estrategia en saques de esquina, es necesario definir el concepto de **bloqueo** [Fun99] como la *acción que realiza un jugador ocupando la zona de un defensor, de manera que otro jugador que realiza el desmarque queda libre de la marca del defensor.*

Una estrategia muy utilizada en saques de esquina es el desplazamiento del balón hacia el **primer palo** [1] [Tur01] (el lado de la portería más próximo a la esquina donde se procederá a poner el balón en juego). Así, un rematador se puede adentrar en el área desde una posición retrasada y que otro jugador realice un bloqueo sobre el defensor que se encarga de marcar al rematador (ver figura 3.10). Al realizar esta táctica, el jugador que recibe el balón se encuentra en una situación idónea para decidir si

- disparar a portería.
- pasar el balón a un compañero que se encuentre en una situación mejor (ver figura 3.11) para que sea este último el que remate a portería.

Esta táctica se basa en que la acumulación de jugadores es menor en el primer palo y fuera del área que dentro de ella [Tur01]. Así pues, será más sencillo que un jugador reciba y controle el balón en una zona donde no haya un número elevado de jugadores que donde si exista.

Además del primer palo, también existen jugadas **al segundo palo** o palo largo (el lado de la portería que está más alejado de la esquina donde se encuentra el balón). Es también, en el segundo palo donde la acumulación de jugadores es menor. Sin embargo, mientras que en los saques al primer palo el desplazamiento del balón puede ser tanto a *ras del suelo* como por encima, en los saques al palo largo el balón siempre debe ser aéreo, para evitar las piernas de los defensores.

La estrategia en este caso[2] consiste en que el jugador encargado de sacar el *corner* eleve el balón hasta el segundo palo donde, previamente, se haya desmarcado el rematador. El

[1] http://www.expertfootball.com/coaching/tactics.php. Accedido el 10 de Septiembre de 2012.
[2] http://www.expertfootball.com/coaching/tactics.php Accedido el 10 de Septiembre de 2012.

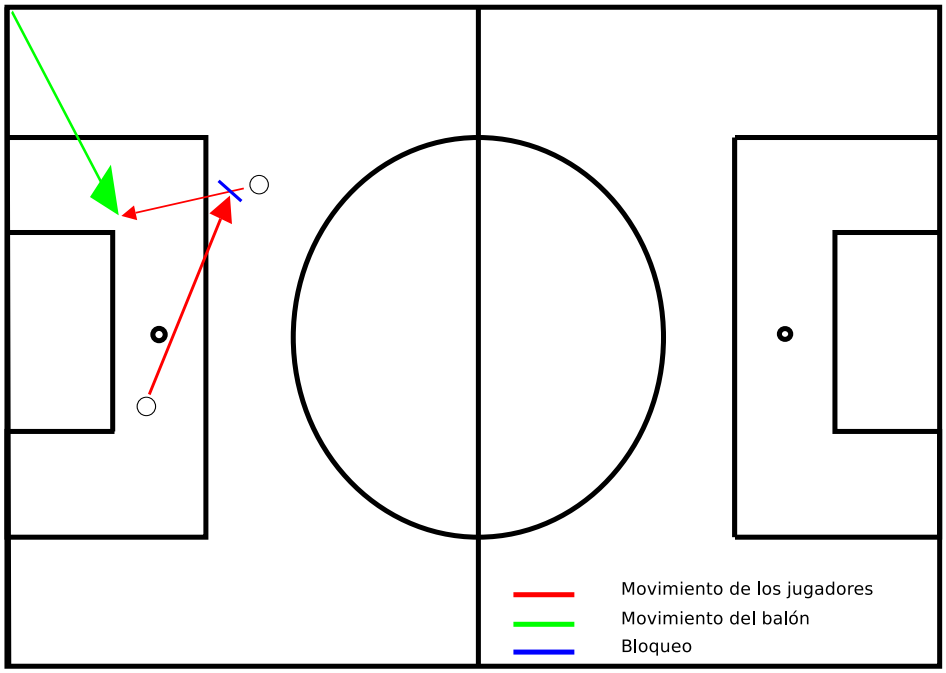


Figura 3.10: Saque de esquina al palo corto

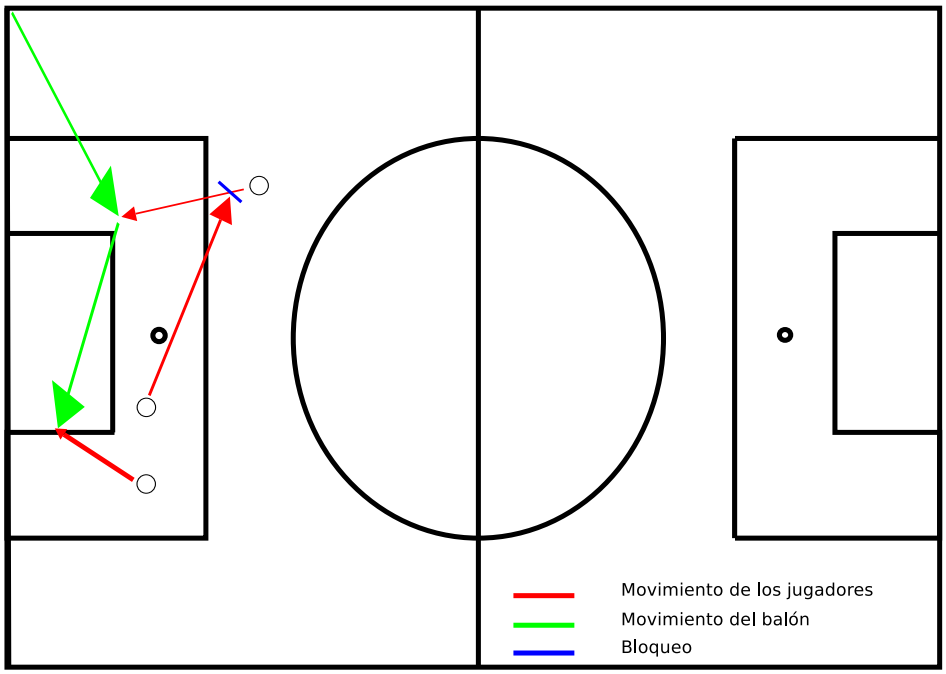


Figura 3.11: Variante de saque de esquina al palo corto

desmarque es posible gracias a la realización de algún bloqueo por parte de un compañero que anule a su defensor. En esta posición ventajosa el rematador puede intentar rematar a portería (ver figura 3.12).

Al igual que en la jugada anterior, esta táctica tiene otra variante que consiste en que cuando el jugador al que va el balón desde la esquina esté en disposición de remate lo envíe

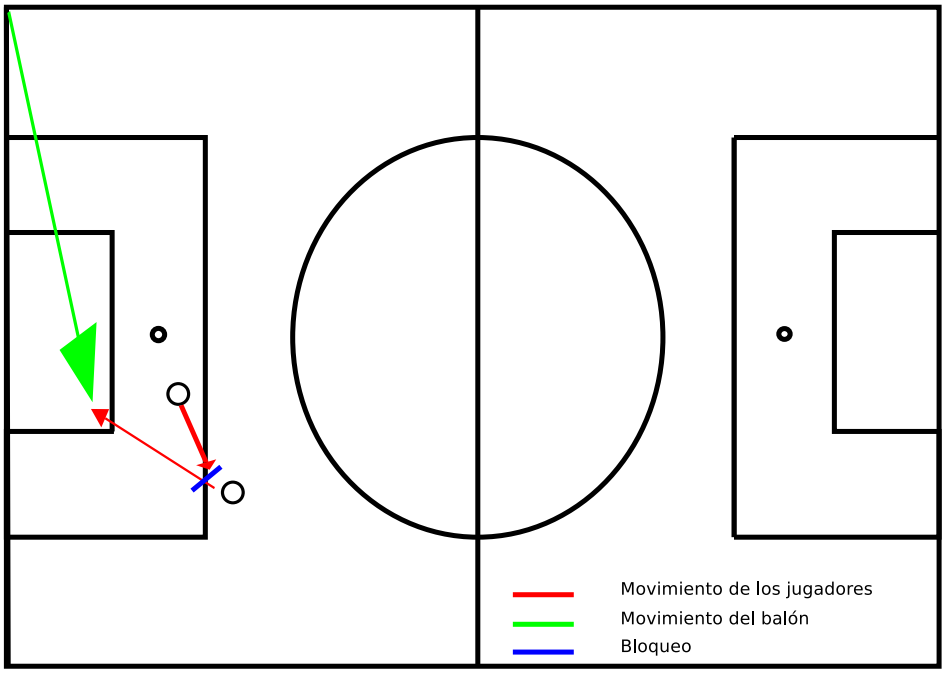


Figura 3.12: Saque de esquina al palo largo

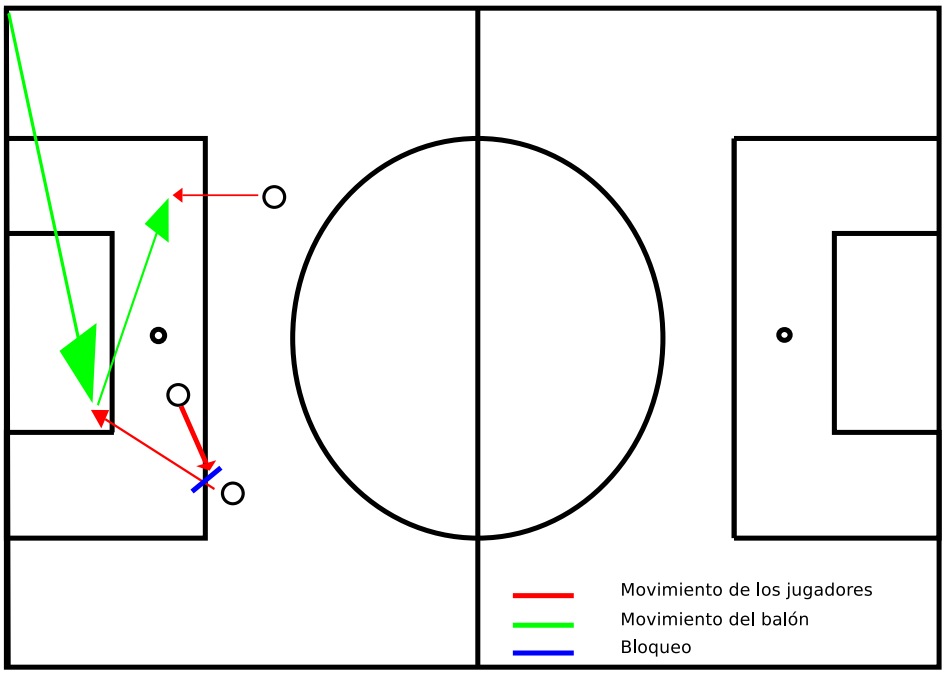


Figura 3.13: Variante de saque de esquina al palo largo

al primer palo, en busca de otro compañero al que le sea más fácil el remate, puesto que seguramente el portero se encuentre vencido intentando evitar el posible remate del primer jugador (ver figura 3.13).

También otra opción muy utilizada es poner el balón en juego con un compañero que se encuentre cerca de la esquina [Tur01]; esto se denomina **sacar en corto** (ver figura 3.14). Un

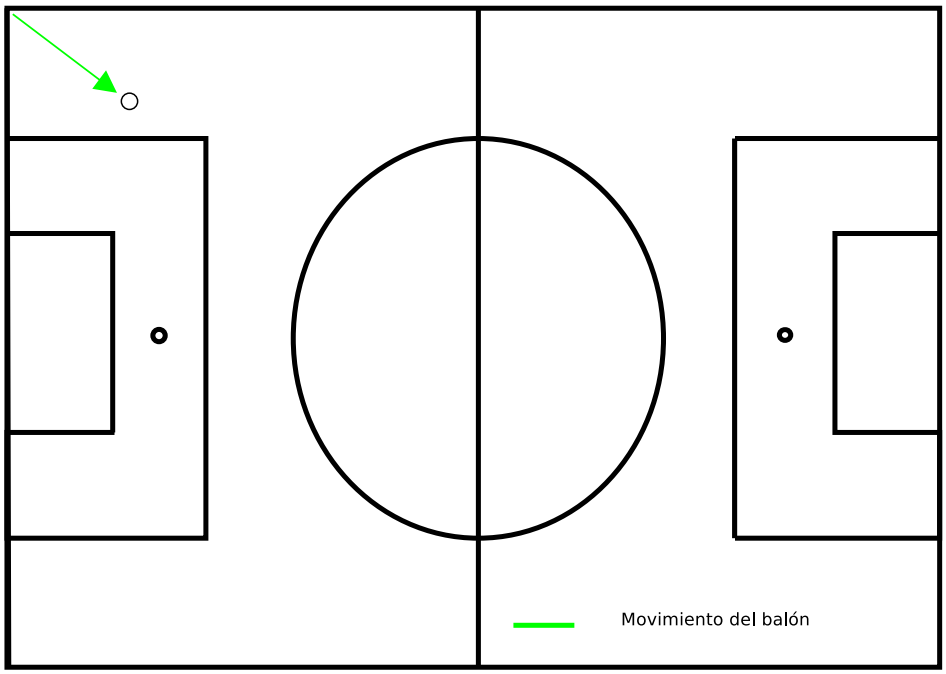


Figura 3.14: Saque en corto

saque en corto tiene dos objetivos principales:

1. **Mantener la posesión del balón**: Esta estrategia se realiza cuando el resultado está ajustado. De esta manera, no se arriesga a la posible pérdida del balón en caso de que la jugada ensayada no se haya ejecutado correctamente.
2. **Centrar la atención hacia la parte del campo dónde se produce el saque de esquina**. Al sacar en corto, algunos defensores intentarán arrebatar el balón al jugador que lo posee, creándose espacios dentro del área que es posible utilizar para intentar marcar un gol.

Contraataques

A menudo, un córner se identifica como una situación ventajosa para los intereses de un equipo. Sin embargo, puede producir un efecto negativo llamado **contraataque**. Un contraataque, es una situación en la que se produce algún error en la estrategia de un equipo dando la posibilidad al rival de aprovechar dicho error. En las acciones a balón parado suele producirse este tipo de situaciones debido, principalmente, a la dificultad de sincronización y falta de precisión en la ejecución de los movimientos de los diferentes jugadores que intervienen en una jugada.

En los córner el peligro de un contraataque aumenta con respecto a otras situaciones de estrategia debido a que es necesario que intervengan un número elevado de jugadores para conseguir marcar un gol. Al aumentar el número de jugadores en ataque, se reduce el número de jugadores con roles defensivos, lo que un fallo en la ejecución de la jugada propiciaría

que el rival pudiese atacar con más jugadores a los que un equipo pueda defender.

3.1.5. Saque de falta directa

Al igual que todas las jugadas a balón parado, los **saques de falta** son una ocasión especial para la aplicación de estrategias que permitan marcar un gol [Fun99]. Además del uso de tácticas, también existe la posibilidad de disponer de un lanzador experto dentro del equipo para poder disparar directamente a puerta.

Los saques de falta directa pueden producirse en cualquier lugar dónde se produzca una falta o infracción, es decir, en cualquier parte del terreno de juego. Las faltas que más próximas estén a portería tiene más probabilidades de acabar en gol que las que se encuentran en una posición más alejada.

En estas situaciones, para evitar el gol del equipo que realiza el saque de falta, el equipo que se defiende normalmente sitúa un número de jugadores a una distancia del balón que se denomina **barrera** [Fou08]. La misión del lanzador es desplazar el balón salvando la barrera.

Al igual que ocurre en los saques de esquina existen jugadas con desplazamiento de balón al palo largo o al palo corto [Tur01]. Por ejemplo, una jugada típica al palo largo implicará que (ver figura 3.15) el lanzador desplace de manera aérea el balón hacia el segundo palo, donde previamente el rematador ha ocupado esa zona y está libre de marca para rematar a portería. Esta jugada es similar a la táctica de saque de esquina al palo largo.

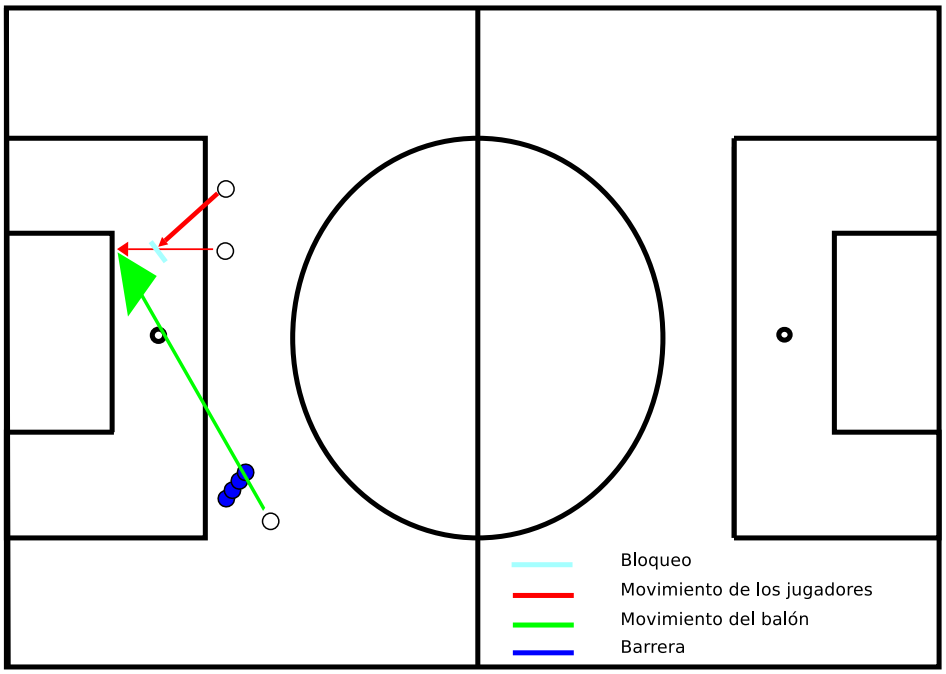


Figura 3.15: Falta al palo largo

Al igual que con los saques de esquina, las faltas también se pueden sacar al palo corto [3](ver figura 3.16). El lanzador se pone el balón en juego hacia el hueco existente entre la barrera y el portero. Otro jugador cercano al lugar donde se encuentra el balón inicia un desmarque a la zona donde va a recibir el balón. En esta zona, el jugador receptor puede elegir entre disparar a portería o pasar el balón a un compañero que se encuentre en una posición más ventajosa para marcar.

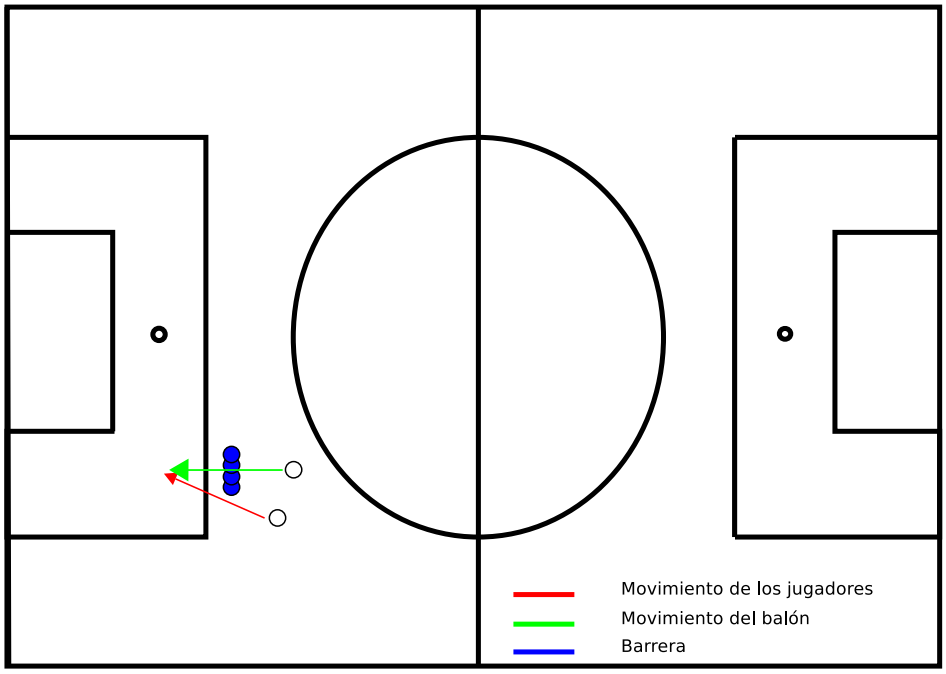


Figura 3.16: Falta al palo corto

Normalmente, esta última acción es más sorpresiva para la defensa, porque si existe algún fallo en la ejecución de la estrategia por parte del equipo atacante se generará una ventaja mayor que si existe un fallo en la estrategia al poner en juego la falta al palo largo.

3.1.6. Comportamientos relevantes

Como se ha especificado anteriormente, el fútbol es un deporte de equipo, pero esto no quiere decir que la componente individual no tenga un especial interés. Tanto es así que existen situaciones en las que un jugador se tiene que comportar de una manera determinada para que el equipo se beneficie. A continuación, se abordan algunas de estas situaciones.

Centro al área

Un centro al área es una acción en la que un jugador que se encuentra escorado en la banda, realiza un pase elevado hacia dentro del área rival con el objetivo de que otro compañero remate a portería. Es habitual que el centro al área se realice desde cualquiera de las bandas

[3]http://www.expertfootball.com/coaching/tactics.php Accedido el 10 de Septiembre de 2012.

y resulta más peligroso cuando más próximo a la línea de fondo del campo rival se encuentre el jugador que lo realiza. El centro al área es un recurso que un equipo no puede despreciar, puesto que esta acción trae consigo una serie de consecuencias como, por ejemplo, retrasar la línea defensiva del rival y, así, jugadores como los centrocampistas puedan modificar su actitud y volverse más ofensivos para entrar como segunda línea del ataque y aumentar el número de efectivos cerca del área contraria.

Un centro al área es similar a un córner, con la salvedad de que el balón está en movimiento y la zona del campo donde se realiza suele ser anterior a la esquina del campo. En la figura 3.17 pueden observarse las posibilidades que ofrece la acción de centrar al área.

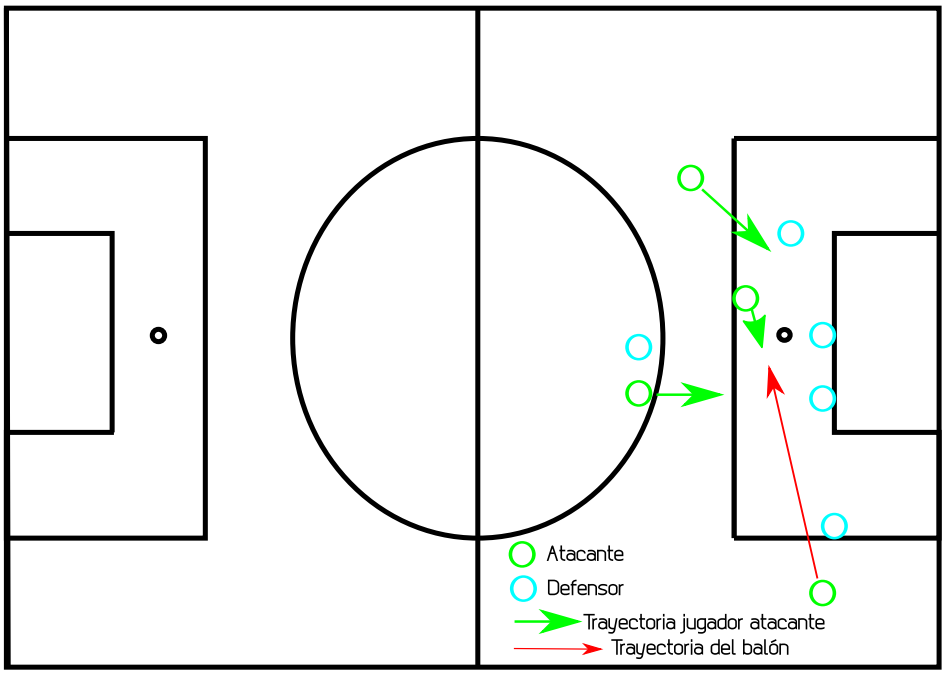


Figura 3.17: Centro al área

Pase de la muerte

A diferencia del centro al área, el pase de la muerte es una acción que un jugador realiza cuando se encuentra escorado en una parte del área rival. El pase de la muerte resulta ser un pase raso a la zona del área dónde un compañero se encuentra libre de marca para que pueda anotar un gol de una manera sencilla, puesto que, el portero del equipo defensor debe salir a tapar el espacio de la portería dónde se encuentra el jugador con la posesión del balón, dejando el resto de portería sin cubrir.

La sentencia *pase de la muerte* es una metáfora, ya que típicamente cuando se produce el equipo defensor encaja un gol. En la figura 3.18 puede observarse una situación en la que se produce el pase de la muerte.

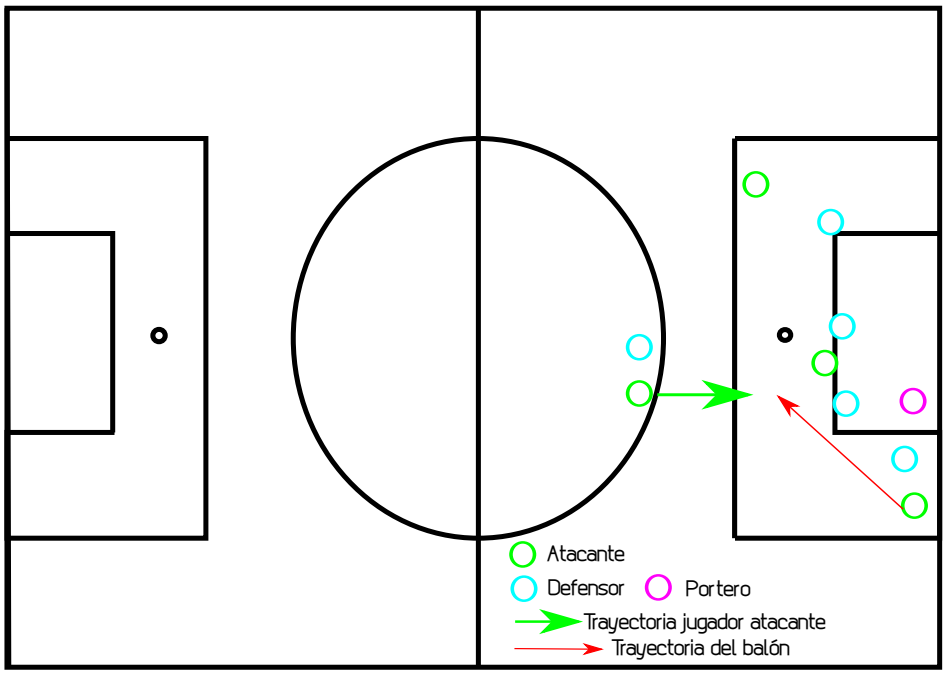


Figura 3.18: Pase de la muerte

Línea del fuera de juego

El **fuera de juego** es una situación en la que un jugador del equipo atacante se encuentra más adelantado que el último defensa en el momento que se efectúa un pase que lo tiene como destinatario. La **línea del fuera de juego** es una línea imaginaria que forma la defensa de un equipo con el objetivo de dejar en fuera de juego a los delanteros del equipo rival. Ambos conceptos se encuentran reflejados en la figura 3.19. Es de vital importancia que la línea del fuera de juego esté bien tirada porque, de lo contrario, un jugador del equipo contrario se puede beneficiar de este error y aprovecharlo para anotar un gol. Como se puede apreciar en la figura 3.20, la línea del fuera de juego está mal realizada lo que da ventaja al jugador del equipo contrario y le permite encarar a portería siendo el portero lo que lo aleja del gol. El hecho de que la línea del fuera de juego esté bien tirada corresponde a una perfecta sincronización por parte de la defensa de un equipo. Un solo jugador que no esté en sincronía con el resto de sus compañeros puede romper la línea del fuera de juego y otorgar ventaja al rival.

Robo crítico

El robo crítico es una situación del juego dónde un jugador se apodera de la posesión del balón en un momento en que la defensa del rival se encuentra en minoría con respecto de los atacantes del equipo atacante. Se denomina robo crítico porque se produce cuando el equipo que lo sufre está iniciando la transición defensa-ataque. En este contexto, sus jugadores se encuentran colocándose en sus puestos para afrontar un nuevo ataque y de repente deben

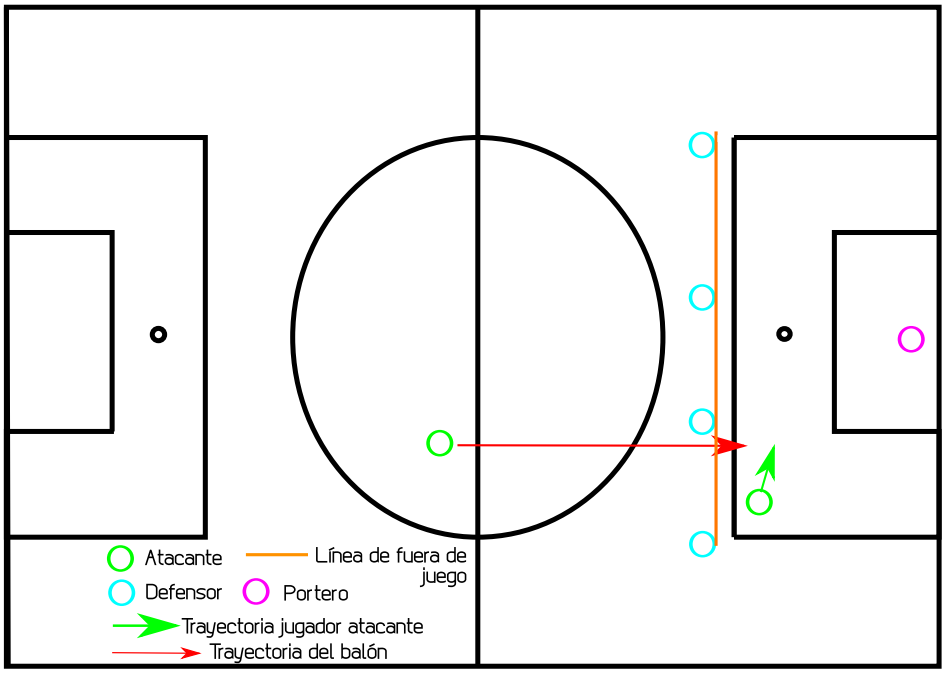


Figura 3.19: Fuera de juego

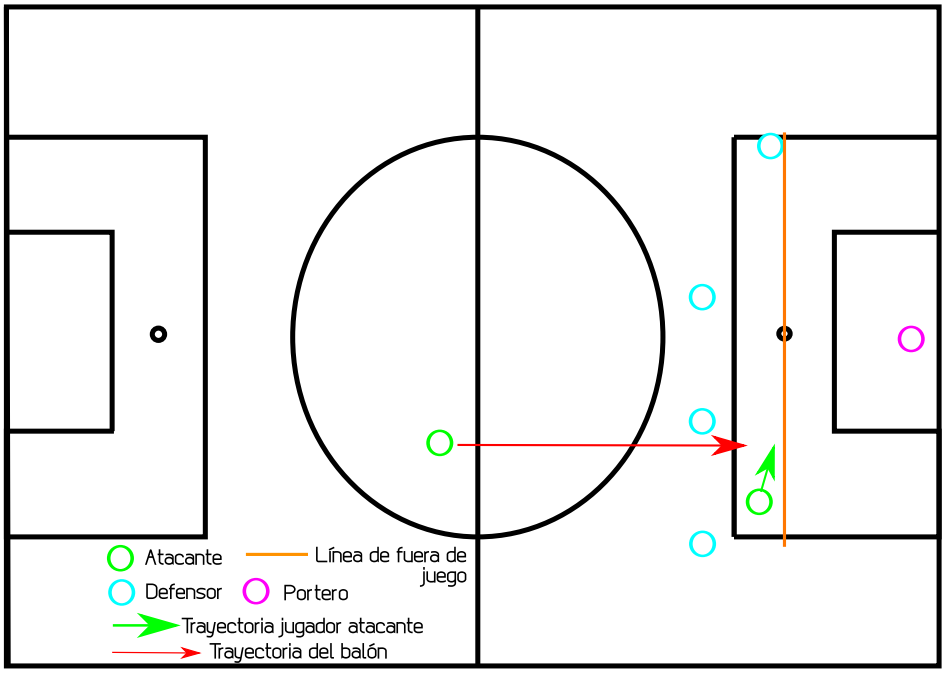


Figura 3.20: Línea de fuera de juego mal tirada

abandonar ese plan y adoptar de nuevo posiciones defensivas. El robo crítico es una situación peligrosa porque suele producirse en posiciones del terreno de juego cercanas al área, con lo que conseguir un gol será, presumiblemente, más sencillo. Para un equipo es importante realizar un número elevado de robos críticos y sufrirlos lo menor posible. En la figura 3.21 puede apreciarse un ejemplo de situación en la que se produce un robo crítico.

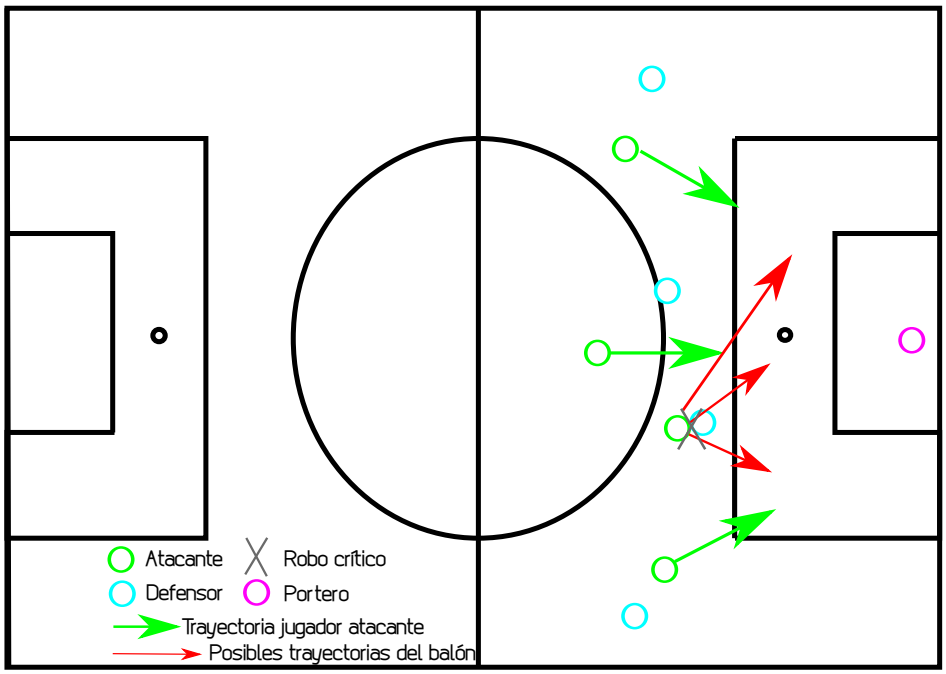


Figura 3.21: Ejemplo de robo crítico

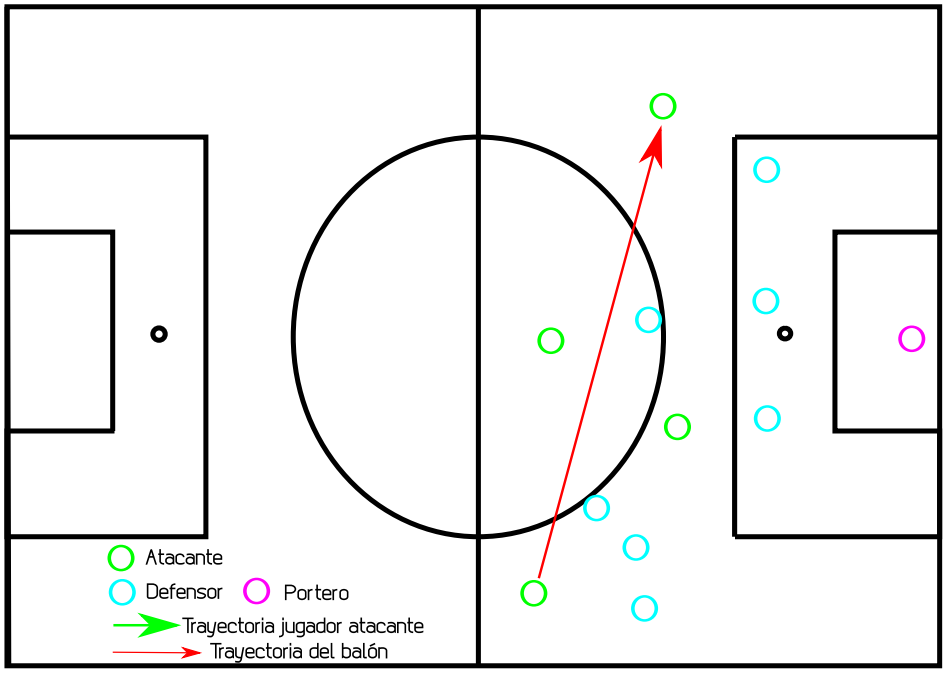


Figura 3.22: Ejemplo de cambio de banda

Cambio de banda

El *cambio de banda* o *cambio de juego* es una acción realizada por un jugador cuando la acumulación de jugadores en una zona del campo es superior a la cantidad de jugadores en otra zona. Se denomina cambio de banda porque el balón se desplaza de una banda a otra. Un cambio de banda permite evitar la presión que sufre un equipo, porque se envía el

balón a una zona del campo del campo dónde la acumulación de contrarios es menor. En la figura 3.22 puede apreciarse una situación en la que se produce un cambio de banda.

3.2. Inteligencia Artificial

3.2.1. Introducción

La **Inteligencia Artificial** es una de las ciencias más recientes [RN04]. El trabajo comenzó poco después de la Segunda Guerra Mundial y el término se acuñó en 1956. En la actualidad, la Inteligencia Artificial abarca una gran variedad de subcampos, que van desde áreas de propósito general, como el aprendizaje y la percepción, a otras más específicas como la construcción de sistemas que determinen la toma de decisiones de un futbolista en el terreno de juego.

La principal función de la Inteligencia Artificial no es sólo intentar comprender cómo piensa el ser humano, sino que va más allá y también se esfuerza en construir entidades inteligentes.

Es difícil dar una definición de Inteligencia Artificial de una manera concreta, pero, podemos entender la IA como la ciencia que establece los fundamentos necesarios para construir sistemas que realicen las siguientes acciones:

- **Piensan y actúan como humanos**. Estos sistemas miden su éxito en términos de la fidelidad en la forma de actuar de los humanos.
- **Piensan y actúan racionalmente**. Estos sistemas, en cambio, se basan en el concepto de racionalidad. Un sistema es racional si hace lo correcto, en función de su conocimiento.

3.2.2. Agentes Inteligentes

En el presente Proyecto Fin de Carrera se considera que la inteligencia está relacionada principalmente con acciones racionales.

Un **agente** es cualquier entidad capaz de percibir su medioambiente con la ayuda de **sensores** y actuar en ese medio utilizando **actuadores** (ver figura 3.23) [RN04].

Es importante definir el término **percepción** que indica que el agente puede recibir entradas en cualquier instante. Un agente tomará una decisión en un momento dado dependiendo de una **serie de percepciones** hasta ese momento.

En términos matemáticos se puede decir que el comportamiento de un agente viene determinado por una función f de un agente que proyecta percepciones en acciones.

$$f : P^* \rightarrow A$$

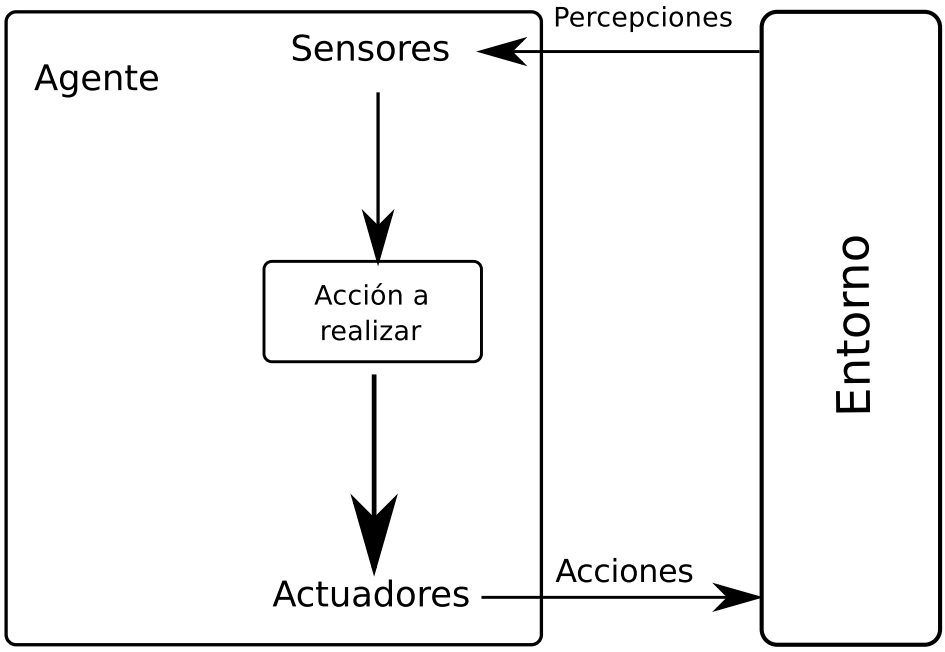


Figura 3.23: Arquitectura de un agente simple

Se propone el siguiente ejemplo con el objetivo de clarificar y afianzar el concepto de agente. Supongamos que existe un sistema que cuando detecta la presencia de una persona una habitación este encienda las luces de la misma. El sistema recibe como una percepción la presencia de una persona. Para obtener dicha percepción, es necesaria la existencia de un sensor, por ejemplo, un sensor de rayos infrarrojos que determine cuando una persona entra a una habitación. Al detectarse la presencia, será un actuador el que active las luces de una habitación mediante un circuito externo que realice esta función.

Un **agente racional** es aquel que hace lo correcto. Lo correcto es aquello que permite al agente obtener un resultado mejor que un comportamiento incorrecto.

Es necesario definir medidas de rendimiento que incluyen los criterios que determinen el éxito en el comportamiento del agente. La **racionalidad** [RN04] en un momento determinado depende de los siguientes aspectos:

- La medida de rendimiento que define el criterio de éxito.
- El conocimiento del medio en el que habita acumulado por el agente.
- Las acciones que el agente puede llevar a cabo.
- La serie de percepciones obtenidas por el agente hasta ese momento.

Un agente racional se puede definir de la siguiente manera [RN04]: *En cada posible secuencia de percepciones, un agente racional deberá emprender aquella acción que supuestamente maximice su medida de rendimiento, basándose en las evidencias aportadas por la secuencia de percepciones y en el conocimiento que el agente mantiene almacenado.* Un

agente racional es aquél que elige la acción **maximiza** su medida de rendimiento dada la secuencia de percepciones y el conocimiento que el agente tiene almacenado [RN04].

Un entorno de trabajo puede entenderse como el "problema" para el que el agente racional es la "solución". Los entornos de trabajo pueden clasificarse en función de las características de los mismos. En [RN04] puede encontrarse la siguiente clasificación en función de las propiedades del entorno.

- **Total o parcialmente observable**. Un entorno será totalmente observable si el agente es capaz de acceder a toda la información del medio. En cambio, un entorno será parcialmente aceptable si el agente sólo puede acceder a una parte de la información.
- **Deterministas o estocásticos**. Si el siguiente estado del medio está totalmente determinado por el estado actual y la acción ejecutada por el agente, entonces se dice que el entorno es determinista; de otra forma es estocástico.
- **Episódico o secuencial**. En un entorno episódico la elección de la acción por parte del agente depende sólo del episodio en sí mismo. Por otro lado, en entornos secuenciales, la decisión presente puede afectar a decisiones futuras. Los medios episódicos son más simples que los secuenciales porque el agente no necesita pensar con el tiempo.
- **Estático o dinámico**. Si el entorno puede cambiar cuando el agente está deliberando, entonces se dice que el entorno es dinámico para el agente; de otra forma se dice que es estático.
- **Discreto o continuo**. Un entorno discreto es aquel en que existe un número finito de acciones y percepciones en el mismo; en otro caso, se dice que el entorno es continuo.
- **Agente individual o multiagente**.

Para poder definir completamente a un agente es necesario una medida de rendimiento, actuadores, sensores y un entorno de trabajo. Estos cuatro conceptos se agrupan en el acrónimo REAS (Rendimiento, Entorno, Actuadores y Sensores) [RN04].

El trabajo de la IA es diseñar el programa del agente que implemente la función del agente que proyecta las percepciones en las acciones. Se asume que este programa se ejecutará en algún tipo de computador con sensores y actuadores, lo cual se conoce como arquitectura. Por tanto se puede definir un agente de la siguiente manera:

$$Agente = Arquitectura + Programa$$

A continuación se enumeran y detallan los distintos tipos de agentes en función su programa [RN04]:

Agentes reactivos simples

Es el tipo de agente más sencillo, puesto que selecciona las acciones a tomar dependiendo del conjunto de percepciones actuales, ignorando las percepciones históricas. Utilizan reglas del tipo **condición-acción** (si el agente recibe una percepción entonces actúa) (ver figura 3.24).

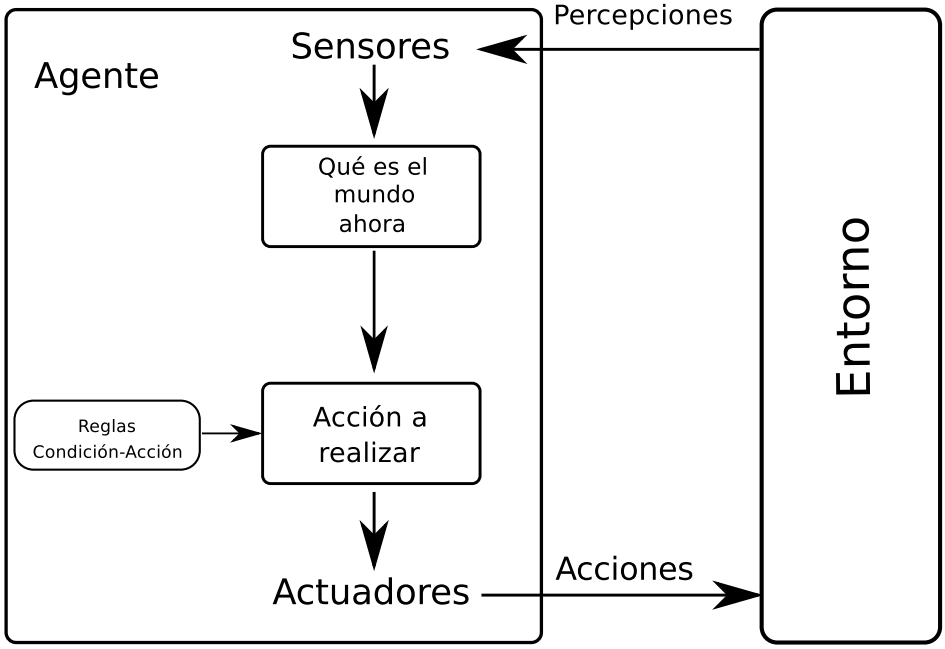


Figura 3.24: Diagrama de un agente reactivo simple

Agentes reactivos basados en modelos

El agente mantiene algún tipo de **estado interno** que dependa de la historia percibida y que, de ese modo, refleje por lo menos alguno de los aspectos no observables del mundo actual. Cuando el agente decida la acción que debe realizar tendrá que valorar la secuencia de estados almacenados, ver cómo evoluciona el mundo en función de la misma y conocer qué efectos pueden ocasionar sus acciones (ver figura 3.25).

Agentes basados en objetivos

Este tipo de agentes complementa a los del tipo anterior, puesto que, además del estado interno, este tipo de agente almacena información sobre una **meta** que describa las situaciones que son deseables. Un agente basado en objetivos almacena información del estado del mundo, así como un conjunto de objetivos que intenta alcanzar, y que es capaz de seleccionar la acción que lo guiará hacia la consecución de sus objetivos (ver figura 3.26).

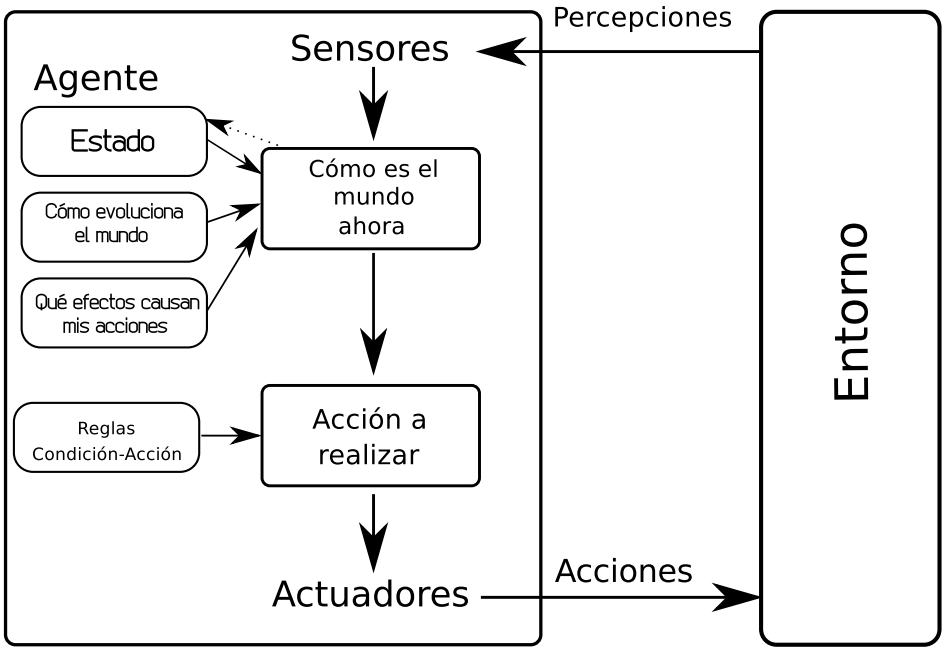


Figura 3.25: Diagrama de un agente reactivo basado en un modelo

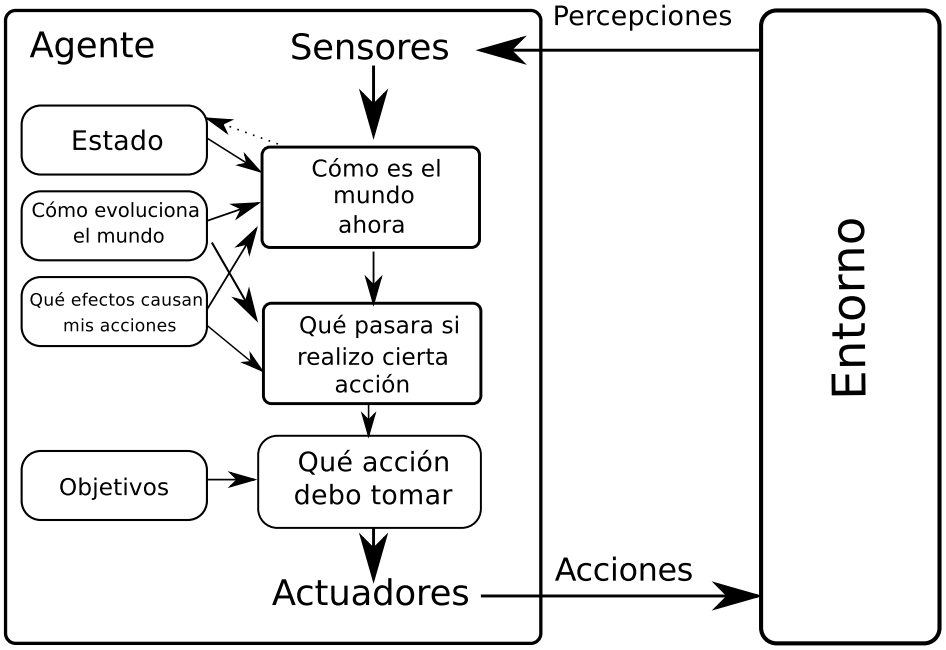


Figura 3.26: Diagrama de un agente reactivo basado en objetivos

Agentes basados en utilidad

Este tipo de agente, además de la información sobre la meta, hace uso de una **función de utilidad** que calcula sus preferencias. Después, selecciona la acción que le lleve a alcanzar la mayor utilidad esperada (ver figura 3.27).

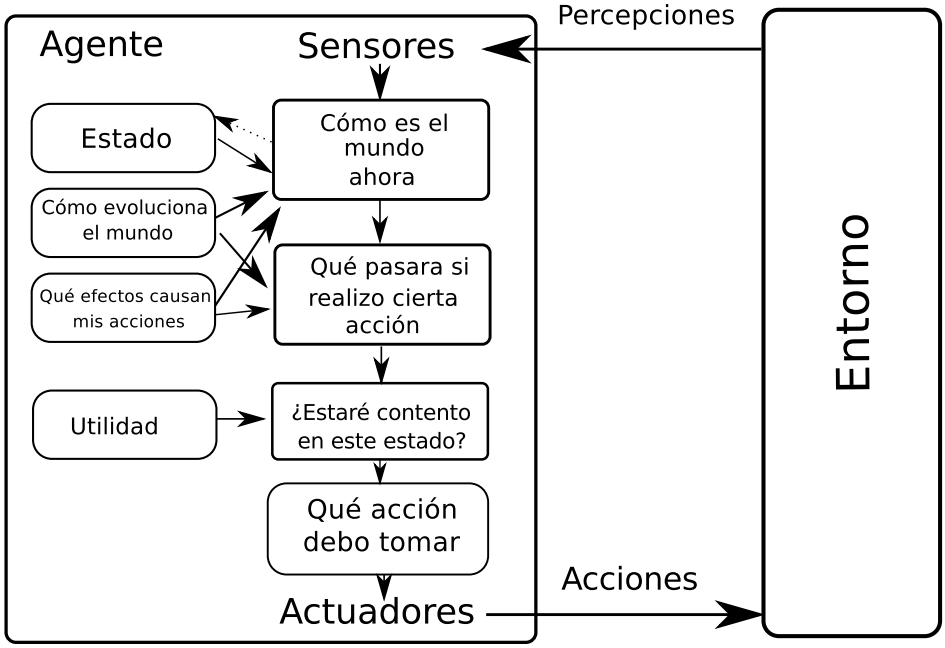


Figura 3.27: Diagrama de un agente reactivo basado en utilidad

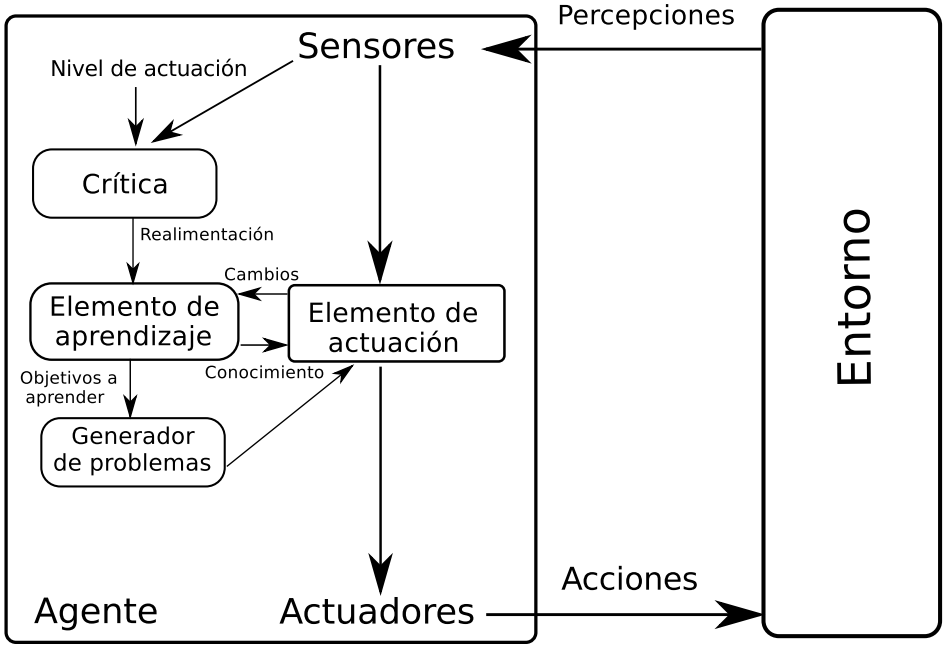


Figura 3.28: Diagrama de un agente que aprende

Agentes que aprenden

Un agente que aprende se puede dividir en cuatro componentes conceptuales, tal y como se muestra en la figura 3.28. Estos componentes son los siguientes:

- **Crítica**. Indica al elemento de aprendizaje qué tal lo está haciendo el agente con respecto a un nivel de actuación fijo. Este componente es necesario porque las percepcio-

nes por sí mismas no prevén un acierto del agente.

- **Elemento de aprendizaje**. Este componente es el encargado de hacer mejoras en el agente. Se realimenta con las críticas sobre la actuación del agente y determina cómo se debe modificar el elemento de actuación para proporcionar mejores resultados.
- **Elemento de actuación**. El objetivo de este componente es la selección de acciones externas.
- **Generador de problemas**. Es el encargado de sugerir acciones que guiarán al agente hacia situaciones nuevas e informativas.

Actualmente, el concepto de agente racional se aplica en muchos y diversos ámbitos. A continuación, se estudiarán diversas técnicas de Inteligencia Artificial con el objetivo de aprender conceptos y técnicas para poder aplicarlos al dominio del deporte en general y del fútbol en concreto.

3.2.3. Lógica difusa

La **lógica difusa** nace a mediados del siglo XX [Zad65] para ampliar el paradigma de la dualidad de valores de verdad de la lógica clásica.

El enfoque de la lógica clásica consiste en el uso de dos valores de verdad: verdadero o falso. Así pues, existen enunciados que pueden ser imprecisos en los que la lógica clásica bivalente no puede actuar de manera correcta. Por ejemplo, el enunciado "Nacho es alto" puede ser correcto en un contexto donde Nacho mide 1.80 y el resto de personas mide 1.70. Pero, sin embargo, el enunciado anterior deja de tener validez si Nacho mide 1.80 y el resto de personas mide 1.90.

Para solucionar problemas de este tipo surge la lógica difusa. La lógica difusa permite ampliar el número de valores de verdad al infinito. De manera que un enunciado puede ser evaluado con distintos valores de verdad. Por ejemplo, el enunciado anterior puede ser evaluado como: Nacho es alto con un valor de verdad del 0.7 y es muy alto con un valor de verdad de 0.3.

Conjuntos difusos y grado de pertenencia

La lógica difusa se fundamenta en clases que juegan un papel importante en el pensamiento humano en dominios como reconocimiento de patrones, comunicación de la información y abstracción [Zad65].

Sea X un espacio de puntos (objetos), con un elemento genérico de X denotado por x. Entonces $X = \{x\}$. Un **conjunto difuso** A en X se caracteriza por una **función de pertenencia** $f_A(x)$ que asocia a cada punto de X un número real en el intervalo [0, 1], donde el valor asociado por la función $f_A(x)$ en x se denomina grado de pertenencia de x al conjunto A. De tal manera que cuanto más se aproxima el valor de $f_A(x)$ a la unidad mayor es el grado

de pertenencia de x a A.

Sobre los conjuntos difusos es posible definir una serie de propiedades [Zad65]:

- Un conjunto difuso A es vacío si y solo si su función de pertenencia es igual a cero en X.
- Dos conjuntos difusos A y B son iguales, escrito $A = B$, si y solo si

$$f_A(x) = f_B(x) \forall x \in X$$

- El complemento de un conjunto difuso A escrito A' está definido como

$$f_{A'} = 1 - f_A$$

- Un conjunto A está **contenido** en B si y sólo si $f_A \leq f_B$

$$A \subset B \leftrightarrow f_A \leq f_B.$$

- **Unión**: La unión de dos conjuntos A y B con sus respectivas funciones de pertenencias $f_A(x)$ y $f_B(x)$ es un conjunto difuso C, escrito $C = A \cup B$, cuya función de pertenencia se relaciona con las de A y B mediante

$$f_C = Max[f_A(x), f_B(x)], x \in X$$

  o abreviado como:

$$f_C = f_A \vee f_B$$

  Es importante recalcar que $\cup$ posee la propiedad asociativa, esto es, $A \cup (B \cup C) = (A \cup B) \cup C$

- **Intersección**: La intersección de dos conjuntos difusos A y B con sus respectivas funciones $f_A(x)$ y $f_B(x)$ es un conjunto difuso C, escrito $C = A \cap B$, cuya función de pertenencia se relaciona con las de A y B mediante

$$f_C = Min[f_A(x), f_B(x)], x \in X$$

  o abreviado como:

$$f_C = f_A \wedge f_B$$

  Al igual que la $\cup$, $\cap$ también cumple la propiedad asociativa.

Además de la unión y de la intersección se pueden definir otras maneras de combinar conjuntos difusos y relacionarlos con otros. Los más importantes son:

- **Producto algebraico**: El producto algebraico de A y B escrito AB y está definido en términos de funciones de pertenencia de A y B por la relación

$$f_{AB} = f_A f_B$$

por tanto,

$$AB \subset A \cap B$$

- **Suma algebraica**: La suma algebraica de A y B escrita $A + B$ y está definida por

$$f_{A+B} = f_A + f_B$$

a condición que la suma $f_A + f_B$ sea menor o igual que la unidad. A diferencia del producto algebraico, la suma algebraica solo tiene sentido cuando la condición $f_A(x) + f_B(x) \leq 1$ se satisfaga para todo x.

- **Diferencia absoluta**: La diferencia absoluta de A y B escrita como $|A - B|$ y esta definido como

$$f_{|A-B|} = |f_A - f_B|$$

Además se han definido funciones con propiedades axiomáticas, que sirven como modelos de intersección y la unión:

- **T-conorma**: Es una operación binaria definida T:[0,1] x [0,1] $\rightarrow$ [0, 1] que puede expresarse así $f_A(x) OR f_B(x) = max[f_A(x), f_B(x)]$ y que cumple:
  1. **Propiedad conmutativa**: T(x, y) = T(y, x)
  2. **Propiedad asociativa**: T(x, T(y, z)) = T(T(x, y), z)
  3. **Elemento neutro**: T(x, 0) = T(0, x) = x
  4. **Monotonía**: si w $\leq$ x e y $\leq$ z entonces T(w, y) $\leq$ T(x, z)
- **T-norma**: Es una operación binaria S:[0,1] x [0,1] $\rightarrow$ [0, 1] que puede expresarse así $f_A(x) AND f_B(x) = min[f_A(x), f_B(x)]$ y que cumple:
  1. **Propiedad conmutativa**: S(x, y) = S(y, x)
  2. **Propiedad asociativa**: S(x, S(y, z)) = S(S(x, y), z)
  3. **Elemento neutro**: S(x, 1) = S(1, x) = x
  4. **Monotonía**: si w $\leq$ x e y $\leq$ z entonces S(w, y) $\leq$ S(x, z)

Reglas

En los sistemas basados en lógica difusa utilizan **reglas** para representar el conocimiento y aproximarse a la forma de raciocinio humano. Una regla puede entenderse como una sentencia condicional del tipo **si-entonces**. Una regla se compone de dos partes:

- Antecedente.
- Consecuente.

Por ejemplo, una regla podría ser la siguiente:

Si el cielo está nublado, entonces lloverá.

En la definición de esta regla podemos apreciar perfectamente el antecedente: "el cielo está nublado" y el consecuente: "lloverá". Tanto en el antecedente como el consecuente pueden utilizarse conectores disyuntivos, negativos o aditivos. Así, podemos construir reglas más complejas. Por ejemplo, la siguiente regla es más compleja:

Si el cielo está nublado y no existe anticiclón, entonces lloverá y hará frío.

Podemos observar que tanto el antecedente como el consecuente son un conjunto de sentencias unidas por diferentes conectores.

El uso de reglas permite trabajar con valores posibles de distintas variables. A cada una de estas variables se les denominará **variables lingüísticas** y al conjunto de los posibles valores que puede tomar se les denomina **etiquetas lingüísticas**. En la regla anterior, vemos que las variables son el estado del cielo, la existencia de anticiclón, la posibilidad de lluvia y el clima. Las dos primeras se definen como variables de entrada al sistema, mientras que las dos últimas son variables de salida. Los sistemas que están formados de esta manera se denominas sistemas de reglas.

Este tipo de sistemas se suele emplear con la ayuda de un **experto** en diferentes dominios. El experto tendrá en cuenta el número de variables que influyen en el dominio de aplicación. Por otra parte, dividirá el conjunto de valores de una variable en tantas etiquetas lingüísticas como considere oportunas. Por último, definirá una serie de reglas que reunirán variables y etiquetas lingüísticas para representar el conocimiento.

Proceso

El proceso que sigue un sistema de reglas basado en lógica difusa es el siguiente [Zad76] [Zad65]:

1. **Borrosificación** (fuzzyfication): Conversión del valor numérico de una variable a la correspondiente etiqueta lingüística de dicha variable.
2. **Inferencia lógica**: Aplicación de las reglas para obtener el grado de pertenencia de las variables de salida de las reglas a cada uno de los conjuntos difusos.
3. **Desborrosificación** (defuzzyfication): Conversión de la etiqueta lingüística de la variable de salida a un valor numérico. Este proceso es el inverso de la borrosificación.

El control difuso

La lógica difusa fue ideada para la representación del conocimiento. Sin embargo, ha adquirido cada vez más importancia en el control de procesos, primero en Japón (1987) y luego en el resto del mundo. Actualmente, la lógica difusa se utiliza en diferentes campos, como pueden ser los electrodomésticos (lavavajillas que evalúan la suciedad de los platos), vehículos (el sistema de frenada del tren bala japonés, sistema de aparcamiento), robótica, etcétera.

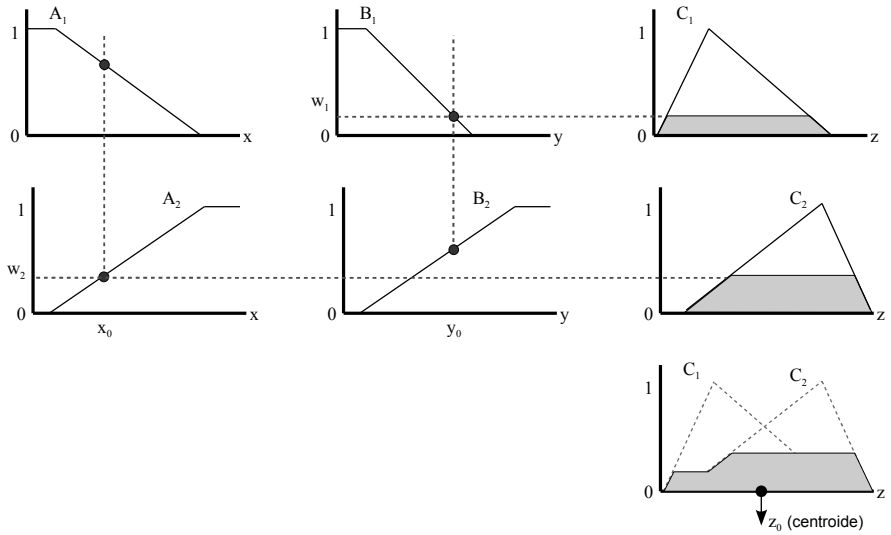


Figura 3.29: Ejemplo de un sistema de evaluación Mamdani

Existen dos métodos de inferencia relevantes que son aplicados en el control automático de procesos:

- **Mamdani** [Mam74]. El método Mamdani se propuso para controlar engranajes mediante un conjunto de reglas lingüisticas obtenidas de la experiencia de operadores humanos. El método de Mamdani es el más adecuado para capturar el conocimiento de un experto, ya que permite describir dicho conocimiento de una manera sencilla e intuitiva. En este método de inferencia una regla tiene la forma *Si x es A e y es B entonces z es C*. Dónde *A*, *B* y *C* son etiquetas lingüísticas de conjuntos difusos pertenecientes a una variable difusa. Un ejemplo de regla en este método es la siguiente *Si estatura = 1.90 entonces tipo de persona = alta*. En la figura 3.29 se detalla gráficamente un sistema que utiliza un método de inferencia de tipo Mamdani. En dicha gráfica se observa cómo el sistema se compone de dos reglas cuyas variables difusas son *X* e *Y*; la salida es otra variable difusa *Z*. El sistema de la gráfica utiliza un sistema de inferencia *MIN-MAX*, es decir, se aplica el operador *T-norma* en la obtención de los conjuntos difusos de salida para cada una de las reglas y se aplica el operador *T-conorma* para la consecución del conjunto difuso de salida a desborrosificar.

- **Takagi-Sugeno** [TS85]. Este método surge debido a la necesidad de desarrollar un sistema para la generación de reglas a partir de conjunto de datos de entrada-salida dado. A diferencia del método anterior, una regla en este tipo de método tiene la forma *Si x es A e y es B entonces* $z = f(x, y)$. Dónde *A* y *B* son etiquetas lingüísticas de conjuntos difusos pertenecientes a una variable difusa y $z = f(x, y)$ es una función. Un ejemplo de regla en este método puede ser **Si X es pequeño entonces Y = 0.3X + 4.7**. El método de Takagi-Sugeno es más eficiente computacionalmente y funciona mejor en sistemas dinámicos y no lineares. El poder expresivo del anterior método se pierde en este otro.

A continuación, se propone un ejemplo de un sistema de este tipo Mamdani al fútbol.

Supongamos que existe un sistema para determinar la fuerza que debe aplicar al balón al disparar un jugador, dependiendo de la velocidad del viento en contra y de la distancia a portería. El sistema, por tanto, se compone de tres variables: i) *la velocidad del viento en contra*, ii) *la distancia a portería del jugador que posee el balón* (serán dos variables de entrada al sistema) y *la fuerza del disparo del jugador* (será una variable de salida del sistema).

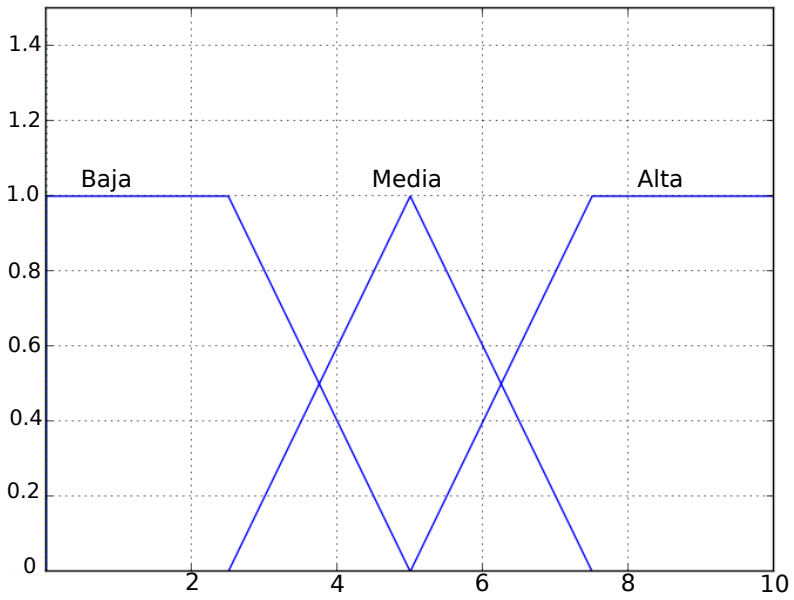


Figura 3.30: Variable difusa *velocidad del viento*

Para dotar de realismo al ejemplo, supongamos que un experto ha determinado que la variable velocidad del viento se divide en tres conjuntos. Cada uno de esos conjuntos difusos tiene una etiqueta lingüística asociada (*baja, media, alta*) (ver figura 3.30).

La variable *distancia a portería* se divide en cinco conjuntos (*muy cerca, cerca, media, lejos, muy lejos*) (ver figura 3.31). La variable de salida *fuerza del disparo* está compuesta por tres conjuntos (*baja, media, alta*)(ver figura 3.32).

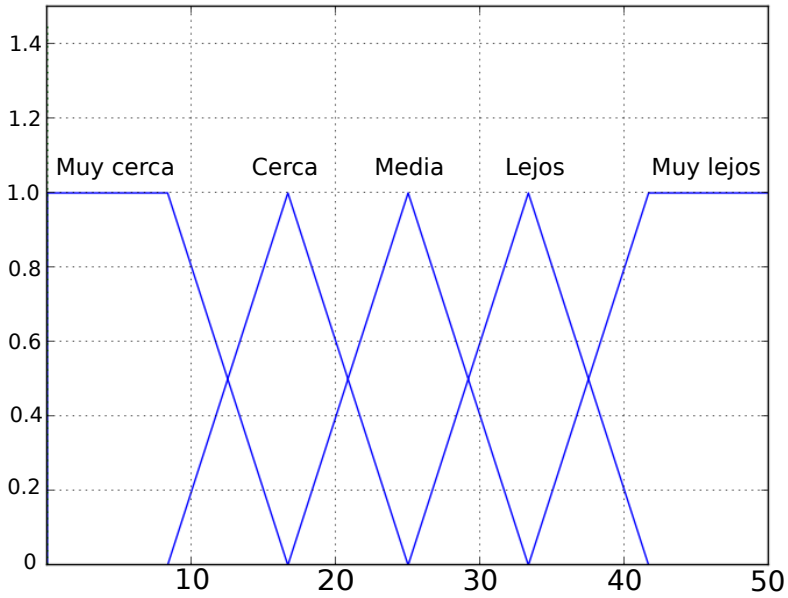


Figura 3.31: Variable difusa *distancia a portería*

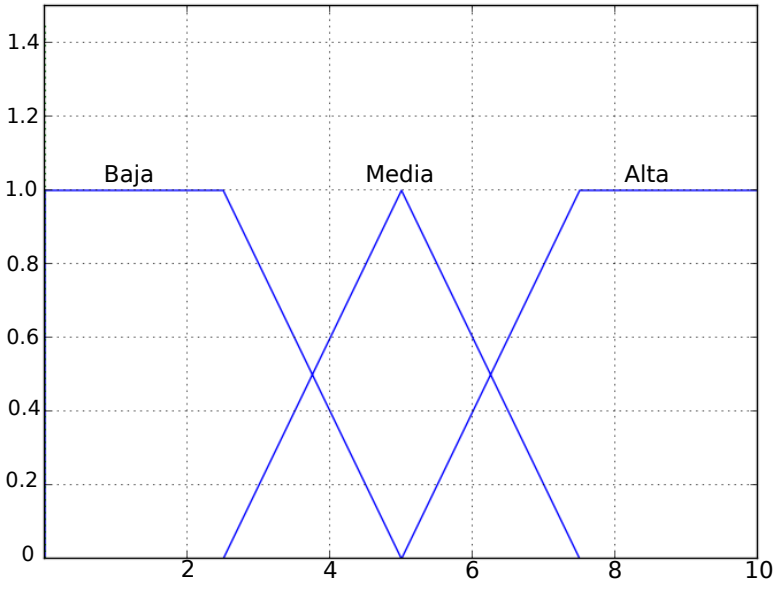


Figura 3.32: Variable difusa *fuerza del disparo*

Con objetivo de simplificar el problema se dispone de dos reglas:

- Si la *velocidad del viento* es media y *la distancia a portería* es media, entonces *la fuerza del disparo* es alta.
- Si la velocidad del viento es alta y la distancia a portería es cerca, entonces la fuerza del disparo es media.

El sistema debe determinar el valor numérico de la variable de salida acción a tomar con los valores de entrada:

- *Distancia a portería del jugador* = 20 m.
- *Velocidad del viento en contra* = 6Km/h.

El **primer paso** es la borrosificación de las entradas. El valor numérico *Distancia = 20 m.* se corresponde con un grado de pertenencia 0.6 para el conjunto difuso *distancia cerca* y con un grado de pertenencia 0.4 para el conjunto difuso *distancia media* como se puede apreciar en la figura 3.33. El valor numérico *velocidad del viento = 6km/h.* se corresponde con un grado de pertenencia 0.6 al conjunto difuso *velocidad media* y con un grado de pertenencia 0.4 al conjunto difuso *velocidad alta* como se puede observar en la figura 3.34.

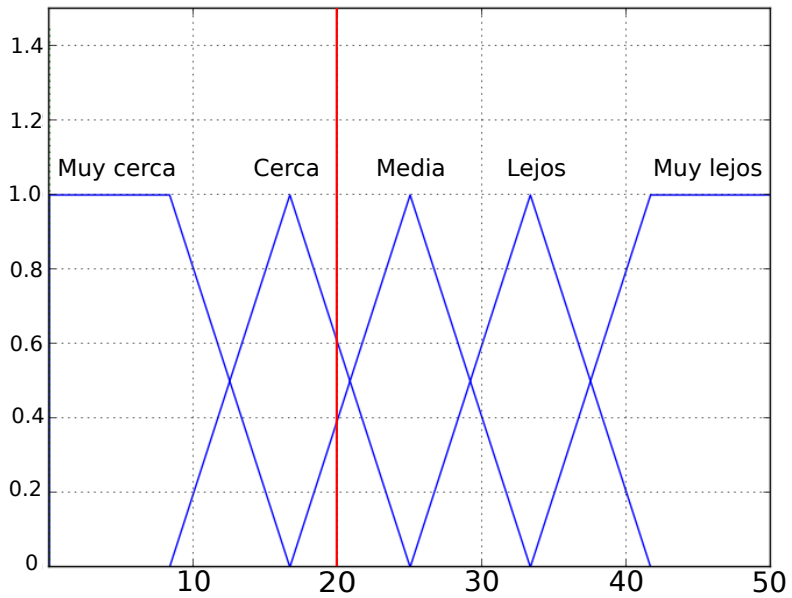


Figura 3.33: Borrosificación de la variable distancia a portería

El **segundo paso** es obtener la conclusión en cada regla. A partir del valor del antecedente obtenido en el primer paso y del conjunto borroso del consecuente, aplicamos un operador borroso de implicación. A continuación se aplicará el operador borroso *T-norma* que realiza la operación *AND* de dos conjuntos difusos.

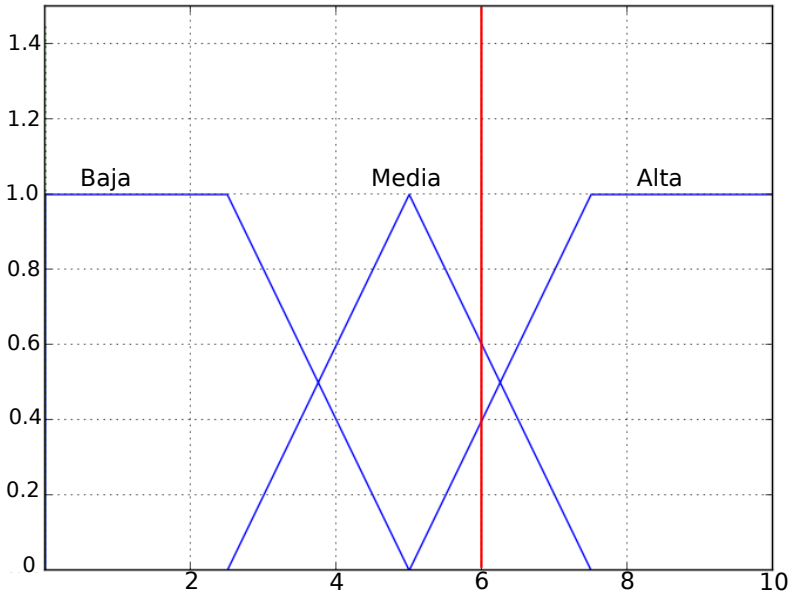


Figura 3.34: Borrosificación de la variable velocidad del viento

Si la *velocidad del viento* es media, y la *distancia a portería* es media entonces la *fuerza del disparo* es alta.

Se aplica el operador *T-norma* y tenemos que el grado de pertenencia al conjunto *fuerza de disparo alta* de la variable de salida fuerza del disparo es la siguiente: $\min\{0.6, 0.4\} = 0.4$. Gráficamente este paso puede apreciarse en la figura 3.35.

Si la *velocidad del viento* es alta y la *distancia a portería* es cerca entonces *la fuerza del disparo* es media.

Se opera de la misma manera que en la regla anterior y obtenemos que el grado de pertenencia al conjunto *fuerza de disparo media* de la variable de salida fuerza de disparo es la siguiente: $\min\{0.4, 0.6\} = 0.4$. (ver figura 3.36).

Una vez se han terminado de evaluar todas las reglas se combinan en un único conjunto borroso utilizando un operador de agregación borrosa, esto puede observarse en la figura 3.37.

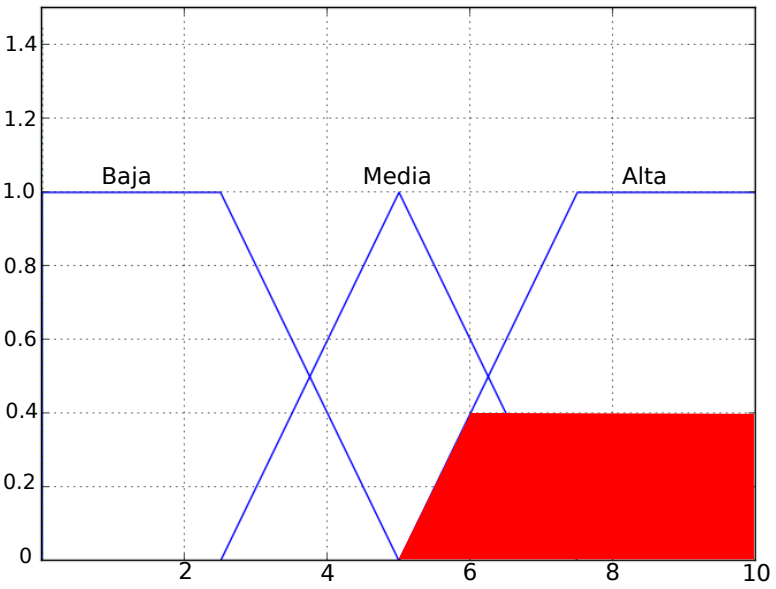


Figura 3.35: Aplicación de la regla 1

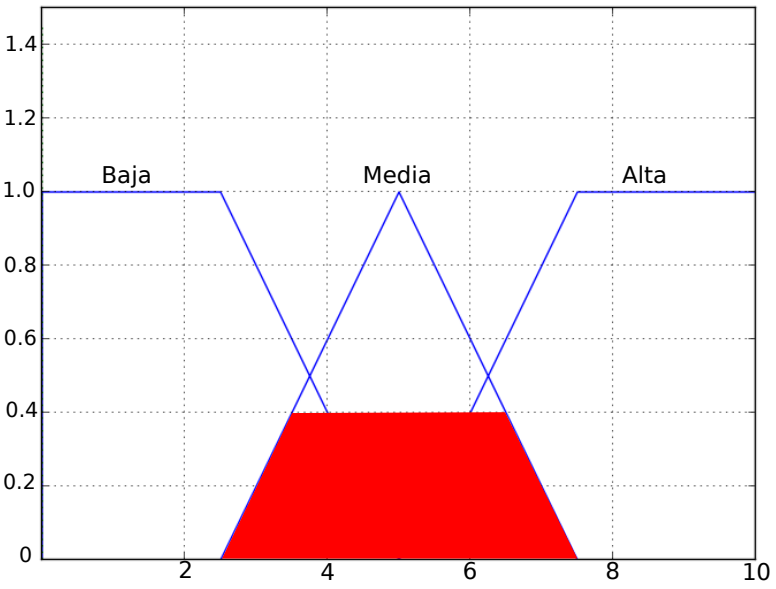


Figura 3.36: Aplicación de la regla 2

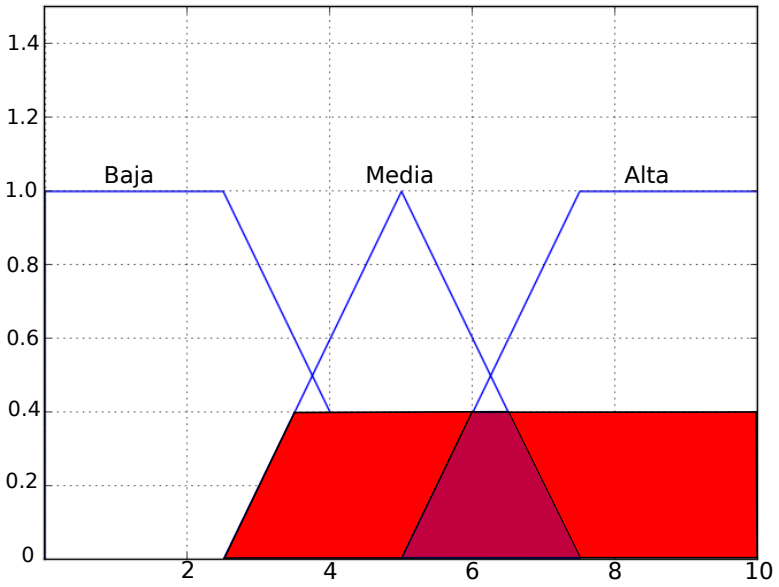


Figura 3.37: Combinación de la evaluación de las reglas

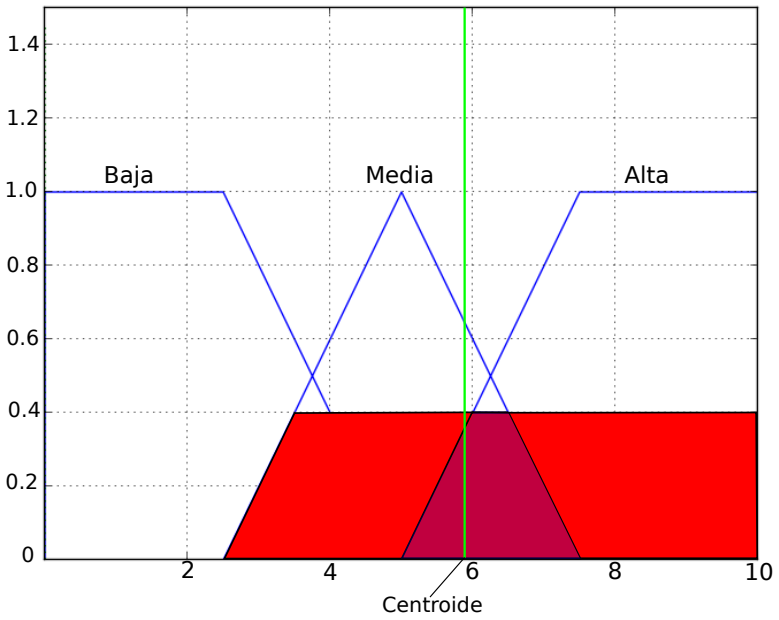


Figura 3.38: Desborrosificación

El tercer y **último paso** es la desborrisificación que, consiste en obtener un valor numérico a partir del conjunto borroso obtenido en el paso anterior. Existen diferentes métodos para llevar a cabo la desborrosificación. En este ejemplo se aplicará el método de centro de sumas.

$$z^* = \frac{\int f_C(z) z \mathrm{d}z}{\int f_C(z) \mathrm{d}z}$$

Si aplicamos este método de desborrosificación tenemos que el valor numérico para la variable difusa de salida fuerza de disparo toma el valor de 5.809, como puede apreciarse en la figura 3.38.

3.3. Sistemas artificiales aplicados al fútbol

Los deportes no están exentos de la presencia de las tecnologías. Un ejemplo es el fútbol. En este contexto, existen diferentes técnicas que se aplican en el fútbol moderno como las que se enumeran a continuación:

- Sistemas de tracking que permiten identificar lo que ha recorrido un jugador en el campo.
- Cámaras personalizadas para cada uno de los jugadores.
- Sistemas que permiten simular partidos.
- Aplicaciones que permiten definir y almacenar tácticas.

El principal cometido de estos sistemas es el de obtener información con el objetivo de encontrar posibles errores en los comportamientos de los jugadores, tácticas, etcétera. Esta información puede ser utilizada para identificar tendencias de los jugadores y aprovechar mejor sus virtudes y corregir los fallos de los mismos. Por ejemplo, si un jugador tiende a ser más ofensivo de lo que se le exige, puede ser que el entrenador se esté equivocando en la posición en la que coloca a dicho jugador en el campo y deba cambiar su rol en el equipo.

En este apartado se describen algunos sistemas artificiales aplicados al fútbol. El objetivo de esta sección consiste en y discutir cómo fútbol y tecnología convergen en ciertos puntos.

3.3.1. Sistemas de tracking

La palabra *tracking* se puede traducir del inglés como *seguimiento*. El concepto de seguimiento queda definido según la RAE [4] como acción y efecto de seguir, y seguir queda definido como [5] ir después o detrás de alguien. Por tanto, un **sistema de *tracking*** puede definirse como un conjunto de reglas o principios basados en ir detrás de algo o alguien. El principal objetivo de los sistemas de tracking es el de automatizar el seguimiento de algo o alguien con objetivos como los que se presentan seguidamente:

- realización de estudios sobre los datos obtenidos por parte del sistema.

[4] http://lema.rae.es/drae/?val=seguimiento Accedido el 6 de Septiembre de 2012.
[5] http://lema.rae.es/drae/?val=seguir Accedido el 6 de Septiembre de 2012.

- análisis de comportamientos.
- predicción de acciones basadas en la información obtenida por el sistema.

Actualmente, las tecnologías para llevar a cabo sistemas de *tracking* más implantadas son las que se describen a continuación:

Sistemas de tracking mediante cámaras

Sistemas de seguimiento mediante cámaras (ver figura 3.39) que, gracias a calibraciones, son capaces de seguir marcas que representan a la persona, animal u objeto a seguir. Mediante las marcas, el sistema de seguimiento puede establecer un modelo que será objeto de estudio en etapas posteriores, y que gracias a él se pueden determinar ciertos datos [BMM+07, XLO04].

La ventaja de estos sistemas es su adaptabilidad a cualquier situación. Gracias a ellos y a las imágenes obtenidas por las cámaras se obtiene la posición de cualquier marca y se almacena junto al tiempo en que se ha tomado la muestra, de manera que resulta sencillo hacer reconstrucciones con esos datos.

En cambio, el principal inconveniente es el número de cámaras a utilizar para que no existan *puntos muertos* en el seguimiento de la escena. La existencia de estos puntos muertos dificultaría la recreación de la escena, puesto que algunas localizaciones estarían ausentes de seguimiento.

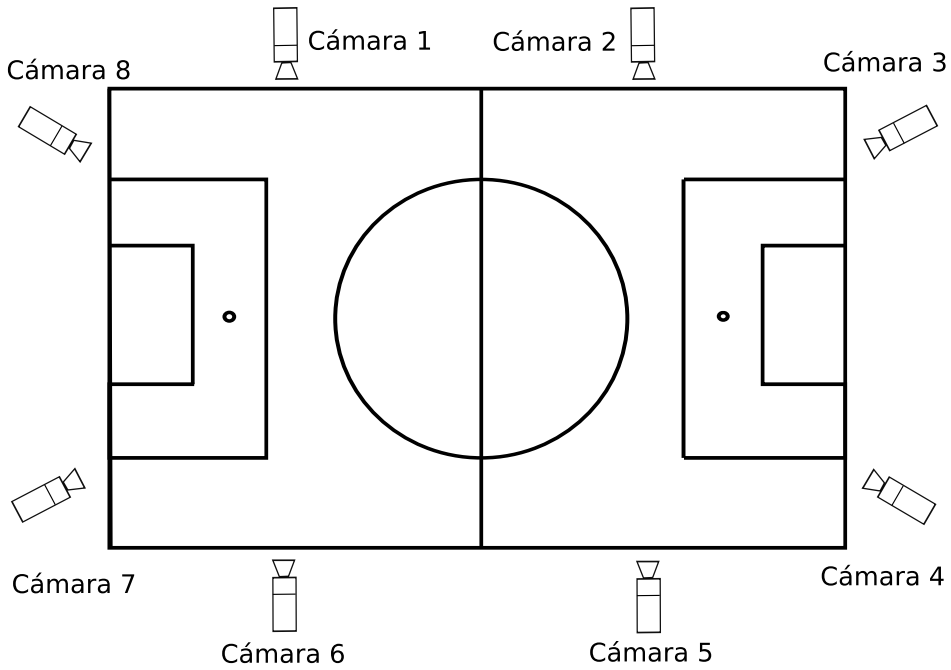


Figura 3.39: Ejemplo Sistema de tracking con cámaras aplicado al fútbol. [XLO04]

En estos sistemas se tratan varios aspectos donde la informática tiene bastante importancia:

- Procesamiento de imágenes y visión por computador.
- Inteligencia Artificial.
- Comunicación entre diferentes partes del sistema.

En general, la mayoría de los sistemas funcionan de la siguiente manera:

1. Gracias a las cámaras se obtiene la posición de una marca. Esta marca puede representar a cualquier cosa que se quiera medir/estudiar. La posición de la marca puede ser relativa con respecto del campo de seguimiento de la cámara o absoluta con respecto al entorno.
2. Si la posición es relativa con respecto al campo de seguimiento de la cámara, una central de cálculo se encargará de obtener la posición absoluta del entorno monitorizado.
3. La posición absoluta se almacena en un soporte digital, normalmente en una base de datos.
4. La información almacenada puede utilizarse para diferentes tareas como la recreación de escenas o incluso para el análisis de datos para la inferencia de nueva información.

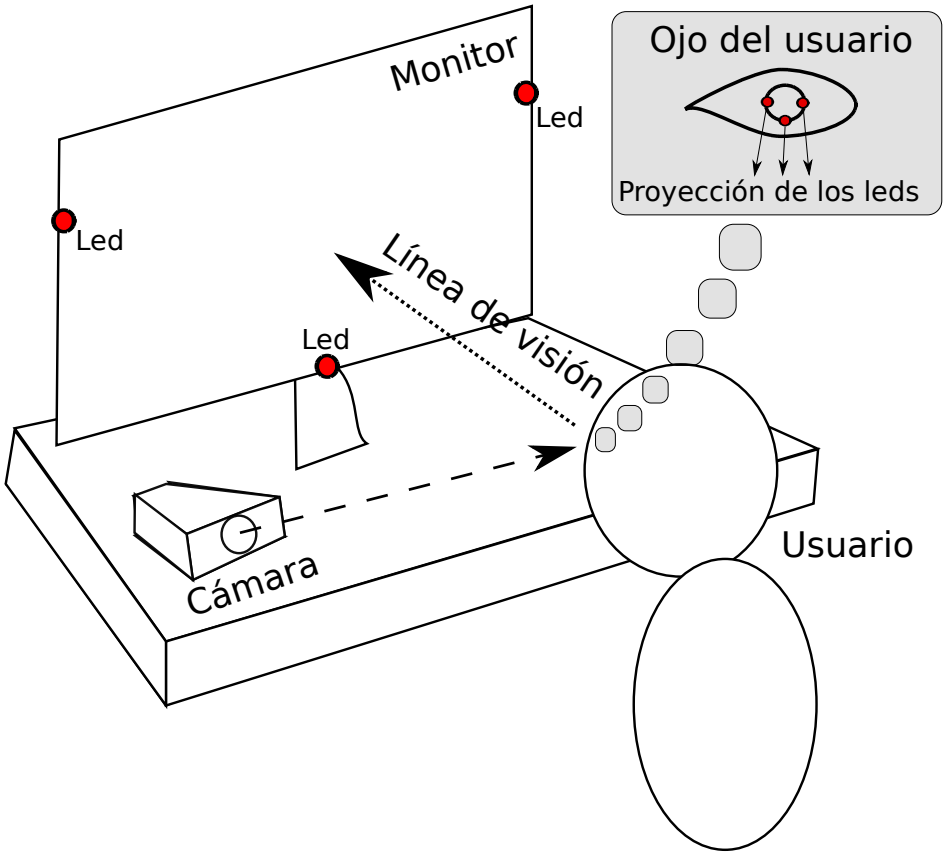


Figura 3.40: Ejemplo de sistema de eye-tracking

Dentro del tipo de sistemas de tracking basados en cámaras, se encuentran los sistemas de *eye-tracking* [Nam01]. Los **sistemas de eye-traking** se utilizan para determinar el movimiento de los ojos de una persona. Para ello disponen de una cámara que monitoriza los movimientos de cada uno de los ojos con objeto de determinar la posición de los ojos en cada instante (ver figura 3.40) para su posterior estudio.

Un sistema de eye-tracking puede utilizarse de dos formas: **pasivamente o activamente**:

- De forma pasiva, un sistema de eye-tracking monitoriza el movimiento de los ojos con el objetivo de recoger datos del movimiento de éstos para posteriormente elaborar análisis.
- De forma activa, este tipo de sistemas provee soporte necesario para la realización de acciones con el movimiento de los ojos.

El funcionamiento de estos sistemas se basan en el siguiente esquema. Una proyección de luz se releja en el ojo. Después, una cámara determina la diferencia entre la reflexión de la pupila y puntos de referencia conocidos para conocer qué está mirando el usuario. Este planteamiento involucra los siguientes métodos [Nam01]:

1. Calibración, permite al sistema de eye-tracking conocer la fisionomía de la persona.
2. Iluminar el ojo para localizar el centro de la pupila.
3. Medir la localización del centro de la pupila.
4. Localizar la posición relativa de la reflexión de la córnea.
5. Calcular la dirección de la línea de visión.

Los sistemas de eye-tracking tienen multitud de usos:

- Aplicaciones científicas que permitan monitorizar que pilotos o luces se encuentran encendidas en salas de control.
- Ayuda a investigadores de mercado que estudian la atención del usuario en diferentes tipos de productos, publicidad, etcétera.
- Sistemas de ayuda a personas incapacitadas que solo pueden utilizar el movimiento de sus ojos para indicar la acción a realizar.
- Sistemas de evaluación de interfaces de usuario.

Sistemas de tracking mediante GPS

Sistemas de seguimiento mediante un receptor GPS que indica la distancia que lo separa de los satélites, para así por medio de cálculo matemático (ver figura 3.41 [6]) determinar

[6] http://sabia.tic.udc.es/gc/Contenidos %20adicionales/trabajos/Peliculas/Mocap/anexo2-1.htm Accedido el 6 de Septiembre de 2012.

la posición y altura a la que se encuentra el receptor. Si almacenamos el conjunto de las posiciones durante un tiempo tendremos la ruta seguida por el receptor.

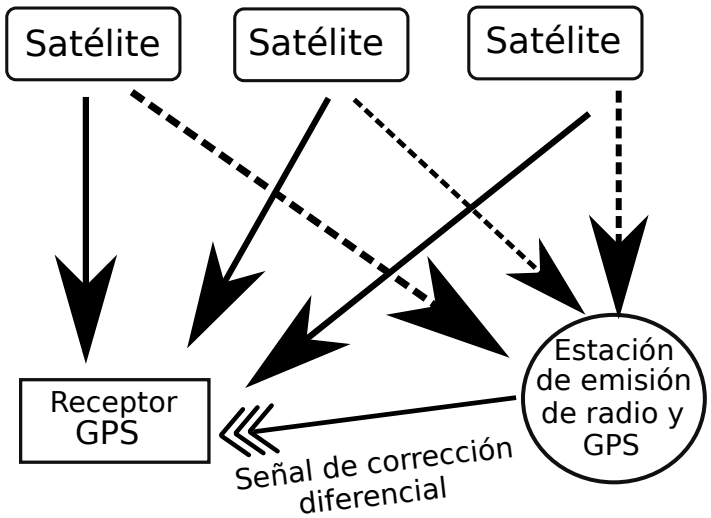


Figura 3.41: Funcionamiento de un GPS diferencial.

El GPS tiene la finalidad de determinar las coordenadas espaciales de puntos respecto de un sistema de referencia. La obtención de los puntos se basa en la determinación de las distancias a cuatro satélites de coordenadas conocidas. Las distancias se obtienen a partir de las señales emitidas por los satélites, las que son recibidas por receptores. Las coordenadas de los satélites son provistas al receptor del sistema [HMN05].

La ventaja de estos sistemas es que, con un único dispositivo, se puede obtener datos respecto a posiciones. Otra de las ventajas es que no existe la presencia de puntos muertos, como pasaba en el anterior sistema de cámaras.

El inconveniente de estos sistemas es que lo que se quiera monitorizar tendrá que ir equipado con el receptor GPS, y esto en algunas situaciones determinadas puede resultar algo incómodo.

Sistemas de tracking en el fútbol

Actualmente, en el fútbol existen diferentes sistemas artificiales que utilizan sistemas de seguimiento, normalmente basados en cámaras:

- En [XLO04] se propone una arquitectura que toma como entrada datos en forma de vídeo obtenidos mediante las cámaras durante un partido. La salida de dicho sistema es el conjunto de las posiciones en tiempo real de los jugadores de fútbol durante un partido. Para ello, se utilizan ocho cámaras estáticamente calibradas alrededor del

estadio. Estas ocho cámaras están conectadas mediante fibra óptica a un servidor que, haciendo uso de la capa *OSI IP*, envía la información al tracker que se encarga de generar un fichero *XML* con la información correspondiente al estado del encuentro. Este fichero se interpreta con el objetivo de obtener como resultado el conjunto de trayectorias seguidas tanto por los jugadores como por el balón.

- En [BGB+07] se presenta el proyecto ASPOGAMO, que determina las posiciones y trayectorias seguidas por los jugadores desde la visión ofrecida por la televisión. Al carecer de un sistema basado en comunicación de cámaras con tracker, se utiliza la estimación probabilística. El proceso se lleva a cabo mediante las soluciones a tres subproblemas:
  - Estimar la posición de la cámara y el nivel de *zoom* de esta.
  - Seguimiento de las trayectorias de los jugadores.
  - Detección de los jugadores en una oclusión.

  En [BGB+07] se pueden apreciar los resultados en detalle. En general, se obtienen buenos resultados en la clasificación de los jugadores. Sin embargo, existen ciertos casos (como la presencia de sombras de los propios jugadores) que hacen más difícil esta clasificación.
- En [MTMH11] presentan una aplicación multicámara capaz de procesar imágenes de alta resolución y extraer características basadas en patrones de color. En el artículo determinan que, debido a que los jugadores al componerse de diferentes colores, pueden aplicarse modelos de mezcla gaussianos para analizar imágenes y vídeo.

  El sistema obtiene la información gracias a las cámaras y mediante la detección de patrones de colores identificar a los jugadores de uno y otro equipo.

3.3.2. Simuladores

Una **simulación**, según [Sha75], es *el proceso de diseño de un modelo de un sistema real y la realización de experimentos con este modelo con el propósito de comprender el comportamiento del sistema o la evaluación de varias estrategias para el funcionamiento del sistema.* Un **simulador** se define como un *aparato que reproduce el comportamiento de un sistema en determinadas condiciones, aplicado generalmente para el entrenamiento de quienes deben manejar dicho sistema* [7].

En la actualidad existen diferentes y diversos simuladores, como pueden ser los siguientes:

- Simuladores de deportes.
- Simuladores de vuelo.

[7] http://lema.rae.es/drae/?val=simulador Accedido el 6 de Septiembre de 2012.

- Simuladores de coches.
- Simuladores de sensaciones.

En este apartado se estudian simuladores que se basen en el deporte, más concretamente en el fútbol.

Los simuladores de fútbol se pueden clasificar de la siguiente manera:

- Simuladores en los que sea el usuario el que especifique el comportamiento de los jugadores previamente a una simulación. Por ejemplo, *Robocup Soccer*, el cual se detalla en el apartado 3.3.2.

  En este tipo de simuladores, el uso de técnicas de Inteligencia Artificial cobra mayor relevancia puesto que el usuario debe especificar modelos de conocimiento para los jugadores del equipo. En estos simuladores se determinarán las acciones que los integrantes de un equipo llevarán a cabo en base a ciertos estímulos simulados como, por ejemplo, ver a un compañero o escuchar órdenes del entrenador.

- Simuladores en los que el usuario tome las decisiones en tiempo real en la propia simulación. Por ejemplo, el simulador *Eat The Wistle* que se detalla en el apartado 3.3.2.

  A diferencia de los simuladores del punto anterior, la habilidad del usuario a la hora de controlar los movimientos de los jugadores dentro del terreno de juego es lo que tendrá mayor importancia.

- Simuladores que mezclen las funcionalidades de los dos tipos anteriores. En este tipo de simuladores el usuario o bien puede definir estrategias y tácticas antes de un encuentro o bien puede intervenir de forma directa controlando manualmente a los jugadores.

  A este tipo de simuladores también pertenecen aquellos en los que se puede definir tácticas para situaciones a balón parado o forma de jugar del equipo (ofensiva o defensivamente).

A continuación se describen diferentes simuladores con el objetivo de realizar estudios sobre las funcionalidades de cada uno, haciendo especial hincapié en el simulador del proyecto RoboCup.

Robocup

El proyecto **RoboCup** [8] se dedica a fomentar diferentes competiciones entre robots, ya sean virtuales o reales [KAK+97]. El objetivo inicial de este proyecto, planteado para el año 2050, es diseñar un equipo de robots humanoides totalmente autónomos que a mediados del siglo XXI sean capaces de ganar (con las reglas oficiales de la FIFA) al ganador de la copa

[8] http://www.robocup.org Accedido el 31 de Agosto de 2012.

del mundo [Nú08]. Actualmente, el proyecto se ha expandido y RoboCup dispone cuatro competiciones:

- **RoboCupSoccer**: Promueve la creación de equipos totalmente autónomos y cooperativos para mostrar comportamientos competitivos avanzados y estrategias. Dentro de RoboCupSoccer se encuentran diferentes ligas:
  - Liga de simulación. Esta competición se caracteriza por la independencia de movimientos por parte de los jugadores (son robots virtuales) que juegan al fútbol en un campo virtual dentro de una simulación. Tiene modalidades tanto de dos como de tres dimensiones.
  - Liga de robots de tamaño pequeño. Esta modalidad consiste en disputar encuentros fútbol por medio de dos equipos compuestos por seis robots de 18cm de ancho por 15cm de alto. Esta liga se centra en la problemática que supone el desarrollo de la inteligencia para un sistema multiagente, así como, la cooperación y el control del mismo en un entorno dinámico.
  - Liga de robots de tamaño mediano. Los partidos de esta modalidad se juegan con dos equipos de seis robots en cada uno. Cada robot posee debe caber en una caja de 52cm. x 52cm. x 80cm. El principal interés en esta competición se centra en la autonomía total de los robots y la cooperación para conseguir resultados.
  - Liga de plataforma estándar. En esta modalidad los equipos compiten con robots idénticos, dichos robots operan de una manera autonómica. El objetivo de esta competición es evaluar los programas software que poseen los robots.
  - Liga de robots humanoides. En esta liga robots autónomos con una estructura corporal y unos sensores semejantes a los humanos juegan al fútbol unos contra otros. En esta competición se persiguen varios objetivos como situaciones en las que un robot debe andar, correr, golpear el balón sin perder el equilibrio, juego en equipo, percepción visual del balón, etcétera.
- **RoboCupRescue**: Utiliza robots que realizan rescates de personas. Los robots son semiautomáticos y capaces de realizar trazados en un entorno complejo. Esta competencia se divide en dos competiciones:
  - Liga de rescate real mediante robots. Es una liga por equipos con un único objetivo: desarrollar y demostrar las capacidades de los robots en ocasiones de emergencia. Algunos de las capacidades que se pretenden comprar en esta modalidad son localizar víctimas y comprobar su estado, establecer comunicación con las víctimas, entregar medicinas o productos a las víctimas, comprobar el estado de las estructuras, etcétera. La competición consiste en la realización de 7-10 misiones de 20-30 minutos cada una desde varios puntos de inicio con el objetivo de encontrar al mayor número de víctimas.

    - Liga de rescate simulado. Esta competición persigue dos objetivos. El primero es el de desarrollar simuladores que sean realistas y que emulen la problemática que supone un desastre en el mundo real. El segundo es desarrollar agentes inteligentes y robots que ejecuten acciones correctas en el tipo de escenarios anteriores.

- **RoboCup@Home**: Esta competencia la componen los robots que ayudan a las personas en la vida diaria en casa. Es la competición internacional anual más larga de robots de servicio autónomos. En esta competición se utiliza un conjunto de *tests* para la evaluación tanto de las habilidades de los robots, como el rendimiento de los mismos en un entorno real no estandarizado como es una casa.
- **RoboCupJunior**: Esta modalidad pretende motivar a los jóvenes a aprender técnicas y conocimiento necesario en ciencia, tecnología y matemática mediante la creación de robots autónomos. Dentro de RoboCupJunior nos encontrarnos diferentes modalidades.
    - Fútbol.
    - Baile.
    - Rescate.

A continuación se estudia en profundidad la competencia RoboCupSoccer y más concretamente la liga de simulación debido a que se han utilizado datos de esta competición por su semejanza con los partidos reales.

RoboCupSoccer

La liga de simulación de RoboCupSoccer es una de las más antiguas de la competencia [9]. Esta liga se centra en la inteligencia artificial y la estrategia de equipo. En ella podemos encontrar dos ligas más:

- **Liga de simulación en 2D**: La idea principal de esta liga es poder utilizarla como escenario inicial de técnicas que posteriormente puedan ser implementadas en los robots físicos reales de cualquier tipo de liga. En esta liga se enfrentan dos equipos de once agentes (programas de software) (ver figura 3.43) en un estadio virtual de dos dimensiones. Existe un servidor principal (ver figura 3.42) que conoce la información de los partidos (la posición del balón y los jugadores, la visibilidad de cada jugador, etcétera) y el que lleva a cabo la comunicación con cada agente. Los agentes reciben información (sonora, visual o física) del servidor y actúan en consecuencia (girando, acelerando, pasando el balón o disparando a portería). Un partido está divido en 6000 ciclos y cada uno de ellos se ejecuta en un tiempo de 100ms.

  A la vez que se está desarrollando un partido se generan dos *logs*. Uno de ellos contiene la información producida por el servidor, es decir, la información que el servidor envía

[9] http://www.robocup.org Accedido el 31 de Agosto de 2012.

a cada uno de los agentes. Este fichero es costoso de entender para los humanos puesto que, con el objetivo de que las conexiones y el paso de información entre los agentes sea más rápido, los mensajes serán más cortos indicándose cada percepción mediante una letra. El estado se indicará como una cadena de texto que denotará el valor de las variables de cada agente. Por otro lado, el segundo fichero contiene información general acerca de la simulación, dentro de este fichero de log se pueden encontrar los siguientes datos:

- Configuración específica de los jugadores. Se definen las características a las que se tendrán que ceñir tanto los equipos en general como cada uno de los agentes que actúen como jugadores en concreto. Algunas de las características que se pueden observar son: El número de sustituciones que se pueden realizar en una simulación, el incremento de velocidad máxima de un jugador, el incremento máximo de resistencia de un jugador, etc.
- Información sobre los gráficos.
- Simulación. Se almacena la información correspondiente a la simulación del encuentro propiamente dicha. En este fragmento del fichero se puede observar información de cada uno de los jugadores y del balón, en concreto, alguno de los datos que se pueden apreciar son las posiciones y la velocidad en los ejes X e Y relativas al campo de fútbol dispuesto por RoboCup, los ángulos de la cabeza y del cuerpo de un jugador, la calidad de la visión, la resistencia, el número de veces que ha golpeado la pelota o el número de veces que ha acelerado, entre otros.

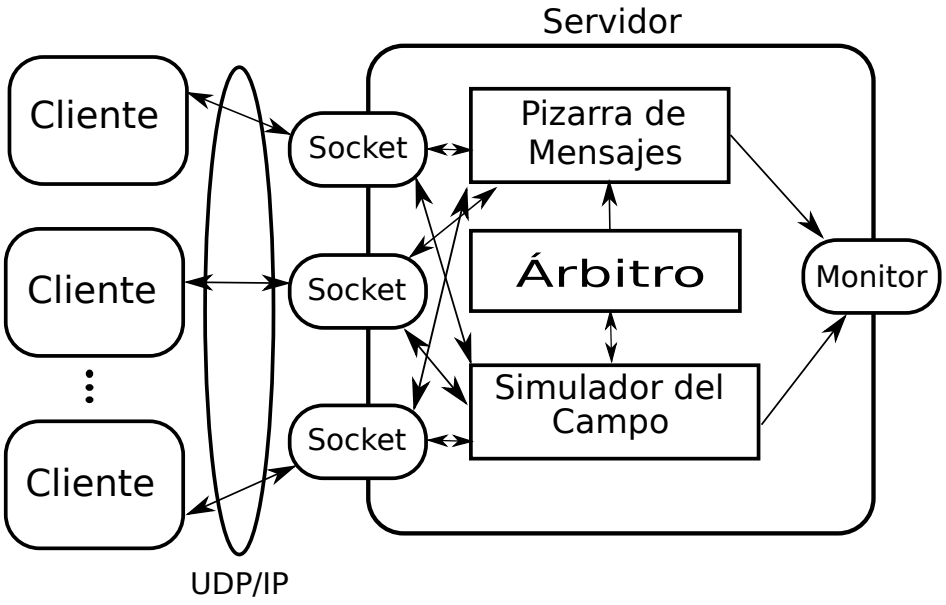


Figura 3.42: Arquitectura RoboCupSoccer: Liga de simulación 2D

Existe una utilidad para el segundo archivo que se denomina **rcg2xml**[10] cuya funcionalidad es convertir los datos de un fichero del segundo tipo en los mismos datos pero en formato XML, lo cual facilita la lectura y comprensión de dichos datos.

Otro componente del cual se dispone es un visualizador de simulaciones, con el cual se pueden reproducir las simulaciones con el objetivo de estudiarlas y extraer información para utilizarla en diferentes partidos. Este simulador contiene diferentes opciones como pueden ser cambiar el color de cualquier componente del campo, ya sea el color del césped, líneas o vestimenta de los equipos, entre otros. El visualizador también permite mostrar los números de los jugadores, ver el ángulo de visión de los jugadores o hacer zoom.

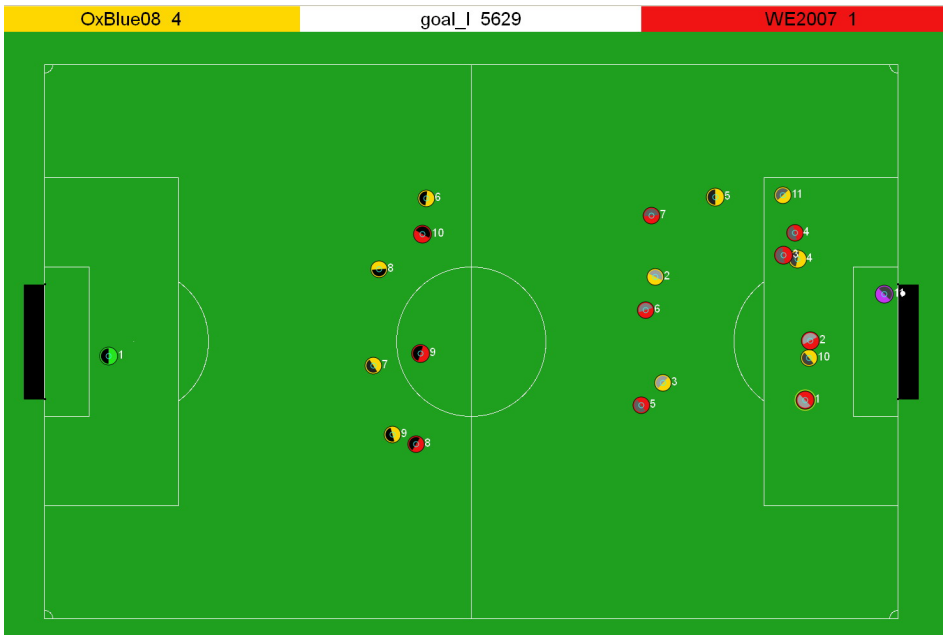


Figura 3.43: Simulador 2D de la RoboCupSoccer 2D

En la actualidad existen varios agentes o APIS (que facilitan el desarrollo de dichos agentes):

- **Biter**: Framework que permite disponer de un mapa de coordenadas absolutas del terreno de juego y los objetos que hay en él. Posee comportamientos simples de agentes y algunas utilidades para iniciar la formación de un equipo y, en base a ésta, crear comportamientos complejos. También implementa una arquitectura genérica de agentes que puede extenderse para cualquier uso [BV01].
- **Atan**[11]: Es una biblioteca software que permite focalizar los esfuerzos en la implementación del comportamiento de los clientes en lugar de centrar nuestra aten-

[10] http://sourceforge.net/projects/sserver/files/rcsslogplayer/
[11] http://robocup-atan.github.com/atan/ Accedido el 31 de 2012

ción en aspectos de más bajo nivel (como por ejemplo el proceso de traducción de mensajes por parte del servidor)

- **Liga de simulación en 3D**: A diferencia de la anterior liga, esta incluye una dimensión más, lo que supone un aumento de la complejidad de la física de la simulación y de las percepciones de los agentes.

  En un principio los agentes eran esféricos, aunque actualmente los agentes son humanoides. En realidad, el principal interés de esta liga es crecer rápidamente para que la investigación pueda ser más avanzada y permitir realizar experimentos con robots reales.

A continuación, se detallan las tácticas utilizadas por los finalistas de la *RoboCup Soccer 2D* del año 2012.

WrightEagle

WrightEagle es un equipo que participa asiduamente en las competiciones de RoboCup en la modalidad de simulación de fútbol 2D [BZL$^+$12]. Fue creado en 1998 y ha obtenido 7 trofeos en los últimos 7 años, hecho que denota la importancia de este equipo en la competición.

Los desarrolladores del equipo toman la modalidad de simulación 2D de fútbol de RoboCup como un problema típico de sistemas multiagente y su principal meta a largo plazo es la investigación en la toma de decisiones y otros desafíos en la Inteligencia Artificial.

En el año 2012, han aplicado diferentes técnicas para mejorar el equipo:

- **Detección de la alineación del equipo rival**: Este apartado es de vital importancia en el desarrollo de un encuentro. Conocer la alineación y la forma de juego del rival hace que sea posible determinar ciertas acciones que puede llevar a cabo y anticiparse a ellas.

  Para implementar esta técnica el equipo de WrightEagle utiliza la triangulación **Delaunay** [BCvKO08] (ver figura 3.44 [12]). La técnica consiste en una red de triángulos que cumple que la circunferencia circunscrita de cada triángulo de la red no contiene a ningún otro vértice de otro triángulo. Con esta técnica se modela la alineación del rival y se clasifica en tipos predefinidos como 4-4-2, 4-3-3, etcétera.

- **Localización de Monte Carlo**: Los agentes en la liga de simulación 2D tienen la dificultad de que sólo pueden recibir observaciones locales y de sonido. En WrightEagle han implementado un sistema que permite a los agentes estimar el estado basándose en creencias. Una creencia es una distribución de probabilidad sobre el espacio de estados. Con $b(s)$ se denota la posibilidad que el entorno se encuentre en un estado s.

[12] http://en.wikipedia.org/wiki/File:Delaunay_circumcircles_centers.svg

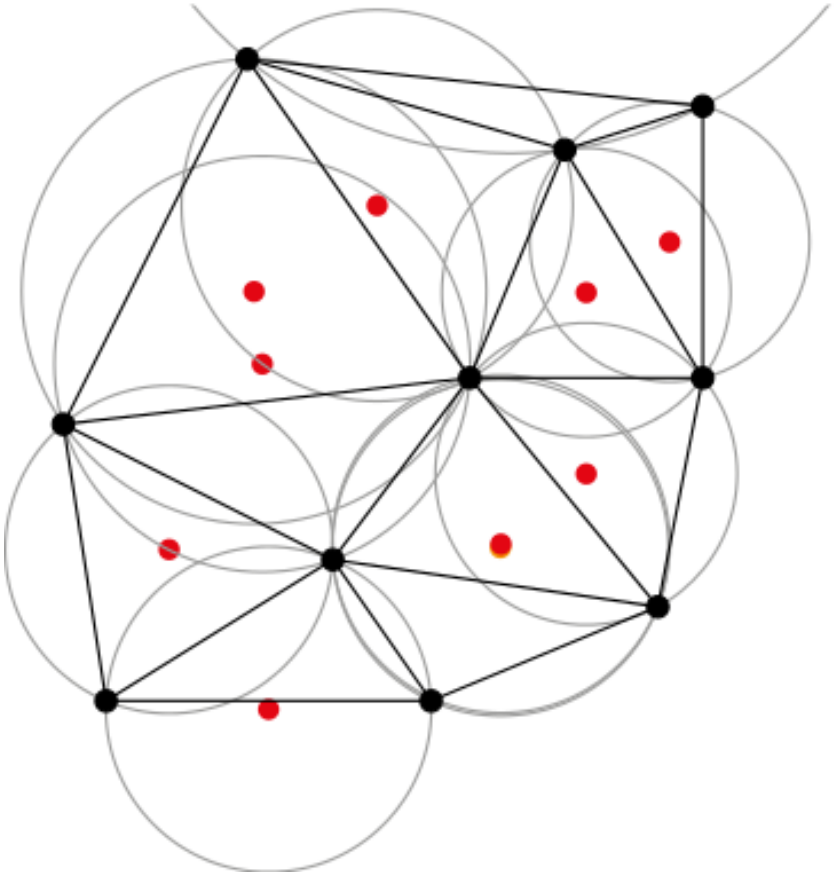


Figura 3.44: Triangulación Delaunay

Así la creencia se puede expresar como

$$b(s) = \sum_{0 \leq i \leq 22} b_i(s[i])$$

donde s es el vector de estados y s[i] es el vector de estados parciales para el objeto i y $b_i(s[i])$ es la distribución marginal para s[i]. $b_i(s[i])$ Un conjunto de m_i muestras ponderadas puede utilizarse para aproximar b_i como

$$b_i(s[i]) \approx \{x_{ij}, wij\}_{j=1...m_i}$$

donde x_{ij} es un estado de muestra para el objeto i, y w_{ij} representa la probabilidad que un objeto este en dicho estado.

Al comienzo de cada ciclo estas muestras se actualizan por procedimientos de Monte Carlo. El estado s actual del entorno es estimado por:

$$s[i] = \sum_{i \leq j \leq m_i} w_{ij} x_{ij}$$

- **Transmisión de comportamientos y posicionamiento multi-etapa**: El comportamiento de posicionamiento se centra en el problema de identificar la posición en la que se encuentran los jugadores cuando no poseen el balón y están atacando. El comportamiento de posicionamiento es esencial en la táctica y estrategia del equipo. Por ejemplo, si el agente 10 piensa que el agente 8 (que posee el balón) puede pasarle el balón al agente 7 en un futuro próximo, el agente 10 puede avanzar a una posición cercana al agente 7 para facilitar el ataque.

  Para llevar a cabo esta técnica, WrightEagle implementa un seguimiento del balón representado mediante una secuencia ordenada en el tiempo de agentes que han tenido la posesión del mismo. Con esta aproximación se facilita el uso de diferentes estructuras, como árboles que permitan búsqueda de estados favorables o algoritmos de búsqueda voraz para especificar el agente al cual pasar. Esta aproximación funciona de manera limitada porque pueden existir estados que estén lejanos en el futuro.

HELIOS2012

HELIOS2012 es un equipo de fútbol que, al igual que *WrightEagle*, participa asiduamente en la *RoboCup Soccer 2D* [ASN+12]. El equipo lleva participando en dicha competición desde el año 2000 y, alcanzó el primer puesto en el año 2010 y 2012. Para este último año, plantearon una serie de tácticas que lo llevaron a obtener el primer puesto. Algunas de estas tácticas se resumen a continuación:

- ***Framework* para la búsqueda de secuencias de acciones**. Este *framework* genera y evalúa un número de secuencias de acciones desarrollado por múltiples agentes en un entorno continuo de estado-acción. Las acciones generadas se almacenan como un nodo en un árbol de búsqueda. Un recorrido desde la raíz a una hoja representa una secuencia de acciones que define la táctica. El *framework* genera secuencias de acción y evalúa sus valores. En la implementación de las tácticas del equipo, se emplea una búsqueda de *el primero, el mejor* como algoritmo de búsqueda en el árbol.

- **Disminución de las oscilaciones en la táctica de planificación**. La oscilación de la toma de decisiones se define en los siguientes términos. Cuando el poseedor del balón mantiene la posesión más de un cyclo, ocurre alguna de las siguientes cosas; se cambia el tipo de acción, el objetivo del jugador cambia o el error de la posición objetivo se encuentra por encima de un umbral antes definido. Es importante disminuir las oscilaciones de la toma de decisiones, con el fin de estabilizar el comportamiento del agente.

  La evaluación de un valor *e* se lleva a cabo mediante la siguiente función:

$$e' = e * exp(-k\frac{|p_{t_n} - p_{t_m}|}{1 + (t_n - t_m)})$$

dónde e' es el valor de la evaluación, t_n y t_m representan el ciclo actual y el ciclo de la anterior decisión realizada, respectivamente, p_{t_n} y p_{t_m} representan la posición actual del objetivo y la posición del objetivo en la misma profundidad del árbol de la anterior decisión tomada, y k es un parámetro no negativo real para cambiar el efecto del tiempo y la distancia.

Otros equipos pueden emplear otras técnicas, pero se pretende dar una visión general acerca de cómo la Inteligencia Artificial puede aplicarse en el dominio del fútbol.

El anterior equipo pone de manifiesto ciertas técnicas de Inteligencia Artificial que pueden llevarse a cabo con los diferentes *frameworks* o bibliotecas para la creación de agentes y diseño de su comportamiento.

ETW

ETW (*Eat the Whistle*) [13] es un simulador 2D libre de fútbol desarrollado por Hurricane Studios como un producto comercial para la plataforma Amiga, que fue lanzado bajo la licencia GPL versión 2 en el año 2002 (ver figura 3.45). ETW es compatible con diferentes sistemas operativos como Windows, Mac o GNU/Linux gracias a la biblioteca SDL[14][PL02].

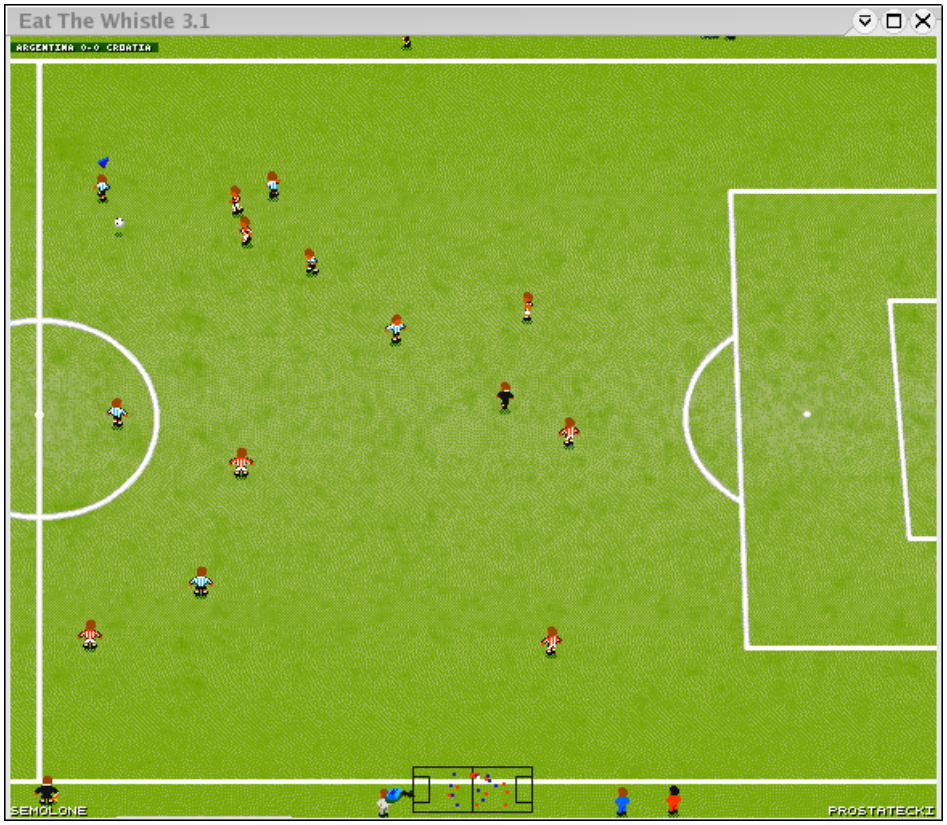


Figura 3.45: Simulador Eat The Whistle

[13] http://www.ggsoft.org/etw/. Accedido el 31 de Agosto de 2012.
[14] www.libsdl.org

Al ser un simulador de fútbol, el usuario interviene de manera directa en la simulación de un partido, indicando a los jugadores lo que tienen que hacer en cada momento. Para ello, el simulador provee varios puntos de acceso de datos por parte del usuario, como diferentes tipos de joystick o el teclado.

El simulador tiene diferentes opciones, como las que se citan a continuación:

- Condiciones atmosféricas. Es posible simular el terreno de juego con lluvia, nieve, sol, etcétera.
- Diferentes tipos de terrenos de juego.
- Posibilidad de jugar partidos amistosos, torneos, copas, etcétera.
- Repeticiones.
- Radar que indica las posiciones de los jugadores en el terreno de juego.

3.3.3. Sistemas para el análisis de datos y comportamientos

En la actualidad y con la creciente repercusión de las nuevas tecnologías como tabletas, o smartphones, resulta sencillo encontrar aplicaciones que permitan definir o analizar comportamientos en diferentes deportes. En concreto, el fútbol con sus respectivas variantes, es uno de los deportes más explotados por las nuevas tecnologías. Aunque, también es posible encontrar herramientas construidas con otras tecnologías.

Algunas de las herramientas que han sido diseñadas para el análisis de datos y comportamientos son las siguientes:

VIENA

VIENA[15] es una herramienta creada por el grupo de investigación CAOS[16] (Control, Aprendizaje y Optimización de Sistemas) que permite las siguientes acciones:

- Visualización gráfica de simulaciones de RoboCup, mediante representación dinámica de datos.
- Representación de las propiedades de los jugadores.
- Detección de las acciones de los jugadores.
- Configuración dinámica de las diferentes parámetros visuales.
- Configuración dinámica de la heurística del área de entrada.
- Análisis del comportamiento de un equipo utilizando diferentes algoritmos.

En esta herramienta se pueden encontrar diferentes propuestas para la obtención de un modelo de comportamiento de un equipo de fútbol, a partir de una serie de secuencias de acciones

[15] http://www.caos.inf.uc3m.es/ viena/
[16] http://www.caos.inf.uc3m.es/

básicas futbolísticas que representan su comportamiento [IPLS09]. Algunas de las técnicas utilizadas por VIENA son las siguientes:

- Técnica basada en la construcción de una estructura denominada *trie*. Con esta técnica, se consideran relevantes aquellas acciones que se repiten en varias ocasiones. En esta técnica, tienen especial importancia las dependencias temporales. De manera que, se utiliza una estructura que identifica de manera sencilla los eventos futbolísticos atómicos más relevantes con sus dependencias con los eventos anteriores y posteriores. Esta técnica. utiliza una estructura de datos denominada como *trie*. Un *trie* es un tipo de árbol de búsqueda similar usado por las tablas de página en los sistemas de memoria virtual [IPLS09].

  El proceso de creación de un modelo utilizando esta técnica son:

  - División de acciones futbolísticas.
  - Almacenamiento de la subsecuencias en el *trie*.
  - Cálculo de la relevancia de las subsecuencias.
  - Obtención del modelo.

- Técnica de modelado utilizando un método de *FT-FDI* (Frecuencia Términos - Frecuencia de Documentos Inversa). Este término, es muy utilizado en áreas como recuperación de información y minería de texto. En estos entornos, la importancia de una palabra incrementa proporcionalmente al número de veces que esta aparece en un documento. Pero, este valor es modificado considerando la frecuencia de la palabra en el total de documentos a considerar [IPLS09].

  En VIENA, se utiliza la frecuencia con la que se ejecuta una acción pero, también qué cantidad de usuarios la han realizado [IPLS09]. Así, para calcular el peso *FT-FDI* para una acción A en un equipo E, utilizaremos la siguiente fórmula [IPLS09]:

$$FT - FDI = Veces(A, E) * log\left(\frac{Jugadores(E)}{Jugadores(E, A)}\right)$$

  dónde:

  - Veces(A,E) es el número total de veces que ha sido ejecutada esa acción A por un equipo E.
  - Jugadores(E,A) es el número de jugadores del equipo E que ha ejecutado A.
  - Jugadores(E) es el número total de jugadores del equipo E.

  Lo que pretende la fórmula anterior es añadir relevancia a las acciones que son ejecutadas por pocos jugadores, por ejemplo, si una acción la realizan los once jugadores, se tiene que $log(\frac{11}{11}) = 0$ lo que indica que dicha acción no tiene relevancia.

Fútbol Coach

Fútbol coach[17] es una aplicación para Android, pensada especialmente para entrenadores y dedicada en exclusiva al fútbol, con la que se pueden realizar tácticas y planificar entrenamientos. Las estrategias, se diseñan estableciendo las posiciones de los jugadores en determinados instantes. La aplicación dispone de un reproductor con el que poder visualizar las jugadas almacenadas. En la figura 3.46 se puede apreciar una captura de pantalla de la aplicación.

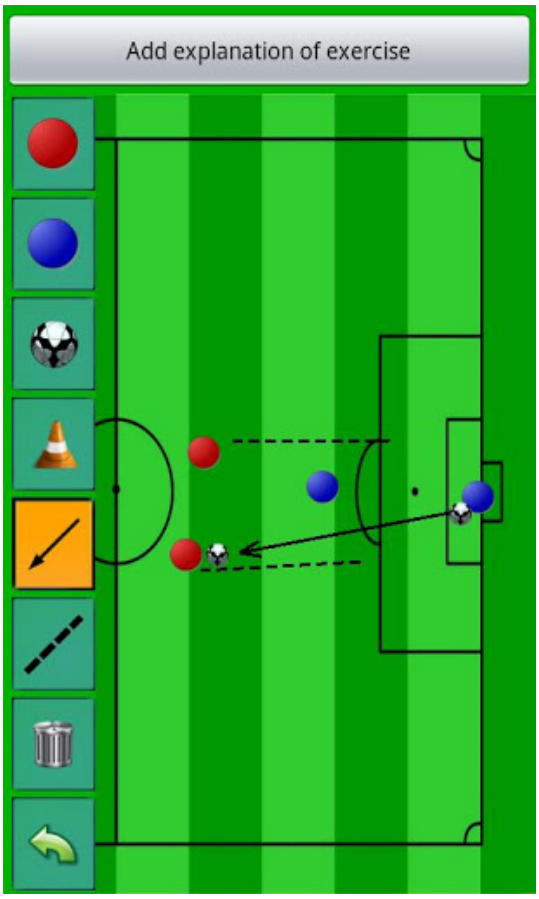


Figura 3.46: Aplicación Fútbol Coach

Gracias a la aplicación es posible planificar estrategias para distintos tipos de jugadas, tanto a balón parado como, faltas, córners o saque de banda además, también se pueden definir jugadas ensayadas como el contraataque. A continuación, se explican los diferentes modos que posee la aplicación:

- **Free**. Este modo, permite utilizar la aplicación como si fuera una pizarra, de manera que, en este modo podemos editar las posiciones de los jugadores en el terreno de juego e interactuar con ellas cambiándolas de posición, de una manera sencilla y cómoda.
- **Strategy**. En esta sección, el usuario puede definir jugadas y almacenarlas. Poste-

[17] https://play.google.com/store/apps/details?id=com.canica.apps.soccer&hl=es

riormente a la creación de la jugada, es posible visualizarla, gracias a un reproductor incorporado en la aplicación.

- **Exercise**. Con esta opción, la aplicación da la oportunidad al usuario de definir diferentes ejercicios que realizar durante una sesión de entrenamiento. En la aplicación, existen diferentes tipos de ejercicios como, ejercicios físicos, de técnica individual, mejora del regate, juego en equipo y disparo. También, es posible que el usuario defina sus propios ejercicios, lo que dota a la aplicación de una gran flexibilidad.
- **Training**: En este modo, es posible planificar entrenamientos asignando ejercicios y determinando el tiempo de duración de cada uno de estos ejercicios.

Coaching Wizard

Coaching Wizard es una aplicación para iOS definida cómo, una *aplicación muy intuitiva y con una apariencia agradable repleta de toda funcionalidad que un entrenador necesite*[18], aunque el desarrollador principal la define como *una aplicación lo suficientemente simple para que cualquier entrenador o padre aficionado a ver a su hijo jugar en un equipo pueda almacenar los datos del encuentro sin perder la atención en el mismo*. En la figura 3.47 se puede apreciar una captura de pantalla de la aplicación.

Figura 3.47: Aplicación *Coaching Wizard*

[18]Según www.coachingwizardapp.com

El principal cometido de esta aplicación es sustituir a las libretas de entrenadores. La herramienta se basa en la creación de un equipo y almacenamiento de las estadísticas del mismo. Para ello, la aplicación permite crear un equipo, definiendo tanto el número de jugadores que componen el equipo, como el nombre de los dichos jugadores jugadores o determinar el color de las vestimentas y dorsales. También permite definir la duración de los partidos.

Al inicio de un partido se puede elegir el equipo titular y la demarcación en el terreno de juego de cada uno de los jugadores. La herramienta iniciará un cronómetro que se detendrá en la duración indicada en la creación del equipo. Este cronómetro es ascendente y tendrá la que será, quizás, la característica más importante de la aplicación, puesto que permitirá anotar los diferentes eventos del partido sin tener que consultar un reloj auxiliar. Además, gracias al uso del cronómetro, se generan estadísticas acerca de cuánto a jugador un jugador, de manera que resulta sencillo calcular promedios en los que intervenga la variable temporal como, número de minutos jugados entre goles anotados, o número de minutos jugados entre número de asistencias, etcétera.

Durante el desarrollo del partido será posible cambiar la situación de los jugadores en el terreno de juego, así como anotar cualquier acción que suceda como disparos a puerta, gol a favor, gol en contra, asistencias, paradas, sustituciones, etcétera. La aplicación permite al usuario disponer de los datos almacenados en cualquier momento, además ofrece la posibilidad de enviar los resultados a diferentes correos. La característica del envío de correo resulta especialmente útil, puesto que el entrenador puede estar ausente por cualquier motivo como enfermedad y puede recibir las estadísticas del partido en cualquier momento.

Capítulo 4

Método de trabajo

En este capítulo se detalla el método de trabajo seguido para el desarrollo del presente Proyecto Fin de Carrera. También se describen las herramientas software y hardware empleadas en la construcción del mismo.

4.1. Metodología de trabajo

La metodología de desarrollo utilizada en este proyecto ha sido *prototipado evolutivo*. Este método de trabajo se apoya en el concepto de **prototipo**, que se define como un modelo funcional de un sistema con el que tiene en común ciertos aspectos. Desde el punto de vista de ingeniería del software un prototipo se define como *el desarrollo de un sistema con la funcionalidad necesaria para abordar diferentes requisitos* [Vli07]. El prototipo se irá depurando y mejorando en cada iteración, añadiendo o modificando funcionalidad según se vayan surgiendo nuevos requisitos.

En un principio, dado el carácter experimental del presente proyecto, no se identificaron todos los requisitos que implementa actualmente el sistema además, muchos de ellos eran incompletos o su definición era demasiado vaga. Por tanto, el proyecto requería el uso de una metodología que se basara en la adaptabilidad y escalabilidad para la definición de nuevos requisitos o la modificación los existentes y que no resultara complicado integrarlos en el sistema. La metodología de prototipado evolutivo aprovecha este planteamiento y realiza la fase de definición de requisitos en base a tres factores:

- Iteración, con el objetivo de descubrir nuevos requisitos o de refinar los que ya se han definido. El uso de iteraciones permite que los prototipos evolucionen hasta construir el sistema final.
- Participación con el usuario [1].
- Uso extensivo de prototipos.

Este método, o proceso de desarrollo, se caracteriza por la repetición de una serie de pasos que se pueden apreciar en la figura 4.1:

[1] En el caso del presente proyecto, el usuario es el director de proyecto David Vallejo Fernández.

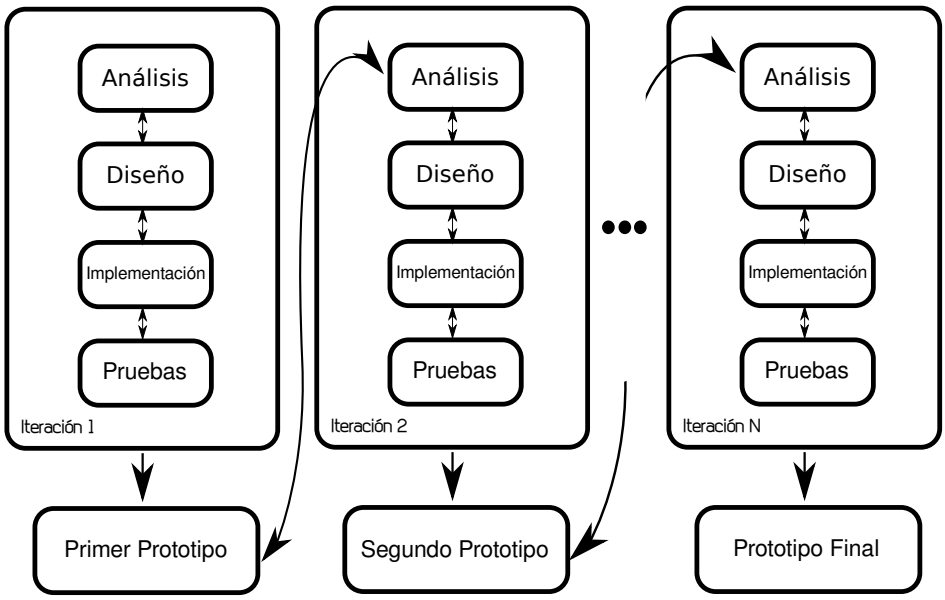


Figura 4.1: Ciclo de desarrollo con Prototipado Evolutivo

1. **Ánalisis**: Puesta en común de los requisitos con el cliente. En esta etapa, desarrollador y cliente acuerdan qué requisitos debe tener el prototipo a implementar. En esta fase también se realizan modificaciones de requisitos en prototipos anteriores, con el objetivo de refinar el sistema final.
2. **Diseño**: Una vez definidos los requisitos en la etapa anterior se diseñan esquemas que describan la estructura del prototipo.
3. **Implementación**: En esta etapa se implementan los componentes descritos en la etapa anterior. En esta fase se genera un prototipo inicial.
4. **Pruebas**: Se prueba el prototipo generado en la fase anterior mediante pruebas automáticas. Cuando se han realizado las pruebas oportunas se genera el prototipo final que será el que se muestre al usuario.
5. **Comprobación del prototipo**: El usuario puede decidir qué nuevas funcionalidades incluir sobre ese prototipo o, por el contrario modificar algunos comportamientos del mismo con el objetivo de refinarlo. Con el prototipo generado y la realimentación del usuario se inicia otra nueva secuencia de estos mismos pasos. Este bucle acabará cuando el usuario esté de acuerdo con el prototipo final.

En el presente proyecto se ha utilizado esta metodología por la siguientes razones:

- Permite el desarrollo del sistema de manera incremental, posibilitando que el desarrollador adquiriera experiencia de iteraciones anteriores.
- Los requisitos no se definen completamente al principio, sino que emergen del desarrollo.

- El usuario final está involucrado desde el principio, de manera que se pueden atender mejor sus exigencias y cambios durante el desarrollo de todo el proceso.
- Los prototipos se generan con frecuencia, de manera que la sensación de avance es mayor que con otras metodologías.

En resumen, el carácter experimental y de investigación hacen que el prototipado evolutivo sea una buena opción para el desarrollo del presente Proyecto Fin de Carrera, debido a que el diseño y desarrollo de un sistema automático de análisis de comportamientos tiene en la actualidad un fuerte componente experimental.

4.1.1. Aplicación de la metodología

La metodología de prototipado evolutivo está ideada para que cualquier equipo de desarrollo. En este caso particular, al tratarse de un único desarrollador no existen restricciones a la hora de llevar a cabo el proceso de desarrollo de un sistema aplicando esta metodología.

El desarrollador del sistema es Gabriel Alises Cano. Por otra parte, David Vallejo Fernández actúa como director del proyecto y usuario final de la aplicación. La generación de los prototipos ha seguido una dinámica ágil, puesto que, durante el desarrollo del proyecto, se mantuvieron reuniones típicamente cada dos semanas. Aunque si la funcionalidad de un prototipo, acordada en la reunión era elevada, la siguiente se posponía otra semana. Por contrarío, si en la reunión se acordaba la realización de cambios o implementación de nuevas funcionalidades mínimas, la siguiente se acortaba una semana.

4.2. Herramientas

A continuación se detallan los recursos, tanto software como hardware, empleados en la construcción del presente proyecto.

4.2.1. Lenguajes de programación

Para la implementación del presente proyecto se han utilizado los siguientes lenguajes:

- **Python:** La gran mayoría del proyecto está codificado en este lenguaje. Python, es un lenguaje que, aunque interpretado, es eficiente para el desarrollo de aplicaciones de una gran variedad de dominios. El lenguaje está orientado a objetos, lo que permite una capa de abstracción, encapsulamiento y modularización en la implementación. Además, la filosofía de *Python* es la de ofrecer facilidades para la generación rápida de prototipos de prototipos de un sistema, lo que se ajusta a la perfección con la metodología de desarrollo empleada.
- **Java:** Es el lenguaje empleado para la codificación del sistema de lógica difusa que emplea el proyecto. Se eligió Java para llevar a cabo esta tarea puesto que ofrece po-

sibilidades como, el desacoplamiento del sistema de lógica difusa con respecto de la aplicación o permitir, escribir el sistema de lógica difusa en un lenguaje estándar creado con dicho propósito. Además, el uso de *Java* permite la utilización de la biblioteca *JFuzzyLogic*, la cuál, ofrece funcionalidad necesaria para el desarrollo de un sistema de lógica difusa.

4.2.2. Software

A continuación se describen las herramientas y bibliotecas empleadas en la elaboración del proyecto:

Sistema Operativo

- **Debian**: Durante parte de la implementación del proyecto se ha utilizado Debian, que se trata de una distribución del sistema operativo GNU/Linux. En concreto, se ha empleado la versión Debian llamada *Debian Squeeze*.
- **Ubuntu**: Es una distribución del sistema operativo GNU/Linux basada en Debian. En Ubuntu se ha desarrollado parte del proyecto. La versión empleada ha sido 12.04, también conocida como *Precise Pangolin*. Las principales razones de la elección de esta distribución son su estabilidad y la facilidad de uso.

Software de desarrollo

- **Emacs**: Para la implementación del proyecto se hace uso de este editor de texto, principalmente, por ser altamente configurable a las necesidades de cada usuario y ofrecer gran variedad de funcionalidades.
- **Eclipse**: Se ha empleado este entorno de desarrollo para la implementación del componente encargado del funcionamiento del sistema de lógica difusa. La principal razón por la que se ha utilizado es la facilidad de desarrollo y depuración de aplicaciones escritas en el lenguaje de programación Java.
- **Make**: Es una herramienta que permite la automatización de tareas que resultan repetitivas para el usuario. En el proyecto se ha hace uso de esta herramienta para eliminar los ficheros de código intermedio generados por el intérprete de Python y para generar la documentación del proyecto.

Gestión de la base de datos

- **SQL**: Es un lenguaje declarativo que permite el acceso a esquemas de bases de datos relacionales. La funcionalidad principal de este lenguaje, es la recuperación y almacenamiento de la información almacenada en un base de datos.
- **MySQL-Query-Browser**: Esta aplicación permite la gestión de bases de datos, tanto

remotas como locales. Durante parte del desarrollo del proyecto, se utilizó para el desarrollo y depuración de consultas a la base de datos del proyecto.

- **MySQL-Workbench**: Al igual que MySQL-Query-Browser esta aplicación permite la gestión de bases datos. Este programa se utiliza con la distribución Ubuntu, ya que MySQL-Query-Browser no se encuentra en los repositorios de dicha distribución.
- **MySQL-Admin**: Este programa permite la administración de bases de datos. Durante el desarrollo del proyecto se ha utilizado para la creación de los esquemas de la base de datos del proyecto, así como para la creación de *backups* de la misma.
- **MySQL-Server**: Esta aplicación lanza el proceso servidor de MySQL, el cual permitirá las conexiones a la base de datos.
- **MySQL-Client**: Este paquete permite establecer una conexión con un servidor MySQL para la posterior realización de consultas a la base de datos.

Documentación y gráficos

- **LaTeX**: Es un lenguaje que permite la generación y maquetación de documentos técnicos. En el proyecto se ha utilizado para la elaboración de la presente documentación. Además, se han utilizado los paquetes **arco-authors**[2] y **arco-pfc**[3] que han facilitado la creación de la documentación.
- **Inkscape**: Es un editor de gráficos vectoriales. En la elaboración del proyecto se ha hecho uso de él para la generación de la gran mayoría de las imágenes de la presente documentación.

Interfaz gráfica

- **Glade**: Es una aplicación para el desarrollo rápido de aplicaciones (*RAD*). Glade permite desarrollar de manera rápida y sencilla interfaces gráficas de usuario para el conjunto de herramientas *GTK+* y el entorno de escritorio *GNOME*.

Bibliotecas

- **Python-MySQLdb**: Es una interfaz compatible con el servidor MySQL que ofrece la API de base de datos de Python. Esta biblioteca es básica para la acceso a la información almacenada en la base de datsos.
- **Python-Pygame**: Es un conjunto de módulos que han sido diseñados para codificar videojuegos. Pygame añade funcionalidad a la biblioteca SDL [4], esto permite crear videojuegos y aplicaciones multimedia. Esta biblioteca se usa en el proyecto con el

[2] https://bitbucket.org/arco_group/arco-authors

[3] https://bitbucket.org/arco_group/arco-pfc

[4] SDL es una biblioteca diseñada para proporcionar acceso a bajo nivel al audio, teclado, ratón, joystick, gráficos en 2D o hardware 3D mediante OpenGL

objetivo de implementar una interfaz gráfica agradable al usuario, convirtiendo los datos de una simulación en jugadores de fútbol que se mueven por un terreno de juego.

- **Python-PyGTK**: Permite la creación de programas con interfaz gráfica de usuario. Al utilizar esta biblioteca la experiencia del usuario con la aplicación se hace más sencilla e intuitiva.
- **Python-PyUnit**: Se trata de un *framework* para llevar a cabo las pruebas unitarias.
- **Build-Essential**: Es un conjunto de herramientas que permiten la creación de archivos binarios. Incluye la herramienta *make*, necesaria para la creación del presente proyecto.
- **Python-Matplotlib**: Permite la generación de gráficas 2D de calidad en una gran variedad de formatos. Esta biblioteca se ha utilizado en el proyecto para la generación de las gráficas de los datos de la simulación, con el objetivo de ofrecer al usuario información concisa acerca de las mismos.
- **jFuzzyLogic**: Es un paquete que escrito en Java que implementa la especificación del lenguaje de control difuso *FCL*. Durante el desarrollo del proyecto se ha utilizado para la implementación del motor del sistema de lógica difusa que permite, obtener nueva información de una simulación haciendo uso de la lógica difusa.
- **jUnit**: Al igual que *Python-PyUnit*, se trata de un *framework* para llevar a cabo pruebas unitarias, con la salvedad que está implementado para Java.

Sistema de control de versiones

Se ha utilizado un sistema de control de versiones *Mercurial*[5] para gestionar los cambios y actualizaciones del software desarrollado. Un sistema de control de versiones permite la gestión de cambios realizados sobre los diferentes elementos que componen un producto.

Los servicios más importantes que debe proveer un sistema de control de versiones son los siguientes:

- Almacenamiento de elementos que componen un producto y sobre los que se, a priori, realizarán los cambios a gestionar.
- Realización de operaciones sobre los elementos almacenados en el sistema como: añadir o eliminar elementos, modificarlos o renombrarlos, entre otras opciones.
- Subsistema de registro que permita almacenar las acciones llevadas a cabo en cada uno de los elementos, siendo posible extraer un estado anterior del producto.

En el campo de la ingeniería del software este tipo de herramientas tienen una importancia capital. Los sistemas de control de versiones resuelven el problema que supone el controlar las distintas versiones del código fuente. Es común que muchas de las organizaciones que desarrollan software tengan diferentes versiones, por ejemplo; una sobre la que se desarrollan

[5] mercurial.selenic.com

nuevas funcionalidades, otra que es la que se ofrece al usuario y que está correctamente probada y otra versión que esté en periodo de pruebas.

La funcionalidad de *Mercurial* se ha combinado con *Redmine*[6], el cual es un software web de gestión de proyectos que incluye, entre otras, las siguientes características:

- Diagramas de Gantt.
- Calendario.
- Notificación mediante correo electrónico.
- Gestión de tareas.
- Integración con Mercurial.

Otros

- **XML**: Es un lenguaje de marcas extensible. En el desarrollo del proyecto se ha empleado para almacenar información, debido a la facilidad con la que se puede analizar su contenido por medio de software.
- **Rcssmonitor**: Es una programa que permite la visualización de las simulaciones extraídas de la *Robocup Soccer 2D*.
- **Rcssserver**: Se trata del servidor al que se deben conectar los agentes de la *Robocup Soccer 2D*, necesario para iniciar simulaciones. Cuando una simulación concluye, es el servidor el que genera un archivo de *log*, el cuál, contiene los datos de la misma.
- **Rcg2XML**: Es una herramienta incluida en el programa *rcssmonitor*, cuya funcionalidad es convertir un fichero con extensión *rcg* en un fichero *XML*, sin pérdida de información.

4.2.3. Hardware

Para el desarrollo del presente proyecto se ha utilizado un computador personal portátil Samsung R519 que posee un procesador Intel Pentium(R) Dual-Core T4300 a 2.10 GHz y una memoria RAM de 3.8 GiB. El computador posee una tarjeta gráfica integrada Intel GMA 4500.

[6] www.redmine.org

Capítulo 5

Arquitectura

En este capítulo se describe detalladamente la arquitecta en la que está basado el presente Proyecto Fin de Carrera. Se trata de una arquitectura *modular* y *adaptativa*, esto es, se trata de una arquitectura divida en módulos interconectados entre sí, sobre la cuál se pueden realizar cambios e incorporar nuevas funcionalidades de una forma sencilla.

La forma de abordar la arquitectura será mediante una visión **top-down**, es decir, se partirá de un nivel de detalle general, que se irá refinando hasta un nivel más específico.

Cabe recordar que el objetivo principal del proyecto es el desarrollo de un **Sistema Inteligente** para el análisis automático del comportamiento de jugadores de fútbol, ofreciendo una herramienta gráfica que facilite dicho análisis de una manera interactiva y sencilla para el usuario.

5.1. Descripción general

La arquitectura (ver figura 5.1) está compuesta por un total cinco módulos, los cuáles, se decriben a continuación:

- **Módulo de preprocesamiento de información**. La funcionalidad de este módulo es transladar la información que se encuentra almacenada en diferentes formatos, a un esquema de base de datos relacional. Resulta más sencillo recuperar y modificar la información almacenada en un esquema de base de datos relacional que en cualquier formato en que se encuentre la información.
- **Módulo de análisis de comportamientos**. El principal objetivo de este módulo es analizar los datos procedentes de una simulación para así inferir nueva información. A parte de la inferencia de este conocimiento, este módulo determina la acción que idealmente deberá llevar a cabo un jugador en base a las definiciones almacenadas en la base de conocimiento. El módulo utiliza los datos introducidos en la base de datos por el anterior módulo y almacena la nueva información (conocimiento) en la misma base de datos.
- **Módulo de representación de la información**. La funcionalidad de este módulo consiste en representar la información almacenada en la base de datos de una manera

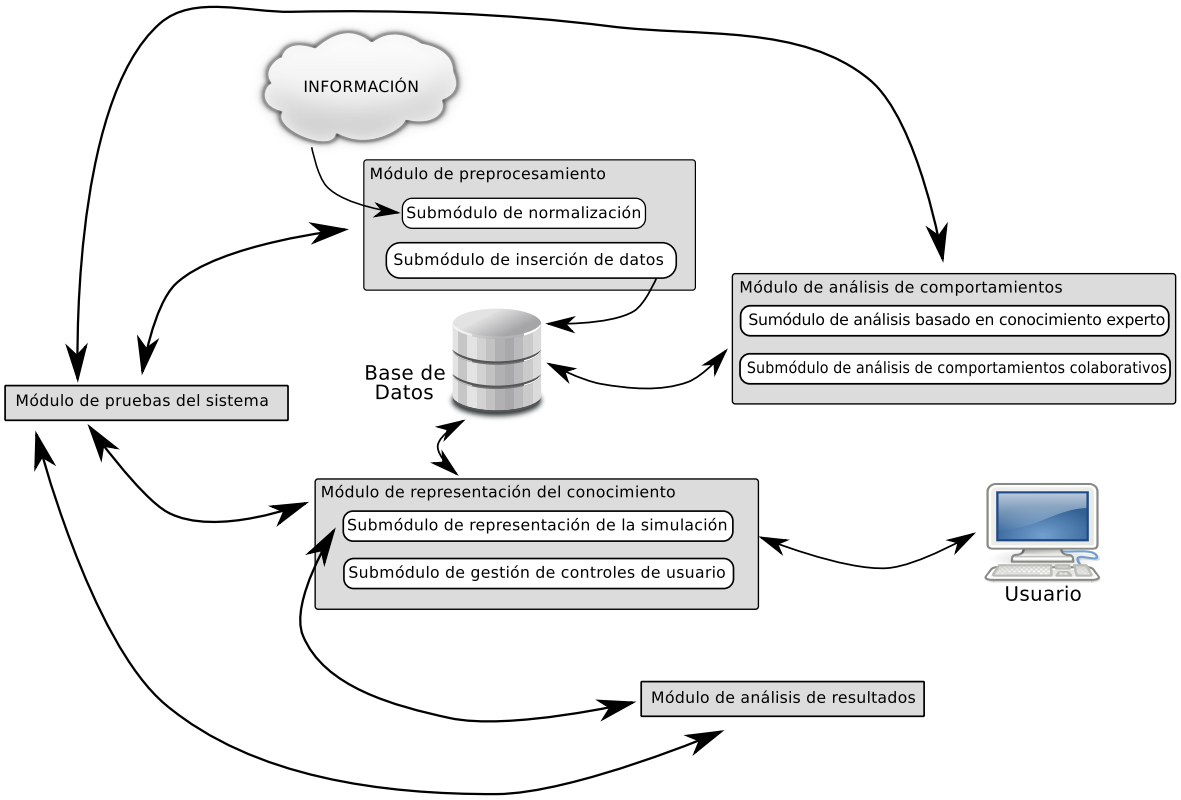


Figura 5.1: Arquitectura del sistema

atractiva y simple para que el usuario pueda interactuar con la herramienta, de una manera cómoda y sencilla. El núcleo de este módulo es una interfaz gráfica de usuario.

- **Módulo de análisis de resultados**. Es el módulo encargado de determinar si un jugador realiza una acción correctamente o no.
- **Módulo de pruebas**. Es el módulo encargado de realizar las pruebas automáticas del sistema. La funcionalidad de este módulo será comprobar que el sistema realice lo que debe hacer y no realice lo que no debe realizar.

Diagrama de casos de uso

Los distintos módulos dotan a la arquitectura de la funcionalidad necesaria para cubrir los requisitos especificados por los distintos casos de uso (ver figura 5.2).

La especificación de los distintos casos de uso es la que sigue:

- **Abrir una simulación**. Cuando un usuario realiza esta acción, una simulación que se encuentra almacenada en la base de datos se carga en memoria. Una vez realizada esta acción, el usuario podrá visualizar una simulación. El usuario puede ejecutar este caso de uso en cualquier momento.

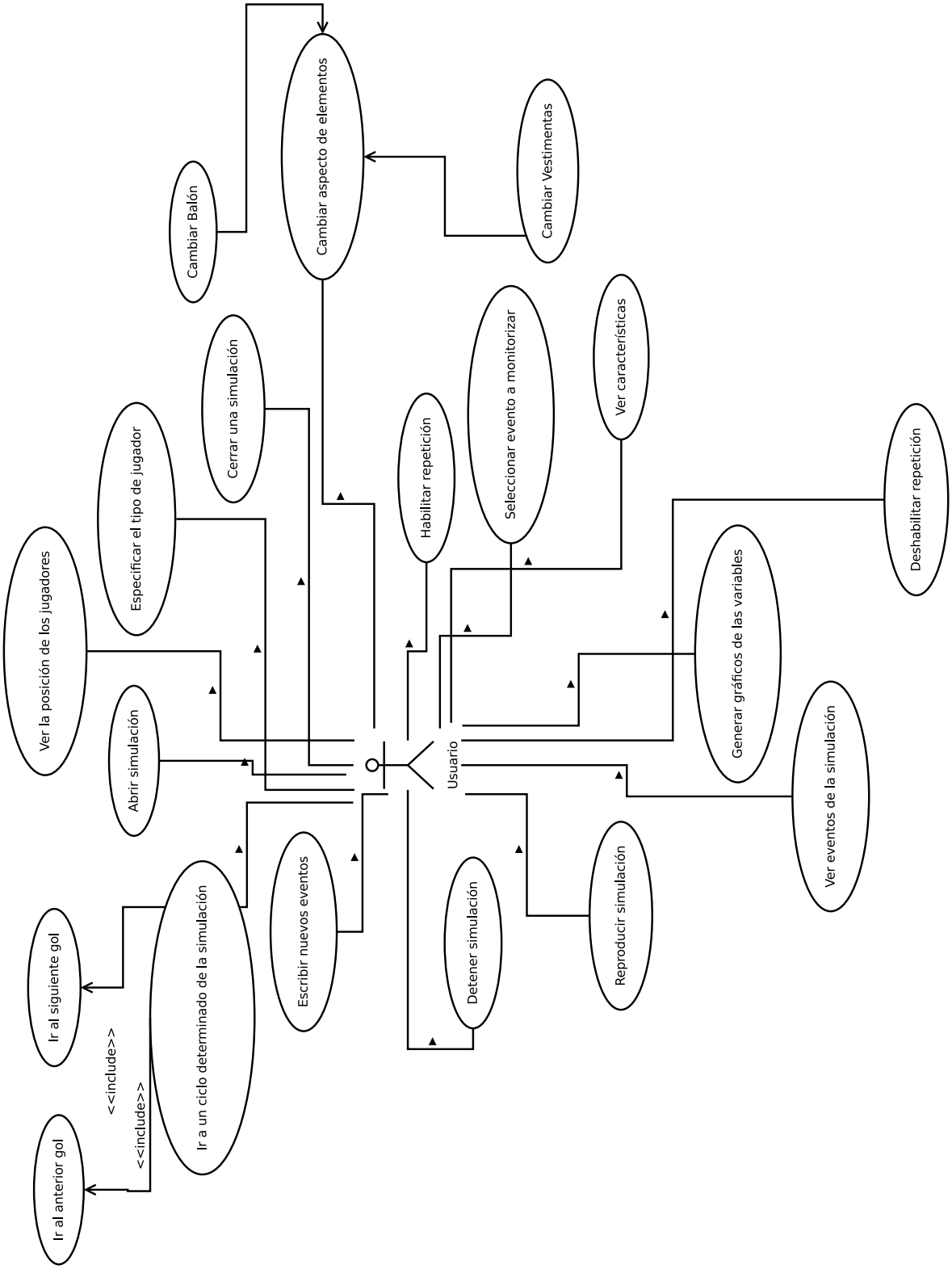


Figura 5.2: Diagrama de casos de uso de la arquitectura

- **Cerrar una simulación**. Al ejecutar este caso de uso, el usuario cerrará una simulación previamente abierta. Esta acción permitirá liberar la memoria del computador los datos de la simulación cargados. Esta acción sólo se podrá llevar a cabo cuando una simulación haya sido abierta.
- **Especificar el tipo de jugador**. El usuario podrá determinar el tipo de cualquier jugador que se encuentre en el terreno de juego. Los tipos de jugador posibles son portero, defensa, centro-campista y delantero. Esta acción puede ser realizada por el usuario en cualquier momento en que una simulación se encuentre cargada.
- **Ver la posición de los jugadores**. El usuario podrá ver el tipo de los jugadores, previamente especificado. Si el tipo de un jugador no está especificado, no se mostrará ningún tipo. Además del tipo, al realizar esta acción, el usuario podrá determinar el número del jugador. Esta acción podrá ser realizada por el usuario en cualquier momento, en que una simulación esté cargada. Si la simulación no se encuentra cargada, esta acción no podrá ser ejecutada.
- **Determinar nuevos eventos**. Al realizar esta acción, un usuario podrá identificar nuevas situaciones que serán reconocidas por el sistema para detectar dichas situaciones en un partido de fútbol. Esta acción podrá ser realizada por el usuario en cualquier momento.
- **Reproducir simulación**. Al ejecutar este caso de uso, un usuario podrá visualizar la simulación desde el punto en que se detuvo. Cuando se carga una simulación se detiene en el primer ciclo, a la espera de que el usuario inicie la reproducción de la información. Esta acción sólo podrá ser ejecutada cuando la simulación se encuentre cargada y detenida.
- **Detener una simulación**. Esta acción tiene una funcionalidad contraria a la acción anteriormente descrita. Al ejecutar este caso de uso, la simulación se detendrá en el ciclo correspondiente. Este caso de uso sólo podrá ser ejecutado cuando la simulación esté cargada y se encuentre reproduciéndose.
- **Habilitar la repetición**. Al ejecutar este caso de uso, el usuario podrá visualizar la repetición de los goles. Cuando se marque un gol, se retrocederá un número de ciclos determinado y se ralentizará la velocidad de la reproducción de la simulación. Así, el usuario podrá visualizar la repetición de los goles de los diferentes equipos que se enfrentan en un partido. Esta acción podrá ser ejecutada cuando la repetición se encuentre deshabilitada, por defecto, al lanzar la aplicación, la repetición se encuentra habilitada.
- **Deshabilitar la repetición**. Esta acción tiene un comportamiento contrario al caso de uso anterior. De manera, que si la repetición está deshabilitada, cuando un equipo marque un gol, se continuará con el siguiente ciclo de la simulación. Esta acción sólo

podrá ser ejecutada cuando la repetición se encuentre habilitada.

- **Generar gráficos de las variables**. Cuando un usuario ejecuta esta acción, es posible determinar tanto los jugadores de cualquier equipo como las variables a generar los gráficos. Una vez especificadas los jugadores y las variables se generará la información correspondiente en forma de gráficos. Esta acción, solo puede ser ejecutada cuando la simulación se encuentre cargada.
- **Ver características de los jugadores**. Al contrario del caso de uso «generar gráficos de las variables» la realización de esta acción permite al usuario generar las características (valores de las variables en un ciclo determinado) de un jugador de forma numérica. Esta acción sólo podrá ser realizada cuando una simulación se encuentre cargada.
- **Ver eventos de la simulación**. La ejecución de este caso de uso permite al usuario visualizar los diferentes eventos de la simulación, así como el ciclo en el cual se producen. Esta acción sólo puede ser ejecutada cuando la simulación se encuentre cargada.
- **Seleccionar evento a monitorizar**. Esta acción permite al usuario determinar el evento del cuál se quiere obtener un *feedback* durante la simulación. Cuando se detecte un evento del tipo seleccionado, la simulación se ralentizará y el jugador que sigue un determinado comportamiento será marcado para facilitar la identificación del mismo al usuario. Esta acción sólo podrá ser realizada cuando la simulación se encuentre cargada.
- **Cambiar aspecto de los elementos**. Cuando se realiza esta acción, un elemento sufre un cambio en su imagen a *renderizar*. Este caso de uso es una generalización de los dos casos de uso siguientes:
  - **Cambiar balón**. La realización de este caso de uso permitirá cambiar la imagen del balón. Los balones a seleccionar serán los balones utilizados en las últimas competiciones europeas.
  - **Cambiar vestimentas**. La ejecución de esta acción permitirá cambiar las equipaciones de cada uno de los dos equipos que se enfrentan en un partido. Las vestimentas a seleccionar serán las que utilizaron en la Eurocopa de 2012.

  Este caso de uso podrá ser ejecutado en cualquier momento.
- **Cambiar a un ciclo de la simulación**. Cuando un usuario ejecuta este caso de uso debe especificar el ciclo al cuál quiere cambiar. La simulación se detendrá en el ciclo determinado. Este caso de uso incluye a dos casos de uso que se describen a continuación:
  - **Cambiar al ciclo del siguiente gol**. Cuando se ejecuta este caso de uso, se lanzará la funcionalidad que cambia a un ciclo determinado de la simulación. Sin

embargo, en este caso, el usuario no especificará ningún ciclo, sino que la simulación se detendrá en el ciclo del siguiente gol.

- **Cambiar al ciclo del anterior gol**. La funcionalidad de este caso de uso es contraria a la funcionalidad del caso de uso anterior. Cuando un usuario ejecuta este caso de uso, la simulación se detendrá en el ciclo que se produjo el último gol.

Este caso de uso sólo podrá ser ejecutado cuando la simulación esté cargada.

Diagrama de clases

El diagrama de clases de la arquitectura es el que se puede apreciar en la figura 5.3. Con el objetivo de no sobrecargar el diagrama, no se muestran los atributos y métodos de las clases. La especificación del diagrama de clases es la siguiente:

- Clase **Point**. Esta clase almacenará la información correspondiente a un punto en el plano sobre el cuál se sitúa el terreno de juego.
- Clase **Region**. Esta clase se compone de una serie de puntos que identifican una región en el plano. El objetivo de esta clase es determinar una región en el plano sobre la cuál se situarán diferentes jugadores y el balón. La utilidad de esta clase será, tal y como se detalla más adelante, para optimizar la búsqueda del jugador que tiene la posesión del balón en un ciclo determinado.
- Clase **Tree**. Esta clase define una estructura de datos en forma de árbol binario que se compone de diferentes regiones. Cuando se carga una simulación se asigna a cada jugador una región del campo en función de la posición del mismo, realizando un recorrido en profundidad en el árbol. De esta manera se determina la región a la que pertenece un jugador en un ciclo determinado.
- Clase **State**. Esta clase almacena la información correspondiente a las características comunes del balón y de los jugadores. Estas características son la posición de los jugadores, la velocidad en cada uno de los ejes de coordenadas y la región que ocupa dentro de campo.
- Clase **State_player**. Esta clase hereda de la clase *State* y almacena la información correspondiente a los jugadores. Los atributos que la definen son el número de oponentes cercanos, si un jugador tiene la posesión del balón o no, la resistencia restante, el ángulo de visión, la orientación del cuerpo.
- Clase **Element**. Esta clase almacena la información correspondiente a un elemento de la simulación. La clase se compone de una lista de estados (la longitud de esta lista será igual al número de ciclos de la simulación). La lista también tiene otros atributos como son el dorsal de un jugador y el tipo de jugador.

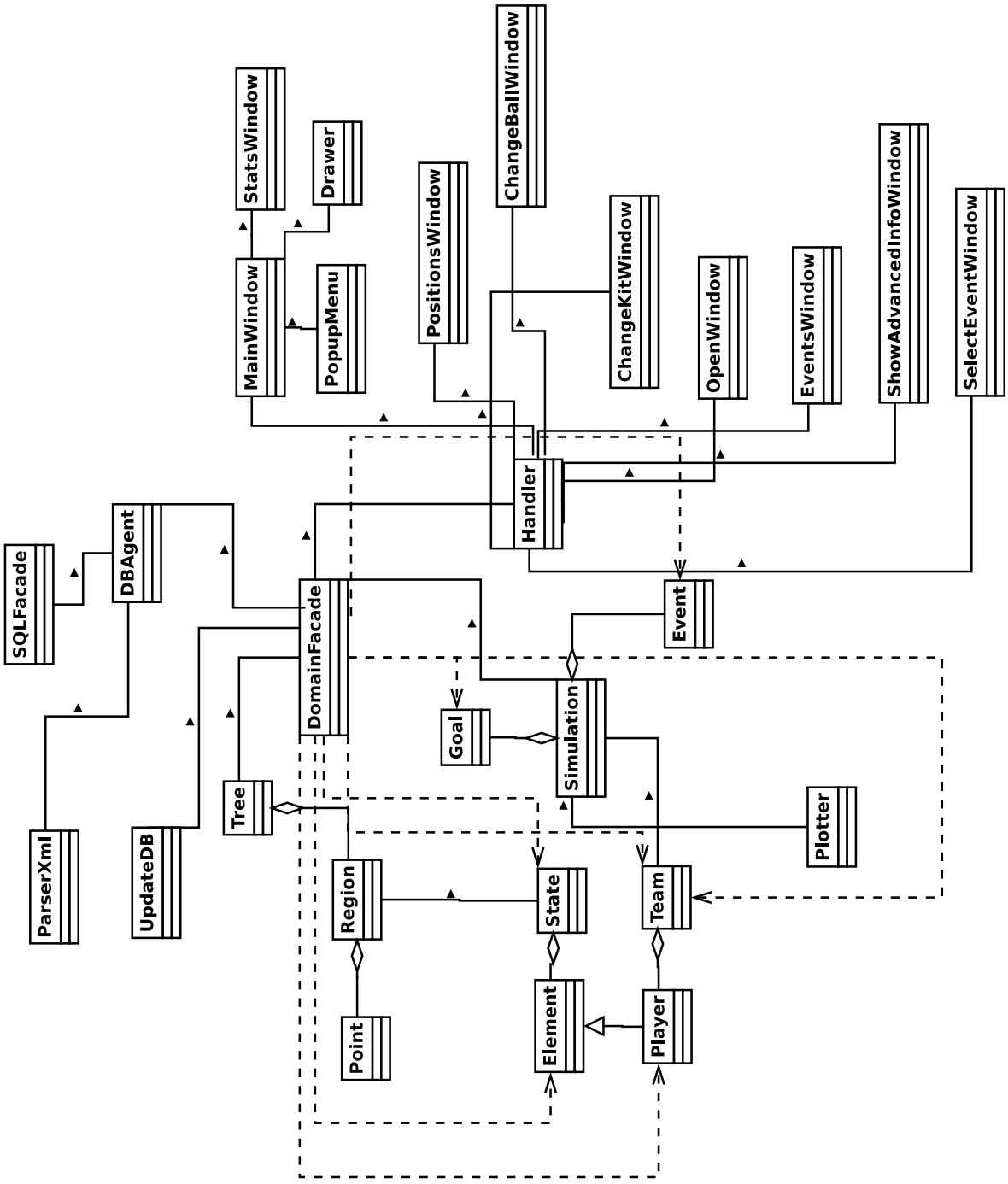


Figura 5.3: Diagrama de clases de la arquitectura

- Clase **Team**. Esta clase se compone de una lista de objetos de la clase *Element* (la longitud de esta lista será igual al número de jugadores que posea un equipo, típicamente once).
- Clase **Goal**. Esta clase contiene la información correspondiente a los goles de la simulación. Es decir, contendrá los atributos del ciclo en el cuál se produce y el resultado parcial de la simulación hasta ese momento.
- Clase **Event**. Esta clase contiene la información correspondiente a los eventos de la simulación. Es decir, contiene el ciclo en el cual se produce el evento, el tipo del evento, el número del jugador que está involucrado en la situación y el identificador del equipo al cuál pertenece dicho jugador.
- Clase **Plotter**. Esta clase contiene la funcionalidad necesaria para llevar a cabo la generación de los gráficos de los valores de las variables de cada uno de los jugadores de los equipos que se enfrentan.
- Clase **Simulation**. Esta clase almacena información y contiene la funcionalidad necesaria para gestionar el partido de fútbol. La clase se compone de una lista de goles (una lista de objetos de la clase *Goal*), otra de eventos (una lista de objetos de la clase *Event*), un generador de gráficos (un objeto de la clase *Plotter*), dos equipos (dos objetos de la clase *Team*) y un balón (un objeto de la clase *Element*). Entre la funcionalidad más destacada de esta clase encuentra cambiar de ciclo de simulación, determinar qué jugador tiene la posesión del balón en cada ciclo y analizar el número de oponentes cercanos a un jugador y determinar diferentes comportamientos colaborativos.
- Clase **UpdateDB**. Esta clase define la funcionalidad necesaria para actualizar la simulación con la información inferida por la clase *Simulation* (determinar quién tiene la posesión del balón, analizar el número de oponentes cercanos y determinar diferentes comportamientos colaborativos).
- Clase **DomainFacade**. Esta clase actúa de fachada entre las capas de dominio, persistencia e interfaz. La principal función de esta clase es la creación de los diferentes objetos que componen una simulación. Esta información se recibirá desde la base de datos, esta clase la procesará y creará los diferentes objetos. Además, cada información que la interfaz tenga que recuperar para mostrar al usuario deberá ser recuperada por esta clase.
- Clase **Parser_XML**. Esta clase contiene la funcionalidad necesaria para reconocer archivos *XML* y procesar la información que contienen. Cuando la información de un archivo *XML* se procesa, se introduce en la base de datos.
- Clase **SQLFacade**. Esta clase contiene la funcionalidad necesaria para interactuar con un esquema de bases de datos *MySQL*. Gracias a esta clase se podrán ejecutar las diferentes consultas a la base de datos.

- Clase **DBAgent**. Esta clase contiene la funcionalidad necesaria para la recuperación, inserción y modificación de la información de la base de datos. Esta clase contendrá un objeto de la clase *SQLFacade* que permitirá la ejecución de las consultas a la base de datos.
- Clase **Handler**. Esta clase contendrá la funcionalidad necesaria para gestionar las diferentes acciones que realice un usuario desde la interfaz gráfica. Esta clase determinará qué ventana debe ejecutar la funcionalidad exigida por el usuario. Además, el manejador posee un objeto de la clase *DomainFacade*, el cuál permitirá recuperar la información de la lógica de dominio para que se distribuya a la ventana correspondiente y se muestre al usuario.
- Clase **PopupMenu**. Esta clase contiene la funcionalidad necesaria para establecer el despliegue de un menú emergente cuando se pulsa sobre un jugador de la simulación. Las opciones con las que se muestra este menú serán configurables mediante parámetros. Durante la simulación, la utilización de este menú es para mostrar u ocultar la ventana de características de los jugadores o para determinar el tipo del jugador.
- Clase **Drawer**. Esta clase contendrá la funcionalidad necesaria para *renderizar* cada ciclo de la simulación.
- Clase **StatsWindow**. Esta clase se utilizará para mostrar las características (valores de las variables de los jugadores) en forma numérica.
- Clase **MainWindow**. Esta clase contendrá la funcionalidad necesaria para llevar a cabo las acciones requeridas por el usuario en la ventana principal de la aplicación. La clase se compondrá de un objeto de la clase *StatsWindow*, un objeto de la clase *Drawer* y otro objeto de la clase *PopupMenu*, en los cuales delegará distintas responsabilidades.
- Clase **PositionsWindow**. Esta clase contendrá la funcionalidad necesaria para mostrar al usuario la información del tipo de los jugadores y el número de los mismos. El *Handler* le proporcionará la información de cada uno de los jugadores de la simulación.
- Clase **ChangeBallWindow**. Esta clase contendrá la información necesaria para cambiar la imagen del balón. Cuando el usuario determina el balón, la ventana notificará al *Handler* y este a su vez, notificará el cambio al objeto de la clase *MainWindow*.
- Clase **ChangeKitsWindow**. La funcionalidad de esta clase es similar a la de la clase anterior. Sin embargo, en lugar de cambiar la imagen del balón, se cambiarán las equipaciones de los equipos.
- Clase **OpenWindow**. Esta clase contendrá la funcionalidad necesaria para albergar la tarea de cargar una simulación. Primero el objeto *Handler* creará esta ventana con la información de cabecera de una simulación (los equipos que se enfrentan, el resulta-

do y la fecha del encuentro). Luego, cuando un usuario selecciona una partido, esta ventana notificará al *Handler*, el cuál recuperará la información de dicha simulación.

- Clase **EventsWindow**. Esta clase contendrá la información necesaria para visualizar los eventos de la simulación. Cuando el usuario desea ver los eventos de la simulación, es el objeto *Handler* el que recuperará la información de los eventos y creará esta ventana con dicha información.
- Clase **SelectEventWindow**. Esta clase contendrá la funcionalidad necesaria para que un usuario pueda seleccionar el evento a monitorizar durante una simulación. Cuando un usuario selecciona el evento a monitorizar, esta clase notificará al *Handler*, el cuál notificará a la *DomainFacade*, la que cambiará el tipo de evento a monitorizar en el objeto de la clase *Simulation*.
- Clase **ShowAdvancedWindow**. Esta clase contendrá la funcionalidad necesaria para la generación de los gráficos de los valores de las variables de cada uno de los jugadores del partido. Cuando un usuario determine los gráficos a mostrar, esta clase notificará al objeto *Handler*, el cuál indicará la notificación a la *DomainFacade*, la que invocará a una función del objeto *Simulation*, siendo este último el que pase los parámetros correspondientes a la clase *Plotter* para la generación de los gráficos.

5.2. Módulo de Procesamiento de Información

Este módulo es el encargado de procesar la información correspondiente a una partido de fútbol. Los datos de los partidos de fútbol se encuentran en diferentes formatos. Los formatos son diferentes y variados, pueden ser audiovisuales, en forma de ficheros de texto, etcétera. Este módulo se organiza en base a un esquema de *parsers*, de tal manera que cuando quiere procesarse la información de un nuevo formato hay que crear un nuevo parser que reconozca dicha información. Así pues, el sistema puede adaptarse a diferentes formatos de entrada de información.

El sistema es capaz de procesar los datos de partidos de fútbol se encuentran almacenados en diferentes ficheros XML (como se puede observar en el anexo D, todos con la misma estructura. La información generada se almacena en las tablas correspondientes de la base de datos.

Actualmente, este módulo está implementado por la clase `Parser_Xml.py`, el cual, es el punto de entrada de información de la arquitectura. La clase utiliza un componente de la biblioteca estándar de *Python* con la que se puede reconocer fácilmente el lenguaje *XML*. El componente en cuestión se denomina `SAX` y permite recorrer secuencialmente un fichero *XML*, sin almacenarlo en memoria. La posibilidad de recorrer un fichero secuencialmente, sin almancenar en memoria la totalidad del contenido del fichero, es la característica es la que inclinó la balanza a la hora de determinar el componente que realizaría esta función.

5.2.1. Submódulo de normalización

Cuando se reconoce la información de un partido de fútbol es necesario normalizar la posición en la que se encuentran los jugadores en el terreno de juego. Este submódulo está planteado de igual manera que el módulo de procesamiento de la información, de tal forma que para un nueva definición de la posición de los jugadores, es necesario crear un nuevo método de normalización para dicha posición. Una vez que se ha realizado la normalización de cualquier formato, es necesario que los posiciones se encuentren en el intervalo [0, 1], de forma algebraica:

$$\forall Jugador \in Simulacion, Jugador.X \in [0,1] AND Jugador.Y \in [0,1]$$

Con la normalización, un jugador puede ser representado en cualquier terreno de juego simplemente ajustando unos parámetros. Además, se puede aproximar mejor la posición de un jugador en el terreno de juego simplemente observando su posición normalizada. Aplicando la normalización, la esquina superior izquierda del campo resulta ser el punto [0, 0] y la esquina inferior derecha el punto [1, 1].

Al analizar la información contenida en un fichero *XML* que se almacenará en la base de datos se determina que los datos correspondientes a las posiciones de los jugadores en el terreno de los jugadores oscilan de la siguiente manera:

$$\forall Jugador \in Simulacion, Jugador.X \in [-56,56] AND Jugador.Y \in [-39,39]$$

Esto es consecuencia de que la información proviene de un dominio concreto, como es el de *RoboCup Soccer 2D*, donde la esquina superior izquierda es el punto [-56. -39] y la esquina inferior derecha se corresponde con el punto [56, 39].

El pseudocódigo de este submódulo puede apreciarse en el listado 5.1

```
1  X_LIMIT -> 56.0
2  Y_LIMIT -> 39.0

4  Si nombreElemento == ''XPos'':
5       x_pos -> attribute.get(''XPos'') + X_LIMIT / (X_LIMIT * 2)
6  Si no Si nombreElemento == ''YPos'':
7       y_pos -> attribute.get(''YPos'') + Y_LIMIT / (Y_LIMIT * 2)
```

Listado 5.1: Pseudocódigo del submódulo de normalización de información

5.2.2. Submódulo de inserción de datos

Una vez se ha reconocido la información, es necesario introducirlos en un esquema de base de datos. Para la inserción se hará uso de los procedimientos de inserción del agente de

la base de datos.

En la clase *Parser_XML* esta funcionalidad se lleva a cabo cuando se detectan diferentes elementos. Dichos elementos son los siguientes:

- *Ball*: Se introduce la información perteneciente al balón. Esta información, se almacena en una estructura de datos auxiliar (que puede observarse en el listado de código 5.2), al mismo tiempo que la biblioteca va leyendo el fichero XML. En cada ciclo de la simulación, esta información es sobrescrita, lo que supone un manejo eficiente de la memoria.

```
1 class Ball:
2   def __init__(self, cycle = int(), simulation_id = int(), x_pos
       = str(), y_pos = str(), x_vel = str(), y_vel = str()):
3     self.cycle = cycle
4     self.simulation_id = simulation_id
5     self.x_pos = x_pos
6     self.y_pos = y_pos
7     self.x_vel = x_vel
8     self.y_vel = y_vel
```

Listado 5.2: Estructura de datos auxiliar que almacena información del balón

Los atributos indican lo siguiente:

- *Cycle*. Almacena el ciclo de simulación correspondiente.
- *Simulation_id*. Determina el identificador de la simulación. Este identificador es necesario para determinar el partido en el que se están insertando los datos.
- *X_pos*. Indica la posición en el eje *X* de coordenadas en un ciclo determinado, tanto del balón como de los jugadores.
- *Y_pos*. Indica la posición en el eje *Y* de coordenadas en un ciclo determinado, tanto del balón como de los jugadores.
- *X_vel*. Indica la velocidad en el eje *X* de coordenadas en un ciclo determinado, tanto del balón como de los jugadores.
- *Y_vel*. Indica la velocidad en el eje *Y* de coordenadas en un ciclo determinado, tanto del balón como de los jugadores.

- *Player*: Al igual que ocurría con la información del balón, para almacenar la información de un jugador, se utiliza otra estructura de datos auxiliar, además de la propia del balón, puesto que hay atributos que son los mismos para un jugador, que para el balón. La estructura de datos, puede observarse en el listado de código 5.3.

```
class Player:
  def __init__(self, cycle = int(), number = int(), simulation_id
      = int(), team_id = int(), stamina = str(), orientation =
     str(), angle = str()):
    self.number = number
    self.team_id = team_id
    self.stamina = stamina
    self.orientation = orientation
    self.angle = angle
```

Listado 5.3: Estructura de datos auxiliar que almacena información de los jugadores

Los atributos de esta estructura de datos auxiliar indican lo siguiente:

- *Number*. Determina el dorsal de un jugador.
- *Team_id*. Determina el identificador del equipo al cual pertenece un jugador.
- *Stamina*. Indica la resistencia restante de un jugador en un ciclo determinado.
- *Orientation*. Indica el ángulo que está girada la cabeza de un jugador en un ciclo determinado.
- *Angle*. Determina el ángulo de visión de jugador en un ciclo determinado.

Tanto la estructura de datos *Ball* como *Player* están empotradas en la clase *Parser_XML*. Estas estructuras almacenarán los datos contenidos en un fichero *XML*, pero no formarán parte de la lógica de dominio.

- *Team*: Cuando se detecta el cierre de un elemento de este tipo, se introduce el nombre del equipo correspondiente. A diferencia de los dos puntos anteriores, no se utiliza una estructura de datos auxiliar, puesto que para insertar un equipo sólo es necesario el nombre.
- *Score*: Indica que uno de los equipos ha marcado un gol.

5.3. Módulo de análisis de comportamientos

La función de este módulo es realizar el análisis de la información de una simulación y determinar el comportamiento correcto de los jugadores. Este módulo es el más importante del sistema, puesto que es aquí donde reside el conocimiento sobre los comportamientos a analizar y la lógica necesaria para ello. Este módulo se divide en dos submódulos que se detallan a continuación.

5.3.1. Submódulo de análisis basado en conocimiento experto

En este módulo se define un sistema experto basado en reglas difusas. El módulo posee las siguientes características:

- **Adaptabilidad**: Este módulo ofrece la posibilidad al usuario de editar sus propias reglas mediante el uso de lógica difusa. Así, la arquitectura puede adaptarse al cualquier evento que el usuario defina.
- **Extensibilidad**: Además de editar las reglas, este módulo permite la inclusión de nuevas reglas por parte del usuario. Así, un usuario que desee determinar si ocurren distintos eventos a los ya definidos puede definir nuevos comportamientos para comprobarlo.
- **Portabilidad**: El uso de este módulo permite que la plataforma donde se ejecute el módulo de análisis de comportamiento sea independiente. El usuario sólo debe definir las reglas en un lenguaje estándar independiente de la misma.

A diferencia de los demás módulos, este submódulo se ha implementado en Java debido a las ventajas que ofrece el uso de la biblioteca *JFuzzyLogic*. El uso de esta biblioteca permite alcanzar las características deseables de este submódulo.

El diagrama de secuencia 5.4 muestra el proceso que sigue este módulo para generar los eventos e introducirlos en la base de datos.

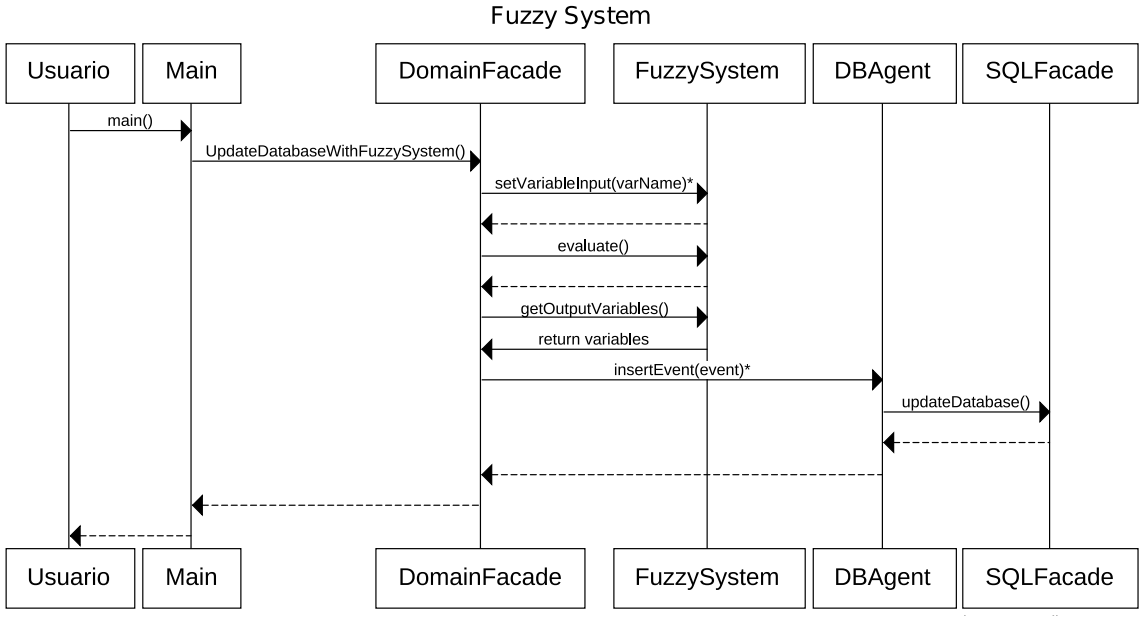


Figura 5.4: Diagrama de secuencia de módulo de lógica difusa

El diagrama de clases que define el diseño de este módulo es el que se puede apreciar en la figura 5.5. En dicha figura, se pueden observar diferentes componentes; por un lado, existe el sistema difuso, que cargará el conocimiento definido en la base de conocimiento y, por otro, estará la interacción con el agente de la base de datos, cuya misión es la de proporcionar e introducir en la base de datos la información, requerida o inferida por dicho

sistema difuso.

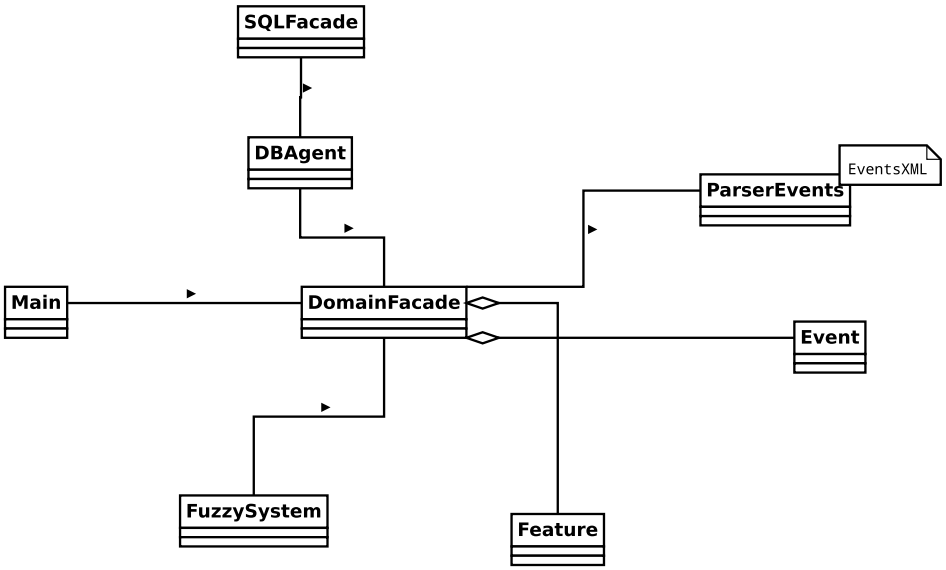


Figura 5.5: Diagrama de clases del módulo del sistema difuso

En el diagrama de clases (ver figura 5.5) se puede apreciar que la fachada de dominio posee dos sistemas de lógica difusa. Esto se debe a que, con el objetivo de facilitar la introducción de conocimiento por parte del usuario, se ha definido una arquitectura de dos niveles de conocimiento. Los niveles, así cómo su cometido quedan definidos a continuación:

Primer Nivel

En este nivel se define el conocimiento más básico y que se encuentra a mayor distancia del lenguaje del usuario experto en fútbol. Es decir, en este punto de la arquitectura se definen las bases sobre las que se asentará el resto del sistema difuso. En este nivel se pueden definir variables que serán utilizadas en otros niveles. En el listado 5.4 se puede ver cómo se definen las variables de **actitud**, **la presión de un jugador** y las **zonas de centro y pase de la muerte** en el terreno de juego.

```
1  DEFUZZIFY Attitude
2            TERM Defensive := (0, 0) (0, 1) (0.25, 1) (0.50, 0);
3            TERM Neutral := (0.25, 0) (0.50, 1) (0.75, 0);
4            TERM Offensive := (0.50, 0) (0.75, 1) (1, 1) (1, 0);
5            METHOD : COG;
6            DEFAULT := 0;
7  END_DEFUZZIFY

9  DEFUZZIFY Pressure
10           TERM Low := (0, 0) (0, 1) (0.25, 1) (0.50, 0);
11           TERM Medium := (0.25, 0) (0.50, 1) (0.75, 0);
12           TERM High := (0.50, 0) (0.75, 1) (1, 1) (1, 0);
13           METHOD : COG;
14           DEFAULT := 0;
```

```
15 END_DEFUZZIFY


18 (* ------------------Attitude Rules--------------*)
19 RULE 1: IF (X_Pos IS VeryLeft) OR (X_Pos IS Left) THEN Attitude IS
      Defensive;
20 RULE 2: IF (X_Pos IS Medium) THEN Attitude IS Neutral;
21 RULE 3: IF (X_Pos IS Right) OR (X_Pos IS VeryRight) THEN Attitude IS
      Offensive;

23 (* ------------------Pressure Rules--------------*)
24 RULE 4: IF (Possession IS Yes) AND (Nearly_players IS High) THEN
      Pressure IS High;
25 RULE 5: IF (Possession IS Yes) AND (Nearly_players IS Medium) THEN
      Pressure IS Medium;
26 RULE 6: IF (Possession IS Yes) AND (Nearly_players IS Low) THEN
27 Pressure IS Low;
28 (*------------------Center Zone------------------*)
29 RULE 7: IF ((X_Pos IS VeryLeft OR X_Pos IS Left) OR (X_Pos IS
      VeryRight OR X_Pos IS Right)) AND (Y_Pos IS VeryDown OR Y_Pos IS
      VeryUp) THEN CenterZone IS Yes;

31 (*----------------Killer Pass Zone-------------------*)
32 RULE 8: IF ((X_Pos IS VeryLeft OR X_Pos IS Left) OR (X_Pos IS
      VeryRight OR X_Pos IS Right)) AND (Y_Pos IS Down OR Y_Pos IS Up)
      THEN KillerPassZone IS Yes;
```

Listado 5.4: Definición de variables Actitud y Presión

Las reglas de la actitud se establecen en base a las particiones difusas que se realizan sobre el terreno de juego. Las particiones del eje *X* de coordenadas son las siguientes:

- *VeryLeft*. Esta partición difusa indica que un jugador se encuentra muy a la izquierda del terreno de juego. Esta partición comprende las zonas de la portería y el área del equipo que ataca de izquierda a derecha.
- *Left*. Esta partición difusa indica que un jugador se encuentra a la izquierda del terreno de juego. Esta partición comprende la zona de tres cuartos del equipo que ataca de izquierda a derecha.
- *Medium*. Esta partición difusa indica que un jugador se encuentra en el medio del terreno de juego. Esta partición comprende la zona del centro del campo.
- *Right*. Esta partición difusa indica que un jugador se encuentra a la derecha del terreno de juego. Esta partición comprende las zonas de tres cuartos del equipo que ataca de derecha a izquierda.
- *VeryRight*. Esta partición difusa indica que un jugador se encuentra muy a la derecha del terreno de juego. Esta partición comprende las zona del área y de la portería del equipo que ataca de derecha a izquierda.

. En la Las particiones del eje *Y* de coordenadas son las siguientes: figura 5.6 se pueden observar dichas particiones difusas:

- *VeryUp*. Esta partición difusa indica que un jugador se encuentra pegado en la banda izquierda del terreno de juego.
- *Up*. Esta partición difusa indica que un jugador se encuentra en la franja superior del terreno de juego.
- *Medium*. Esta partición difusa indica que un jugador se encuentra en la franja media del terreno de juego.
- *Down*. Esta partición difusa indica que un jugador se encuentra en la franja inferior del terreno de juego.
- *VeryDown*. Esta partición difusa indica que un jugador se encuentra pegado a la banda derecha del terreno de juego.

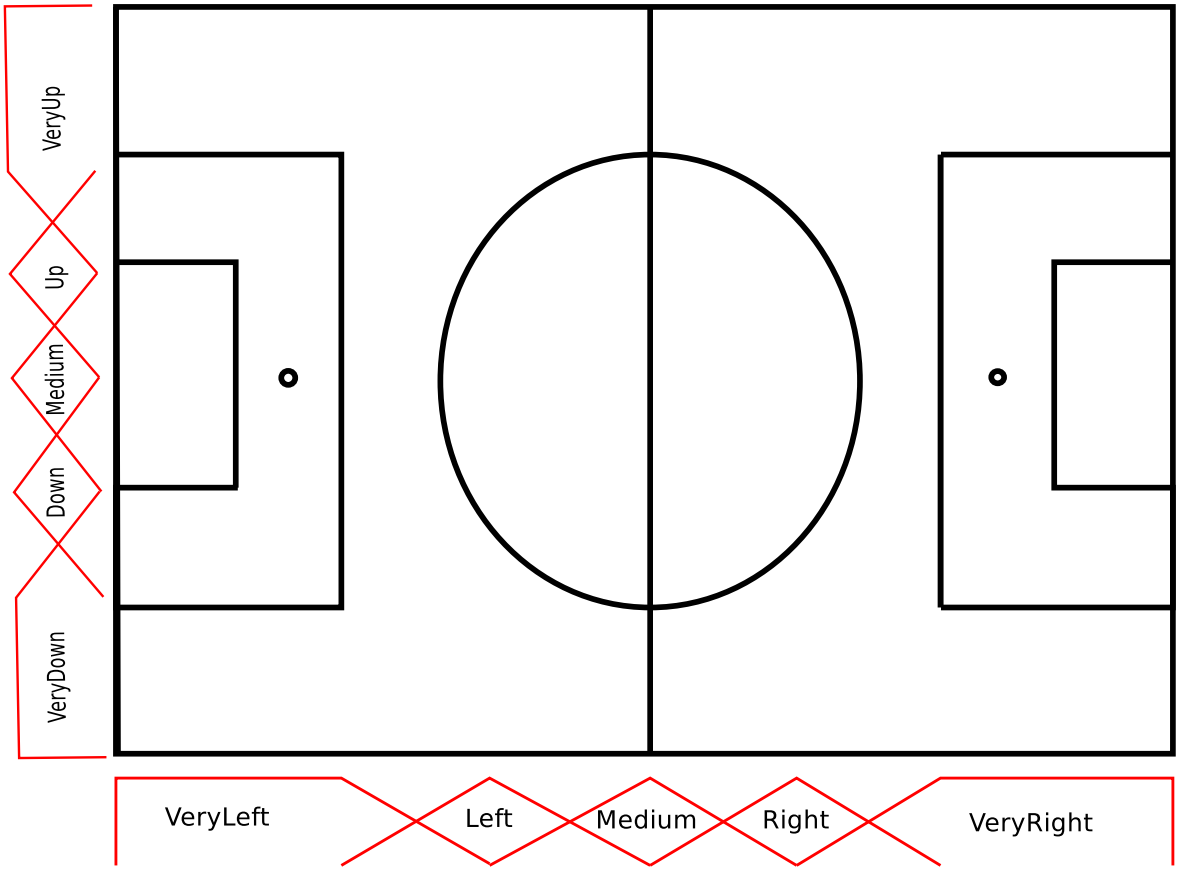


Figura 5.6: Partición difusa del terreno de juego

Por otro lado, la interpretación de las reglas de la presión es sencilla, simplemente, si un jugador tiene la posesión del balón la presión que sufre el mismo será mayor cuantos más oponentes se encuentren cerca de él y, será menor cuantos menos oponentes se encuentren cerca de él. La presión se define con tres conjuntos difusos que son los siguientes:

- *Low*. Este conjunto indica que la presión sufrida por un jugador que tiene la posesión del balón es reducida. La presión será reducida cuando el número de oponentes

cercanos sea uno o ninguno.

- *Medium*. Este conjunto indica que la presión sufrida por un jugador que tiene la posesión del balón es media. La presión será media cuando el número de oponentes cercanos sean uno o dos.
- *High*. Este conjunto indica que la presión sufrida por un jugador que tiene la posesión del balón es elevada. La presión será elevada cuando el número de oponentes cercanos sean dos o más.

La zona del centro se define, al igual que la actitud, utilizando las particiones difusas del terreno de juego. Sin embargo, para la definición de las zonas se utiliza, también, la partición difusa del eje *Y*. De esta manera cuando un jugador se encuentre a la derecha o a la izquierda del terreno de juego y, además, se encuentre cercano a cualquiera de las bandas del terreno de juego, se encontrará en la zona de centrar.

De la misma manera ocurre con la zona de pase de la muerte. Así, cuando un jugador se encuentre a la derecha o a la izquierda del terreno de juego y, además, se encuentre en la franja superior o inferior del terreno de juego, se encontrará en la zona de pase de la muerte.

Segundo nivel

En este nivel las reglas definidas están más cercanas al lenguaje de los expertos en fútbol, y se pueden encontrar expresiones como *cambio de juego*, *robo crítico* o *disparo a portería*. Así, también se pueden encontrar conclusiones sobre qué debe hacer, idealmente, un jugador en un momento determinado, dependiendo del valor de las variables. Las variables definidas en el primer nivel de la arquitectura pueden ser utilizadas en este nivel para determinar nuevos eventos a detectar. En el listado 5.5 se pueden apreciar las reglas en las que se basa la inferencia de nuevos valores para las variables. En dicho listado, vemos cómo la mayoría de las reglas utilizan alguna de las dos variables (actitud y presión) o, en ocasiones, las dos, definidas en el primer nivel.

```
2  DEFUZZIFY CriticalTheft
3          TERM Yes := 1;
4          TERM No := 0;
5          METHOD: COGS;
6          DEFAULT := 0;
7  END_DEFUZZIFY

9  DEFUZZIFY Shoot
10         TERM Yes := 1;
11         TERM No := 0;
12         METHOD: COGS;
13         DEFAULT := 0;
14 END_DEFUZZIFY

16 DEFUZZIFY ChangeGame
17           TERM Yes := 1;
```

```
18            TERM No := 0;
19            METHOD: COGS;
20            DEFAULT := 0;
21  END_DEFUZZIFY

23  DEFUZZIFY Center
24            TERM Yes := 1;
25            TERM No := 0;
26            METHOD: COGS;
27            DEFAULT := 0;
28  END_DEFUZZIFY

30  DEFUZZIFY KillerPass
31            TERM Yes := 1;
32            TERM No := 0;
33            METHOD: COGS;
34            DEFAULT := 0;
35  END_DEFUZZIFY

37  (*------------------Critic Theft---------------*)
38  RULE 1: IF (Possession IS Yes) AND (Pressure IS High) AND (Attitude IS
        Defensive) THEN CriticalTheft IS Yes;

40  (*------------------Shoot Success----------*)
41  RULE 6: IF (Possession IS Yes) AND (Pressure IS Low) AND (Attitude IS
       Offensive) THEN Shoot IS Yes;

43  (*------------------Change Game----------------*)
44  RULE 7: IF (Possession IS Yes) AND (Attitude IS Neutral) AND (
       MidfieldOpponents IS Low) THEN ChangeGame IS Yes;

46  (*-----------------Center Action---------------*)
47  RULE 9: IF (Possession IS Yes) AND (CenterZone IS Yes) AND (Attitude
       IS Offensive) THEN Center IS Yes;

49  (*-----------------Killer Pass Action----------*)
50  RULE 10: IF (Possession IS Yes) AND (KillerZone IS Yes) AND (Attitude
       IS Offensive) THEN KillerPass IS Yes;
```

Listado 5.5: Definición de variables Actitud y Presión

En el listado anterior se definen las variables difusas de las diferentes situaciones a detectar. Todas estas variables son definidas con dos valores, por ello a estas variables se les denomina *singleton*. Las variables son las siguientes:

- *CriticalTheft*. Esta variable determina la situación en que se puede producir un robo crítico del balón. Esta situación se produce cuando un jugador tiene el balón, su actitud es defensiva y la presión es elevada.
- *Shoot*. Esta variable determina la situación en que se puede producir un disparo a portería. Esta situación se produce cuando un jugador tiene el balón, su actitud es ofensiva y la presión es reducida.

- *ChangeGame*. Esta variable determina la situación en que se puede producir un cambio de juego. Esta situación se produce cuando un jugador tiene el balón, su actitud es neutral y el número de jugadores en la franja del campo distinta a la que se encuentra el jugador es reducido.
- *Center*. Esta variable determina la situación en que se puede producir un centro. Esta situación se produce cuando un jugador tiene el balón, se encuentra en zona de centrar, y su actitud es ofensiva.
- *KillerPass*. Esta variable determina la situación en que se puede producir un pase de la muerte. Esta situación se produce cuando un jugador tiene el balón, se encuentra en zona de pase de la muerte, y su actitud es ofensiva.

5.3.2. Submódulo de análisis de comportamientos colaborativos

. A diferencia del módulo anterior, este módulo está implementado en *Python* y no define un sistema experto basado en reglas. Uno de los inconvenientes que surgieron a la hora de utilizar sistemas expertos basados en reglas fue dificultad de definir reglas que determinen el comportamiento colectivo. Debido a esto, se optó por comportamientos mediante código imperativo, ya que resulta más sencillo especificarlos procedimientos secuenciales. Los eventos implementados por este submódulo se encuentran en la clase `Simulation.py`.

A continuación, se detallan los eventos implementados de esta manera.

Línea mal tirada del fuera de juego

Cuando todos los defensores de un equipo se encuentran en línea, se dice que la línea del fuera de juego está bien tirada. Sin embargo, si un defensor se encuentra significativamente más retrasado que la línea que forman los otros defensores del equipo, se dice que la línea del fuera de juego está mal tirada. Una situación de este tipo puede observarse en la figura 5.7.

El método que implementa esta situación es el que se puede apreciar en el listado 5.6.

```
1  def check_correct_line_of_offside(self, cycle):
2      OFFSIDE_DISTANCE = 0.05
3      players_team_left = self.team_left.get_data_cycle(cycle)
4      players_team_left.sort()
5      # The first one should be the goalkeeper.
6      if (players_team_left[2][0] - players_team_left[1][0]) >=
           OFFSIDE_DISTANCE:
7          return self.team_left.id, players_team_left[1][2]
8      players_team_right = self.team_right.get_data_cycle(cycle)
9      players_team_right.sort()
10     if (players_team_right[len(players_team_right) - 2][0] -
           players_team_right[len(players_team_right) - 3][0]) >=
           OFFSIDE_DISTANCE:
11         return self.team_right.id, players_team_right[len(
               players_team_right) - 3][2]
```

Figura 5.7: Línea mal tirada del fuera de juego

```
12        return -1, -1
```

Listado 5.6: Método que determina cuando la línea de juego está mal tirada

El método obtiene la posición en el eje *X* de los jugadores de los dos equipos en un ciclo determinado. Estas posiciones son ordenadas de menor a mayor. Se obtiene la distancia que existe entre el segundo y tercer jugador, ya que el primer jugador que resulta ser el portero no interviene en la línea del fuera de juego. Si la distancia es significativa, entonces se devuelve el dorsal del jugador que causa el mal trazo en la línea.

Pasar a un compañero más adelantado desmarcado

Cuando un jugador que tiene la posesión del balón puede determinar si existe un compañero en un posición más adelantada y además libre de marca, en ese caso, el jugador con el balón puede pasárselo. En la figura 5.8 se puede observar como el jugador que tiene la

posesión del balón puede pasar el balón a un compañero más adelantado desmarcado.

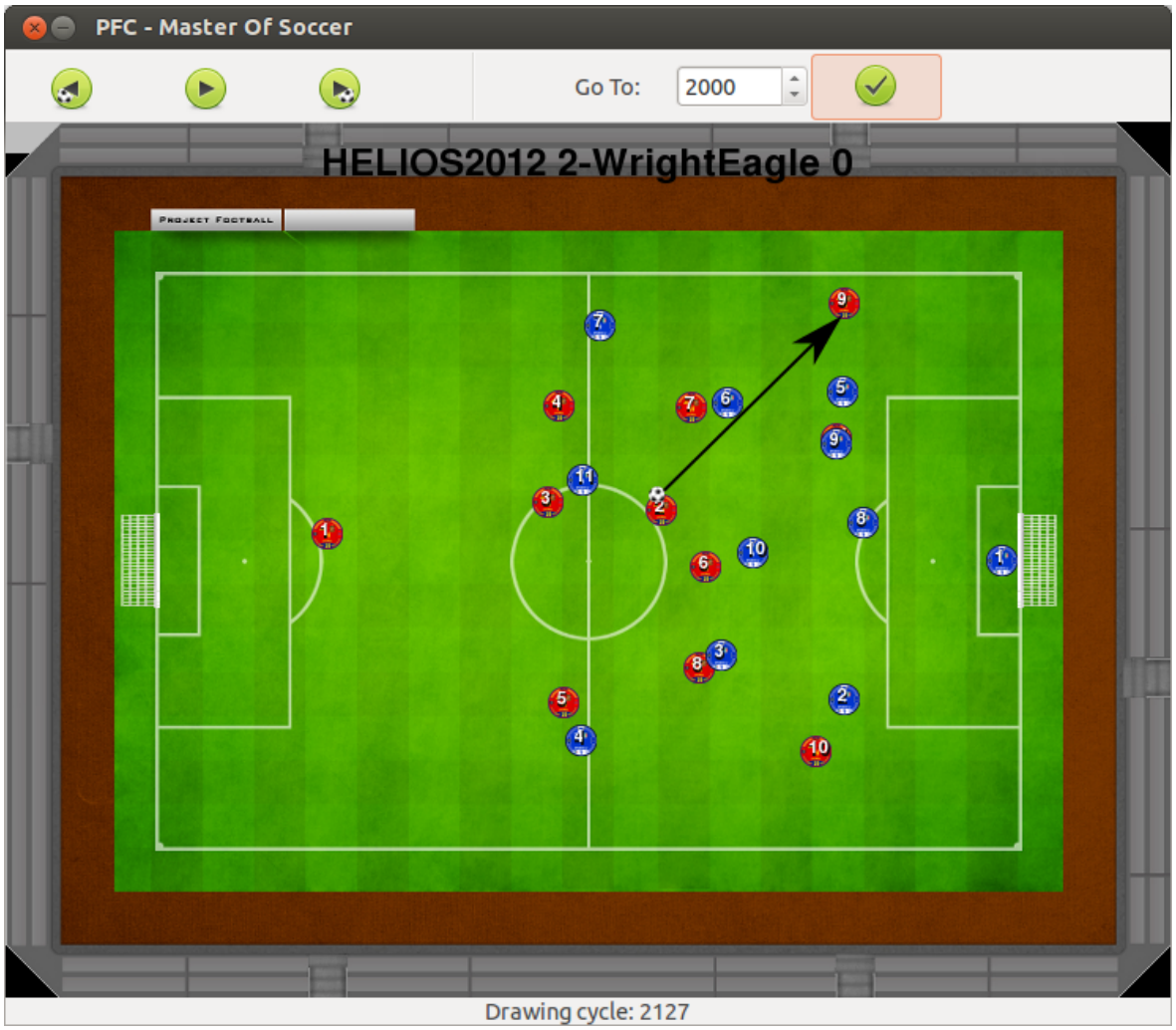


Figura 5.8: Posible pase a un compañero más adelantado desmarcado

El método que implementa esta situación es el que se puede apreciar en el listado 5.7

```
def pass_to_only_teammate(self, cycle, team_id, number_of_player):
    player_data = []
    team_data = []
    result_player = -1
    distance_to_player = MAX_DISTANCE
    if (team_id % 2) == 1:
        player_data = self.team_left.get_player(number_of_player,
            cycle)
        team_data = self.team_left.get_data_cycle(cycle)
    else:
        player_data = self.team_right.get_player(number_of_player,
            cycle)
        team_data = self.team_right.get_data_cycle(cycle)
        team_data = self.__convert(team_data)
    if player_data[11] == 1:
        # If player has the ball...
        for player in team_data:
        if (player[0] > player_data[0]) and (player[12] == 0):
            player_point = Point.Point(player_data[0], player_data[1])
            distance = player_point.distance(player[0], player[1])
            if distance < distance_to_player:
                #if xPos is greater than
                #player with ball possession
                #xPos then...
                #should pass the ball to that player
                result_player = player[2]
                distance_to_player = distance
    return result_player
```

Listado 5.7: Método que determina cuándo hay que pasar a un compañero más adelantado desmarcado

Primero, el método determina el equipo del jugador que tiene el balón (que resulta ser el que se pasa por argumento al método) y se obtienen los datos de los jugadores del mismo en un ciclo determinado. Para cada jugador, si el compañero está más adelantado (si el valor de su posición en el eje *X* es mayor que el del jugador que posee el balón) y está desmarcado, entonces se evalúa la distancia que separa a ambos y se anota. Si la distancia que los separa, es menor que la distancia mínima obtenida para otro jugador, entonces el jugador del balón deberá pasar el balón a dicho compañero.

Determinar la pérdida de posesión

Cuando un jugador pierde la posesión del balón debe reaccionar. La primera reacción que puede realizar el jugador es la moverse. El siguiente método comprueba si un jugador ha perdido la posesión.

```
def check_player_unmarked(self, cycle, team_id, number_of_player):
    res = None
    if (team_id % 2) == 1:
        player_data = self.team_left.get_player(number_of_player,
```

```
            cycle)
    else:
        player_data = self.team_right.get_player(number_of_player,
            cycle)
     possession_threshold = self.get_possession_in_threshold(team_id,
        number_of_player, cycle, 5)
     if possession_threshold != None:
         if player_data[11] == 1 and possession_threshold == False:
             res = "Yes"
     return res
```

Listado 5.8: Método que determina la pérdida de posesión

Al igual que el método anterior, el método determina el equipo, el cual pertenece el jugador que se pasa cómo argumento al método. Cuando esto ocurre, se obtienen los datos correspondientes a dicho jugador. Además, se recupera el dato de si el jugador tiene la posesión unos ciclos más adelante. Si en el ciclo actual el jugador tiene la posesión y dentro de un tiempo no la tiene, significa que el jugador ha perdido el balón o lo ha pasado o a disparado a puerta. Sea cual sea la situación, el jugador deberá moverse. Mediante el método del listado5.9 se determina en tiempo de simulación si el jugador se ha movido o no.

```
def __check_movement(self, cycle, number_of_player, team_id):
    if (team_id % 2) == 1:
        player_data = self.team_left.get_player(number_of_player,
            cycle)
    else:
        player_data = self.team_right.get_player(number_of_player,
            cycle)
    x, y = self.__get_position_threshold(cycle, number_of_player,
        team_id, 10)
    if x != None and y != None:
        p1 = Point.Point(player_data[0], player_data[1])
        if p1.distance(x, y) >= 0.05:
            return "Right"
    return "Wrong"
```

Listado 5.9: Método que analiza si un jugador se ha movido

Para determinar si un jugador realiza un movimiento, simplemente, se recogen datos sobre la posición del jugador en un ciclo y la posición en algunos ciclos después. Si la distancia existente entre dos puntos no es significativa, entonces se determina que el jugador no se ha movido.

Este módulo también es el encargado de obtener los valores de las diferentes variables. Las variables que se obtienen de los datos de la simulación son las siguientes:

- **Posesión del balón**. Esta variable determina qué jugador tiene la posesión del balón en un ciclo determinado, en caso que la tenga alguno. Para implementar un método que refleje inteligentemente qué jugador tiene la posesión del balón se ha utilizado una estrategia utilizando estructuras de datos basadas en árboles [AMHH04]. La estrategia se basa en *árboles octales*. El algoritmo idealmente está basado para el *rendering 3D*,

pero se ha adaptado de manera que pueda utilizarse en un terreno de juego de dos dimensiones. El algoritmo consiste en que una caja es *cortada* simultáneamente por sus tres ejes, formando ocho cajas de menor área, con lo que se consigue una estructura regular que puede hacer que la recuperación de la información sea más eficiente.

El algoritmo empleado para la obtención eficiente del jugador que tiene la posesión del balón se apoya en la siguiente idea básica. La posesión del balón sólo la puede tener un jugador que se encuentre muy próximo al balón, de manera que resulta imposible que un jugador que se encuentre alejado del balón tenga la posesión del mismo. Para determinar qué jugadores se encuentran cercanos al balón de una manera eficiente, se divide el terreno de juego en varias regiones. Las regiones de terreno de juego resultan de cortar el mismo en sus diferentes ejes (véase la figura 5.9).

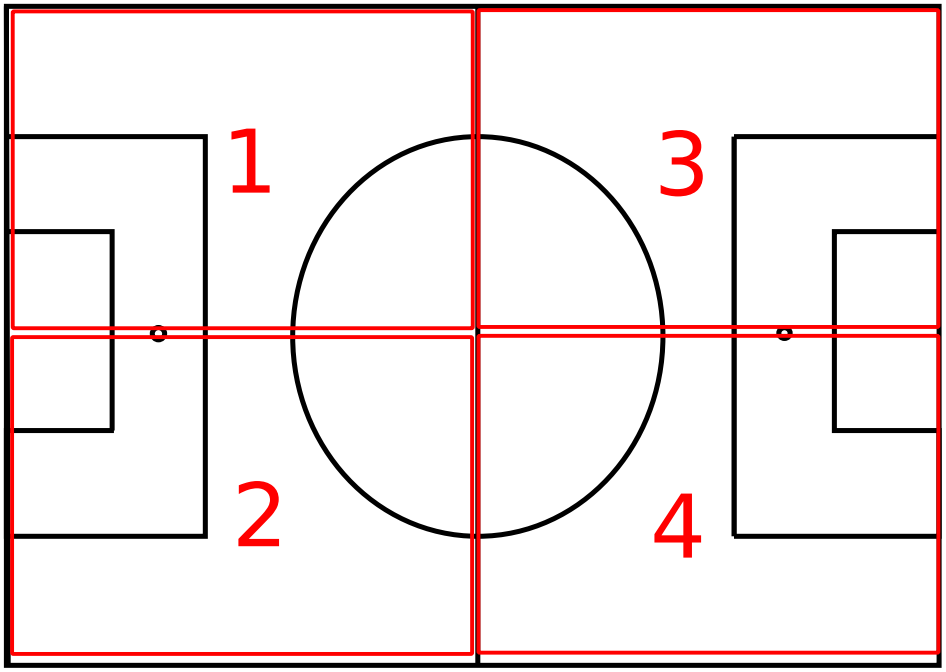


Figura 5.9: Terreno de juego divido en regiones

Cuando la simulación se carga, a cada jugador se le asigna una región, dependiendo de su posición. Para determinar que jugador tiene la posesión, bastará con analizar los jugadores de la región dónde se encuentre el balón, descartando el resto de jugadores del terreno de juego. De esta manera, el número de comprobaciones se reduce significativamente. El código que realiza este comportamiento es el que se puede encontrar en el listado 5.10.

```
1  def get_possession(self, cycle):
2    res = None
3    distance_cmp = MAX
4    ball_data = self.ball.get_data_cycle(cycle)
5    ball_region = ball_data[4]
6    players_in_region = self.__get_players_in_region(ball_region,
       cycle)
```

```
7     for player in players_in_region:
8       player_point = Point.Point(player.states[0][0], player.states
            [0][1])
9       distance = player_point.distance(ball_data[0], ball_data[1])
10      if distance <= MIN_POSS_DIST:
11        if distance < distance_cmp:
12          distance_cmp = distance
13          res = player
14    return res
```

Listado 5.10: Método que determina el jugador que posee la posesión del balón

En la figura 5.10 se puede observar como es el jugador número 6 del equipo rojo quién tiene la posesión del balón.

- **Jugadores oponentes cercanos**. Esta variable determina el número de jugadores oponentes que se encuentran en los alrededores de un jugador determinado. De esta manera, se determina si un jugador está marcado no. A diferencia del caso anterior, obtener el valor de esta variable no se puede optimizar de la misma manera, puesto que los jugadores cercanos a un determinado jugador pueden encontrarse en otras regiones y al realizar la comprobación por regiones no se considerarían. El código que obtiene el valor de esta variable se puede apreciar en el listado 5.11.

```
1   def get_nearly_players(self, cycle, number_of_player, team):
2     res = 0
3     player_data = []
4     if (team % 2) == 1:
5       player_data = self.team_left.get_player(number_of_player,
            cycle)
6       for player in self.team_right.get_data_cycle(cycle):
7         player_point = Point.Point(player_data[0], player_data[1])
8         distance = player_point.distance(player[0], player[1])
9         if distance <= 0.1:
10          res += 1
11    else:
12      player_data = self.team_right.get_player(number_of_player,
            cycle)
13      for player in self.team_left.get_data_cycle(cycle):
14        player_point = Point.Point(player_data[0], player_data[1])
15        distance = player_point.distance(player[0], player[1])
16        if distance <= 0.1:
17          res += 1
18    return res
```

Listado 5.11: Método que determina el número de jugadores cercanos

En la figura 5.11 se puede observar como el jugador número 11 del equipo rojo tiene oponentes cercanos.

- **Oponentes en distinta franja del terreno de juego con respecto a un jugador**. Esta variable se calcula para poder inferir cuándo hay que realizar un cambio de juego. Al estar el terreno de juego normalizado, la forma de calcular el valor de esta variable resulta bastante sencillo, puesto que el valor que identifica la mitad de un campo es 0.5. El código que realiza esta función es el que se puede apreciar en el listado 5.12

Figura 5.10: El jugador número 6 tiene la posesión del balón

Figura 5.11: El jugador número 11 tiene 3 oponentes cercanos

```
def __get_number_of_opponents(self, midfield, opponents):
    res = 0
    for player in opponents:
        opponent_midfield = self.__get_midfield(player[1])
        if midfield != opponent_midfield:
            res += 1
    return res
```

Listado 5.12: Método que determina el número de jugadores en el campo contrario

En la figura 5.12 se puede observar como hay 7 jugadores del equipo azul en la franja inferior del campo.

5.4. Módulo de representación de la información

La finalidad de este módulo consiste en presentar al usuario la información de una manera atractiva y sencilla. Este módulo está compuesto por dos submódulos que cooperan para representar la información de una forma dinámica.

Este módulo se compone de prácticamente la totalidad del código fuente dedicado a la interfaz gráfica de usuario. El módulo está escrito en **Python** utilizando la biblioteca **PyGTK** que, a su vez, interacciona con la biblioteca *PyGame*.

5.4.1. Submódulo de representación de la simulación

Para la representación de la simulación se utiliza un submódulo, porque es necesaria la combinación de dos tecnologías distintas. Una de ellas es la biblioteca mencionada anteriormente, *PyGTK*. La otra es la biblioteca *PyGame*. La interacción entre estas dos bibliotecas no es sencilla, puesto que cada una tiene un objetivo específico bien definido. *PyGTK* se utiliza para crear interfaces gráficas, en la que la interacción del usuario se basa en la navegación por diferentes ventanas. Mientras que, con *PyGame*, la interacción con el usuario, típicamente, se basa en una ventana en la que se muestran los gráficos. Para aclarar la diferencia existente entre las dos bibliotecas, se propone el siguiente ejemplo. Un programa comercial de gestión de una organización sería más adecuado que estuviera realizado utilizando *PyGTK*; sin embargo, un juego sería común que estuviera implementado utilizando *PyGame*.

Por tanto, como las posibilidades que ofrecen las dos bibliotecas son necesarias para el desarrollo de la herramienta, es necesaria su integración. Para ello, es necesario un *hack* puesto que *PyGTK* no nació con el objetivo de albergar otras tecnologías. Así pues, dentro de la ventana principal de la herramienta se genera dinámicamente una área para pintar (un componente de la biblioteca *PyGTK* llamado *DrawingArea*) que contendrá la funcionalidad de la biblioteca *PyGame*. En el desarrollo del sistema, sobre el componente *DrawingArea* de *PyGtk* se dibuja el campo de juego y cada uno de los jugadores, así como el resultado actual.

Figura 5.12: Siete jugadores en la franja inferior del medio campo

Figura 5.13: Pantalla principal con una simulación cargada

En la figura 5.13 se puede observar una captura de pantalla de la aplicación. En el componente *DrawingArea* se habilitan los controladores para los distintos eventos de usuario, como son el de pulsación de botones y el de movimiento del ratón. Al ser este área un componente de *PyGtk*, el resto de la interfaz gráfica puede interactuar con el, y este a su vez notificar las diferentes acciones a la biblioteca *PyGame*.

5.4.2. Submódulo de gestión de controles de usuario

Este submódulo se encarga de manejar la interactividad del usuario con la interfaz gráfica y mostrar la información que dicho usuario requiera.

La implementación de este submódulo pasa, en su mayoría, por la clase *Handler*, que se encarga de manejar la mayor parte de las interacciones del usuario con la interfaz. Esta clase contiene un objeto llamado *builder* que cargará el fichero de definición de la interfaz gráfica generado por *Glade*. Este fichero contiene la totalidad de la definición de la interfaz gráfica. El objeto *builder* proporciona métodos para realizar las siguientes acciones:

- Añadir el fichero de definición de la interfaz gráfica. Este fichero será un XML creado por la aplicación *Glade*. El método que realiza esta acción se denomina *add_from_file()*.

- Conectar las señales. Esta acción permite definir la petición que genera la producción del evento de diferentes acciones como hacer click sobre un botón, interactuar con la opción de un menú o cerrar una ventana, entre otras. El método en cuestión se denomina *connect_signals()*.
- Tener acceso a los elementos definidos en fichero. Este comportamiento es especialmente útil, puesto que una invocación al método *get_object()*, devuelve el objeto correspondiente con todos sus métodos y sus propiedades, ya sean, ventanas, botones, opciones del menú, etcétera.

En el listado de código 5.13 se puede apreciar como se instancia el objeto *builder*, se añade el fichero de definición de la interfaz gráfica, cómo se conectan las señales y por último cómo se obtiene el objeto definido con el nombre *show_advanced_info* en el fichero de definición.

```
1 class Handler:
2     def __init__(self):
3         filename = "../interface/mos_interface.glade"
4         self.builder = gtk.Builder()
5         self.builder.add_from_file(filename)
6         self.builder.connect_signals(self)
7         ...
8         self.show_advanced_info_option = self.builder.get_object("
              show_advanced_info")
```

Listado 5.13: Objeto *builder*

De la misma manera que se obtiene el elemento *show_advanced_info* mediante el objeto *builder*, se obtienen el resto de elementos que componen la interfaz gráfica. Los elementos más relevantes, de cara al usuario, son las ventanas de la aplicación. Las ventanas que componen la aplicación y que se obtienen con el *builder* son las siguientes:

- Ventana principal de la aplicación. La ventana principal de la aplicación será la que se genere al iniciar la aplicación. La clase que contiene la funcionalidad de esta ventana se denomina *MainWindow*. En la figura 5.14 se puede observar la ventana principal de la aplicación.
- Ventana de cambio de balón. La funcionalidad de esta ventana es permitir al usuario cambiar la apariencia del balón. La ventana está construida con una caja de combo, que incluye los posibles balones a seleccionar. Bajo la caja de combo, se encuentra una pequeña imagen que determina el balón seleccionado. La imagen del balón cambia cuándo se detecta el evento *changed* de la caja de combo. La clase que soporta la funcionalidad de esta ventana a se denomina `ChangeBallWindow`. En el listado de código 5.14 pueden apreciarse los métodos que manejan tanto, la gestión del evento cómo la adición de los balones a la caja de combo.

Figura 5.14: Pantalla principal

```
1  def combo_changed(self):
2    active_text = self.combo.get_active_text()
3    self.image.set_from_file(balls[active_text])

5  def add_combo_info(self):
6      self.combo.clear()
7      liststore = gtk.ListStore(str)
8      for entry in balls.keys():
9          liststore.append([entry])
10     self.combo.set_model(liststore)
11     render = gtk.CellRendererText()
12     self.combo.pack_start(render, True)
13     self.combo.add_attribute(render, 'text', 0)
14     self.combo.set_active(3)
```

Listado 5.14: Métodos de la ventana `ChangeBallWindow`

En la figura 5.15 se detalla el proceso del cambio de balón, en forma de diagrama de secuencia.

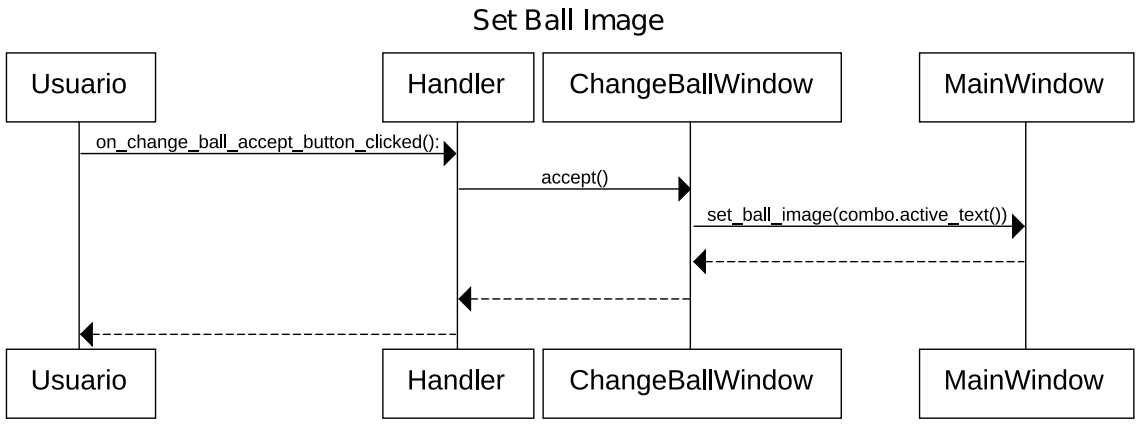


Figura 5.15: Diagrama de secuencia de la pulsación del botón *aceptar* en la ventana *Cambiar balón*

- Ventana de información de los eventos de la simulación. La funcionalidad de esta ventana es la de ofrecer al usuario la lista de los eventos que se producen en la simulación. Se trata de una ventana simple, construida con una tabla de dos columnas; una de ellas indica la descripción del evento y la otra el ciclo en que se produce. Al constructor de la ventana se le pasa como argumento los eventos de la simulación y, al crearse la ventana, dichos argumentos se añaden a la tabla. La clase que soporta esta funcionalidad es que se denomina cómo *EventsWindow*. La figura 5.16 determina el proceso de recuperación de los eventos de la simulación.

  En el listado de código 5.15 se puede observar como se añaden los eventos a la tabla.

```
2  def __fill_table_with_events(self, events):
3      for event in events:
4          self.table.attach(gtk.Label(str(event[0])), self.row,
               self.row + 1, self.column, self.column + 1)
5          self.table.attach(gtk.Label(str(event[1])), self.row + 1,
               self.row + 2, self.column, self.column + 1)
6          self.row = 0
7          self.column += 1
```

Listado 5.15: Método que añade los eventos a la tabla

- Ventana de información de las posiciones de los jugadores. La funcionalidad de esta ventana reside en la clase *PlayersPositionsWindow*. La ventana está construida por veintidós etiquetas por equipo, de las cuales once son para indicar los números de los jugadores y otras once para indicar las posiciones de los jugadores en el terreno de juego. Al producirse la petición, generada por el evento que supone mostrar la ventana, se escriben todas las etiquetas. Se escriben justo en ese momento, debido a que el usuario puede especificar la posición de un jugador en el terreno de juego en cualquier momento. De esta manera, se crear el dinamismo necesario para llevar a

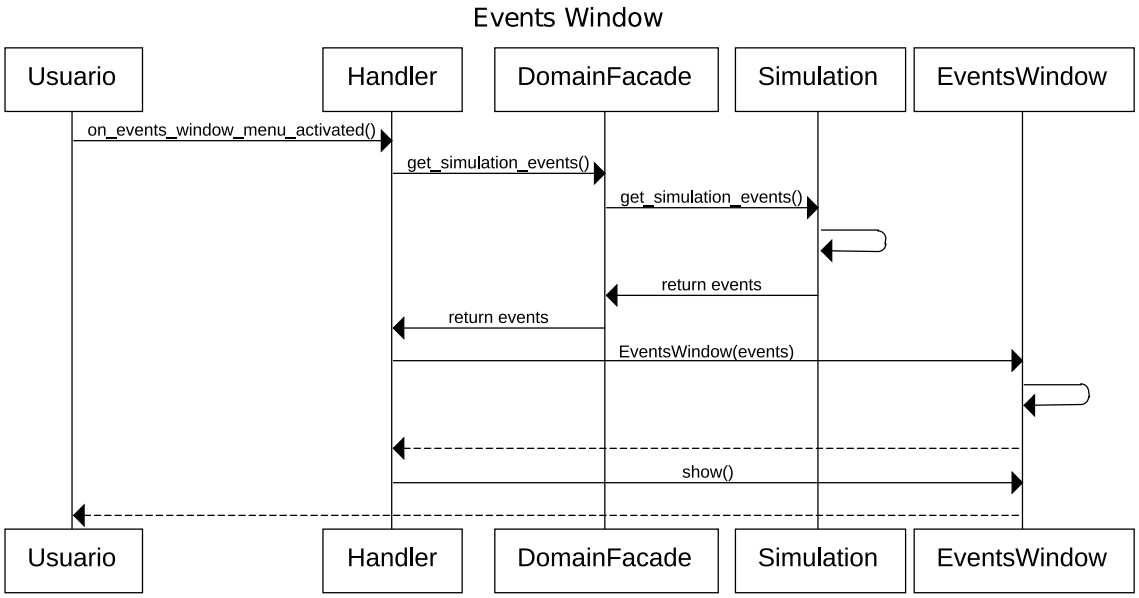


Figura 5.16: Diagrama de secuencia de la pulsación del botón *Show Simulation Events* en el menú de la ventana principal

cabo esta tarea. En el listado de código 5.16 se puede apreciar el proceso de escritura de las etiquetas, cuando se recibe una petición de mostrar la ventana.

```
1  def show(self, players_numbers, players_types):
2      ...
3      self.write_labels_types(players_types)
4      self.window.show_all()
5
6
7  def write_labels_types(self, players_types):
8      cont = 0
9      for label in self.type_labels_team_left:
10         if players_types[0][cont] != None:
11             label.set_text(str(players_types[0][cont]))
12         cont += 1
13     cont = 0
14     for label in self.type_labels_team_right:
15         if players_types[1][cont] != None:
16             label.set_text(str(players_types[1][cont]))
17         cont += 1
```

Listado 5.16: Método que escribe las posiciones de los jugadores

La figura 5.17 especifica la forma en que se recuperan los datos de los jugadores.

- Ventana para seleccionar el evento a analizar. La funcionalidad de esta ventana reside en la clase *SelectEventWindow*. La ventana se construye con una serie de botones de radio, que permiten seleccionar el evento, el cuál tendrá una representación de su análisis durante su simulación. Al constructor de la ventana se le pasa como argumento

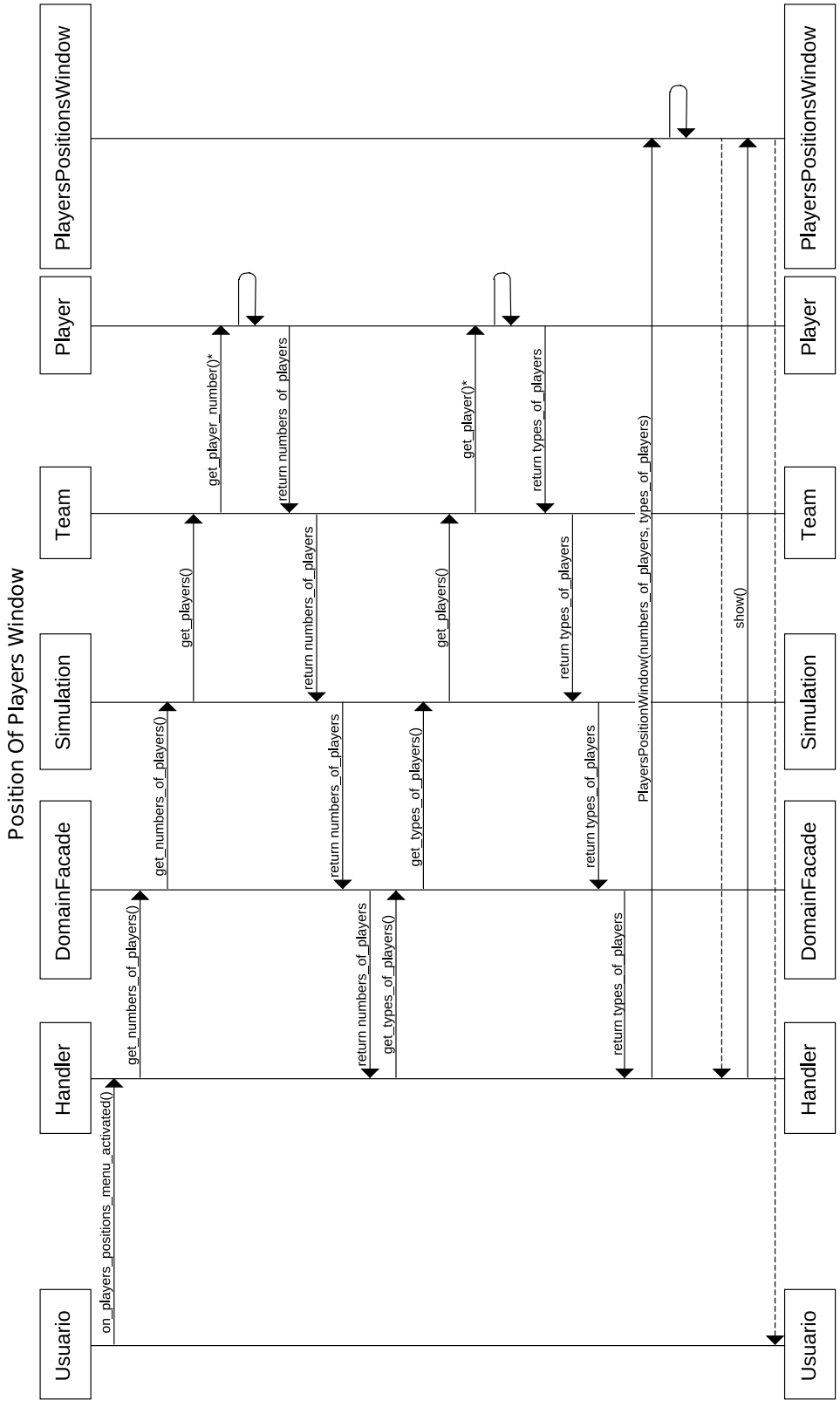


Figura 5.17: Diagrama de secuencia de la generación de la ventana que muestra las posiciones de los jugadores

los eventos de la simulación. Al crear la ventana, se genera una cantidad de botones de radio, igual a la cantidad de eventos y, se asocia un evento con un botón de radio. En el listado 5.17 se puede apreciar como se crea la lista de botones de radio y se asocian con los distintos eventos.

```
1  def __create_radio_buttons(self):
2      res = []
3      button = gtk.RadioButton(None, self.events[0])
4      self.vbox.add(button)
5      res.append(button)
6      for i in range(1, len(self.events)):
7          button = gtk.RadioButton(button, self.events[i])
8          self.vbox.add(button)
9          res.append(button)
10     return res
```

Listado 5.17: Método que genera los botones de radio asociados a los eventos de la simuación

La figuras 5.18 y 5.19 determinan los procesos que sigue el sistema para la recuperación de los tipos de eventos y la secuencia de pasos que sigue la herramienta para seleccionar el tipo de evento a monitorizar.

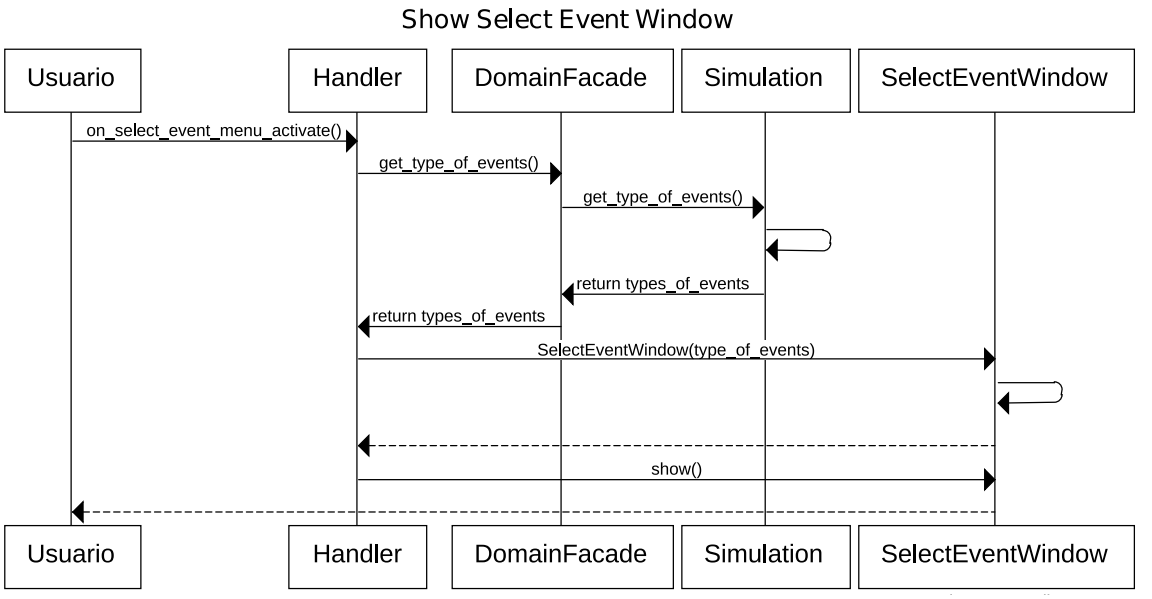


Figura 5.18: Diagrama de secuencia de la generación de la ventana que muestra los tipos de eventos

- Ventana para cambiar las vestimentas de los jugadores. La funcionalidad de esta ventana es la personalizar el traje de los dos equipos que se enfrentan en la simulación. La ventana está construida con dos cajas de combo, que determinan la vestimenta de cada uno de los dos equipos. Cuando el usuario especifica una equipación para un equipo,

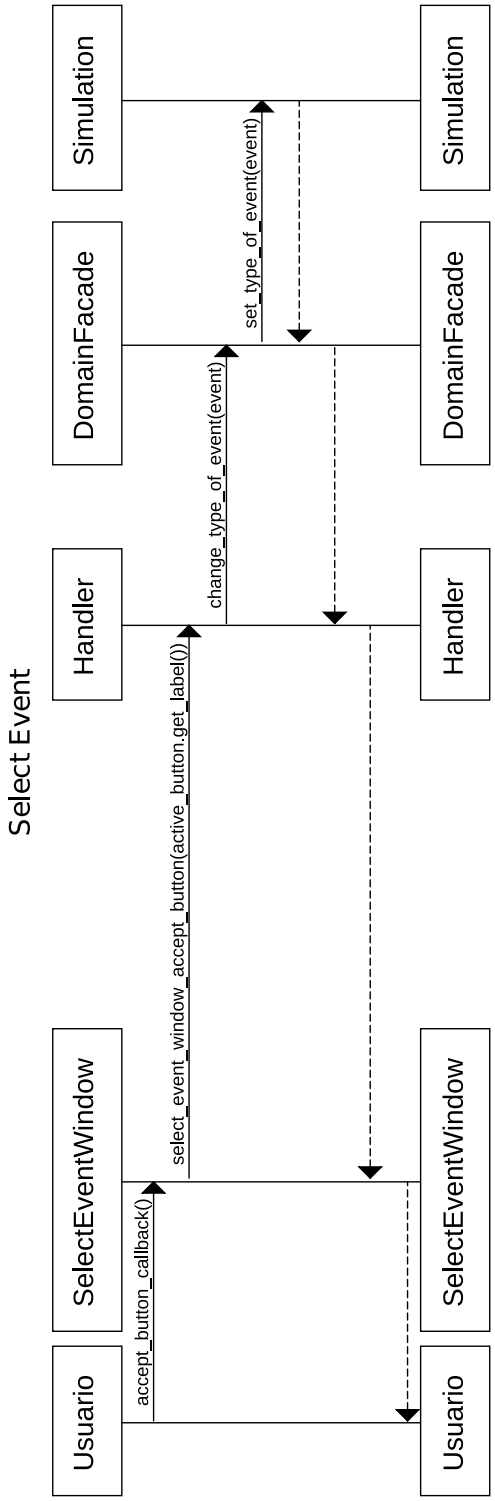


Figura 5.19: Diagrama de secuencia de la pulsación del botón *Ok* en la ventana de seleccionar el tipo de evento

automáticamente se carga una imagen situada bajo la caja de combo correspondiente. Dicha imagen sirve al usuario para previsualizar la forma en que vestirá un equipo. El código que gestiona esta ventana, se encuentra en la clase denominada como *Change-*

KitsWindow. La manera de operar de este ventana es similar a la de la ventana de que cambia la apariencia del balón. La figura 5.20 indica el proceso de creación de esta ventana.

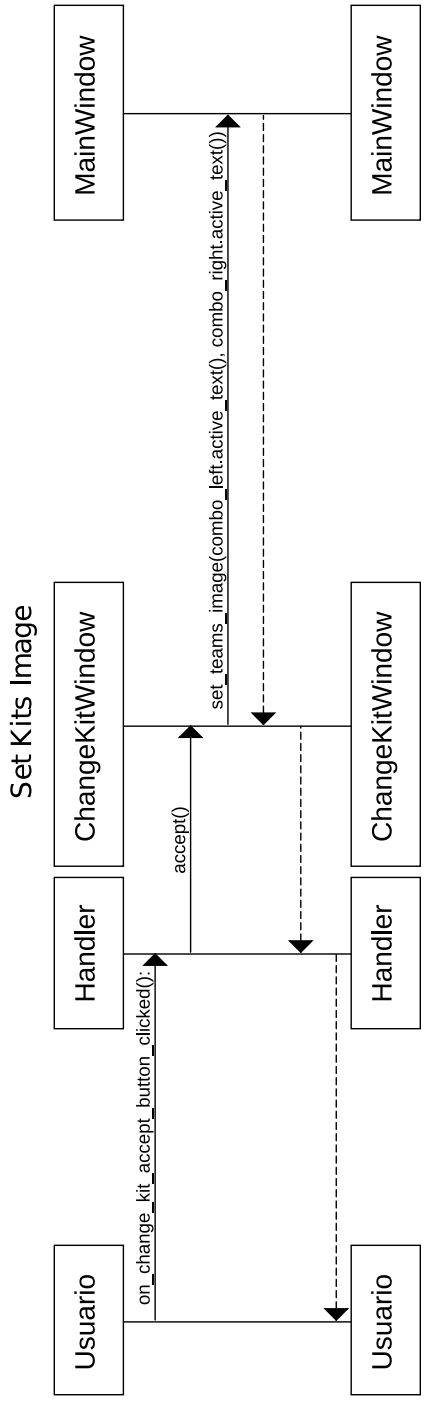


Figura 5.20: Diagrama de secuencia de la pulsación del botón *aceptar* en la ventana *cambiar vestimenta*

- Ventana para cargar las simulaciones. La funcionalidad de esta ventana es esencial en la herramienta, ya que proporciona la interfaz necesaria para que el usuario pueda

cargar una simulación. La ventana se compone de una caja de combo que contiene las simulaciones que se encuentran almacenadas en la base de datos. Cuando un usuario indica la simulación, podrá ver la fecha de dicha simulación, así como el resultado de la misma. Cuando esta ventana se crea, la caja de combo se completa con el total de simulaciones. La funcionalidad de esta ventana reside en la clase *OpenWindow*. La manera en que se crea esta ventana se puede observar en la figura 5.21. La serie de pasos que se siguen en la arquitectura cuando el usuario pulsa el botón *Ok*, se pueden observar en la figura 5.22.

- Ventana para mostrar características avanzadas de la simulación en forma de gráficos. El objetivo de esta ventana es ofrecer al usuario una forma de seleccionar las características de los distintos jugadores de la simulación, con la finalidad de generar gráficos que permitan ver la evolución de dicho jugador a lo largo del partido.

  En el listado de código 5.18 se muestra el método que gestiona el evento de la pulsación del botón *Ok* que genera los gráficos. El código de esta ventana reside en la clase *ShowAdvancedWindow*.

```
def __ok_button_callback(self, widget):
    team_left_active_numbers = []
    team_right_active_numbers = []
    active_vars = []
    # Almacenamiento de los jugadores
    # y caracteristicas marcadas.
    for number in self.numbers_left:
        if number.get_active():
            team_left_active_numbers.append(number.get_label())
    for number in self.numbers_right:
        if number.get_active():
            team_right_active_numbers.append(number.get_label())
    for var in self.vars:
        if var.get_active():
            active_vars.append(var.get_label())
    if self.check_save_charts.get_active():
        path = self.filechooser.get_filename()
    else:
        path = ""
    self.window.hide()
    # Generates the charts for each
    # mark player and variable
    for number in team_left_active_numbers:
        for var in active_vars:
            self.handler.plot(var, path, number, self.
                team_left_name)
     for number in team_right_active_numbers:
         for var in active_vars:
            self.handler.plot(var, path, number, self.
                team_right_name)
```

Listado 5.18: Manejador del evento de pulsación sobre el botón *Ok*

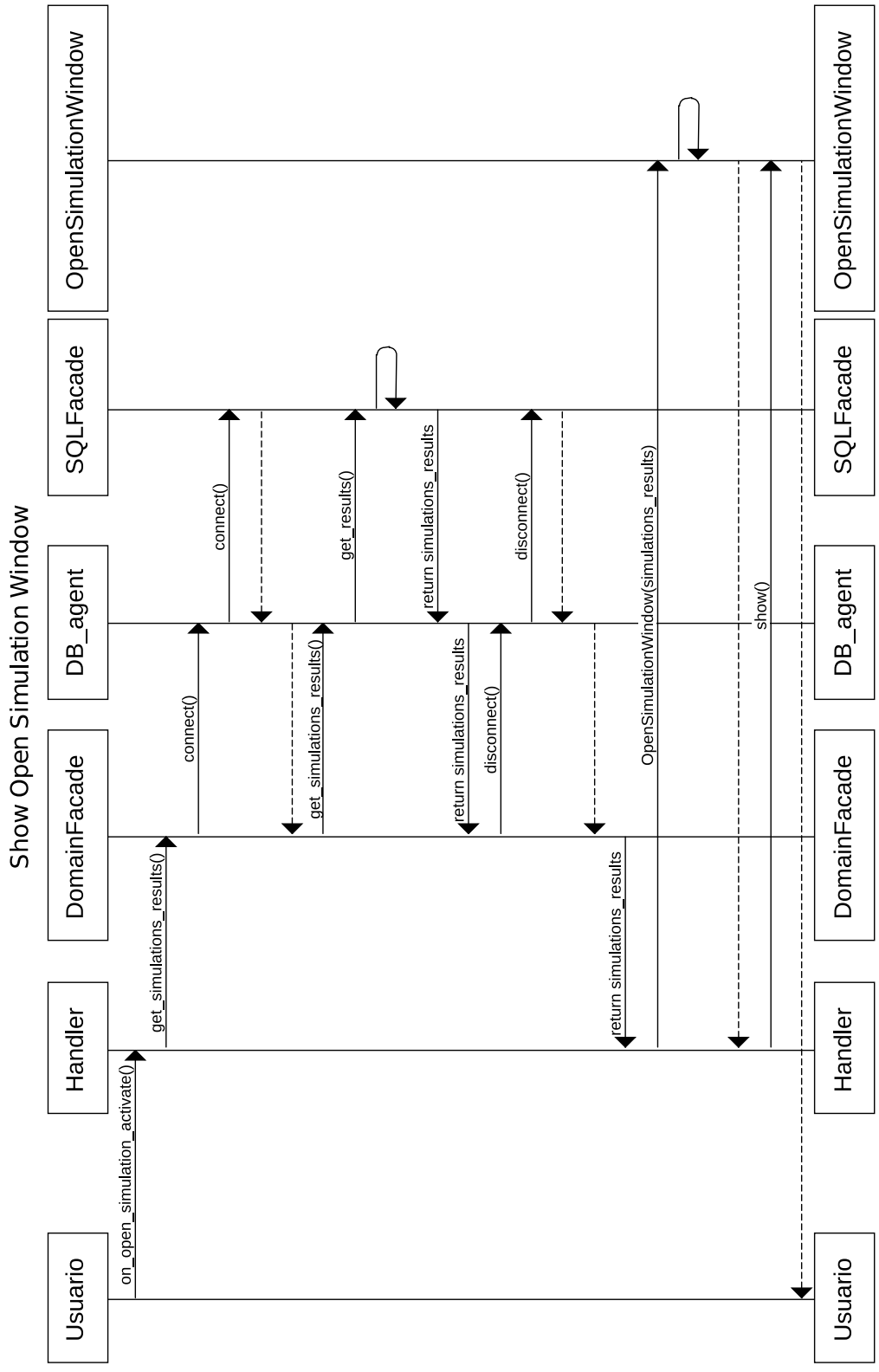


Figura 5.21: Diagrama de secuencia de la generación de la ventana de abrir una nueva simulación

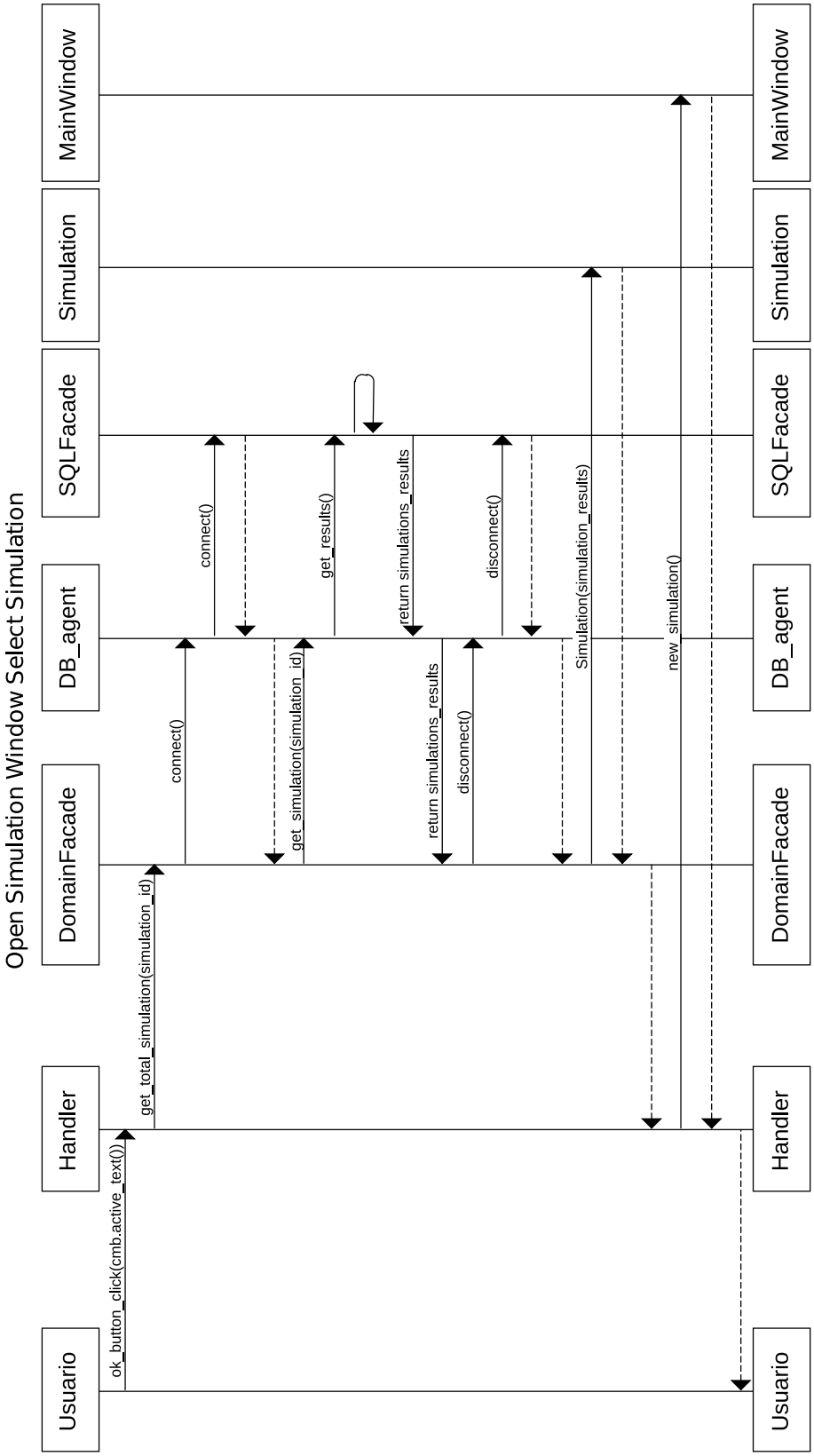


Figura 5.22: Diagrama de secuencia de la pulsación del botón *Ok* en el menú de la ventana de abrir una nueva simulación

Las figuras 5.23 y 5.24 muestran los procesos de creación de esta ventana y de generación de los gráficos.

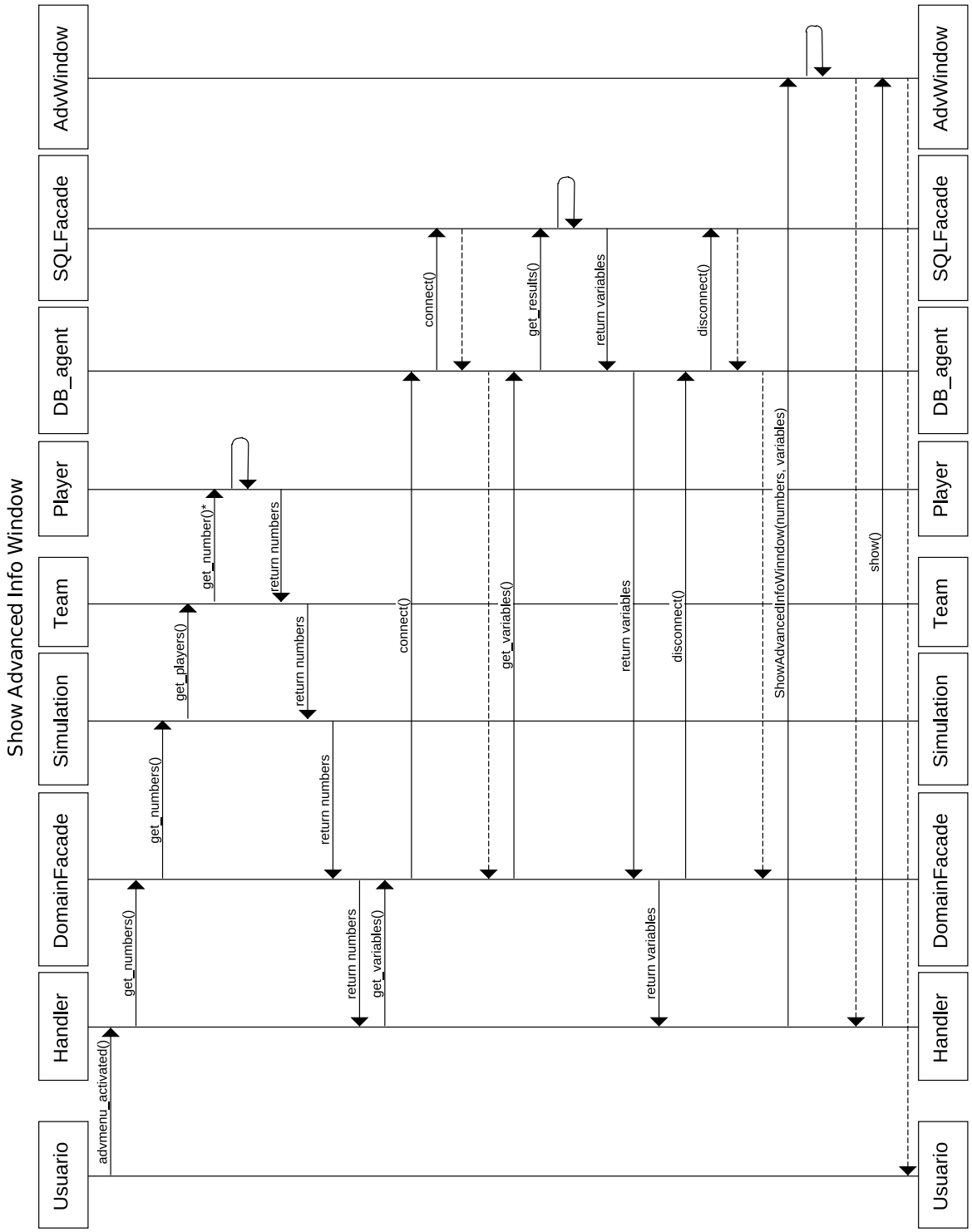


Figura 5.23: Diagrama de secuencia de la generación de la ventana que muestra las posibilidades de generación de gráficos

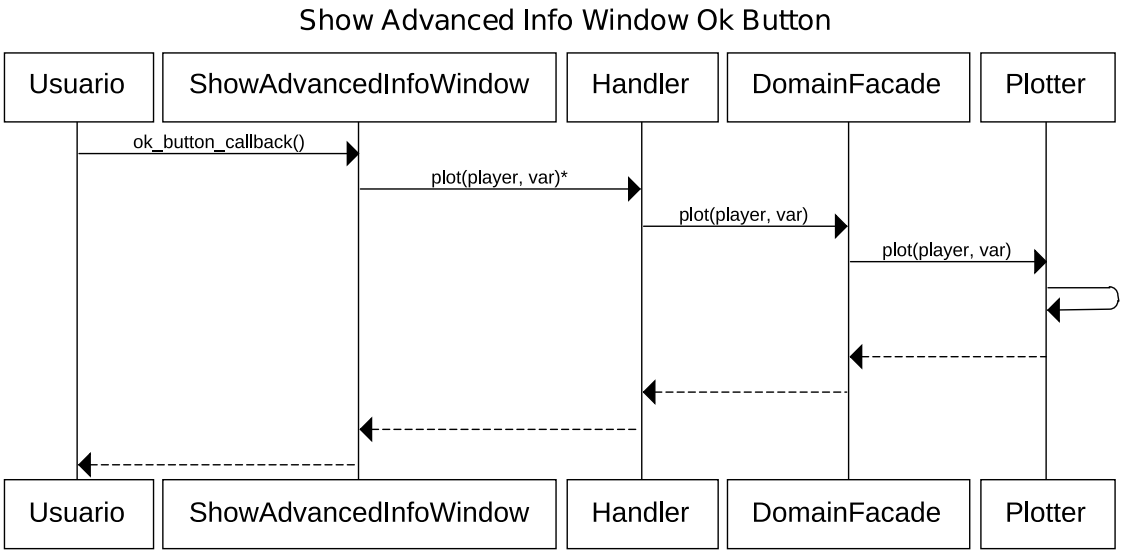


Figura 5.24: Diagrama de secuencia de la pulsación del botón *Ok* en la ventana de estadísticas avanzadas

- Ventana que genera información avanzada. Esta ventana muestra el valor de las variables almacenadas en la base de datos para un jugador determinado. La información se genera a medida que transcurre la información. La ventana aparecerá dinámicamente, a la derecha de la principal. Esta ventana está compuesta de una tabla, la cual será el elemento de presentación de los datos.

  Con el objetivo de aumentar la eficiencia del sistema, se recuperan datos del jugador cada cinco ciclos de simulación. De esta manera, la recuperación de datos se produce cuando transcurren cinco ciclos desde la última recuperación. Esto supone una mejora en la eficiencia de la aplicación, puesto que, si se tiene que recuperar información cada ciclo de simulación, el número de llamadas a procedimientos será cinco veces mayor que con el actual enfoque. Esto repercute a la postre en un retardo innecesario en tiempo de respuesta para el usuario. El código que soporta la funcionalidad de esta ventana se puede encontrar en la clase *StatsWindow*.

  Además, con la finalidad de facilitar al usuario el seguimiento de la información generada, se propone el siguiente código en la tabla de información:

  - **Verde**: El verde representa el ciclo actual de simulación. De manera que, los datos que se encuentren de este color, serán los que el jugador posea en el ciclo actual.
  - **Azul**: El color azul representa el ciclo en el cual se recuperaron, por última vez, los datos de la simulación.

  La figura 5.25 detalla el proceso de recuperación de los atributos de un jugador.

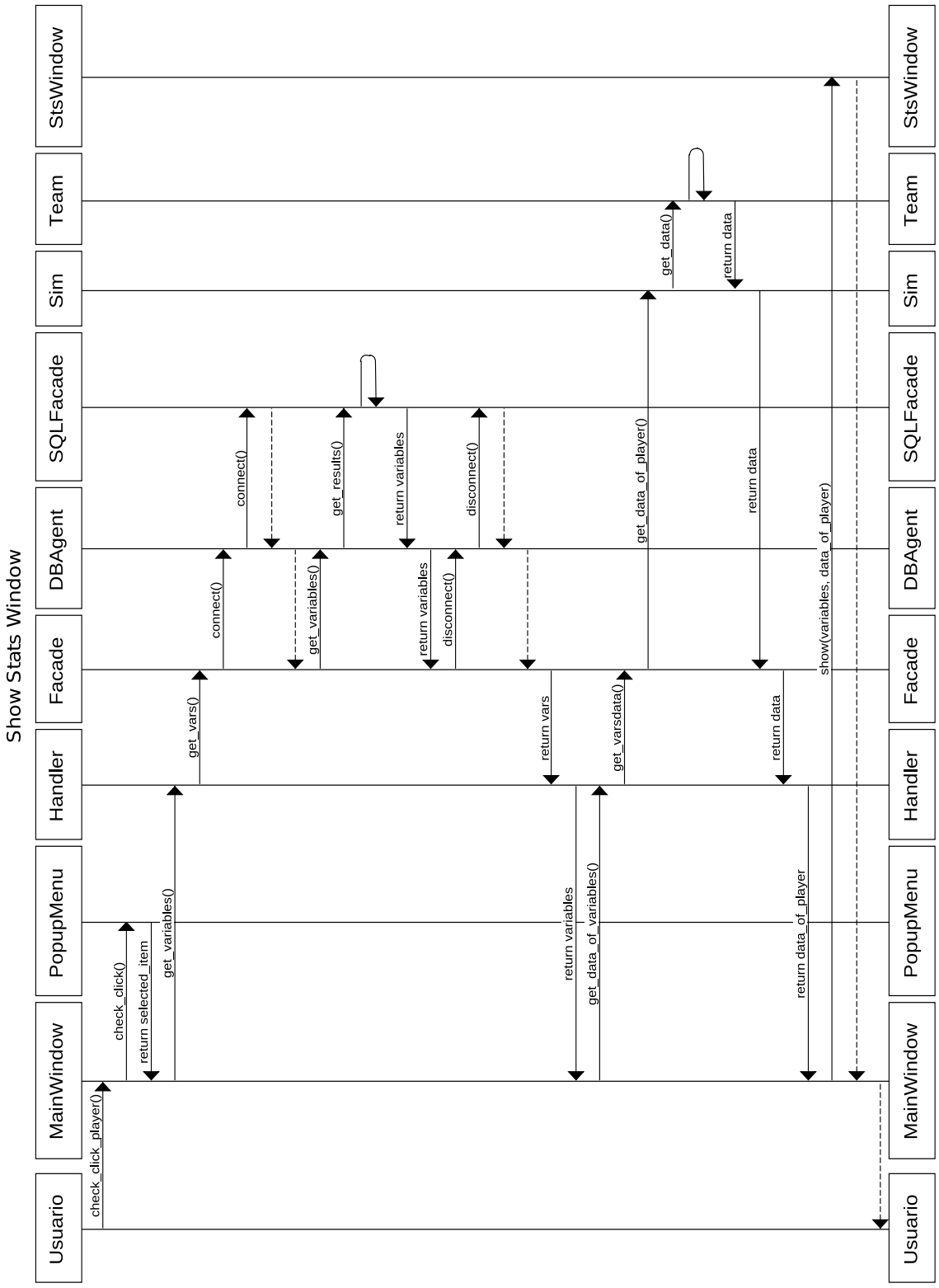


Figura 5.25: Diagrama de secuencia de la generación de la ventana que muestra las estadísticas de un jugador

5.5. Módulo de análisis de resultados

Este módulo es el responsable de analizar los eventos generados en los módulos anteriores. Se encarga, principalmente, de decidir cuándo un jugador de la simulación realiza una acción correctamente o no. Además, de acuerdo a las definiciones de la base de conocimiento o del código implementado para los comportamientos colaborativos, el objetivo del módulo es indicar al usuario dicha decisión. La manera de interacción con el usuario, se basa en un código de colores, los cuales, determinan la corrección de la acción del jugador. El esquema de colores es el que se explica a continuación:

- **Verde**. El color verde se utiliza para determinar cuándo un jugador ha realizado una acción correctamente. Si un jugador actúa de acuerdo con la decisión tomada por el módulo, entonces aparecerá, bajo dicho jugador, un círculo verde parpadeante (ver figura 5.26).

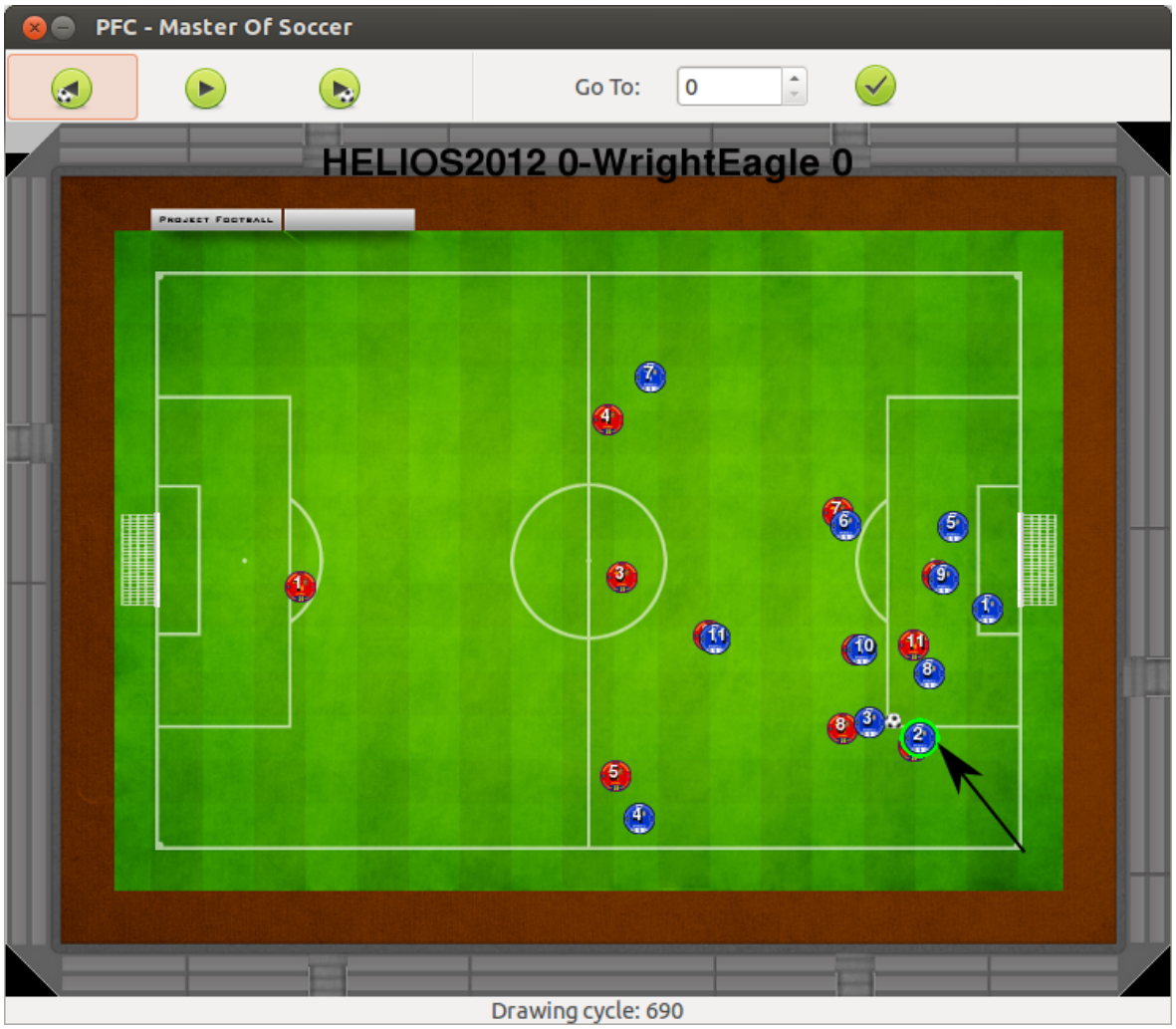


Figura 5.26: El jugador 2 del equipo azul realiza una acción correcta

- **Rojo**. El color rojo determina un comportamiento no adecuado por parte del jugador que ha tomado una decisión. De la misma manera que en el anterior caso, un círculo parpadeante, en este caso de color rojo, aparecerá bajo el jugador que haya tomado la decisión erróneamente (ver figura 5.27).

Figura 5.27: El jugador 10 del equipo rojo realiza una acción incorrecta

Otra funcionalidad del módulo es la ralentización del tiempo de simulación cuando va a ocurrir un evento próximamente. Esto permite al usuario prestar mayor atención en la decisión que va a tomar un jugador. La ralentización del tiempo de simulación durará hasta que el jugador haya tomado una decisión. En el listado de código 5.19 se puede observar la función encargada de cambiar el tiempo de la simulación. El método puede encontrarse en la clase *MainWindow*.

```
1 def update_timer(self, refresh_rate):
2   self.refresh_rate = refresh_rate
3   gobject.source_remove(self.timer_key)
```

```
4     self.timer_key = gobject.timeout_add(self.refresh_rate, self.
          get_sim_info)
```

Listado 5.19: Método que cambiar el tiempo de simulación

Para determinar qué decisión es la que debe tomar un jugador, se definen diferentes tipos de comportamientos:

- Comportamientos **generales**. Este tipo de comportamientos definen situaciones generales. Este tipo de comportamientos, son menos restrictivos a la hora de imponer condiciones para determinar la corrección de la acción de un jugador que los comportamientos específicos. A continuación, se detallan los diferentes tipos de comportamientos generales.
  - **Tipo 1**. El comportamiento correcto de un jugador que realiza una acción que da lugar a un evento de tipo 1, es no mantener la posesión del balón después de realizar la acción. Por ejemplo, un comportamiento de este tipo sólo determina si el jugador se separa del balón, en un tiempo determinado, es decir, no comprueba que el jugador le pase a un compañero o efectúe un disparo a portería.
  - **Tipo 2**. Cuando un jugador realiza una acción que de lugar a un evento de este tipo, el comportamiento correcto es que el balón debe sobrepasar la franja del mediocampo horizontal. Esta forma de actuar está ideada, principalmente, para cambios de juego, aunque también puede utilizarse para pases en diagonal, ya que la mayoría de ellos atraviesan la franja del mediocampo.
  - **Tipo 3**. Un comportamiento de este tipo, indica que la decisión que la decisión tomada por un jugador siempre es errónea. Por ejemplo, cuando un jugador tira mal la línea del fuera de juego, se produce una acción que siempre es incorrecta, puesto que, la línea del fuera de juego no se debería trazar mal nunca, idealmente.
  - **Tipo 4**. Un comportamiento de este tipo indica que la decisión que la decisión tomada por un jugador siempre es correcta.
- Comportamientos **específicos**. Utilizar un comportamiento de este tipo implica determinar, de manera muy concreta, la acción que debe realizar el jugador. Los comportamientos definidos de esta manera son los que se detallan a continuación.
  - **Tipo 1001**. Un comportamiento definido con este tipo indica que, el jugador que realiza una acción que da lugar a la producción de un evento que requiere este comportamiento, debe desmarcarse cuando pasa el balón.
  - **Tipo 1002**. Un comportamiento de este tipo determina el pase a un compañero más adelantado que se encuentre libre de marca. Para poder señalizar el evento correctamente, en la conclusión del evento debe aparecer el jugador que debe recibir el balón.

El método que se puede apreciar en el listado 5.20 determina el tratamiento del evento correspondiente.

```
def __check_event_type(self, event_type, cycle, number_of_player,
    team_id, event_conclusion):
     correct_action = ""
     if event_type == 1:
         # Type 1: To Keep or not to keep the ball possession
         possession_threshold = self.get_possession_in_threshold(
             team_id, number_of_player, cycle, 15)
         correct_action = self.__check_possession(possession_threshold)
     elif event_type == 2:
         # Type 2: Check if the ball is in the
         # different midfield after few cycles.
         correct_action = self.__check_change_of_game(cycle)
     elif event_type == 3:
         # Type 3: This event conclusion is always wrong.
         correct_action = "Wrong"
     elif event_type == 4:
         # Type 4: This event conclusion is always right
         correct_action = "Right"
     elif event_type == 1001:
         # Type 1001: Check movement.
         correct_action = self.__check_movement(cycle, number_of_player
             , team_id)
     elif event_type == 1002:
         # Type 1002: Pass to an only teammate.
         only_teammate = int(event_conclusion.split(":")[0].split()[2])
         possession_threshold = self.get_possession_in_threshold(
             team_id, only_teammate, cycle, 20)
         if possession_threshold == True:
             correct_action = "Right"
         else:
             correct_action = "Wrong"
     return correct_action
```

Listado 5.20: Método que determina el tipo de evento a tratar

5.6. Módulo de pruebas del sistema

Este módulo contiene las pruebas automatizadas del sistema. La funcionalidad de este método es la de probar que el software implementado realiza **lo que debe hacer y no realiza lo que no debe realizar**. Este módulo busca la detección de los defectos en el software. La detección y resolución de los mismos dota de mayor calidad al software realizado.

En este módulo existen pruebas para gran parte de los métodos implementados en la lógica de dominio y la persistencia de la aplicación. De la parte de la interfaz se han realizado pruebas de aceptación con el director del proyecto, ya que dicha parte no se presta a ser probada fácilmente por medio de pruebas automáticas.

La implementación de este módulo se encuentra definida en la carpeta *test* del *CD* adjunto a esta documentación. Las pruebas analizan las partes que determinan la funcionalidad de la aplicación. El módulo contiene las siguientes clases:

- **PointTest**. Esta clase contiene las pruebas de la clase *Point*. Los test más significativos de esta clase son los de los métodos que comprueban la equivalencia entre dos puntos y la distancia entre dos puntos.
- **SQLFacadeTest**. Contiene la funcionalidad necesaria para probar la clase *SQLFacade*. Esta clase contiene cinco pruebas que definen las pruebas para los métodos de conectar y desconectar de obtener información y actualizar la base de datos y pruebas de la comprobación de generación de excepciones.
- **db_agent_test**. Contiene la funcionalidad necesaria para la probar la clase *db_agent*. Todos los métodos de esta última clase se encuentran probados.
- **DomainFacadeTest**. En esta clase se prueban los métodos de la fachada del dominio, que recuperan información y definen comportamientos de la interfaz del usuario.
- **SimulationTest**. Estas pruebas analizan el comportamiento de los métodos de la clase *Simulation* que determinan el valor de variables como la presión o los jugadores contrarios cercanos.
- **PlotterTest**. Esta clase se realizan las pruebas del generador de gráficos (clase *Plotter*).
- **RegionTest**. Comprueba el comportamiento de la clase *Region*.
- **TreeTest**. Comprueba el comportamiento de la clase que divide el campo en un árbol binario (clase *TreeTest*).

Este módulo también contiene una fichero de código que ejecuta todas las clases de prueba, con el objetivo de facilitar la labor de realizar dichas pruebas una a una. El fichero se denomina *AllTest.py* y su código puede apreciarse en el listado 5.21.

```
1 test_all = unittest.TestSuite()
2 for testcase in glob.glob('./*.py'):
3     mod_aux = testcase.split("./")[1]
4     mod = mod_aux.split(".py")[0]
5     if mod != "AllTest":
6         test_all.addTest(unittest.TestLoader().loadTestsFromName(mod))
7 unittest.TextTestRunner(verbosity = 2).run(test_all)
```

Listado 5.21: Código que lanza todas las pruebas

La funcionalidad del código consiste en crear un conjunto de pruebas, añadir una a una las pruebas que resultan ser ficheros con extensión de *Python* de la carpeta mediante un bucle y ejecutar las pruebas utilizando el método `run` de la clase `TestRunner`.

5.7. Patrones de diseño

Los **patrones de diseño** son una solución simple y elegante a problemas comunes del diseño orientado a objetos [GHJV96]. Así pues, no es necesario volver a determinar una solución frente a uno de estos problemas. Simplemente se estudia el problema y se adopta la solución para ese problema.

Los requisitos que debe poseer una solución para que pueda ser considerada como un patrón de diseño son dos; **efectividad** y **reusabilidad** [GHJV96]. Esto quiere decir que una solución debe haber resuelto problemas con éxito y, además, dicha solución se puede aplicar a problemas con semejantes restricciones.

Los patrones se dividen en tres tipos [GHJV96]:

- **De creación**: Este tipo de patrones propone soluciones para los problemas que supone la creación de objetos.
- **De estructura**: Los patrones de estructura se utilizan para una correcta composición de las clases.
- **De comportamiento**: Estos patrones caracterizan las formas en las que interactúan y reparten responsabilidades las distintas clases u objetos.

5.7.1. Patrones en la arquitectura

En la arquitectura del presente proyecto se han aplicado diferentes patrones de diseño. A continuación, se enumeran y describen dichos patrones.

Modelo Vista Controlador (MVC)

Es un patrón de arquitectura de las aplicaciones. Este patrón separa la lógica de negocio de la interfaz de usuario, facilitando la evolución por separado de ambos aspectos, lo que supone un incremento de la reutilización y flexibilidad [Ree03].

El patrón se compone de varios módulos, como son los siguientes (ver figura 5.28):

- Un **Modelo**: El modelo contiene el núcleo funcional de la aplicación. El modelo proporciona funciones para acceder a sus datos y que son utilizadas por las vistas para adquirir dichos datos y mostrarlos al usuario.
- Varias **Vistas**: Las vistas presentan la información al usuario. Las diferentes vistas presentan información del modelo en diferentes maneras.
- Varios **Controladores**: Los controladores definen las entradas del usuario como eventos. Un controlador implementa procedimientos de atención a eventos, dichos procedimientos serán invocados cuando se produzca un evento. Los eventos se convierten en peticiones al modelo, que procesará esa petición de una manera u otra.

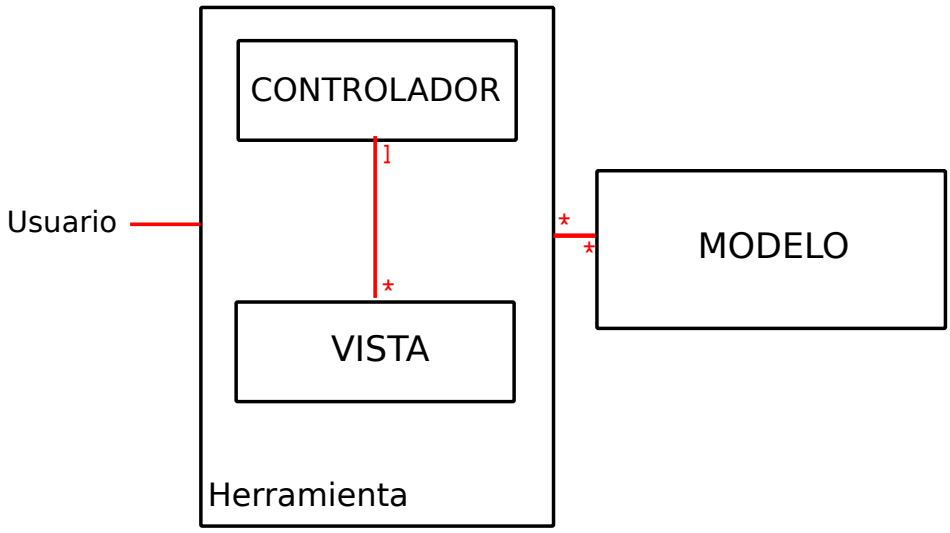


Figura 5.28: Patrón MVC

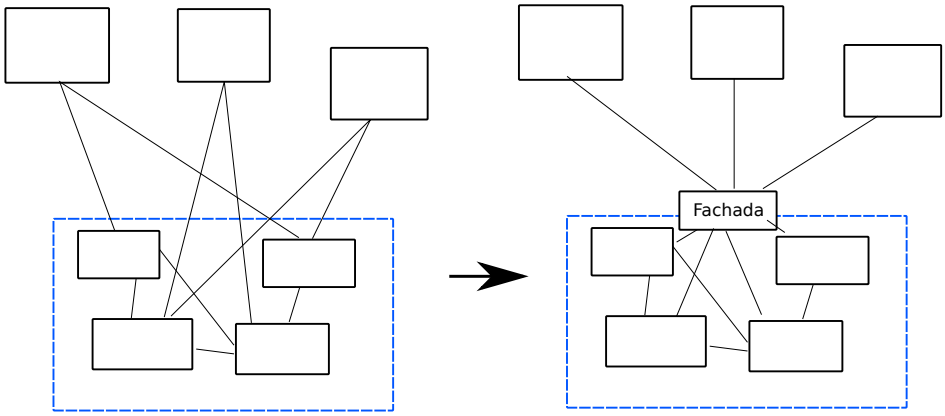


Figura 5.29: Patrón Fachada

Este patrón es el más importante de los aplicados, ya que define la arquitectura de la aplicación. Cuando un usuario interactúa con la herramienta, un controlador recoge la acción e invoca el procedimiento necesario de una vista determinada, así como, del modelo.

Fachada

El patrón fachada proporciona una interfaz unificada para un conjunto de interfaces de un subsistema [GHJV96]. Define una interfaz de alto nivel que hace que el subsistema se más fácil de usar. El patrón fachada se ha utilizado en la aplicación para establecer una separación entre el código perteneciente a la interfaz gráfica, el de la lógica de dominio y el código destinado a interactuar con el esquema de base de datos.

Aplicar el patrón fachada tiene el principal objetivo de mantener un **bajo acoplamiento** entre clases y una **cohesión alta** para cada clase. Para mantener un bajo acoplamiento, es necesario que las relaciones entre clases sean las menores posibles. Una clase con alta cohesión

significa que la clase es responsable de proveer funcionalidad, de tal manera, que no existen dos clases con responsabilidades similares o que se complementen.

Un ejemplo de un uso correcto se puede apreciar en la figura 5.29.

Iterator

Proporciona una manera de acceder a los elementos de un objeto secuencialmente [GHJV96].

En la aplicación este patrón se utiliza en múltiples situaciones, aunque la más representativa es para recorrer los jugadores de un equipo con sus respectivas características. Un ejemplo de uso de este patrón se puede observar en el código 5.22.

```
1 def get_player(self, number_of_player, cycle):
2     # Iterator...
3     for player in self.players:
4         if player.number == number_of_player:
5             return player
```

Listado 5.22: Patrón iterator

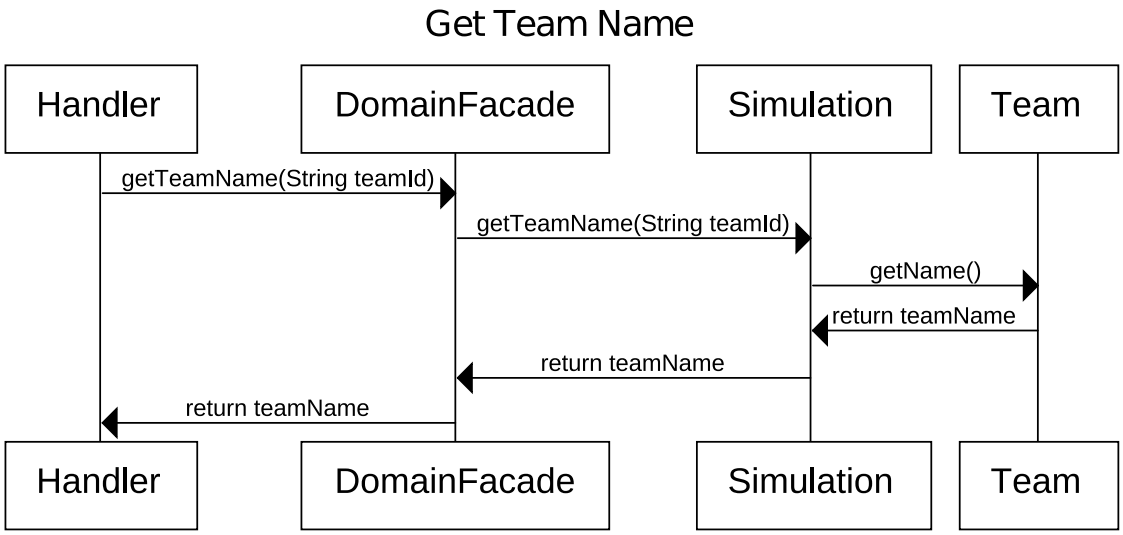


Figura 5.30: Patrón Chain of responsability

Chain of responsibility

La principal característica de este patrón es la de encadenar peticiones por parte de los objetos emisores, para que sean recibidos por objetos receptores que se encarguen de procesar la petición o prolongar la cadena emitiendo la petición a otros objetos. La idea de este patrón es la de desacoplar los emisores y receptores, por medio de múltiples objetos que manejen una petición. La petición pasa a través de una cadena de objetos dónde uno de ellos

la procesa.

La aplicación de este patrón es imprescindible en aplicaciones con interfaz gráfica de usuario, puesto que, desde que se interactúa con la aplicación hasta que se obtiene respuesta, es necesaria una secuencia de eventos que provea dicha respuesta. El procesamiento de los eventos será responsabilidad de los objetos que componen la aplicación.

En la aplicación se ha utilizado este patrón en múltiples ocasiones, una de las más representativas puede apreciarse en el diagrama de secuencia que se corresponde con la figura 5.30.

Singleton

La aplicación de este patrón asegura que una clase tenga únicamente una instancia, y proporciona un punto de acceso global a la misma [GHJV96]. Una buena solución para implementar este patrón es hacer que la propia clase sea responsable de almacenar una sola instancia de ella misma. Cuando se requiera una instancia de la clase, será la propia clase la que la proporcione. Así, se asegurará que no se crean nuevas instancias.

En el sistema se ha utilizado el patrón *singleton* para el desarrollo del agente de la base de datos.

Capítulo 6

Evolución, Resultados y Costes

Este capítulo evalua el trabajo llevado a cabo desde que comenzó el proyecto hasta que llegó a la etapa de mantenimiento. Como parte de ello, se estudiará la aplicación de la **metodología** realizada, se analizarán los **resultados** obtenidos a la hora de desplegar el sistema en un caso de estudio concreto y se estimarán los **costes** del desarrollo.

6.1. Evolución

Esta sección pone de manifiesto las distintas iteraciones empleadas desde el inicio del proyecto hasta el final del mismo. Cada una de ellas describe, además de la información relevante, los diferentes hitos alcanzados.

En primer lugar, debido a las múltiples alternativas que se planteaban a la hora de la realización del Proyecto Fin de Carrera, se acordó el dominio del mismo. Tras determinar la temática del proyecto, se decidió que el primer problema a solucionar debería ser la obtención de datos de partidos de fútbol. Esta información tendría que ser analizable y reconocible mediante software, de manera que se pudiera inferir nueva información de forma automática. Las diferentes alternativas de obtención de la información pasaron por vídeos, hasta datos que una empresa inglesa se negó a dejar por cuestiones de privacidad. Tras evaluar las diferentes opciones, se decidió estudiar algún simulador libre con el objetivo de modificar su código fuente para obtener las posiciones de los jugadores en el campo, en cada instante de simulación. Así fue cómo se descubrió el simulador *RoboCup Soccer 2D* y se procedió al estudio de las diferentes posibilidades que ofrecía su arquitectura.

Una vez se determinó la fuente de datos, se acordaron los objetivos iniciales del proyecto, dichos objetivos se enumeran a continuación y quedan detallados en el capítulo 2:

- La arquitectura debe estar basada en bibliotecas libres y estándares abiertos para conseguir un sistema multiplataforma.
- La arquitectura debe ser adaptable a cualquier dominio monitorizado.
- La arquitectura debe ser modular y que resulte fácil modificar y agregar módulos para conseguir una alta expansibilidad.

- La arquitectura debe ser fácilmente escalable para dar soporte a los máximos usuarios posibles.
- La arquitectura debe emplear patrones de diseño en la resolución de problemas planteados en situaciones similares.
- La arquitectura debe soportar la inclusión de nuevos comportamientos por parte del usuario.
- La herramienta deberá poseer una interfaz gráfica atractiva para que la interación, por parte del usuario, sea sencilla y cómoda.

Tras la definición de los requisitos iniciales de la arquitectura, dió comienzo el desarrollo de la misma (2 de Junio de 2012). El desarrollo de la misma ha seguido varias iteraciones. A continuación se detallan estas iteraciones.

6.1.1. Iteraciones

Iteración nº1

Al comienzo de esta iteración se crea el árbol de directorios en que se estructura el proyecto. Además, se construye el *Makefile* que elimina los archivos de *bytecode* generados por el intérprete de *Python*. Seguidamente, se determinó que todo el proyecto debería estar gestionado por un sistema de control de versiones. Se siguieron los pasos necesarios para la implantación de un sistema de este tipo que soportara la estructura del proyecto, así como los cambios que iban sucediendo.

El objetivo de esta iteración se resume en la creación del módulo de preprocesamiento de la información. En esta iteración se determinó que la entrada de información al sistema consistiría en un sistema de *parsers* que reconocieran diferentes formatos en los que se almacena la información. Así pues, es en la primera iteración donde se definió el primer *parser* para el reconocimiento de la información almacenada en ficheros *XML*. Dicho *parser* obtiene la información de dichos ficheros de manera automática y la almacena en un esquema de base de datos relacional. El resultado de esta iteración es, por un lado, la clase *ParserXML* y, por otro, el comienzo de la funcionalidad del agente de base de datos, que permite que se introduzca información relativa a la simulación en la base de datos. Mientras se implementa esta funcionalidad, se va defiendo el diseño de la base de datos. Además, también se implementan las pruebas tanto del *parser* como de la funcionalidad del agente.

El resultado de esta iteración no supuso ningún prototipo de la aplicación. Sin embargo, se construyó el esqueleto inicial de la herramienta.

Iteración nº2

Cuando la primera iteración estuvo completa, se implementó la primera aproximación de las clases de dominio. La funcionalidad en esta iteración debería soportar la instanciación de

objetos con la información almacenada en la base de datos. Así se crearon las clases *State Object*, *Player*, *Team*, *Goal*, *Simulation* y *DomainFacade*.

Con respecto al agente, se incluyó funcionalidad que permitiera la recuperación de la información de una simulación. El proceso de recuperación de los datos de un partido es costoso en términos computacionales, de manera que se estudiaron diferentes técnicas de optimización en las consultas, lanzadas por parte del agentes a la base de datos. También se creó un script (*RestoreDB*) que permitiera eliminar toda la información almacenada en las tablas de la base de datos y, así, volver a un estado inicial en caso que hubiera algún error en la inserción de los datos en dicha base de datos. Al mismo tiempo que se iba implementando la funcionalidad de cada clase se creaban las pruebas automáticas que comprobarán los posibles defectos que fueran cometiendo.

El resultado de esta iteración, al igual que en la anterior, no presenta ningún prototipo de la herramienta, pero define funcionalidad necesaria que será utilizada en posteriores iteraciones.

Iteración nº3

En esta iteración se implementa la funcionalidad necesaria para la creación de una primera aproximación de la interfaz de usuario. Por tanto, el objetivo de esta iteración se centra en la creación de un prototipo básico de la interfaz de usuario.

En la tercera iteración se construye el esqueleto de la aplicación de manera estática, utilizando *Glade*. Esto da lugar a la creación del fichero *mos_interface.glade* que contiene el prototipo de la interfaz de usuario. También se implementan las primeras versiones de las clases *Handler*, *MainWindow*, *OpenSimulation* y *Drawer*. El objetivo de estas clases es proveer la funcionalidad necesaria para, por medio de la interfaz gráfica, recuperar la información correspondiente a una simulación, mostrar a los jugadores en movimiento en el terreno de juego, y la gestión de los botones de «ir al siguiente gol», «ir al anterior gol» e «ir a un ciclo específico», así como del botón de «reproducir/parar».

El resultado de la iteración es un primer prototipo de la herramienta. En la figura 6.1 se puede observar el primer prototipo de la interfaz gráfica.

Iteración nº4

Esta iteración estuvo destinada a implementar las utilidades necesarias para obtener información de la simulación en forma de ventanas de la interfaz gráfica. Además, se desarrollaron diferentes aspectos que dotaran a la herramienta de la posibilidad de personalización por parte del usuario. Así en esta iteración se crearon las clases *ChangeBallWindow*, *ChangeKitWindow*, *PlayerPositionsWindow*. En esta iteración se determinó la posibilidad de que el usuario especificara el tipo de jugador en la simulación, con el objetivo de responder a las preguntas ¿dónde está el delantero? ¿quién es un defensor? Para ello, se determinó la

Figura 6.1: Primer Prototipo de la herramienta

creación de un menú emergente que se desplegara al pulsar con el botón izquierdo del ratón sobre un jugador, y que contuviera los diferentes tipos de jugadores (portero, defensor, mediocampista y delantero). Esto dio lugar a la implementación de la clase *PopupMenu*.

Además, se implementó la funcionalidad necesaria de la clase *StatsWindow* que, además de ofrecer información acerca de los valores de las variables de un jugador, servía de depuración.

Por último, se pensó en integrar el comportamiento de la clase *StatsWindow* con el de *PopupMenu*, para que el usuario tuviera la posibilidad de determinar las características de un jugador al hacer click sobre el. Así pues, en lugar de poseer una ventana con la información de un jugador, se determinó que sería una mejor opción acoplar dicha ventana a la ventana principal y que se mostrase cuando el usuario lo decidiera oportuno, mediante las opciones correspondientes en el menú emergente.

El resultado de esta iteración es el de un segundo prototipo de la interfaz, pero más refinado y con muchas más opciones y posibilidades que el prototipo anterior. Además, se produce un cambio en la imagen del terreno de juego, que da lugar a una mejora estética de la aplicación (véase la figura 6.2).

Figura 6.2: Segundo prototipo

Iteración nº5

En esta iteración se decidió que el conocimiento del experto debía residir en una base de conocimiento. Al plantearse las primeras situaciones a detectar, se hizo necesario establecer una división en dos capas de dicha base de conocimiento. Una capa sería de nivel más bajo, en la cuál se definirían las variables difusas más concretas. La otra capa está relacionada con dar soporte a conceptos más generales del fútbol, tales como, el cambio de juego o el disparo a portería. Se decidió que esta última capa pudiera utilizar la información generada por la capa de más bajo nivel. De esta manera, surgió la necesidad de implementar un sistema

difuso de dos niveles. Ambos niveles se encuentran en los archivos *PlayerKnowledge.fcl* y *PlayerActions* de la carpeta *kb* del *CD* adjunto a esta documentación.

En esta iteración se da soporte a la primera aproximación de la detección de diferentes situaciones de una simulación. El objetivo de esta iteración es el de inferir nueva información, a partir de la información disponible y utilizando la base de conocimiento en dos capas. En esta iteración se implementa el sistema de lógica difusa en Java, por tanto se crean las clases *FuzzySystem*, *Feature*, *Event*. Además, se implementan los métodos necesarios para el agente de la base de datos y la fachada de domino.

La primera aproximación llevada a cabo en esta iteración fue la crear un sistema difuso propio que generará nueva información. Pero, al ser la adaptabilidad, extensibilidad y el uso de estándares objetivos de la arquitectura, se descartó esa idea inicial y se decidió utilizar el lenguaje estándar *FCL*. Una vez tomada esa decisión, se buscó la manera de que el sistema difuso se integrara con el resto del proyecto, utilizando el mismo lenguaje de programación (*Python*), pero las bibliotecas encontradas no reconocían correctamente el FCL. De manera que, se utilizó *Java*.

Una vez elaborado el software que soportaba la creación de sistemas difusos se idearon los primeros eventos a detectar, los cuales eran el del disparo a portería y el de la detección de un robo crítico. Para poder describir correctamente los eventos, era necesario información auxiliar, que determinará ciertas situaciones en el terreno de juego. Por ejemplo, un jugador no tirará siempre que posea la posesión del balón, sino que deberá poseer el balón y estar en una zona del campo adecuada para la realización del golpeo a puerta.

Por parte de la interfaz, se implementa la funcionalidad necesaria para determinar cuándo ocurre un evento y señalar al jugador que realiza la acción. Además, se desarrolla una ventana que permite al usuario determinar los diferentes eventos de una simulación. La clase que soporta esta última funcionalidad es *EventsWindow*.

El nuevo prototipo resultante contiene la funcionalidad del segundo prototipo y además se representa la forma en que un jugador realiza correcta o incorrectamente una acción de manera gráfica.

Iteración 6

En la iteración anterior se definen los eventos de disparo y robo crítico. En esta iteración se definen otros eventos como son los del centro, el pase de la muerte y el cambio de juego. Sin embargo, existen situaciones en las que el empleo de un sistema difuso no es una opción adecuada, como en la detección del fuera de juego o el pase a un compañero, debido a que influye la situación de más jugadores. Estos eventos son importantes en un partido de fútbol. Por tanto, tras reuniones con el director de proyecto se decidió que la mejor solución para determinar este tipo de situaciones sería implementar los diferentes eventos mediante código

Python, debido a que el uso de un sistema basado en reglas difusas era poco flexible para el comportamiento colaborativo.

En la parte de la interfaz gráfica, en esta iteración se decide implementar la clase *ShowAdvancedWindow*, que permite al usuario la generación de gráficos de los valores de una simulación.

Por tanto, el prototipo resultante contendrá toda la funcionalidad del anterior más la implementación de esta ventana y la detección de nuevos eventos.

Iteración 7

En esta iteración se implementa la funcionalidad para que los eventos puedan ser especificados por el usuario en ficheros XML. Para ello, se modifica la clase Java *DomainFacade* y desarrolla la clase *ParserEvents* que reconoce los diferentes eventos creados por el usuario. En esta iteración se dota a la arquitectura de la flexibilidad suficiente para la inclusión de nuevos eventos generados por el usuario.

En la interfaz gráfica se añade la ventana (clase *SelectEventWindow*) de selección del tipo evento cuyo cometido es el de dar la posibilidad al usuario de determinar el tipo de evento que se quiere monitorizar durante el desarrollo de la simulación. En esta etapa se decide también que cuando ocurra un evento se ralentice el tiempo de simulación, para que se pueda analizar detenidamente el comportamiento de un jugador. También se corrigen diferentes defectos de otras iteraciones que no se habían detectado anteriormente.

En resumen, en esta iteración se genera el último prototipo con la funcionalidad de los anteriores, defectos corregidos y ampliándolo con la ventana de selección de eventos.

Iteración 8

En esta iteración se genera la documentación correspondiente al presente Proyecto Fin de Carrera. Para la generación de la documentación se crean las imágenes que componen la documentación. Para la maquetación de la documentación se utiliza el paquete *arco-pfc*. En esta iteración se construyen los vídeos de demostración de la herramienta, además de la presentación de la misma. Esta resulta la última iteración del presente Proyecto Fin de Carrera.

6.1.2. Fechas

El período de realización de cada una de las diferentes iteraciones puede observarse en la figura 6.3.

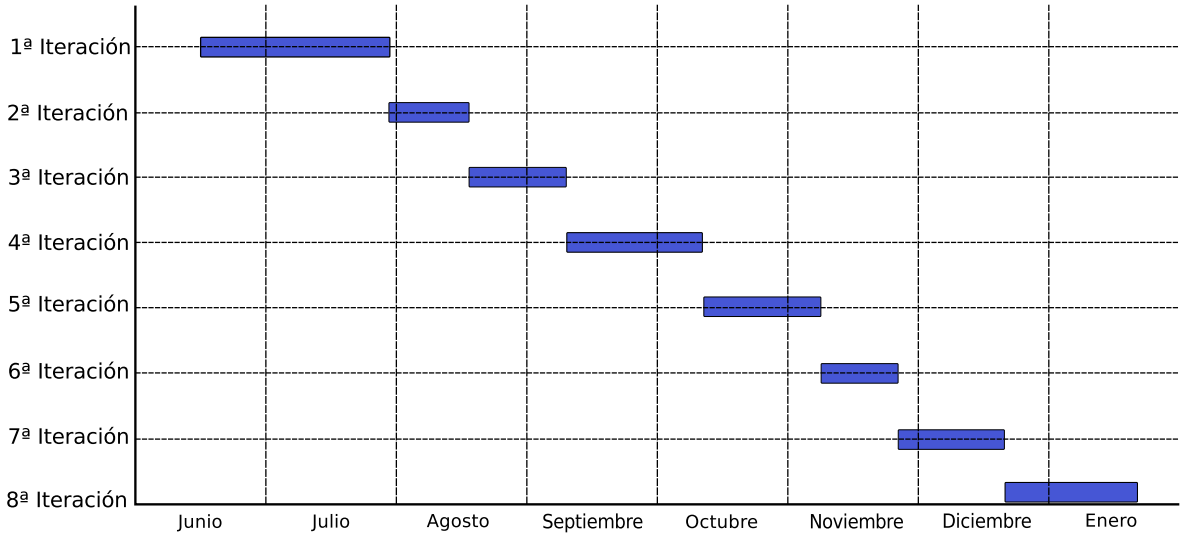


Figura 6.3: Iteraciones en el desarrollo del sistema

6.2. Resultados

Los resultados acerca del proyecto tratan de ayudar a evaluar en qué medida se ha logrado cumplir con los objetivos y requisitos impuestos. Debido a que el trabajo realizado consiste en un *Sistema Inteligente de Análisis de Comportamientos en el Fútbol*, ha sido necesario construir un caso de estudio del sistema recoger estadísticas e información acerca del funcionamiento y rendimiento de éste.

En las siguientes secciones se detalla el caso de estudio realizado, analizando los resultados obtenidos. Además, se estudian diferentes estadísticas del proyecto, tales como el número de líneas o el uso del repositorio. También se analiza el grado de cobertura, a través de pruebas automáticas, que se tiene de la implementación.

6.2.1. Caso de estudio: Final de RoboCup 2012

Introducción

Para determinar la eficacia del sistema desarrollado, se propone analizar un encuentro entre dos equipos de agentes software. El partido se corresponde con la final de la competición *RoboCup Soccer 2D* que tuvo lugar en México. El partido se disputa entre los equipos *HELIOS2012* [ASN+12] y *WrightEagle* [BZL+12] y culmina con un resultado de 4 a 1 favorable al primer equipo.

Se ha elegido este partido para su estudio siguiendo la hipótesis de que **las competiciones oficiales más actuales tendrán que haber implementado estrategias más eficaces que las de hace unos años**. Cuando una competición termina se ponen a disposición de los usuarios los ficheros de simulación, además de las estrategias de los equipos. Así, cualquier persona

interesada puede acceder a ellos. Al tratarse de una competición, los diferentes equipos tratarán de explotar al máximo las debilidades de los demás y para ello pueden utilizar los datos de competiciones pasadas. Por tanto, cuanto más actual sea una simulación más información se tendrá y mejor se prepararán los equipos.

El objetivo de este caso de estudio es analizar la simulación y determinar los eventos que se producen. Además se analizará el comportamiento seguido por los diferentes agentes y se determinarán tanto acciones correctas como incorrectas detectadas por la aplicación.

Configuración del sistema difuso

El sistema difuso se ha configurado de la siguiente manera

- **Conocimiento de bajo nivel**. Se corresponde con las siguientes características:
  - **Variables de entrada**:
    - *X_Pos*: Indica la posición en el eje *X* de un jugador. El dominio de definición de esta variable es en el intervalo [0, 1].
    - *Y_Pos*: Indica la posición en el eje *Y* de un jugador. El dominio de definición de esta variable es en el intervalo [0, 1].
    - *Possession*: Indica si un jugador tiene o no la posesión del balón. El dominio de definición de esta variable se especifica de la siguiente manera; 1 si tiene la posesión y 0 en caso contrario.
    - *Nearly_players*: Indica el número de jugadores oponentes cercanos. El dominio de definición de esta variable es en el intervalo [0, 4].
  - **Variables de salida**:
    - *Attitude*: Indica la actitud de un jugador en el campo. El dominio de definición de esta variable es en el intervalo [0, 1].
    - *Pressure*: Indica la presión a la que está sometido un jugador. El dominio de definición de esta variable es en el intervalo [0, 1].
    - *MidField*: Indica la franja del campo en que se sitúa un jugador. El dominio de definición de esta variable se define de la siguiente manera; 1 si el jugador se encuentra en la franja superior del campo ó 0 en caso contrario.
    - *CenterZone*: Indica si un jugador se encuentra en posición para efectuar un centro. El dominio de definición de esta variable se define de la siguiente manera; 1 si se encuentra en la zona para efectuar un centro ó 0 en caso contrario.
    - *KillerPassZone*: Indica si un jugador se encuentra en posición para efectuar el pase de la muerte. El dominio de definición de esta variable se define de la siguiente manera; 1 si se encuentra en la zona para efectuar un pase de la muerte ó 0 en caso contrario.
- **Conocimiento de alto nivel**: Se corresponde con las siguientes características:

- **Variables de entrada**: Las variables de entrada se corresponden con las variables de salida de la capa de conocimiento de bajo nivel. Además se utiliza la siguiente variable:
  - *MidfieldOpponents*: Esta variable indica los oponentes que se encuentran en la franja de campo contrario. El dominio de definición de esta variable es en el intervalo [0, 11].
- **Variables de salida**: Las variables de salida serán las que se identifiquen como los eventos de la simulación. Por tanto, el dominio de definición de estas variables se define de la siguiente manera; 1 si se produce el evento y 0 en caso contrario. Las variables son las siguientes:
  - *CriticalTheft*: Indica si se puede producir un robo crítico o no.
  - *Shoot*: Indica si se puede efectuar un disparo a portería o no.
  - *ChangeGame*: Indica si se puede realizar un cambio de juego o no.
  - *Center*: Indica si se puede llevar a cabo un centro o no.
  - *KillerPass*: Indica si se puede producir un pase de la muerte o no.

Eventos detectados

En la simulación se detectan un total de 886 eventos, divididos de la siguiente manera:

- **Número de eventos analizados a través de comportamientos colaborativos**: 295. A su vez, en este tipo de eventos, se pueden clasificar en distintos tipos, tal y como puede apreciarse en la tabla 6.1.

  Algunos de los eventos generados por código fuente detectados son los que se pueden apreciar en las figuras 6.4, 6.5.

Tipo de Evento	**Cantidad**
Línea de fuera de juego mal trazada	8
Pase a un compañero adelantado desmarcado	151
Movimiento tras pérdida de balón	136
Total	295

Tabla 6.1: Clasificación de los eventos analizados a través en comportamientos colectivos

Figura 6.4: Detección de la línea mal trazada del fuera de juego

En la figura 6.4 se puede observar una captura de pantalla sobre como el sistema detecta la línea mal trazada del fuera de juego. El jugador cuyo comportamiento es incorrecto es el que lleva dorsal 5 del equipo *HELIOS2012*, el cuál es el equipo que viste de color rojo. En la figura también se puede apreciar como existe otro jugador de mismo equipo cuyo comportamiento no es adecuado, dicho jugador es que lleva el dorsal número 3. El sistema software evalúa la posición del último defensor del equipo y determina si la distancia entre dicho jugador y la posición en el eje X del siguiente jugador es superior al umbral. La situación en que se encuentran estos dos defensores obliga al número 4 a retrasar su posición, puesto que si no lo hace el rival número 7 estaría libre de marca con lo que, podría encarar al portero sin ningún impedimento.

La línea del fuera de juego la forman los jugadores con dorsales 4, 2, 8 y 10 y, si los jugadores 5 y 3 adelantaran su posición hasta la línea del fuera de juego dejarían en una posición antirreglamentaria a los jugadores 11 y 4 del equipo *WrightEagle* y

no podrían intervenir en la jugada, puesto que se encontrarían en fuera de juego. En la figura se puede observar que existe un gran espacio a la espalda del número 4 del equipo rojo, y otro en la frontal del área. Estos espacios podrían ser aprovechados por los delanteros. Si el jugador número 3 del equipo azul, el cuál va a recibir el balón, da un pase en profundidad a cualquiera de esos espacios, los delanteros solo tendrían el impedimento de un defensor y del portero para anotar un gol. Si la línea del fuera de juego estuviera bien trazada en todo momento, un pase al espacio no sería tan crítico, puesto que la defensa encimaría al delantero con al menos dos defensas (un central y un lateral, típicamente).

Por otro lado, se ve perfectamente como el equipo *WrightEagle* tiene trazada correctamente la línea del fuera del juego. La línea de este equipo la forman los defensores con dorsales 5, 9, 8 y 2. En el caso que el equipo rojo recuperase el balón y efectuara un pase rápido a un espacio, el delantero estaría marcado férreamente por dos o tres defensores, lo que haría imposible que recibiera el balón.

El fuera de juego es una de las tácticas más poderosas en el fútbol, puesto que permite que no intervengan los rivales más adelantados que la línea del fuera de juego. Además, si en una jugada no pueden intervenir los delanteros porque se encuentren en fuera de juego, un equipo se vería en la obligación de pasar el balón hacia atrás o en horizontal, dando lugar a una posible pérdida de la posesión del balón.

En la figura 6.5 se puede apreciar una captura de pantalla sobre como el sistema detecta, correctamente, una situación en que un compañero puede efectuar un pase a un compañero adelantado libre de marca. En este caso, el jugador con la posesión de balón es el número 7 del equipo *HELIOS2012* y puede pasar el balón al compañero número 9. Este último jugador se encuentra en una posición más adelantada y además libre de marca, puesto que el defensor número 5 se encuentra unos cuantos metros de distancia.

Además, en la figura se puede apreciar como la acción de pase al jugador número 9 es una de las pocas opciones que tiene el jugador número 7, puesto que el resto de sus compañeros de equipo están sometidos a estrechos marcajes por parte de la defensa rival. El único jugador que no está marcado de esta manera es el jugador número 3. Sin embargo, ese jugador está en una posición muy distante como para pasarle el balón. El comportamiento de la defensa es correcto, ya que ha encerrado a los jugadores de *HELIOS2012* en la banda izquierda, imposibilitando que se desplace el balón a otra zona del campo. Sin embargo, si el jugador 7 finalmente pasa el balón al jugador 9, el equipo rojo tendría una posibilidad de anotar gol, puesto que, si este último jugador efectúa un pase rápido y preciso a las espaldas de los números 5 y 9 de *WrightEagle*, permitiría que el jugador número 11 pudiera rematar a portería.

Otra opción posible en caso que el número 9 recibiera el balón sería efectuar un regate

Figura 6.5: Detección del pase a un compañero más adelantado

sobre el número 5 y posteriormente realizar un centro al centro del área que podría ser rematado bien por el número 11 o por el número 10. En las dos posibilidades de jugada anteriores, existe el riesgo de perder la posesión del balón, pero, en caso que esto ocurriera no sería demasiado peligroso, ya que el balón se encontraría muy distante de la portería.

Efectuar un pase a un compañero cuya posición es más adelantada y además este libre de marca siempre será una importante opción a considerar, puesto que sirve para adelantar la posición del balón a una zona segura. Además, cuando el compañero desmarcado reciba el balón, siempre tendrá al menos dos jugadas, bien mantener la posesión del balón retrasando el balón al jugador que le ha pasado el balón, o bien efectuar algún tipo de regate sobre la marca que le encime.

- **Número de eventos analizados a partir de la base de conocimiento**: 591. Al igual que el apartado anterior, se pueden clasificar en distintos tipos (véase tabla 6.2).

Tipo de Evento	Cantidad
Disparo	101
Robo crítico	30
Centro al área	240
Pase de la muerte	130
Cambio de juego	90
Total	591

Tabla 6.2: Clasificación de las eventos analizados a través de la base de conocimiento

Algunos de los eventos de usuario pueden apreciarse en las figuras 6.6, 6.7 6.8, 6.9.

Figura 6.6: Detección del posible cambio de juego

En la figura 6.6 se puede apreciar como el sistema detecta, correctamente, la situación en que un jugador pueda cambiar el juego. En este caso, el jugador 2 de *HELIOS2012*, que es el que posee la posesión del balón, tiene la opción de efectuar un pase largo a la otra banda donde se encuentra su compañero con el dorsal número 9. El pase debe efectuarse elevando el balón para que pueda superar a los rivales.

El cambio de juego es una de las pocas opciones que posee el jugador número 2, puesto que los defensores de *WrightEagle* están efectuando una presión adelantada cubriendo la mayoría de huecos. Otras posibles opciones, serían efectuar un pase en corto al número 3, pero este

pase podría ser interceptado por el jugador número 11 de *WrightEagle* y comprometería la defensa de *HELIOS2012*. La otra posible opción es efectuar un pase al espacio que existe a la espalda del compañero 9, para que fuera recogido por el jugador número 4. Esta opción tampoco resulta ser la mejor, puesto que si el pase no es preciso o se realiza con más fuerza de la necesaria impediría que el jugador número 4 obtuviera el balón, y se generará un saque de banda en una zona del campo peligrosa para *HELIOS2012*.

La mejor opción es el cambio de juego al número 9, puesto que en esa zona existe menos sobrecarga de rivales, ya que la mayoría de ellos se encuentra en la parte inferior del terreno de juego. Además, en caso que el pase no fuera preciso, la zona en donde se perdería la posesión del balón no sería tan peligrosa como la zona de pérdida en las otras opciones.

En cambio, si el pase resulta preciso resultaría muy efectivo, ya que la mayoría de los jugadores de *WrightEagle* están comprometidos en la presión adelantada, dejando la defensa desguarnecida. Si el jugador número 9 recibiera el balón, tendría mucho espacio a la espalda de la defensa rival para iniciar una carrera o para que el número 11 iniciara un desmarque que le permitiera encarar al portero mano a mano.

El cambio de juego es siempre una opción a considerar en caso que el equipo contrario se encuentre presionando la salida del balón o que la acumulación de jugadores en una franja del campo sea significativamente elevada con respecto a la acumulación de jugadores en la otra franja del campo. La intención con la que debe realizarse un cambio de juego es la liberar al equipo de la zona dónde intenta acorralar el juego la defensa rival.

En la figura 6.7 se puede apreciar como el sistema detecta, correctamente, la situación en que un jugador pueda disparar a puerta. En este caso, el jugador número 4 de *WrightEagle*, que es el que posee la posesión del balón, tiene la opción de disparar a portería. Además, se puede apreciar como el portero está ligeramente adelantado, lo que permitiría que el atacante elevara el balón por encima, efectuando una vaselina.

En la situación de la figura 6.7, una de las pocas opciones de las que dispone el jugador número 4 de *WrightEagle* es la disparar a puerta. También, este jugador podría avanzar con el balón, pero se le podría echar encima el jugador número 5 de *HELIOS2012*, y robarle el balón. En este caso, una pérdida de balón sería peligrosa, puesto que, aunque el balón se encuentre lejos de la portería de *WrightEagle*, el jugador número 10 de *HELIOS2012* se encuentra libre de marca y, los jugadores 9 y 11 de ese mismo equipo se encuentran en una disposición idónea para el contraataque.

Si, finalmente el jugador 4 realizase el disparo a portería, podrían ocurrir diferentes situaciones, que se marcara gol o que el portero rechazara el balón a saque de esquina. Estas situaciones se producirían en los mejores casos. Sin embargo, en el peor de los casos, el balón acabaría en posesión del portero, bien porque bloque el balón, o bien porque el disparo sea defectuoso. En estos dos casos, lo único que ocurriría sería la pérdida del balón en una

Figura 6.7: Detección del posible tiro a puerta

zona en la que no existe posibilidad de contraataque por parte de *HELIOS2012*.

El disparo a puerta es siempre la principal opción que debe considerar un delantero cuando esté cerca del área rival libre de marca. Además, típicamente un delantero tendrá más capacidad para realizar disparos efectivos, con lo que las posibilidades de que el portero detenga el balón disminuyen, aumentando las posibilidades de anotar un gol o de generar un saque de esquina.

En la figura 6.8 se puede apreciar como el sistema detecta, correctamente, la situación en que un jugador pueda efectuar un centro aĺ área. En este caso, el jugador número 8 de *HELIOS2012* se encuentra efectuando un saque de esquina. Este jugador puede efectuar un centro al área. El destinatario del balón bien podrá ser el compañero que se encuentra marcado por el defensor número 2 de *WrightEagle*, o bien podrá utilizarse alguna táctica de equipo para los saques de esquina.

Las dos posibles opciones que posee el jugador que va a efectuar el saque de esquina son el pase en corto al compañero número 11 o el centro al área. Tal y como se ha visto en el capítulo 3, el pase en corto es una opción que da lugar a segundas jugadas dentro del área, así pues lo más probable, es que cuando se efectúe el saque en corto, el jugador número 11 de *HELIOS2012* devuelva el balón al compañero número 8 y este centre al área.

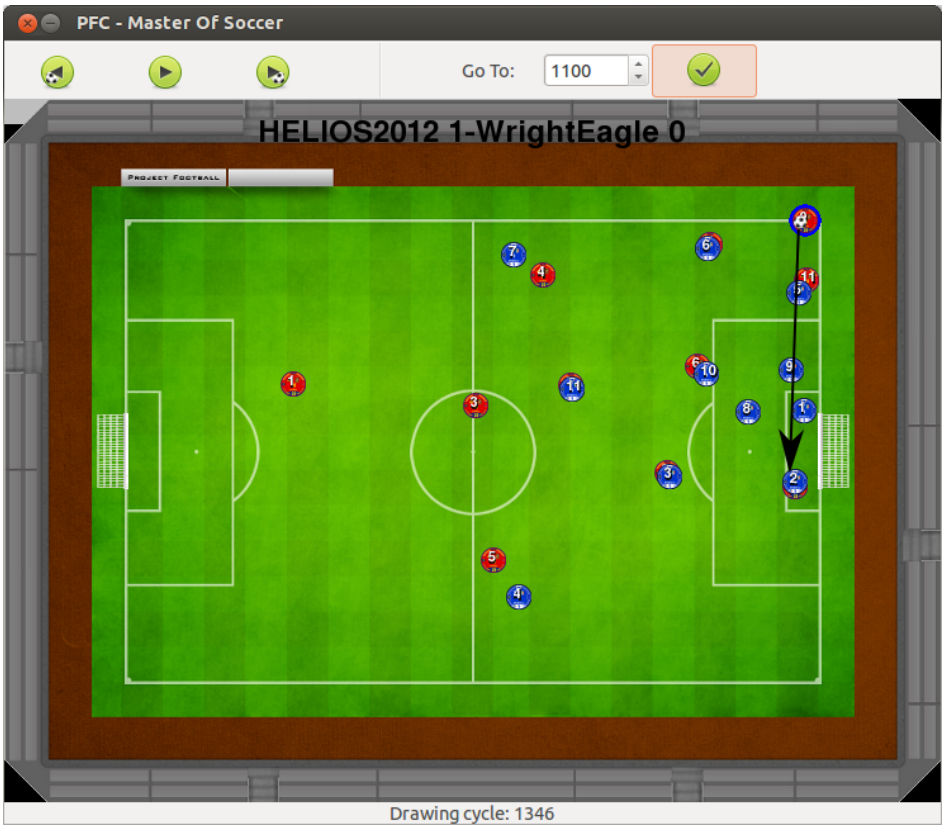


Figura 6.8: Detección del posible centro

En caso que el jugador realice el centro al área, existen muchas jugadas ensayadas, que permiten la generación de espacios que pueden ser aprovechados por diferentes compañeros. En esta situación, una posible jugada ensayada podría ser el desplazamiento del balón hacia el punto de penalti dónde, el compañero que se encuentra marcado por el jugador número 3 de *WrightEagle* realizase un desmarque y rematase en solitario. Otra posible jugada podría ser que el compañero que sufre la marca del jugador número 3 azul realizase un bloqueo sobre el jugador número 10 del mismo equipo, lo que permitiría que el compañero número 6 se quedase solo en la frontal del área, con lo que si el centro va a esa posición dicho jugador podría disparar a portería.

En definitiva, cuando un jugador se encuentra escorado en una banda una opción que siempre debe considerar es centrar al área, puesto que o bien un compañero remata o hay posibilidad de una segunda jugada dentro del área o se produce un fallo en la defensa rival que de ventaja. En caso que el centro sea defectuoso y se pierda la posesión, la peligrosidad de contraataque no es tan elevada como si un rival te roba el balón en la banda.

En la figura 6.9 se puede apreciar como el sistema detecta, correctamente, la situación en la que un jugador puede sufrir un robo de balón que resulte crítico para los intereses de un equipo. En este caso, el jugador de *HELIOS2012*, señalado con un círculo azul, puede sufrir un pérdida de balón por parte de los defensores 6, 5 y 10 de *WrightEagle*.

Figura 6.9: Detección del posible robo crítico

En este caso si el jugador de *HELIOS2012* perdiera el balón, el equipo azul se encontraría en una situación idónea para marcar gol, puesto que el equipo rojo se encuentra dando salida al balón y no está bien colocado en el campo. Si se produjese el robo de balón, el jugador que se apoderase de la posesión del balón podría efectuar un pase tanto al jugador número 7 como al jugador número 11 a la espalda de la defensa de *HELIOS2012*, dejando como único obstáculo para evitar el tanto al portero. Por otro lado, también se podría efectuar un pase al jugador número 3, que se encuentra completamente liberado de cualquier marcaje. Si este último jugador recibiera el balón crearía un desajuste defensivo, puesto que si el defensor número 5 corre a quitarle el balón, el atacante azul número 4 quedaría solo. Sin embargo, si es el defensor número 2 el que intentase recuperar el balón, el atacante número 3 azul podría efectuar un pase al hueco para el desmarque del atacante número 11 de *WrightEagle*.

Sin embargo, si las situaciones de robo crítico se resuelven con rapidez y precisión permiten obtener ventaja con respecto al rival. En este caso, si el jugador con la posesión del balón, realizase un pase al compañero 6, que está libre de marca, se tendría una situación de peligro para *WrightEagle*, puesto que tres atacantes (6, 11 y 10) se enfrentarían a tres defensores (9, 8 y 2) con mucho espacio a las espaldas de estos últimos. Si el jugador número 6 recibe el balón, existen más jugadas diferentes, por ejemplo que el jugador rojo número 9 marcado por el defensor número 5, iniciase un desmarque. Este desmarque le permitiría avanzar muchos

metros hacia la portería rival con la posesión del balón.

Cuando se detecta un robo crítico un jugador debe actuar con precisión y rapidez y no perder el balón. La posibilidad de un robo crítico está ligada tanto a la posición del campo en que se pueda producir, como al número de defensores que hostigan al poseedor del balón. Al ir muchos oponentes a arrebatar la posesión del balón, muchos jugadores del equipo quedarán libres de marca y podrán ser receptores del balón, dando lugar a otras posibles jugadas de ataque.

6.2.2. Estadísticas del proyecto

Líneas de código

A modo informativo, la tabla 6.3 determina las líneas de código escritas en cada uno de los lenguajes que se han empleado para el desarrollo de la aplicación. Para obtener estos datos se ha empleado la herramienta libre `cloc`[1], que contabiliza el número de líneas escritas de un proyecto en cada lenguaje.

Lenguaje	Ficheros	Líneas en blanco	Comentarios	Total
Python	51	642	138	3948
Java	14	191	189	970
XML	5	0	0	176
Make	2	0	0	5
Total	72	833	327	5099

Tabla 6.3: Estadísticas del proyecto

Uso del repositorio

Este apartado permite determinar la actividad del desarrollador de la aplicación en el sistema de control de versiones empleado. La figura 6.10 indica, por un lado y de color rojizo, el número de *commits* y, en color azul, el número de cambios que ha sufrido el software. Estas estadísticas son generadas mes a mes, lo que permite realizar un estudio de ello y determinar que el mes más relevante en este contexto del software fue Diciembre. Si se estudian las tareas desarrolladas en el apartado 6.1, se llega a la conclusión que existen más cambios debido a la incorporación de los eventos por código, a la detección y resolución de defectos que no se habían encontrado en etapas anteriores y al inicio de la documentación, lo que genera un número de cambios significativamente mayor que el resto de meses.

Sin embargo, el mes que contiene más *commits* es Julio. Esto se debe a que es la etapa del proyecto donde se implementaron las primeras aproximaciones de la mayoría de las clases que componen el proyecto.

[1] http://sourceforge.net/projects/cloc/

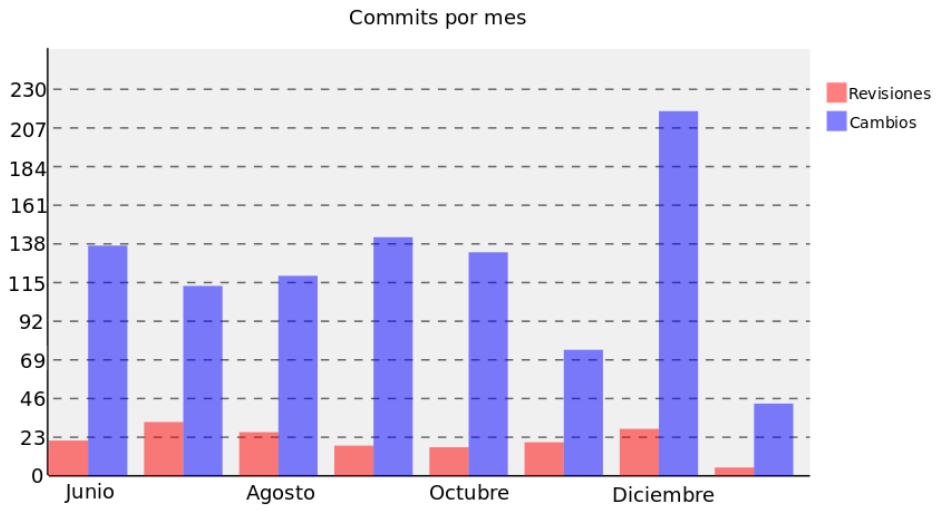


Figura 6.10: Commits y cambios por mes

6.2.3. Pruebas

La utilización de bibliotecas de prueba aporta muchas ventajas. Una de las más importantes es determinar la cobertura del código ofrecida por los *test* automáticos. Esto permite tener una idea general de la cantidad de código que realiza lo que debe hacer y no realiza lo que no debe hacer. Además, si una nueva funcionalidad se incorpora, sería suficiente con lanzar las pruebas automáticas para determinar si dicha funcionalidad causa errores en el comportamiento de la aplicación.

Las tablas 6.4 y 6.5 determinan la cobertura de código en las capas de persistencia y dominio de la aplicación. Acerca de la interfaz no se tienen datos exactos sobre la cobertura del código, debido a la imposibilidad que supone realizar pruebas automáticas en la misma. En cambio, se han realizado pruebas de aceptación con el director del proyecto y el resultado ha sido satisfactorio.

La cobertura media del código se encuentra por encima del 80 % lo que permite concluir que la mayor parte del código se encuentra cubierto por las pruebas.

Se han realizado 65 *test* divididos de la siguiente manera:

- **Número de *tests* en de las clases *Python***: 49

- **Número de *tests* en de las clases *Java***: 16

El número de test en las clases *Python* es significativamente mayor, debido a que el número de líneas escritas utilizando este lenguaje de programación es también significativamente superior al número de líneas escritas en *Java*.

Clase	Cobertura
DomainFacade	66 %
Goal	100 %
Object	81 %
ParserXML	97 %
Plotter	96 %
Point	88 %
Region	100 %
Simulation	45 %
State	71 %
StatePlayer	89 %
Team	80 %
Tree	74 %
RestoreDB	93 %
SQLFacade	83 %
DBAgent	95 %
Event	100 %

Tabla 6.4: Cobertura de los *Test* de las clases Python

Clase	Cobertura
FuzzySystem	81.25 %
ParserEvents	75 %
DBAgent	85.71 %
SQLFacade	50 %

Tabla 6.5: Cobertura de los *Test* de las clases Java

Elemento	Precio	Total	PrecioTotal
Desarrollador	25€/h	528	13200€
Computador portátil Samsung R159	399€/unidad	1	399
			13599€

Tabla 6.6: Precio del proyecto

6.3. Coste

El desarrollo del proyecto ha tenido una duración de 7 meses. Si se asume que son 4 horas / día la media de horas empleadas al día y que el sueldo de una persona con conocimientos de programador y de gestión de conocimientos es 25 €/hora. El coste del desarrollo del proyecto es a 13200 €. A ese total hay que añadir el precio del computador utilizado, es decir, 399€. El precio total del proyecto es 13599€(véase tabla 6.6).

Capítulo 7

Conclusiones y Propuestas

En este capítulo se expone una valoración acerca de los objetivos alcanzados, teniendo en cuenta la construcción de la arquitectura del sistema y los resultados obtenidos tras la aplicación del mismo. Además, se realizan una serie de propuestas futuras para ampliar y mejorar las funcionalidades de *Sidane*. Para concluir, se realiza una valoración personal acerca de lo que ha supuesto la realización del presente Proyecto Fin de Carrera.

7.1. Objetivos alcanzados

La motivación principal de este proyecto es el gran impacto social y económico que posee la industria deportiva en los últimos tiempos y, particularmente el fútbol. Como se ha visto en el capítulo 1, el fútbol genera alrededor de 500.000 millones de dólares anuales y es seguido por millones de personas en todo el mundo. Tanto es así, que las últimas estimaciones determinan que alrededor de 270 millones de personas practican este deporte en todo el mundo.

El objetivo principal era ofrecer un sistema inteligente para el análisis automático de comportamientos de jugadores de fútbol. Para que esto sea posible, ha sido necesario el estudio de muchas herramientas que permitieran desarrollar de la mejor manera posible este sistema.

Actualmente, existe una carencia de este tipo de sistemas debido a su carácter experimental. No resulta sencillo desarrollar un sistema que identifique situaciones y evalúe automáticamente el comportamiento de diferentes entidades en dichas situaciones. La complejidad reside en la gran cantidad de acciones que es posible realizar en una misma situación. Por ejemplo, en el contexto del fútbol, el comportamiento correcto de un jugador que se encuentre desmarcado dentro del área rival, deberá disparar a portería. Sin embargo, el jugador podría realizar otras acciones diferentes como pasar atrás o avanzar con el balón, en estos casos, el comportamiento del jugador no sería el adecuado.

Una de las principales ventajas de las que ha gozado el desarrollo del proyecto han sido las continuas reuniones con el director de proyecto. Estas reuniones han sido necesarias tanto para idear soluciones a diferentes problemas como para evaluar la forma de implementación

requisitos ya cubiertos. Esto dota a la herramienta de mayor calidad.

Uno de los requisitos del sistema era la **modularidad**. Este objetivo se encuentra cubierto, puesto que se han creado diferentes módulos, los cuáles, tienen una responsabilidad e interactúan con los demás para la realización de las tareas del sistema. Así, el mantenimiento y la ampliación del sistema no resulta costosa.

Otro de los requisitos era la creación de una **arquitectura adaptativa** y **extensible** basada en patrones de diseño. Tal y cómo se puede observar en el capítulo 5, este objetivo se ha cubierto utilizando diferentes patrones de diseño que se aplican en la ingeniería del *software*.

También se ha cubierto el objetivo de crear una **interfaz interactiva** y atractiva y que resultara **fácil de utilizar** para el usuario. El usuario puede ver la totalidad de la información almacenada en la base de datos a golpe de ratón. Un ejemplo de ello, se puede apreciar en el anexo A, el cuál especifica un manual de usuario de la aplicación con las diferentes opciones que posee la aplicación. En dicho manual, se encuentran imágenes de las diferentes ventanas que posee la herramienta.

La herramienta es totalmente **portable**, puesto que, se han utilizado estándares para su implementación, así como, lenguajes de programación y bibliotecas portables. Algunas de las herramientas se han descartado, puesto que, no ofrecían las características de portabilidad requeridas por el proyecto.

El sistema se ha integrado con el sistema *Redmine*, el cuál, es utilizado por diversos proyectos del grupo de investigación *Oreto*. Por tanto, el objetivo de **distribución** ha sido cubierto gracias al uso de esta herramienta.

En el capítulo 6 se construye un **caso de estudio** basado en la final de la competición de la *RoboCup Soccer 2D* del año 2012. Esto ha logrado la **generación de resultados** de importancia para la evaluación del rendimiento de la herramienta. Los resultados han sido comparados y gracias a ellos se obtienen diferentes comportamientos seguidos por los jugadores a lo largo de un partido. Los resultados obtenidos han sido satisfactorios. Por una parte, han sido identificados una gran variedad de comportamientos colectivos. Resulta especialmente significativo las pocas ocasiones en que se traza de manera incorrecta la línea de fuera de juego en comparación con otros eventos. Dentro del contexto del caso de estudio (la final de la *RoboCup Soccer 2012*) se puede concluir que el comportamiento sobre la línea de fuera de juego está correctamente implementado por parte de los agentes que disputan el partido. Sin embargo, existen muchas situaciones en las que se ha identificado que se debe pasar el balón a un compañero adelantado o que hay que realizar un movimiento tras la pérdida del balón. En el transcurso de la simulación, los agentes tuvieron un comportamiento correcto y realizaron la acción apropiada en muchas ocasiones, aunque existen diferentes situaciones en las que siguieron un comportamiento erróneo. Las principales causas de que los agentes

no realizarán la acción esperada se debe, en la mayoría de los casos, a que *RoboCupSoccer* se trata de un simulador, en el cuál las características de los jugadores se encuentran acotadas por diferentes parámetros. Uno de estos parámetros que influye significativamente en el simulador es el del campo de visión de los jugadores. En la realidad, un jugador de fútbol puede divisar la totalidad del campo desde cualquier parte del mismo, independientemente de la posición en la se encuentre. Esto no ocurre así en el simulador, los agentes no ven más haya de lo que le permita el parámetro de configuración encargado de limitar la visión, lo que supone un inconveniente a la hora de realizar diferentes acciones.

Desde el punto de vista de los comportamientos definidos por el usuario, resultan significativas las pocas ocasiones en que se puede producir un robo crítico. Esta información indica que esta situación resulta demasiado peligrosa para los intereses de un equipo y la intentan evitar en la mayoría de los casos. No ocurre así con las otras situaciones definidas. Es especialmente significativo el número de centros al área que se pueden producir y se debe, principalmente, a que los agentes de la simulación están implementados de tal manera que, aprovechar el juego por bandas supondrá una ventaja a la hora de disputar un partido. Los comportamientos definidos a nivel de usuario están ligados al conocimiento almacenado en las bases de conocimiento. Dicho conocimiento puede ser más o menos restrictivo en función de la definición de las variables contenidas en el. Con unas variables especificadas de una manera más estricta se obtendrán menos situaciones que si estuvieran especificadas de una forma más flexible, aunque las situaciones serán más precisas. Para el caso de estudio el conocimiento se ha definido buscando un compromiso entre las situaciones a detectar y la flexibilidad de las variables.

En resumen, tanto el **objetivo general**, como los **objetivos específicos** del capítulo 2 han sido cubiertos en su gran mayoría. Esto demuestra el nivel de implicación y compromiso tanto del desarrollador de la herramienta, como del director de proyecto.

7.2. Propuestas y trabajo futuro

Aunque los objetivos propuestos se han cumplido, el proceso de desarrollo del software no es estático sino que evoluciona con el tiempo. Así, se pueden apreciar algunas líneas de trabajo futuro que completen o mejoren las funcionalidades que ofrece *Sidane*. Las líneas de trabajo identificadas de esta manera son las siguientes:

- **Implementar nuevos comportamientos**. La herramienta cuenta con la definición de tres comportamientos colectivos, cuya implementación resulta especialmente costosa utilizando el sistema difuso. Dichos eventos se corresponden con la detección de la línea de fuera de juego mal trazada, del pase a un compañero más adelantado desmarcado y del movimiento tras la pérdida del balón.

Se podrían estudiar otros comportamientos que se realicen en el fútbol e implementarlos en la herramienta. Para ello, sólo haría falta pensar dicho comportamiento y escribir nueva funcionalidad en la clase *Simulation*.

- **Puesta en contacto con firmas comerciales**. Para que el proyecto pueda evolucionar son necesarios diferentes recursos, tanto económicos como de infraestructura o marketing, entre otros. Por tanto, sería imprescindible para continuar el desarrollo del proyecto obtener financiación de organizaciones comerciales.
- **Hacer más amigable para el usuario la forma de definir los eventos**. Actualmente, la forma en se definen los eventos puede resultar compleja de cara a usuarios que no tengan conocimientos informáticos medios y no tengan conocimiento de lógica difusa.

  Para abordar esta nueva funcionalidad, se podría implementar una interfaz gráfica sencilla que permitiera a los usuarios definir nuevos comportamientos, sin tener que conocer la lógica difusa.
- **Implementar un sistema de *log* para el análisis de comportamientos**. En la herramienta se pueden observar todos los eventos de una simulación y el ciclo en que se producen. Sin embargo, un comportamiento no se evalúa hasta que no ocurre en el desarrollo de la simulación. De esta manera, un usuario puede determinar si el comportamiento realizado por un jugador es correcto o no.

  Se podría construir un sistema que escribiera un *log*, al cargar la simulación para determinar el número de comportamientos correctos e incorrectos. Dicho archivo de log almacenaría también el tipo de evento y el ciclo en que se produce.
- **Reducir el tiempo de espera que supone la generación de nueva información**. Actualmente, el tiempo empleado por esa tarea es bastante elevado, puesto que genera información de comportamientos para cada ciclo de simulación, con el consiguiente trasiego que supone con la base de datos.

  Para abordar este trabajo se podría llegar a un compromiso entre generación de información y tiempo utilizado. Aunque, habría que tener en cuenta que la información generada es utilizada, tanto para la detección de los eventos de usuario como para el análisis de comportamientos de los jugadores.
- **Implementar más opciones de personalización de la interfaz gráfica**. Actualmente, sólo se puede personalizar las imágenes de los equipos de la simulación y del balón. La idea sería implementar la funcionalidad necesaria para que se pudiera personalizar el campo en que se desarrolla la simulación, determinar la rotación de las imágenes según el ángulo del cuerpo o pintar el ángulo de visión de los jugadores en el campo.

  Estos datos se pueden encontrar en una simulación y son almacenados en la base de datos. Habría que implementar primitivas en el agente para poder recuperar dichos datos y utilizarlos para dibujar los jugadores en el terreno de juego.

- **Adaptar la herramienta a las tecnologías Web y móviles**. La herramienta sólo se puede ejecutar como aplicación de escritorio, con lo que queda bastante aislada de las tecnologías que están copando el mercado informático en estos momentos.

  Es común que la mayoría de las personas que estuvieran interesadas en el uso de esta aplicación, posean dispositivos móviles o tengan conexión a internet. Por tanto, sería interesante crear aplicaciones tanto móviles como web, que permitieran el análisis de las simulaciones. De esta manera, la aplicación podría llegar a ser utilizada por mayor número de personas.

7.3. Valoración personal

El desarrollo del presente Proyecto Fin de Carrera, además de servir para completar mis estudios superiores, me ha ayudado a obtener otro punto de vista sobre cómo se construyen las sistemas software con una complejidad significativa. Desde mi punto de vista y sin tener en cuenta las prácticas en empresa, el Proyecto Fin de Carrera es lo más similar al entorno laboral a lo que me he enfrentado a lo largo de la titulación universitaria, y como tal lo he encarado. Si de algo me enorgullezco en la realización de este proyecto es de que siempre he cumplido con los plazos que se han propuesto para la realización de una tarea, sin tener que retrasar ni un sólo día una reunión con mi director de proyecto. Me he demostrado a mí mismo que con esfuerzo y tesón, las cosas que resultan imposibles en un principio, no lo son tanto en realidad.

Desde el punto de vista tecnológico, he trabajado con herramientas que me hacen disfrutar con lo que hago. Algunas de estas herramientas son las que permiten probar el código automáticamente. En materia de pruebas, la realización de este proyecto me ha ayudado a implementar en la práctica conceptos que había aprendido en le teoría. Una de las conclusiones que extraigo acerca de la realización de este proyecto es que me resulta difícil confiar en el código que no está probado mediante pruebas automáticas. Por cada defecto de código probado mediante pruebas automáticas, he tenido que corregir muchos otros de código no probado de esta manera.

En conclusión, el resultado obtenido es satisfactorio. Prueba de ello son los resultados obtenidos. Espero que esta herramienta pueda seguir creciendo con el trabajo futuro propuesto y que, finalmente, pueda servir para solventar muchas de las necesidades de diferentes equipos de fútbol, que es para lo que ha sido creada.

ANEXOS

Anexo A

Manual de Usuario de Sidane

En este anexo se muestra el manual de usuario de para instalar, configurar y ejecutar Sidane.

A.1. Configuración de la base de datos

Para la correcta configuración de la base de datos, se recomienda el uso del programa *MySQL-Workbench*. Este manual se ha apoyado en el uso de este programa.

Para iniciar la configuración de la base de datos, ejecutamos *MySQL-Workbench* y se generará una ventana semejante a la que podemos observar en la figura A.1.

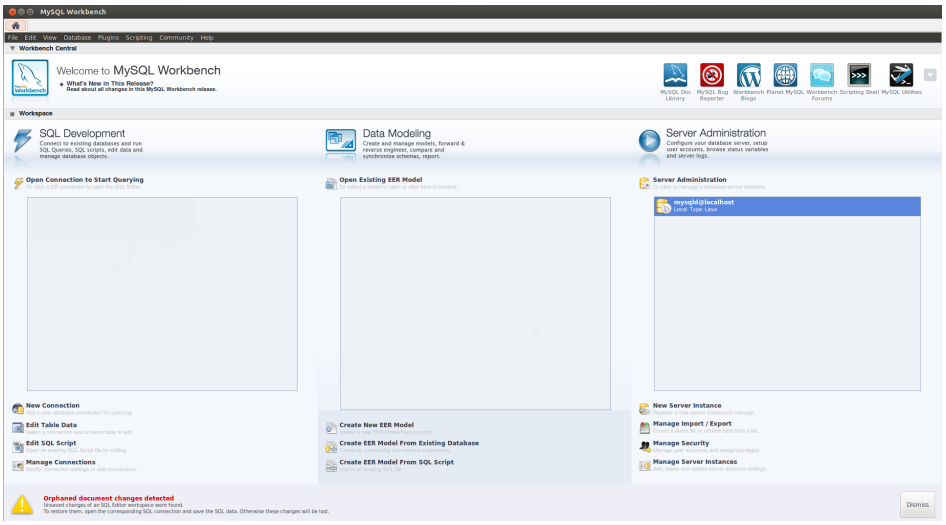


Figura A.1: Ventana principal *MySQL-Workbench*

Los pasos necesarios para configurar la base de datos son los siguientes:

1. Hacer click sobre la opción ***New Connection***.
2. Se abre una ventana como la que se puede observar en la figura A.2. En esta ventana, se introducen los datos necesarios para que *Sidane* pueda conectarse correctamente a la base de datos. Los datos a introducir deberán ser los mismos que se han especificado en el fichero *Constants.py* (ver listado A.1). Cuando se hayan introducido todos los datos correspondientes, se pulsa sobre el botón *Ok*.

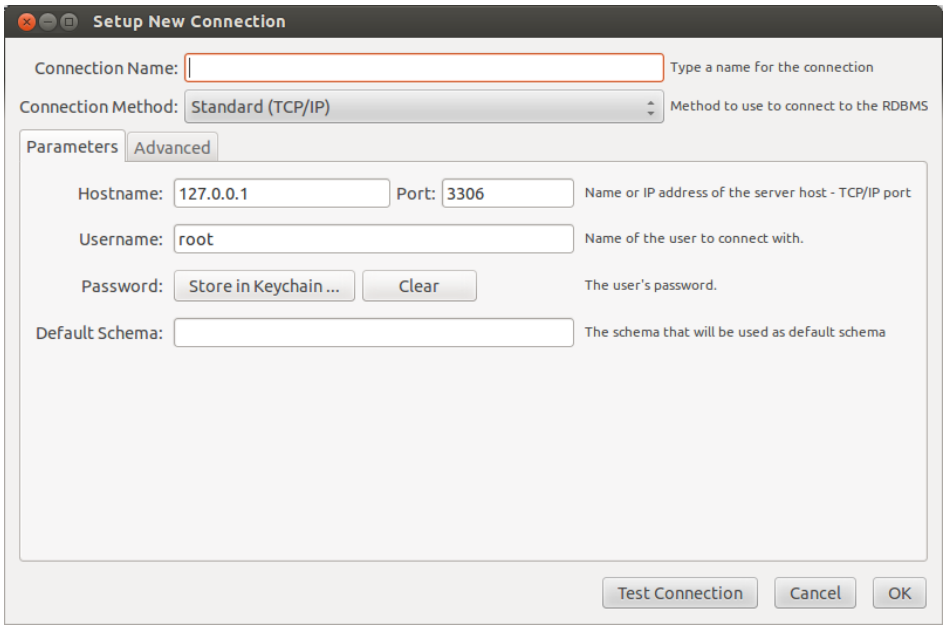


Figura A.2: Ventana de *Nueva conexión*

```
1       DB_HOST = ''localhost''
2       DB_USER = ''root''
3       DB_PASS = ''pass''
4       DB_SCHEMA = ''MasterSoccer''
```

Listado A.1: Fichero de configuración de los parámetros de la base de datos.

3. Estando de nuevo en la pantalla principal, se hace doble click sobre la nueva conexión creada y se observa la pantalla que se puede apreciar en la figura A.3.

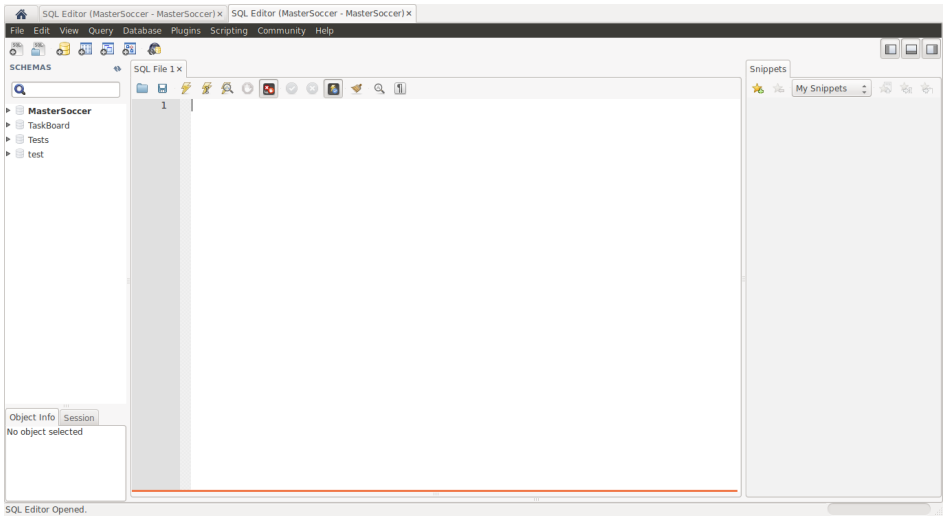


Figura A.3: Ventana de realización de consultas.

4. Se hace click en el icono indicado por la figura A.4. El fichero a cargar será el denominado *tables.sql* que se encuentra en la carpeta *db* de *Sidane*. Una vez seleccionado, se pulsa el botón abrir, lo que hará que el código sql se genere.

Figura A.4: Icono de introducción de código *SQL*

5. Una vez que el código este generado, se hace click sobre el icono con forma de rayo (ver figura A.5), lo que hará que se ejecute el código sql y tengamos la base de datos lista para comenzar a utilizarse.

Figura A.5: Icono de ejecución del código *SQL*

A.2. Almacenar una simulación en la base de datos

Para almacenar una simulación es necesario seguir tres sencillos pasos, estos pasos son los siguientes:

- **Almacenar la simulación**. Para la realización de este paso es necesaria la ejecución del `Parser_xml`. Para ello, mediante la consola de comandos se accede al directorio `domain/`, en el cuál ejecutaremos el comando `./Parser_xml.py <simulación.xml>`. Varias simulaciones se pueden encontrar en el directorio `simulations/`. Este proceso puede durar entre dos y tres minutos.

  Una vez se ejecuta este primer paso, se puede lanzar la herramienta para reproducir la simulación sin eventos. Además de la reproducción, se podrán ver las características de los jugadores, generar gráficos o especificar tipos de jugadores, entre otras cosas.

- **Generar información de la simulación**. Cuando se almacena una información, se guarda la información contenida en el fichero *XML*. Sin embargo, para poder detectar los eventos de una simulación, es necesario generar nueva información a partir de la ya almacenada. Para ello, mediante la consola de comandos se accede al directorio `domain/`, en el cuál ejecutaremos el comando `./Upadte_database.py`. De esta manera, se generará nueva información en la base de datos, además de los eventos generados por código fuente.

Cuando la ejecución del comando termine, se podrá utilizar la herramienta para determinar los eventos generados por código y analizar el comportamiento de los jugadores.

- **Introducir los eventos de usuario**. Este es el último paso y, para su realización, es necesaria la ejecución de los dos pasos anteriores. Para llevar a cabo la inserción de los eventos de usuario, es necesario navegar en la línea de comandos hasta el directorio `exec/`. En ese directorio, se ejecutará el comando `java -jar FuzzySystem.jar`. Este comando se encargará de reconocer e interpretar el contenido de los ficheros *kb/PlayerKnowledge.fcl*, *kb/PlayerActions.fcl* y *kb/events.xml*, para generar la información relativa a los eventos definidos por el usuario.

  Cuando se ejecute este paso, la simulación se encontrará completamente analizada, y se podrán visualizar todos los análisis de los comportamientos que realizan los jugadores.

A.3. Uso de la aplicación

Cuando se inicie la aplicación, se observa una ventana como la que se puede apreciar en la figura A.6. Dicha ventana, es la ventana principal de la herramienta, sobre la cual, se puede acceder a las distintas opciones la aplicación. A continuación, se detallan cada una de las posibilidades que ofrece el programa.

Figura A.6: Ventana principal

A.3.1. Menú *File*

En este menú se pueden encontrar las opciones relativas a la gestión de la simulación, tales como, cargar una simulación o cerrarla.

Abrir una simulación

Esta opción puede encontrarse como la entrada denominada *Open Simulation...*. Al interactuar con el botón izquierdo del ratón sobre dicha opción, se observa una ventana como la que puede apreciar en la figura A.7. El objetivo de esta ventana es cargar una simulación que se encuentre en la base de datos. Para ello, pinchando sobre la caja de combo (numerada con un uno en la figura A.7), se desplegará una serie de simulaciones, pinchando sobre alguna de ellas, se seleccionará. Por último, haciendo click con el botón izquierdo del ratón sobre el botón *Ok* (numerado con un dos en la figura A.7), se carga la simulación, viéndose en la ventana principal el primer ciclo de la simulación (ver figura A.8.

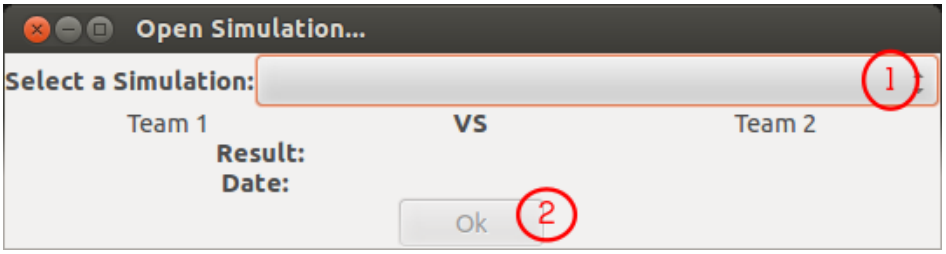


Figura A.7: Ventana *Open simulation...*

Cerrar una simulación

En la aplicación, esta opción se denomina *Close Simulation...*. Al hacer click sobre esta opción, la simulación cargada se eliminará, de manera que no se puede volver a interactuar con ella hasta que la volvamos a cargar, tal y como se vio en A.3.1.

Salir

Identificada como *Exit* en la herramienta, esta opción sirve para cerrar la misma.

A.3.2. Menú *View*

En este menú se pueden encontrar las opciones referentes a la personalización de partes de la interfaz. Este menú posee las siguientes opciones:

Cambiar las equipaciones de los jugadores

Esta funcionalidad permite modificar el aspecto de los jugadores dentro del campo. Así pues, se da la posibilidad al usuario de personalizar la simulación con los equipos que desee.

Figura A.8: Simulación cargada

Actualmente, la aplicación contiene las equipaciones, tanto en su modalidad local como visitante, correspondientes a los equipos que disputaron la Eurocopa de 2012.

Al hacer click sobre esta opción, se generará una nueva ventana semejante a la que puede apreciarse en la figura A.9. Esta ventana posee dos cajas de combo (numeradas en la figura A.9 como uno y dos), mediante las cuales se podrá seleccionar la vestimenta de cada equipo. Una vez determinada, se cambiará la vestimenta de cada uno de los equipos por la seleccionada (en la figura A.9, números tres y cuatro se puede ver un ejemplo de ello), pero, para que tenga efecto en la simulación se deberá pulsar el botón *Accept* (numerado con un cinco en la figura A.9).

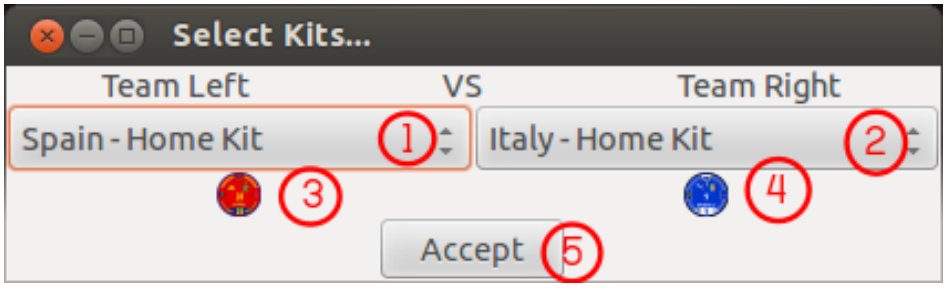


Figura A.9: Ventana *Change Kits...*

Cambiar el balón

Esta funcionalidad es semejante a la vista en A.3.2, con la excepción de, que en lugar de modificar la apariencia de los jugadores en el campo, se modifica la del balón. Los balones que se pueden seleccionar corresponden a los utilizados en algunas de las últimas competiciones como, la Eurocopa de 2008, el mundial de 2002 o la Liga de 2012, entre otros.

Al pulsar sobre esta opción, se abrirá una nueva ventana con la misma apariencia que la figura A.10. Dicha ventana se compone de una caja de combo (númerada en la figura A.10 con un uno) y un botón con la etiqueta *Accept* (numerado en la figura A.10 con un dos). De la misma manera que en la opción A.3.2, se puede seleccionar el balón en la caja de combo, pero, que se vea reflejado en la simulación, será necesario pulsar sobre el botón.

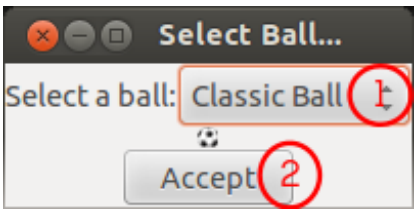


Figura A.10: Ventana *Change Ball...*

Posiciones de los jugadores

Esta funcionalidad permite al usuario observar el rol de los jugadores de cada uno de los dos equipos que componen una simulación, así como, la vestimenta y el nombre de cada equipo. Esta opción sólo estará disponible al cargar una simulación (seguir los pasos de A.3.1).

Al pinchar sobre esta opción se generará una ventana como la que se puede apreciar en la figura A.11. Esta ventana es meramente informativa.

Eventos de la simulación

Esta parte de la herramienta permite al usuario identificar todos los eventos que ocurren durante la simulación. Esta ventana incluye la descripción de los eventos, así como, el ciclo en que se producen, facilitando al usuario, la tarea de descubrirlos por el mismo. Al ejecutar esta opción se abrirá una ventana como la que se puede apreciar en la figura A.12.

Realimentación de los eventos

En esta ventana se puede determinar qué eventos se muestren al usuario en la simulación. Así pues, cuando el usuario seleccione un tipo de evento y éste ocurra, la simulación se ralentizará y se generará un círculo intermitente de diferentes colores bajo el jugador que realizará el evento. Los colores del círculo serán:

Figura A.11: Ventana *Positions of Players...*

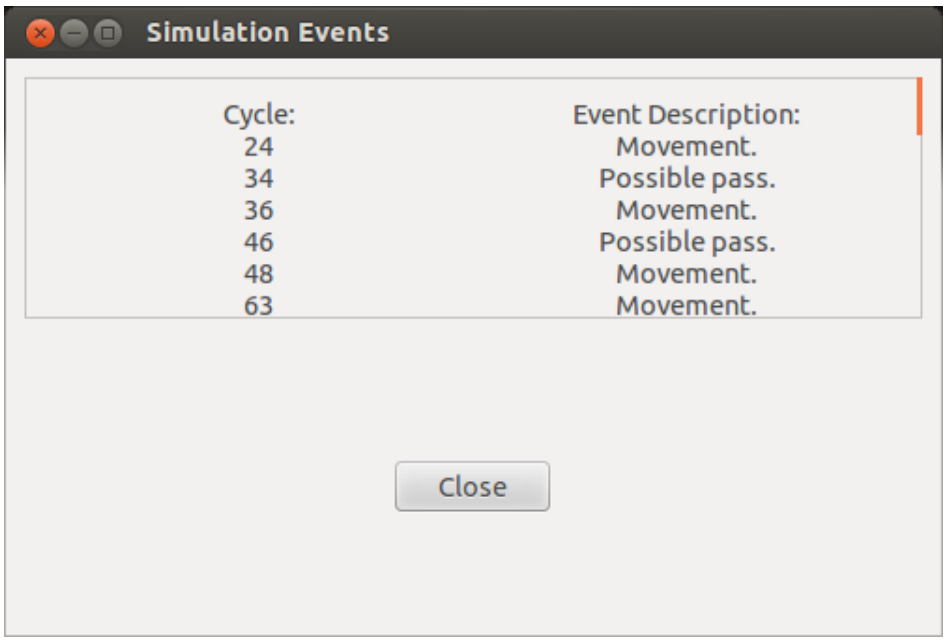


Figura A.12: Ventana *Show Events...*

- Azul. Indica que el evento va a ocurrir inminentemente. Este círculo permite al usuario concentrarse en el jugador que realizará un evento concreto.
- Verde. Si el círculo pasa de azul a verde, significa que el jugador sigue un comportamiento definido como correcto, es decir, el jugador ha realizado la acción correcta.
- Rojo. Si el círculo pasa de azul a rojo, significa que el jugador sigue un comportamiento definido como incorrecto, es decir, el jugador debería haber realizado otra acción distinta a la que ha realizado.

Para seleccionar la realimentación de los eventos, el usuario debe seleccionar un evento de entre los disponibles en una simulación y posteriormente pulsar sobre el botón *Ok*.

La ventana de retroalimentación de los eventos se puede apreciar en la figura A.13.

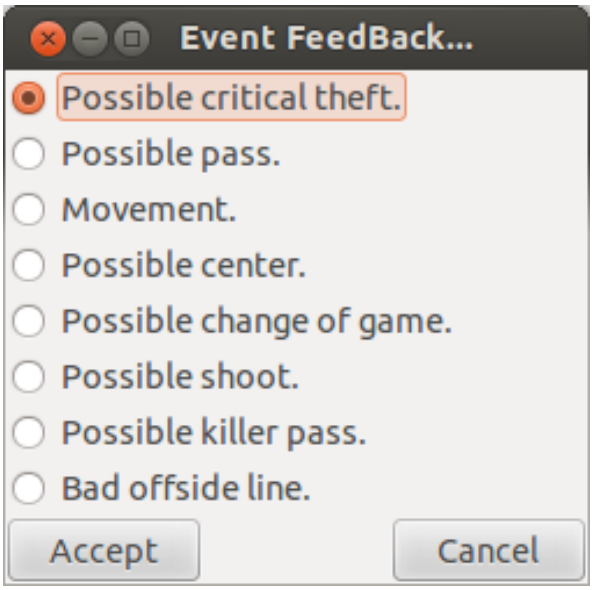


Figura A.13: Ventana *Select Feedback...*

Mostrar información avanzada

En esta ventana se permitirá al usuario generar gráficas de los datos de la simulación. En la ventana podremos seleccionar cualquier jugador de cada equipo, así como, cualquier variable disponible de la simulación. Las variables de la simulación son:

1. *Xpos*. Indica el recorrido horizontal que un jugador realiza en el campo.
2. *YPos*. Indica el recorrido vertical que un jugador realiza en el campo.
3. *Xvel*. Indica la velocidad aplicada por un jugador en el eje X.
4. *Yvel*. Indica la velocidad aplicada por un jugador en el eje Y.
5. *Possession*. Muestra cuando un jugador ha obtenido la posesión del balón.
6. *Nearlyplayers*. Indica los oponentes cercanos a un jugador.
7. *Midfieldopponents*. Indica los jugadores que se encuentran en la franja del campo opuesta a un jugador.
8. *Stamina*. Indica el cansancio de un jugador.
9. *Orientation*. Muestra la orientación del cuerpo de un jugador.
10. *Angle*. Determina el ángulo de visión de un jugador.

. El usuario podrá almacenar las gráficas en un dispositivo de almacenamiento, marcando la opción *Save Charts...*, o no almacenarlas, no marcando dicha opción, en cuyo caso, las gráficas se generarán en diferentes ventanas. Si habilitamos la opción de guardar las gráficas, el usuario podrá determinar el directorio (mediante la caja de combo, numerada con), en el cual se almacenarán. Una vez habilitadas las opciones, pulsaremos el botón *Ok* para llevar a cabo la generación de gráficas.

La ventana de mostrar información avanzada de la simulación se puede apreciar en la figura A.14.

A.3.3. Menú *Help*

Acerca de...

Interactuando con esta opción se genera una ventana que información para el usuario acerca del creador de esta herramienta. Dentro de esta ventana se especifica la licencia de la misma.

A.3.4. Interactuando en una simulación

Cuando iniciamos la aplicación, podemos ver una serie de botones que, si el usuario los pulsa, no notará ningún efecto en ellos. Este comportamiento se debe a que es necesario cargar una simulación (como se ha visto en refsubsubsec:cargar) antes. Cuando la simulación se encuentre cargada, el usuario podrá interactuar con los botones, cuyo comportamiento es el siguiente:

- Botón ir al anterior gol. Este botón tiene la propiedad de que si hacemos click sobre el, la simulación cambiará el ciclo actual al ciclo en el cual, se haya producido el último gol con respecto al ciclo actual. En caso, que no se haya producido un gol anterior al ciclo actual, la simulación empezará desde el inicio de la misma.

- Botón de reproducir. Este botón tiene dos usos, dependiendo si la simulación se encuentra reproduciéndose o no. Si la simulación se encuentra detenida, este botón tendrá el aspecto común de los reproductores para iniciar la simulación y, mediante su pulsación se consigue que la simulación avance. Por otro lado, si la simulación se encuentra en reproducción, el botón tendrá la apariencia común de los reproductores para parar la simulación e, interactuando con este botón, la simulación se detendrá.

- Botón ir al siguiente gol. Este botón tiene la propiedad de que si hacemos click sobre el, la simulación cambiará el ciclo actual al ciclo en el cual, se produzca el siguiente gol con respecto al ciclo actual.

- Botón *Go To:*. La funcionalidad de este botón es que la simulación cambie el ciclo actual, por el ciclo indicado en la caja de texto anterior a dicho botón.

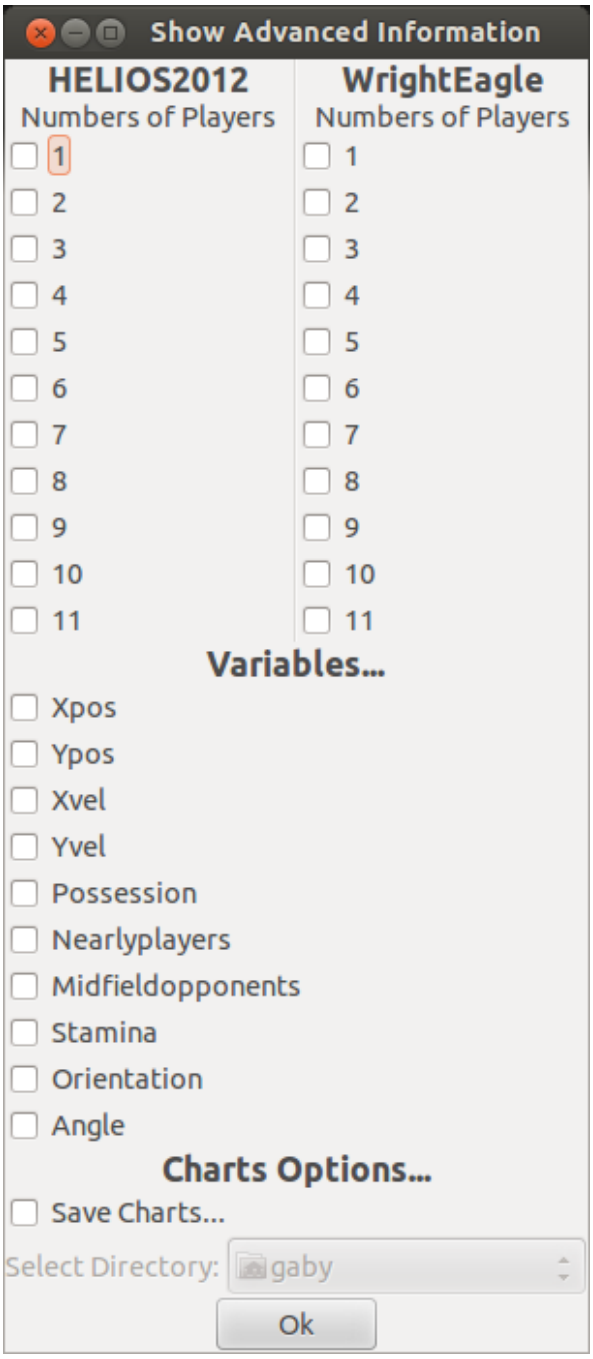


Figura A.14: Ventana *Show Advanced Information*

Además de estos botones, el usuario puede interactuar con los jugadores de la simulación cuando esta este detenida. Haciendo click, sobre un jugador el usuario puede seleccionar alguna de estas opciones:

- Mostrar información avanzada del jugador. Esta opción permite que se habilite una ventana, mediante la cual se pueda comprobar información de dicho jugador. Información del tipo, posición en el eje X, posición en el eje Y, velocidad en el eje X, velocidad en el eje Y, posesión, oponentes cercanos, oponentes en el centro del campo contrario,

resistencia, orientación del cuerpo y el ángulo de visión.

- Seleccionar el tipo de jugador. Esta opción permite al usuario determinar el tipo de jugador, con el objetivo de que el propio usuario tenga facilidad de identificar a los jugadores en el terreno de juego. Cuando el usuario pasa el puntero del ratón por encima de esta opción, el menú se expande, permitiendo al usuario identificar al jugador con tipos como portero, defensa, centrocampista o delantero.

  Cuando el usuario selecciona un tipo, hará que el jugador sobre el cual ha hecho click previamente posea una letra encima de él indicando su tipo. Esta información es persistente, de manera que, aunque el usuario cierre la simulación, el tipo de jugador se mantendrá.

- Ocultar información avanzada. Esta opción sólo estará disponible cuando la ventana de información avanzada de un jugador se encuentre visible. Cuando el usuario haga click sobre esta opción, dicha ventana se ocultará y esta opción desaparecerá el menú, hasta que el usuario vuelva a habilitar la ventana.

Con el objetivo de proporcionar una realimentación al usuario, la herramienta de una **barra de información** que permitirá al usuario comprobar el estado de la herramienta en todo momento. Esta barra se verá actualizada con cada acción que el usuario realice. Esta barra de información se encuentra bajo el área destinada al campo de juego.

Anexo B

Definición de Eventos Por Parte del Usuario

Los eventos puede ser definidos por parte del usuario. Para ello, se utilizan tres archivos que se pueden encontrar en la carpeta *kb/*. A continuación se detallan los archivos que se deben configurar para la definición de eventos.

B.1. Archivo PlayerKnowledge.fcl

Este archivo se correspone con el nivel bajo del sistema difuso. En este archivo se define la información de bajo nivel sobre la que se apoyará la capa de alto nivel del sistema difuso.

La configuración de este archivo se basa en la escritura de un sistema difuso utilizando el lenguaje de control difuso (ver anexo E. Los elementos de un sistema difuso se clasifican en tres grupos:

- Variables de entrada.
- Variables de salida.
- Reglas.

Para la configuración de un sistema difuso utilizando este archivo se dispone de las siguientes variables de entrada:

- **Posición en el eje *X***. Esta variable determina la posición de un jugador en el campo en el eje *X*. Los valores de esta variable se encuentran normalizados, por lo que se definen en el intervalo [0, 1], dónde el 0 representa la línea de meta del equipo que ataca de izquierda a derecha y 1 representa la línea de meta del equipo que ataca de derecha a izquierda. Para utilizar esta variable se utilizará el identificador `X_Pos`.
- **Posición en el eje *Y***. Esta variable determina la posición de un jugador en el campo en el eje *Y*. Los valores de esta variable se encuentran normalizados, por lo que se definen en el intervalo [0, 1], dónde el 0 representa la banda superior y el 1 representa la banda inferior. Para utilizar esta variable se utilizará el identificador `Y_Pos`.
- **Velocidad en el eje *X***. Esta variable determina la velocidad de un jugador en el eje *X*. Los valores de esta variable no se encuentran normalizados, porque dichos valores son dependientes del contexto de la simulación. Para utilizar esta variable se utilizará

el identificador `X_vel`.

- **Velocidad en el eje *Y***. Esta variable determina la velocidad de un jugador en el eje *Y*. Los valores de esta variable no se encuentran normalizados, porque dichos valores son dependientes del contexto de la simulación. Para utilizar esta variable se utilizará el identificador `Y_vel`.
- **Resistencia**. Esta variable determina la resistencia restante de un jugador en el campo. El valor máximo de esta variable es dependiente de los parámetros de configuración de RoboCup, sin embargo, suele ser 8000, siendo el valor mínimo 0. Para utilizar esta variable se utilizará el identificador `Stamina`.
- **Orientación**. Esta variable determina el ángulo de orientación del cuerpo con respecto a la cabeza de un jugador en el campo. Los valores de esta variable se definen en el intervalo [-90, 90]. Para utilizar esta variable se utilizará el identificador `Orientation`.
- **Ángulo de visión**. Esta variable determina el ángulo de visión de un jugador en el campo. Los valores de esta variable se definen en el intervalo [-180, 180]. Para utilizar esta variable se utilizará el identificador `Angle`.
- **Posesión del balón**. Esta variable determina si un jugador tiene o no la posesión del balón. Los valores de esta variable son; 1 si el jugador tiene la posesión y 0 en caso contrario. Para utilizar esta variable se utilizará el identificador `Possession`.
- **Oponentes cercanos**. Esta variable determina el número de oponentes que se encuentran cercanos a un jugador en el campo. Los valores de esta variable se definen en el intervalo [0, 11]. Para utilizar esta variable se utilizará el identificador `Nearly_players`.
- **Oponentes en la franja campo contraria**. Esta variable determina el número de oponentes en la mitad del campo superior si el jugador se encuentra en la mitad inferior, o el número de oponentes en la mitad inferior del campo, en caso que el jugador se encuentre en la mitad superior. Los valores de esta variable se definen en el intervalo [0, 11]. Para utilizar esta variable se utilizará el identificador `Midfield_opponents`.
- **Número del jugador**. Esta variable determina el dorsal de un jugador en el campo. Los valores de esta variable son dependientes del contexto de la simulación. Para utilizar esta variable se utilizará el identificador `Number`.

A continuación se propone un ejemplo de configuración de un sistema difuso utilizando algunas de las variables antes mencionadas.

```
2  FUNCTION_BLOCK PlayerKnowledge

4  VAR_INPUT
5          X_Pos: REAL;
6          Y_Pos: REAL;
```

```
7           Possession: REAL;
8           Nearly_players: REAL;
9   END_VAR

11  VAR_OUTPUT
12          Attitude: REAL;
13          Pressure: REAL;
14          MidField: REAL;
15          CenterZone: REAL;
16          KillerPassZone: REAL;
17  END_VAR

19  FUZZIFY X_Pos
20          TERM VeryLeft := (0.0, 0.0) (0.0, 1.0) (0.16667, 1) (0.3334,
                0);
21          TERM Left := (0.16667, 0) (0.3334, 1) (0.5, 0);
22          TERM Medium := (0.3334, 0) (0.50, 1) (0.6667, 0);
23          TERM Right := (0.50, 0) (0.6667, 1) (0.83334, 0);
24          TERM VeryRight := (0.6667, 0) (0.83334, 1) (1, 1) (1, 0);
25  END_FUZZIFY

27  FUZZIFY Y_Pos
28          TERM VeryUp := (0.0, 0.0) (0.0, 1.0) (0.16667, 1) (0.3334, 0);
29          TERM Up := (0.16667, 0) (0.3334, 1) (0.5, 0);
30          TERM Medium := (0.3334, 0) (0.50, 1) (0.6667, 0);
31          TERM Down := (0.50, 0) (0.6667, 1) (0.83334, 0);
32          TERM VeryDown := (0.6667, 0) (0.83334, 1) (1, 1) (1, 0);
33  END_FUZZIFY

35  FUZZIFY Possession
36          TERM Yes := 1;
37          TERM No := 0;
38  END_FUZZIFY

40  FUZZIFY Nearly_players
41            TERM Low := (0, 0) (0, 1) (1, 1) (2, 0);
42            TERM Medium := (1, 0) (2, 1) (3, 0);
43            TERM High := (2, 0) (3, 1) (4, 1) (4, 0);
44  END_FUZZIFY

46  DEFUZZIFY Attitude
47            TERM Defensive := (0, 0) (0, 1) (0.25, 1) (0.50, 0);
48            TERM Neutral := (0.25, 0) (0.50, 1) (0.75, 0);
49            TERM Offensive := (0.50, 0) (0.75, 1) (1, 1) (1, 0);
50            METHOD : COG;
51            DEFAULT := 0;
52  END_DEFUZZIFY

54  DEFUZZIFY Pressure
55            TERM Low := (0, 0) (0, 1) (0.25, 1) (0.50, 0);
56            TERM Medium := (0.25, 0) (0.50, 1) (0.75, 0);
57            TERM High := (0.50, 0) (0.75, 1) (1, 1) (1, 0);
58            METHOD : COG;
59            DEFAULT := 0;
60  END_DEFUZZIFY

62  DEFUZZIFY MidField
63            TERM Up := 1;
```

```
64            TERM Down := 0;
65            METHOD: COGS;
66            DEFAULT := 0;
67  END_DEFUZZIFY

69  DEFUZZIFY CenterZone
70            TERM Yes := 1;
71            TERM No := 0;
72            METHOD: COGS;
73            DEFAULT := 0;
74  END_DEFUZZIFY

76  DEFUZZIFY KillerPassZone
77            TERM Yes := 1;
78            TERM No := 0;
79            METHOD: COGS;
80            DEFAULT := 0;
81  END_DEFUZZIFY

83  RULEBLOCK Knowledge
84            AND: MIN;
85            OR: MAX;
86            ACCU: MAX;
87            (* ----------------Position Rules-------------*)
88            RULE 1: IF (X_Pos IS VeryLeft) OR (X_Pos IS Left) THEN
                 Attitude IS Defensive;
89            RULE 2: IF (X_Pos IS Medium) THEN Attitude IS Neutral;
90            RULE 3: IF (X_Pos IS Right) OR (X_Pos IS VeryRight) THEN
                 Attitude IS Offensive;

92            (* ----------------Pressure Rules-------------*)
93            RULE 4: IF (Possession IS Yes) AND (Nearly_players IS High)
                 THEN Pressure IS High;
94            RULE 5: IF (Possession IS Yes) AND (Nearly_players IS Medium
                 ) THEN Pressure IS Medium;
95            RULE 6: IF (Possession IS Yes) AND (Nearly_players IS Low)
                 THEN Pressure IS Low;

97            (*----------------Midfield Rules--------------*)
98            RULE 7: IF (Y_Pos IS VeryUp) OR (Y_Pos IS Up) THEN MidField
                 IS Up;
99            RULE 8: IF (Y_Pos IS VeryDown) OR (Y_Pos IS Down) THEN
                 MidField IS Down;

101           (*----------------Center Zone-----------------*)
102           RULE 9: IF ((X_Pos IS VeryLeft OR X_Pos IS Left) OR (X_Pos
                 IS VeryRight OR X_Pos IS Right)) AND (Y_Pos IS VeryDown
                 OR Y_Pos IS VeryUp) THEN CenterZone IS Yes;

104           (*---------------Killer Pass----------------*)
105           RULE 9: IF ((X_Pos IS VeryLeft OR X_Pos IS Left) OR (X_Pos
                 IS VeryRight OR X_Pos IS Right)) AND (Y_Pos IS Down OR
                 Y_Pos IS Up) THEN KillerPassZone IS Yes;

107 END_RULEBLOCK

109 END_FUNCTION_BLOCK
```

Listado B.1: Ejemplo de archivo *PlayerKnowledge.fcl*

Con dicha configuración se puede apreciar que las **variables de entrada** son las siguientes:

- Posición en el eje *X*.
- Posición en el eje *Y*.
- Posesión.
- Oponentes cercanos.

. Las variables de salida son las siguientes:

- Actitud de un jugador en el campo.
- Presión a la que está sometido un jugador.
- Ocupación de la franja superior o inferior del mediocampo.
- Ocupación de la zona de centro.
- Oucpación de la zona de pase de la muerte.

B.2. Archivo PlayerActions.fcl

El archivo *PlayerActions.fcl* se corresponde con la capa de nivel alto del sistema difuso. En este archivo se determinan los eventos de una simualación. La forma de definirlo es idéntica a la de el archivo *PlayerKnowledge.fcl*, puesto que, en este archivo se describe otro sistema difuso.

Sin embargo, en este archivo, se pueden utilizar como variales de entrada, tanto las variables explicadas en la sección anterior, como las variables de salida del anterior sistema. A continuación se propone un ejemplo de configuración de un archivo *PlayerActions.fcl*.

```
1  FUNCTION_BLOCK PlayerActions
2
3  VAR_INPUT
4          Pressure: REAL;
5          Attitude: REAL;
6          Possession: REAL;
7          MidField: REAL;
8          CenterZone: REAL;
9          KillerPassZone: REAL;
10         MidfieldOpponents: REAL;
11 END_VAR
12
13 VAR_OUTPUT
14     CriticalTheft: REAL;
15     Shoot: REAL;
16     ChangeGame: REAL;
```

```
    Center: REAL;
    KillerPass: REAL;
END_VAR

FUZZIFY Pressure
        TERM Low := (0, 0) (0, 1) (0.25, 1) (0.50, 0);
        TERM Medium := (0.25, 0) (0.50, 1) (0.75, 0);
        TERM High := (0.50, 0) (0.75, 1) (1, 1) (1, 0);
END_FUZZIFY

FUZZIFY Attitude
        TERM Defensive := (0, 0) (0, 1) (0.25, 1) (0.50, 0);
        TERM Neutral := (0.25, 0) (0.50, 1) (0.75, 0);
        TERM Offensive := (0.50, 0) (0.75, 1) (1, 1) (1, 0);
END_FUZZIFY

FUZZIFY Possession
        TERM Yes := 1.0;
        TERM No := 0.0;
END_FUZZIFY

FUZZIFY MidField
        TERM Up := 1;
        TERM Down := 0;
END_FUZZIFY

FUZZIFY CenterZone
        TERM Yes := 1;
        TERM No := 0;
END_FUZZIFY

FUZZIFY KillerPassZone
        TERM Yes := 1;
        TERM No := 0;
END_FUZZIFY

FUZZIFY MidfieldOpponents
        TERM Low := (0, 0) (0, 1) (2, 1) (5, 0);
        TERM Medium := (3, 0) (5, 1) (7, 0);
        TERM High := (5, 0) (7, 1) (11, 1) (11, 0);
END_FUZZIFY

DEFUZZIFY CriticalTheft
        TERM Yes := 1;
        TERM No := 0;
        METHOD: COGS;
        DEFAULT := 0;
END_DEFUZZIFY

DEFUZZIFY Shoot
        TERM Yes := 1;
        TERM No := 0;
        METHOD: COGS;
        DEFAULT := 0;
END_DEFUZZIFY

DEFUZZIFY ChangeGame
          TERM Yes := 1;
```

```
75            TERM No := 0;
76            METHOD: COGS;
77            DEFAULT := 0;
78  END_DEFUZZIFY

80  DEFUZZIFY Center
81            TERM Yes := 1;
82            TERM No := 0;
83            METHOD: COGS;
84            DEFAULT := 0;
85  END_DEFUZZIFY

87  DEFUZZIFY KillerPass
88            TERM Yes := 1;
89            TERM No := 0;
90            METHOD: COGS;
91            DEFAULT := 0;
92  END_DEFUZZIFY


95  RULEBLOCK Possibilities
96            AND: MIN;
97            OR: MAX;
98            ACCU: MAX;

100           (*------------------Critic Theft---------------*)
101           RULE 1: IF (Possession IS Yes) AND (Pressure IS High) AND (
                 Attitude IS Defensive) THEN CriticalTheft IS Yes;

103           (*------------------Shoot Success---------*)
104           RULE 2: IF (Possession IS Yes) AND (Pressure IS Low) AND (
                 Attitude IS Offensive) THEN Shoot IS Yes;

106           (*------------------Change Game----------------*)
107           RULE 3: IF (Possession IS Yes) AND (Attitude IS Neutral) AND
                  (MidfieldOpponents IS Low) THEN ChangeGame IS Yes;

109           (*-----------------Center Action--------------*)
110           RULE 4: IF (Possession IS Yes) AND (CenterZone IS Yes) AND (
                 Attitude IS Offensive) THEN Center IS Yes;

112           (*-----------------Killer Pass Action---------*)
113           RULE 5: IF (Possession IS Yes) AND (KillerPassZone IS Yes)
                 AND (Attitude IS Offensive) THEN KillerPass IS Yes;

115 END_RULEBLOCK

117 END_FUNCTION_BLOCK
```

Listado B.2: Ejemplo de archivo *PlayerKnowledge.fcl*

En este ejemplo se aprecia cómo se utilizan las variables de entrada explicadas en la anterior sección y las variables de salida del anterior sistema difuso. Se utilizan las variables «posesión» y «oponentes en la franja del campo contraria» y también, se utilizan las variables «presión», «actitud», «mediocampo», «zona de centro» y «zona de pase de la muerte», definidas como variables de salida del anterior sistema.

Las variables de salida de este sistema se corresponden con la definición de eventos que detectan el robo crítico del balón, el disparo a portería, el cambio de juego, el centro y el pase de la muerte.

B.3. Archivo Events.xml

La finalidad de este archivo es la de indicar a la herramienta cuáles son los eventos que se han definido en las dos capas del sistema difuso, para ello, se emplea el lenguaje estándar *XML*.

Los eventos se construyen con las etiqutetas <event>y </event>y cada evento debe tener asociadas las siguientes características:

- **Nombre**. Define el nombre del evento. Este nombre deberá ser igual que el nombre de la variable de salida del archivo *PlayerActions.fcl* que determine este evento. El nombre se especifica dentro de las etiquetas <name>y </name>.
- **Tipo**. Define el tipo del evento. El tipo será necesario para analizar el comportamiento del jugador en la simulación. Existen diferentes tipos, los cuales se explican a continuación:
  - Tipo **1**: No mantener la posesión del balón. Cuando se define un evento de este tipo, el jugador con la posesión del balón debe perder la posesión del mismo, en favor de los intereses del equipo.
  - Tipo **2**: Cambio de juego. Cuando se produce un evento de este tipo, trás unos cuantos ciclos, el balón deberá haber atravesado la divisoria de los campos inferior y superior.
  - Tipo **3**: Siempre incorrecto. Idealmente, un evento de este tipo no se debería producir nunca.
  - Tipo **4**: Siempre correcto. Cuando ocurre un evento de este tipo, significa que alguna acción está bien hecha.
  - Tipo **5**: Mantener la posesión del balón. Cuando se define un evento de este tipo, el jugador con la posesión del balón debe mantener la posesión del mismo, en favor de los intereses del equipo.

  El tipo de los eventos se especifica entre las etiquetas <type>y </type>.
- Descripción. Determina la descripción del evento. La descripción se determina entre las etiquetas <descr>y >/descr>.

Cada uno de los eventos se encuentran definidos entre las etiquetas <system>y </system>.

Un ejemplo de archivo *events.xml* puede observarse a continuación:

```
<?xml version="1.0"?>
<system>
        <event>
                <name>ChangeGame</name>
                <type>2</type>
                <descr>Possible change of game.</descr>
        </event>
        <event>
                <name>Center</name>
                <type>1</type>
                <descr>Possible center.</descr>
        </event>
        <event>
                <name>KillerPass</name>
                <type>1</type>
                <descr>Possible killer pass.</descr>
        </event>
        <event>
                <name>CriticalTheft</name>
                <type>1</type>
                <descr>Possible critical theft.</descr>
        </event>
        <event>
                <name>Shoot</name>
                <type>1</type>
                <descr>Possible shoot.</descr>
        </event>
</system>
```

Listado B.3: Ejemplo de archivo *events.xml*

Anexo C

Introducción de situaciones en la base de datos

Este anexo detalla los pasos que se sigen para la introducción de las situaciones detectadas durante un partido de fútbol en la base de datos.

Desde el punto de vista del código fuente, la función más importante es la definida en la clase fachada del dominio para actualizar las tablas de la base de datos e insertar los eventos y cuyo código es el que se puede apreciar en el listado de código C.1. Dicha función, obtiene cada uno de los jugadores de un equipo en una simulación y, cada cierto umbral de tiempo, recoge sus características con el objetivo de detectar o no si se producen eventos para un ciclo determinado.

```
1  private void updatePlayersFeaturesOfTeam(int lastSimulationId, int
       teamId, int threshold)
2  {
3    agent.connect();
4    List<Integer> numbersOfPlayers = agent.getNumbersOfPlayers(
       lastSimulationId, teamId);
5    int firstCycle = agent.getFirstCycle(lastSimulationId);
6    int lastCycle = agent.getLastCycle(lastSimulationId);
7    int i = firstCycle;
8    while (i <= lastCycle)
9    {
10     for (Integer number : numbersOfPlayers)
11     {
12       List<Feature> featuresOfPlayerInCycle = agent.getPlayer(
           lastSimulationId, teamId, number, i);
13       for (Feature feature : featuresOfPlayerInCycle)
14       {
15         lowFuzzySystem.setVariableInput(feature.getDescr(), feature.
             getValue());
16         mediumFuzzySystem.setVariableInput(feature.getDescr(), feature
             .getValue());
17       }
18       lowFuzzySystem.evaluate();
19       List<Variable> outputVariables = lowFuzzySystem.
           getOutputVariables();
20       for (Variable var : outputVariables)
21       {
22         double defuzzyValue = var.defuzzify();
23         if (var.getName().equals("Attitude"))
24         {
25           if ((teamId % 2) == 0)
```

```
26                defuzzyValue = 1 - defuzzyValue;
27            }
28            mediumFuzzySystem.setVariableInput(var.getName(), defuzzyValue
                  );
29          }
30          mediumFuzzySystem.evaluate();
31          this.insertEvents(lastSimulationId, i, number, teamId);
32          mediumFuzzySystem = new FuzzySystem(fileMediumFuzzySystem);
33          lowFuzzySystem = new FuzzySystem(fileLowFuzzySystem);
34        }
35        i += threshold;
36      }
37      agent.disconnect();
38    }
```

Listado C.1: Cabecera de la función que actualiza la base de datos con la información de la lógica difusa

La función vista en C.1 se apoya en otra que determina los eventos a insertar en la base de datos, como los eventos son definidos por el usuario, es necesario que se determinen los eventos de alguna manera, para ello se utiliza un fichero *XML* (los pasos necesarios para definir eventos se pueden encontrar en el anexo B). La función que detecta si hay que almacenar nuevos eventos, obtiene todos los eventos definidos por el usuario e identifica si se ha producido alguno, en caso afirmativo, almacena el evento en la base de datos. La función en cuestión puede obervarse en el listado de código C.2.

```
1     private void insertEvents(int simulationId, int cycle, int
          numberOfPlayer, int teamId)
2     {
3       List<Variable> outputVariables = highFuzzySystem.
            getOutputVariables();
4       for (Variable var : outputVariables)
5         for (Event e: this.__events)
6           if (var.getName().equals(e.getName()) && Math.rint(var.
                getValue()) == 1)
7             agent.insertEvent(simulationId, teamId, numberOfPlayer,
8             cycle, e.getDescr(), e.getConclusion(), e.getType());
9     }
```

Listado C.2: Función que determina los eventos a introducir en la base de datos

De esta manera, se introducen los eventos en la base de datos. Cabe destacar, la forma en que se obtienen los eventos definidos por el usuario. Estos eventos se almacenan en la variable `__events` y se obtienen *parseando* un fichero XML que los almacena. La función que proporciona los eventos se encuentra en la clase `ParserEvents.java` y cuyo código puede encontrarse en el listado C.3.

```

2   public List<Event> parseEvents()
3   {
4     List<Event> events = new ArrayList<Event>();
5     try {
```

```
6       File file = new File(this.file);
7       DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
8       DocumentBuilder db = dbf.newDocumentBuilder();
9       Document doc = db.parse(file);
10      doc.getDocumentElement().normalize();
11      NodeList nodeLst = doc.getElementsByTagName("event");
12      for (int s = 0; s < nodeLst.getLength(); s++)
13      {
14        Node fstNode = nodeLst.item(s);
15        if (fstNode.getNodeType() == Node.ELEMENT_NODE)
16        {
17          Element fstElmnt = (Element) fstNode;
18          String eventName = getAttribute(fstElmnt, "name");
19          String eventType = getAttribute(fstElmnt, "type");
20          String eventDescr = getAttribute(fstElmnt, "descr");
21          String eventConclusion = getAttribute(fstElmnt, "conclusion");
22          events.add(new Event(eventName, eventType, eventDescr,
                eventConclusion));
23        }

25      }
26    }
27    catch (Exception e)
28    {

30    }
31    return events;
32  }
```

Listado C.3: Función que parsea los eventos del usuario

Esta función se apoya en otra función que se encuentra en la misma clase llamada `getAttribute` y cuyo código se puede observar en el listado C.4.

```
1   private String getAttribute(Node node, String nameOfAttribute)
2   {
3     Element fstElmnt = (Element) node;
4     NodeList fstElmntLst = fstElmnt.getElementsByTagName(nameOfAttribute
          );
5     Element elmnt = (Element) fstElmntLst.item(0);
6     NodeList attr = elmnt.getChildNodes();
7     return ((Node) attr.item(0)).getNodeValue();
8   }
```

Listado C.4: Función que obtiene los elementos de un fichero XML

De esta manera los eventos definidos por los usuarios son analizados e introducidos en la base de datos.

Anexo D

Archivos de Simulación XML

En este anexo se detalla la información que contienen los ficheros XML correspondientes a la simulaciones de *RoboCup Soccer 2D*. Estos archivos contendrán la información necesaria para que la herramienta pueda inferir nuevo conocimiento o recrear la simulacion, entre otras funcionalidades. Por tanto, los ficheros XML de simulación resultan ser un componente importante en el proyecto, que han de ser estudiados a fondo para no obviar ningún dato que nos puedan proporcionar.

Podemos establecer una partición en el contenido del documento, clasificándolo en tres partes bien diferenciadas:

D.1. Configuración de parámetros

En esta parte del documento, se detallan los parámetros de la simulación, es decir, se simulan diferentes impedimentos físicos reales mediante valores de variables que supondrán límites para los jugadores virtuales. Con el objetivo de facilitar la estructura del documento, este parte del documento, se puede establecer una división a la hora de configurar los parámetros.

Parámetros definidos por el servidor

En esta parte del archivo, se pueden observar reglas generales que afectarán a todos los jugadores por igual. Por ejemplo, en estos parámetros se puede establecer el máximo de resistencia de un jugador, la probabilidad de desvío de un disparo, o la probabilidad de que un portero haga una parada. Esta parte del documento se encuentra acotada por las etiqueta *<ServerParam>* y *</ServerParam>*. Cada uno de los parámetros estarán definidos por las etiquetas *<Param>* y *</Param>*. Algunas de los parámetros que se pueden definir en esta parte del documento se encuentran en el listado de código D.1

```
1 <ServerParam>
2 <Param name="gwidth">14.02</Param>
3 <Param name="inertia_moment">5</Param>
4 <Param name="psize">0.299988</Param>
5 <Param name="pdecay">0.399994</Param>
6 <Param name="prand">0.0999908</Param>
7 <Param name="pweight">60</Param>
8 <Param name="pspeed_max">1.2</Param>
```

```
<Param name="paccel_max">1</Param>
<Param name="stamina_max">32000</Param>
<Param name="stamina_inc">45</Param>
<Param name="recover_init">1</Param>
<Param name="recover_dthr">0.299988</Param>
<Param name="recover_min">0.5</Param>
<Param name="recover_dec">0.0019989</Param>
<Param name="effort_init">1</Param>
<Param name="effort_dthr">0.299988</Param>
<Param name="effort_min">0.599991</Param>
<Param name="effort_dec">0.00498962</Param>
<Param name="effort_ithr">0.599991</Param>
<Param name="effort_inc">0.00999451</Param>
<Param name="kick_rand">0</Param>
</ServerParam>
```

Listado D.1: Parámetros del servidor

Parámetros específicos de los jugadores

En esta parte del documento, se definen características específicas de los jugadores. Algunas de las características que podemos encontrarnos son: El incremento de velocidad máxima de un jugador, el incremento máximo de resistencia de un jugador o el margen de error al patear el balón, entre otras. En el listado D.2 se pueden observar las características específicas de los jugadores.

Esta parte del documento está comprendida por las etiquetas *<PlayerParam>* y *</PlayerParam>*. Al igual que los parámetros del servidor, cada uno de los parámetros estarán definidos por las etiquetas *<Param>* y *</Param>*.

```

<PlayerParam>
<Param name="player_types">18</Param>
<Param name="subs_max">3</Param>
<Param name="pt_max">1</Param>
<Param name="allow_mult_default_type">0</Param>
<Param name="player_speed_max_delta_min">0</Param>
<Param name="player_speed_max_delta_max">0</Param>
<Param name="stamina_inc_max_delta_factor">0</Param>
<Param name="player_decay_delta_min">-0.1</Param>
<Param name="player_decay_delta_max">0.1</Param>
<Param name="inertia_moment_delta_factor">25</Param>
<Param name="dash_power_rate_delta_min">0</Param>
<Param name="dash_power_rate_delta_max">0</Param>
<Param name="player_size_delta_factor">-100</Param>
<Param name="kickable_margin_delta_min">-0.1</Param>
<Param name="kickable_margin_delta_max">0.1</Param>
<Param name="kick_rand_delta_factor">1</Param>
<Param name="extra_stamina_delta_min">0</Param>
<Param name="extra_stamina_delta_max">50</Param>
<Param name="effort_max_delta_factor">-0.004</Param>
<Param name="effort_min_delta_factor">-0.004</Param>
<Param name="random_seed">1316714976</Param>
<Param name="new_dash_power_rate_delta_min">-0.0012</Param>
```

```
25 <Param name="new_dash_power_rate_delta_max">0.0008</Param>
26 <Param name="new_stamina_inc_max_delta_factor">-6000</Param>
27 </PlayerParam>
```

Listado D.2: Parámetros de los jugadores

La primera característica definida es el número de tipos posibles para los jugadores, y es que en una simulación los jugadores pueden tener diferentes tipos, los cuales permiten a los jugadores unas características u otras. En el listado D.3 se pueden observar las características correspondientes para el tipo 0. Un jugador que esté asignado a un tipo, tiene definidos diferentes valores de variables que no podrá sobrepasar a lo largo de la simulación. Por ejemplo, en D.3 se puede observar que la velocidad máxima de un jugador es de 1.05 o que el incremento de resitstencia es de 45, entre otros valores.

```

2 <PlayerType id="0">
3 <Param name="player_speed_max">1.05</Param>
4 <Param name="stamina_inc_max">45</Param>
5 <Param name="player_decay">0.4</Param>
6 <Param name="inertia_moment">5</Param>
7 <Param name="dash_power_rate">0.006</Param>
8 <Param name="player_size">0.3</Param>
9 <Param name="kickable_margin">0.7</Param>
10 <Param name="kick_rand">0.1</Param>
11 <Param name="extra_stamina">50</Param>
12 <Param name="effort_max">1</Param>
13 <Param name="effort_min">0.6</Param>
14 </PlayerType>
```

Listado D.3: Parámetros del servidor

D.2. Información sobre los gráficos

En este fragmento del fichero se definen las partes gráficas que serán utilizadas para dibujar la interfaz correspondiente a la simulación RoboCup 2D. En el listado D.4 se puede observar un ejemplo, de la estructura de esta parte del documento.

```

2 <MsgInfo>
3 (team_graphic_l (8 3 &quot;8 8 2 1&quot; &quot;$ c red&quot; &quot;*
4 c #FF3333&quot; &quot;$$$$$$$$&quot; &quot;$$$$$$$$&quot; &quot
    ;$$$$$$$$&quot;
5  &quot;$$$$$$$$&quot; &quot;$$$$$$$$&quot; &quot;$$$$$$$$&quot; &quot
    ;$$$$$$$$
6 &quot; &quot;********&quot;)</MsgInfo>
7 <MsgInfo>(team_graphic_l (8 2 &quot;8 8 10 1&quot; &quot;$ c red&quot;
    &quot;
8 &amp; c #FF2222&quot; &quot;* c #FF3333&quot; &quot;= c #FF4444&quot;
   &quot;2
9  c #FF8888&quot; &quot;5 c #FFBBBB&quot; &quot;8 c #FFCCCC&quot; &quot
```

```
     ;9 c
#FFDDDD&quot; &quot;q c #FFEEEE&quot; &quot;w c gray100&quot; &quot;
    wwwwwwww&quot
; &quot;wwwwwwww&quot; &quot;wwwwwwqw&quot; &quot;wwwwwqw9&quot; &quot
    ;wwwww5=$&quot
; &quot;wwq8*$$$&quot; &quot;w2&amp;$$$$$&quot; &quot;$$$$$$$$&quot;)<
    /MsgInfo>
<MsgInfo>(team_graphic_l (8 1 &quot;8 8 9 1&quot; &quot;$ c red&quot;
    &quot;*
 c #FF3333&quot; &quot;= c #FF4444&quot; &quot;- c #FF5555&quot; &quot
    ;; c
#FF6666&quot; &quot;: c #FF7777&quot; &quot;2 c #FF8888&quot; &quot;q
    c
#FFEEEE&quot; &quot;w c gray100&quot; &quot;$$$$$$$$&quot; &quot
    ;$$$$$$$$&quot
; &quot;$$$$$$$$&quot; &quot;*=-;:222&quot; &quot;wwwwqwww&quot; &quot
    ;
wwwwwwww&quot; &quot;wwwwwwww&quot; &quot;wwwwwwww&quot;)</MsgInfo>
<MsgInfo>(team_graphic_l (8 0 &quot;8 8 5 1&quot; &quot;$ c red&quot;
    &quot
;% c #FF1111&quot; &quot;= c #FF4444&quot; &quot;2 c #FF8888&quot; &
    quot;8
c #FFCCCC&quot; &quot;$$$$%=28&quot; &quot;$$$$$$$$&quot; &quot
    ;$$$$$$$$&quot;
 &quot;$$$$$$$$&quot; &quot;$$$$$$$$&quot; &quot;$$$$$$$$&quot; &quot
    ;$$$$$$$$&quot
; &quot;$$$$$$$$&quot;)</MsgInfo>
<MsgInfo>(team_graphic_l (7 3 &quot;8 8 8 1&quot; &quot;$ c red&quot;
    &quot;%
 c #FF1111&quot; &quot;&amp; c #FF2222&quot; &quot;* c #FF3333&quot; &
    quot;; c
#FF6666&quot; &quot;: c #FF7777&quot; &quot;3 c #FF9999&quot; &quot;9
    c
#FFDDDD&quot; &quot;$$$:;%$$&quot; &quot;$&amp;$$$$$&quot; &quot
    ;93$$$$$$&quot;
 &quot;9$$$$$$$&quot; &quot;&amp;$$$$$$$&quot; &quot;$$$$$$$$&quot; &
    quot;
$$$$$$$$&quot; &quot;********&quot;)
</MsgInfo>
```

Listado D.4: Parámetros del servidor

D.3. Simulación

En esta parte del fichero XML se almacena la información correspondiente a la simulación del encuentro. La simulación comienza con un evento que indica el comienzo del partido, dicho evento puede apreciarse en el listado de código D.5.

```
<PlayMode>kick_off_l</PlayMode>
```

Listado D.5: Evento de inicio de una simulación

Aparte del evento que se puede observar en el listado D.5, existen diferentes valores que pueden aparecer entre las etiquetas <PlayMode> y </PlayMode>. Algunos de estos valores

pueden ser indicar el saque de un penalty, saque de falta, saque de córner, fuera de juego, etcétera. La lista de posibles valores, que puede tomar el modo de juego se define en el listado D.6.

```
"before_kick_off"          "time_over"
"play_on"                  "kick_off_l"
"kick_off_r"               "kick_in_l"
"kick_in_r"                "free_kick_l"
"free_kick_r"              "corner_kick_l"
"corner_kick_r"            "goal_kick_l"
"goal_kick_r"              "goal_l"
"goal_r"                   "drop_ball"
"offside_l"                "offside_r"
"penalty_kick_l"           "penalty_kick_r"
"first_half_over"          "pause"
"human_judge"              "foul_charge_l"
"foul_charge_r"            "foul_push_l"
"foul_push_r"              "foul_multiple_attack_l"
"foul_multiple_attack_r"   "foul_ballout_l"
"foul_ballout_r"           "back_pass_l"
"back_pass_r"              "free_kick_fault_l"
"free_kick_fault_r"        "catch_fault_l"
"catch_fault_r"            "indirect_free_kick_l"
"indirect_free_kick_r"     "penalty_setup_l"
"penalty_setup_r"          "penalty_ready_l"
"penalty_ready_r"          "penalty_taken_l"
"penalty_taken_r"          "penalty_miss_l"
"penalty_miss_r"           "penalty_score_l"
"penalty_score_r"
```

Listado D.6: Eventos posibles en una simulación

Este evento que indica el modo de juego, va seguido de la definición de los dos equipos, tal y como puede observarse en el listado de código D.7.

```
<Team side="l"><Name>TeamName1</Name></Team>
<Team side="r"><Name>TeamName2</Name></Team>
```

Listado D.7: Definición de los dos equipos

A continuación y desde la definición de los equipos al final del fichero se encuentran los datos de los jugadores y del balón, ciclo por ciclo. Podemos encontrar desde la posición en coordenadas cartesianas hasta el número de veces que un jugador ha efectuado un disparo. La información perteneciente a un ciclo de simulación puede encontarse en el listado D.8

```
<ShowInfo time="1"> // Numero de ciclo
<Ball> // Informacion del balon
<X>0</X><Y>0</Y> // Coordenadas X, Y del balon en el ciclo n
<VelX>0</VelX><VelY>0</VelY> // Velocidades en X y en Y en el ciclo n
</Ball>
<Player side="l" unum="1" type="0"> // Informacion del jugador
                                    // con dorsal 1 y tipo 0
<X>-49</X><Y>0</Y> // Coordenadas X, Y del jugador en el ciclo n
```

```
<VelX>0</VelX><VelY>0</VelY> // Velocidades en X y en Y en el ciclo n
<BodyAng>-67.909</BodyAng> // Angulo del jugador en el ciclo n
<HeadAng>0</HeadAng> // Angulo de la cabeza del jugador en el ciclo n
<ViewWidth>180</ViewWidth> // Ancho (angulo) de vision del jugador
<ViewQual>high</ViewQual> // Calidad de la vision del jugador
<Stamina>8000</Stamina> // Resistencia del jugador en el ciclo n
<Effort>1</Effort>
<Recovery>1</Recovery>
<Count>
<Kick>0</Kick> // Cuenta de las veces que ha disparado
               // el jugador en el ciclo n
<Dash>0</Dash> // Cuenta de las veces que ha acelerado
               // el jugador en el ciclo n
<Turn>221</Turn> // Cuenta de las veces que ha girado
                 // el jugador en el ciclo n

<Say>0</Say> // Cuenta de las veces que ha hablado
               // el jugador en el ciclo n
<TurnNeck>222</TurnNeck> // Cuenta de las veces que ha girado el
   cuello
                         // el jugador en el ciclo n
<Catch>0</Catch> // Cuenta de las veces que ha
                 // detenido el balon el jugador en el ciclo n
<Move>1</Move>   // Cuenta las veces que se ha cambiado
                 // de posicion el jugador.
<ChgView>1</ChgView> // Cuenta el numero de veces que
                     // el jugador ha cambiado la vision
</Count>
</Player>
...
<Player>
...
</Player>
</ShowInfo>
...
<ShowInfo time="n">
...
</ShowInfo>
```

Listado D.8: Información de un ciclo de simulación

Anexo E

Fuzzy Control Language

Introduction

The theory of Fuzzy Logic in the application of control is named Fuzzy Control. The Fuzzy Control is emerging as a technology that can enhance the capabilities of industrial automation, and is suitable for control level tasks generally performed in Programmable Controllers (PC).

Fuzzy Control is based upon practical application knowledge represented by so called linguistic rule bases, rather than by analytical (either empirical or theoretical) models. Fuzzy Control can be used when there is an expertise that can be expressed in its formalism. That allows to take opportunity of available knowledge to improve processes and perform a variety of tasks, for instance

- control (closed or open loop, single or multi-variable, for linear or non linear systems)
- on-line or off-line setting of control systems' parameters
- classification and pattern recognition
- real-time decision making (send this product to machine A or B ?)
- helping operators to make decisions or tune parameters
- detection and diagnosis of faults in systems

Its wide range of applications and natural approach based on human experience makes Fuzzy Control a basic tool that should be made available to Programmable Controller users as a standard.

Fuzzy Control can also in a straightforward way be combined with classical control methods.

The application of Fuzzy Control can be of advantage in such cases where there is no explicit process model available, or in which the analytical model is too difficult to evaluate (e.g. multiple input multiple output systems) or when the model is too complicated to evaluate it in real time.

Another advantageous feature of Fuzzy Control is, that human experience can be incorporated in a straightforward way. Also it is not necessary to model the whole controller with Fuzzy Control: sometimes Fuzzy Control just interpolates between a series of locally linear models, or dynamically adapts the parameters of a "linear controller", thereby rendering it non linear, or alternatively just "zoom in" onto a certain feature of an existing controller that needs to be improved.

Fuzzy Control is a multi-valued Control, no longer restricting the values of a Control proposition to "true" or "false". This makes Fuzzy Control particularly useful to model empirical expertise, stating, which control actions have to be taken under a given set of inputs.

The existing theory and systems already realised in the area of Fuzzy Control differ widely in terms of terminology (definitions), features (functionalities) and implementation (tools).

The goal of this Standard is to offer the manufactures and the users a well defined common understanding of the basic means to integrate Fuzzy Control applications in the Programmable Controller languages according to Part 3, as well as the possibility to exchange portable Fuzzy Control programs among different programming systems.

To achieve this, Annex A of the Standard gives a short introduction to the theory of Fuzzy Control and Fuzzy Control as far as it is necessary for the understanding of the Standard. It may be helpful for readers of this Standard which are not familiar with Fuzzy Control theory to read the Annex A first.

The standard terminology is defined in clause 3, the integration of Fuzzy Control application into the Standard Languages of Programmable Controllers in clause 4, and a three level set of features with their functionality and textual representation in clause 5. This clause 5 defines the syntax and the semantics of the standardised features in form of the textual Fuzzy Control Language (FCL) using the definitions from Part 3 (Programmable Controller Languages) like Data Types and Function Blocks.

Fuzzy Control is used from small and simple applications up to highly sophisticated and complex projects. To cover all kinds of usage by this Part the features of a compliant Fuzzy Control system are mapped into defined conformance classes described in clause 6.

The Basic Class defines a minimum set of features which has to be achieved by all compliant systems. This facilitates the exchange of Fuzzy Control programs.

Optional standard features are defined in the Extension Class. Fuzzy Control programs applying these features can only be fully ported among systems using the same set of features, otherwise a partial exchange may be possible only. The Standard does not force all compliant systems to realise all features in the Extension Class, but it supports the possibility of (partial) portability and the avoidance of the usage of non-standard features. Therefore a compliant system should not offer non-standard features which can be meaningfully realised by using standard features of the Basic Class and the Extension Class.

In order not to exclude systems using their own highly sophisticated features from complying with this Standard and not to hinder the progress of future development, the Standard permits also additional non-standard features which are not covered by the Basic Class and the Extension. However, these features need to be listed in a standard way to ensure that they are easily recognised as non-standard features.

The portability of Fuzzy Control applications depends on the different programming systems and also the charcteristics of the control systems. These dependencies are covered by the Data Check List to be delivered by the manufacturer.

1 Scope

This part of IEC 1131 defines a language for programming of Fuzzy Control applications which use programmable controllers.

2 Normative references

The following standard documents contain provisions which, through reference in this text, constitute provisions of this part of IEC 1131. At the time of publication, the editions indicated below were valid. All such documents are subject to revision. Parties to agreements based on this part of IEC 1131 are therefore encouraged to apply the current editions of standards. Members of IEC and ISO maintain registers of current standard documents.

IEC 1-1581-FDIS 1996, International Electrotechnical Vocabulary, clause 351: Automatic Control
IEC 1131 Part 1: 1992, General information
IEC 1131 Part 2: 1992, Equipment requirements and tests
IEC 1131 Part 3: 1993, Programming languages
IEC 1131 Technical Report, Guidelines for users and implementers of IEC 1131-3

3 Definitions

Further definitions for language elements are given in Part 3.

3.1 **accumulation** [or **result aggregation**]: Combination of results of *linguistic rules* in a final result.

3.2 **aggregation**: Calculation of the degree of accomplishment of the *condition* of a rule.

3.3 **activation**: The process by which the degree of fulfilment of a *condition* acts on an output fuzzy set.

NOTE - Also known as composition.

3.4 **conclusion:** The output of a *linguistic rule*, i.e. the actions to be taken (the THEN part of an IF..THEN Fuzzy Control rule).

NOTE - Also known as consequent.

3.5 **condition**: A composition of *linguistic terms* forming the IF-part of a rule.

3.6 **crisp set**: A crisp set is a special case of a *Fuzzy set*, in which the *membership function* only takes two values, commonly defined as 0 and 1.

3.7 **defuzzification**: Conversion of a *Fuzzy set* into a numerical value.

3.8 **degree of membership**: membership function value.

3.9 **fuzzification**: Conversion of an input value into degrees of membership for the membership functions defined on the variable taking this value.

3.10 **Fuzzy Control**: A type of control in which the control algorithm is based on *Fuzzy Logic* (IEV 351-07-51 modified)

3.11 **Fuzzy Logic**: Collection of mathematical theories based on the notion of *Fuzzy set*. *Fuzzy Control* is a branch of multi-valued Control.

3.12 **Fuzzy Control operator**: Operator used in Fuzzy Logic theory

3.13 **Fuzzy set**: A *Fuzzy set* A is defined as the set of ordered pairs $(x,\mu A(x))$, where x is an element of the universe of discourse U and $\mu A(x)$ is the *membership function*, that attributes to each $x \in U$ a real number $\in [0,1]$, describing the degree to which x belongs to the set.

3.14 **inference**: Application of *linguistic rules* on input values in order to generate output values

3.15 **linguistic rule**: IF-THEN rule with *condition* and *conclusion*, one or both at least linguistic.

3.16 **linguistic term**: In the context of Fuzzy Control *linguistic terms* are defined by *Fuzzy sets*

3.17 **linguistic variable**: Variable that takes values in the range of *linguistic terms*.

3.18 **membership function**: A function which expresses in which degree an element of a set belongs to a given Fuzzy subset (see IEV 351-07-52).

NOTE - Also known as antecedent.

3.19 **singleton**: A singleton is a *Fuzzy set* whose *membership function* is equal to one at one point and equal to zero at all other points.

3.20 **rule base**: Collection of *linguistic rules* to attain certain objectives.

3.21 **weighting factor**: Value between 0..1, that states the degree of importance, credibility, confidence of a linguistic rule.

4 Integration into the Programmable Controller

The Fuzzy Control applications programmed in Fuzzy Control Language FCL according to clause 5 of this Part of the Standard shall be encapsulated in Function Blocks (or Programs) as defined in IEC1131 Part 3, Programming Languages. The concept of Function Block Types and Function Block Instances given in Part 3 apply to this part.

The Function Block Types defined in Fuzzy Control Language FCL shall specify the input and output parameters and the Fuzzy Control specific rules and declarations.

The corresponding Functions Block Instances shall contain the specific data of the Fuzzy Control applications.

Function Blocks defined in Fuzzy Control Language FCL can be used in Programs and Function Blocks written in any of the languages of Part 3, e.g. Ladder Diagram, Instruction List, etc. The data types of the input and output parameters of the Function Block or Program written in FCL shall match those of the corresponding “calling environment“ as illustrated in Figure 4-1.

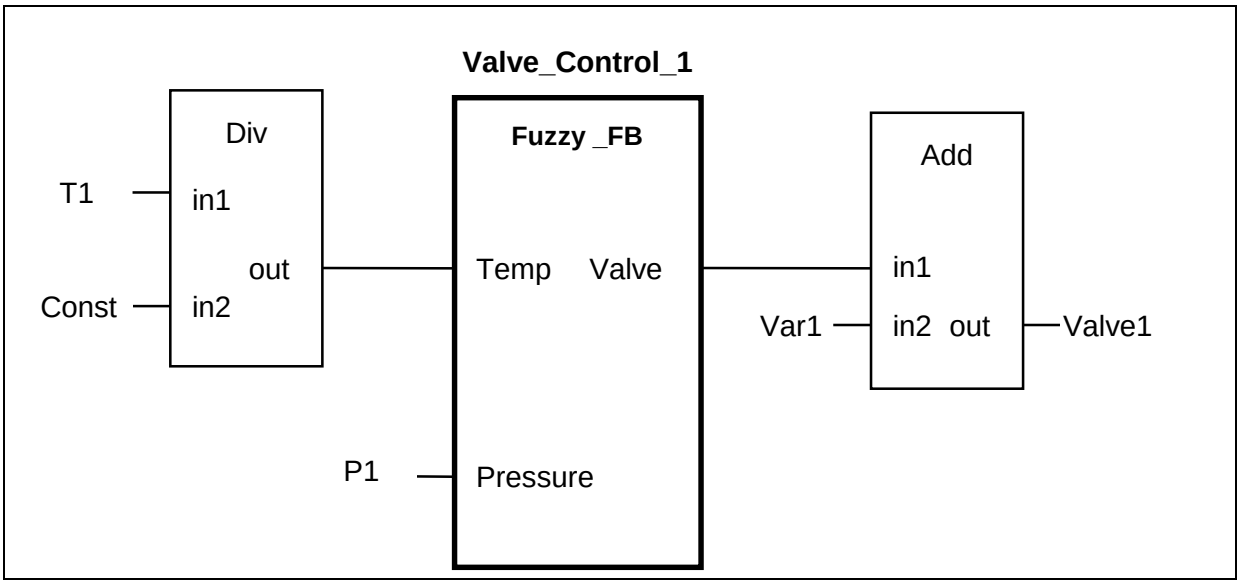


Figure 4-1: Example of a Fuzzy Control Function Block in FBD representation

In this example Valve_Control_1 is a user defined Function Block Instance of the Function Block Type Fuzzy_FB. The Function Block Type Fuzzy_FB may be programmed in Fuzzy Control Language FCL according to clause 5 of this Part. The Function Block Fuzzy_FB is used here in a program or a Function Block which is represented in the graphical language FBD (Function Block Diagram) of Part 3.

5 Fuzzy Control Language FCL

5.1 Exchange of Fuzzy Control programs

The definition of the Fuzzy Control Language FCL is based on the definitions of the programming languages in Part 3. The interaction of the Fuzzy Control algorithm with its program environment causes it to be “hidden“ from the program. The Fuzzy Control algorithm is therefore externally represented as a Function Block according to Part 3. The necessary elements for describing the internal linguistic parts of the Fuzzy Control Function Block like membership functions, rules, operators and methods have to be defined according to clause 5.

The language elements of FCL standardise a common representation for data exchange among Fuzzy Control configuration tools of different manufacturers shown in figure 5.1-1. Using this common representation every manufacturer of programmable controllers can keep his hardware, software editors and compilers. The manufactuer has only to implement the data interface into his specific editor. The customer would be able to exchange Fuzzy Control projects between different manufacturers.

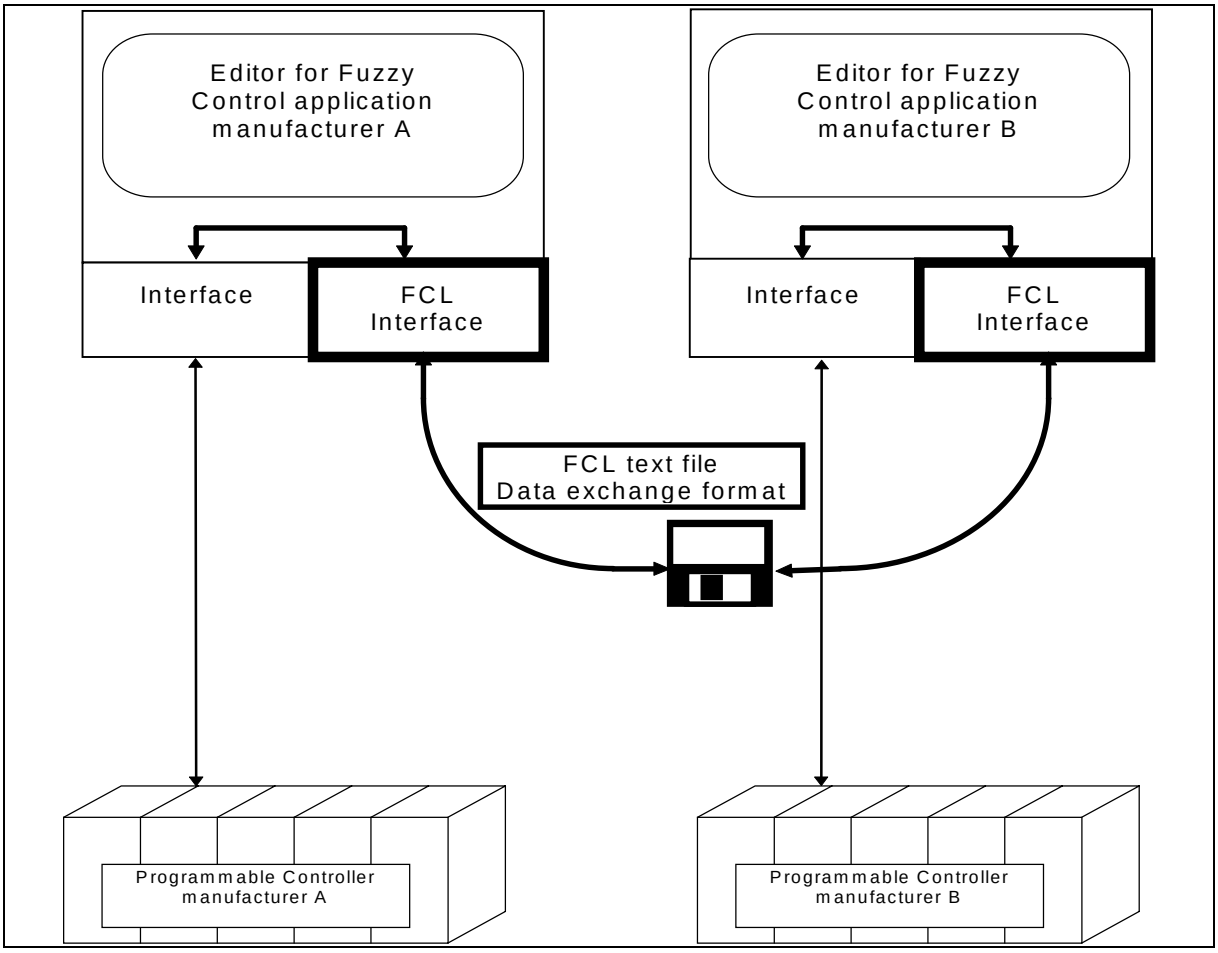


Figure 5.1-1: Data exchange of Programs in Fuzzy Control Language (FCL)

5.2 Fuzzy Control Language elements

Fuzzy control language elements in this clause are described using examples. The detailed production rule is given in clause 5.4.

5.2.1 Function Block interface

According to clause 4 the external view of the Fuzzy Function Block requires that the following standard language elements of Part 3 shall be used:

FUNCTION_BLOCK *function_block_name*	Function block
VAR_INPUT *variable_name*: *data_type*; .... END_VAR	Input parameter declaration
VAR_OUTPUT variable_name: *data_type*; .... END_VAR	Output parameter declaration
.... VAR *variable_name: data_type*; END_VAR END_FUNCTION_BLOCK	Local *variables*

With these language elements it is possible to describe a function block interface. The function block interface is defined with parameters which are passed into and out of the function block. The data types of these parameters shall be defined according to Part 3.

Figure 5.2.1-1 shows an example for a Function Block declaration in Structured Text (ST) and Function Block Diagram (FBD) languages.

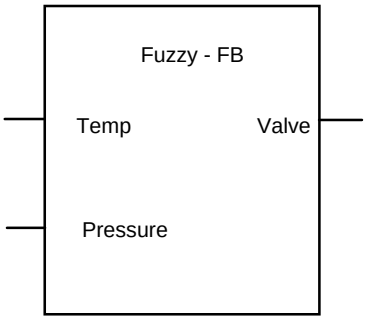
FUNCTION_BLOCK Fuzzy-FB VAR_INPUT Temp: REAL; Pressure: REAL; END_VAR VAR_OUTPUT Valve: REAL; END_VAR END_FUNCTION_BLOCK	
Structured Text (ST)	*Function Block Diagram (FBD)*

Figure 5.2.1-1: Example of a Function Block interface declaration in ST and FBD languages

5.2.2 Fuzzification

The values of the input variables have to be converted into *degrees of membership* for *the membership functions* defined on the variable. This conversion is described between the keywords FUZZIFY and END_FUZZIFY.

```
FUZZIFY variable_name
        TERM term_name := membership_function ;
        ....
END_FUZZIFY
```

After the keyword FUZZIFY the name of a variable which is used for the fuzzification shall be named.

This is the name of a previously defined variable in the VAR_INPUT section. This *linguistic variable* shall be described by one or more *linguistic terms*. The *linguistic terms* introduced by the keyword TERM described by *membership functions* in order to fuzzify the variable. A *membership function* is a piece-wise linear function. It is defined by a table of points.

membership_function ::= (*point i*), (*point* j), ...

Every point is a pair of the values of the variable and the membership degree of that value separated by a comma. The pairs are enclosed in parentheses and separated by commas.

point i ::= *value* of input i / *variable_name* of input i , *value* i of membership degree

With this definition all simple elements e.g. ramp and triangle can be defined. The points shall be given in ascending order of variable value. The membership function is linear between successive points. The degree of membership for each term is therefore calculated from the crisp input value by the linear interpolation between the two relevant adjacent membership function points.

The number of points can vary, but its maximum number is restricted according to the clause 6 conformance classes.

Example of *membership function* with 3 points for *linguistic term* "warm":

TERM warm := (17.5, 0.0), (20.0, 1.0), (22.5, 0.0);

If the value of a *linguistic variable* is less than the first base point in the look-up table all values below the first point in the lookup table shall have the same membership degree as defined at the first point.

If the value of a *linguistic variable* is greater than the last base point in the lookup table all values greater that the last point in the lookup table shall have the same membership degree as defined at the last point.

Definition of ramps:

```
FUZZIFY temp;
    TERM cold :=    (3, 1) (27, 0);
    TERM warm :=    (3, 0) (27, 1);
END_FUZZIFY
```

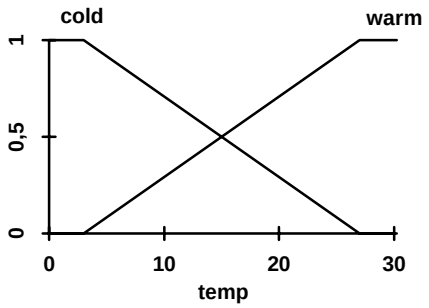


Figure 5.2.2-1: Example of ramp terms

In order to adapt the Fuzzy Control application on-line the base points in the membership functions can be modified. This can be done using variables which are input to the function block. These variables have to be declared in the VAR_INPUT section of the function block. An example for the use of variables for the definition of the points the membership functions is given in fig. 5.2.2-2.

NOTE - The values of membership functions points at runtime may be out of sequence.

```
VAR_INPUT
        temp: REAL;                          (* this input shall be fuzzified *)
        pressure: REAL;                      (* this input shall be fuzzified *)
        bp_warm1, bp_warm2 : REAL;           (* these inputs are for on-line adaptation *)
END_VAR
FUZZIFY temp
        TERM warm := (bp_warm1, 0.0), (21.0, 1.0),  (bp_warm2, 0.0);
..
END_FUZZIFY
```

Figure 5.2.2-2: Example of usage of variables for membership functions

5.2.3 Defuzzification

A *linguistic variable* for an output variable has to be converted into a value. This conversion is described between the keywords DEFUZZIFY and END_DEFUZZIFY.

After the keyword DEFUZZIFY the variable which is used for the *defuzzification* shall be named. This is the name of a previous defined variable in the VAR_OUTPUT section.

```
DEFUZZIFY variable_name
        TERM term_name := membership_function ;
        defuzzification_method ;
        default_value ;
        [range ;]
END_DEFUZZIFY
```

The definition of *linguistic terms* is given in clause 5.2.2 Fuzzification.

Singletons are special *membership functions* used for outputs in order to simplify the *defuzzification*. They are described only by a single value for the *linguistic term*. In figure 5.2.3-1 examples of terms are given.

Definition of *singletons:*

```
DEFUZZIFY valve
        TERM drainage := -100;
        TERM closed   := 0;
        TERM inlet    := 100;
END_DEFUZZIFY
```

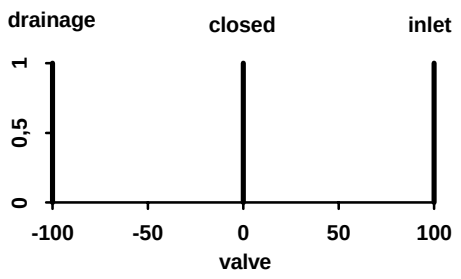


Figure 5.2.3-1: Example of singleton terms

The defuzzification method shall be defined by the language element Method.

```
METHOD :  defuzzification_method ;
```

The following defuzzification methods are possible.

Table 5.2.3-1: Defuzzification methods

Keyword	Explanation
COG	Centre of Gravity (Note 1)
COGS	Centre of Gravity for Singletons
COA	Centre of Area (Notes 2 and 3)
LM	Left Most Maximum
RM	Right Most Maximum

NOTE 1 - Centre of Gravity is equivalent to Centroid of Area

NOTE 2 - Centre of Area is equvalent to Bisector of Area

NOTE 3 - COA is not applicable if singletons are used.

Table 5.2.3-2: Formulae for Defuzzification methods

COG	$U = \frac{\int_{Min}^{Max} u\,\mu(u)\,du}{\int_{Min}^{Max} \mu(u)\,du}$
COGS	$U = \frac{\sum_{i=1}^{p} [u_i \mu_i]}{\sum_{i=1}^{p} [\mu_i]}$
COA	$U = u', \int_{Min}^{u'} \mu(u)du = \int_{u'}^{Max'} \mu(u)du$
RM	$U = \sup(u'), \mu(u') = \sup_{u \in [Min,Max]} \mu(u)$
LM	$U = \inf(u'), \mu(u') = \sup_{u \in [Min,Max]} \mu(u)$

where:

U :	result of defuzzification
u :	output variable
p :	number of singletons
μ :	membership function after accumulation
i :	index
Min :	lower limit for defuzzification
Max :	upper limit for defuzzification
sup :	largest value
inf:	smallest value

If the degree of membership is 0 for all *linguistic terms* of an output variable, that means: no rule for this variable is active. In that case the *defuzzification* is not able to generate a valid output. Therefore it is possible to define a default value for the output. This default value is the value for the output variable only in the case when no rule has fired.

DEFAULT := *value* | NC;

After the keyword DEFAULT the value shall be specified. Otherwise the key word NC (no change) shall be specified to indicate that the output shall remain unchanged if no rule has fired.

The range is a specification of a minimum value and a maximum value separated by two points.

RANGE := (minimum *value* .. maximum *value*);

The range is used for the specification of minimum and maximum values of an output variable. This is not applicable if singletons are used for output membership functions. In other cases, the RANGE is used for limiting each membership function to the range of each output variable and should be defined to avoid unpredictable output values.

If there is no range defined the default range shall be the range of the data type of the variable specified in Part 3.

5.2.4 Rule block

The inference of the fuzzy algorithm shall be defined in one or more rule blocks. For proper handling and to cater for the possibility of splitting the rule base into different modules, the use of several rule blocks is possible. Each rule block has a unique name.

Rules shall be defined between the keywords RULEBLOCK and END_RULEBLOCK.

```
RULEBLOCK ruleblock_name
        operator_definition ;
        [activation_method ;]
         accumulation_method ;
         rules ;
END_RULEBLOCK
```

The Fuzzy operators are used inside the rule block.

```
operator_definition ::= operator : algorithm
```

To fulfill de Morgan's Law, the algorithms for operators AND and OR shall be used pair-wise e.g. MAX shall be used for OR if MIN is used for AND.

Table 5.2.4-1: Paired algorithms

operator **OR**		operator **AND**	
keyword for **Algorithm**	**Algorithm**	keyword for **Algorithm**	**Algorithm**
MAX	Max ($\mu_1(x)$, $\mu_2(x)$)	MIN	Min($\mu_1(x)$, $\mu_2(x)$)
ASUM	$\mu_1(x)+\mu_2(x) - \mu_1(x)\ \mu_2(x)$	PROD	$\mu_1(x)\ \mu_2(x)$
BSUM	Min(1, $\mu_1(x) + \mu_2(x)$)	BDIF	Max (0, $\mu_1(x) + \mu_2(x) -1$)

An example of rule blocks:

```
RULEBLOCK first
        AND : MIN;
        ..
END_RULEBLOCK
RULEBLOCK second
        AND : PROD;
        ..
END_RULEBLOCK
```

The following language element defines the method of the activation:

```
ACT : activation_method;
```

The following activation methods are available:

Table 5.2.4-2: Activation methods

Name	Keyword	Algorithm
Product	**PROD**	**$\mu_1(x)\ \mu_2(x)$**
Minimum	**MIN**	**$Min(\mu_1(x), \mu_2(x))$**

NOTE -The activation method is not relevant for singelton.

The following language element defines the method of the accumulation.

ACCU : *accumulation_method*;

The following *accumulation_methods* are possible:

Table 5.2.4-3: Accumulation methods

Name	**Keyword**	**Formula**
Maximum	MAX	$Max(\mu_1(x), \mu_2(x))$
Bounded Sum	BSUM	$Min(1, \mu_1(x) + \mu_2(x))$
Normalised Sum	NSUM	$\frac{\mu_1(x) + \mu_2(x)}{Max(1, MAX(\mu_1(x') + \mu_2(x')))}$

The inputs of a rule block are *linguistic variables* with a set of *linguistic terms*. Each term has a degree of membership assigned to it.

Inside the rule block the rules are defined. Each begins with the keyword RULE followed by a name for the rule and shall be concluded by a semicolon. Each rule has a unique number inside the rule block.

RULE *numbers* : IF condition THEN conclusion [WITH weighting factor];

The rule itself shall begin with the keyword IF followed by the *condition*. After the *condition* the *conclusion* follows beginning with the keyword THEN.

It is possible to combine several *subconditions* and input variables in one rule. The purpose of variables is to permit fuzzy degrees of membership to be imported into the Fuzzy Function Block. All of them shall be defined between the kewords IF and THEN and combined by the operators with the keywords AND, OR or NOT .

The priority of the operator is handled according to boolean algebra given in Table 5.2.4.1.

Table 5.2.4-4: Priority of operators

Priority	**operator**
1	( ) parenthesis
2	NOT
3	AND
4	OR

Simplified example for a rule:

RULE 1 : IF *subcondition1* AND *variable1* OR *variable2* THEN *conclusion*;

In the Basic Level of conformance the OR operation can be implemented by defining two rules:

RULE 3 : IF *subcondition 1* OR *subcondition 2* THEN *conclusion*;
replaced by:
RULE 3a : IF *condition 1* THEN conclusion;
RULE 3b : IF *condition 2* THEN conclusion;

The subcondition begins with the name of a *linguistic variable* followed by the keyword IS with an optional NOT and one *linguistic term* of the *linguistic variable* used in the *condition.*

Subcondition := *linguistic_variable* IS [NOT] *linguistic_term*

Note: The *linguistic terms* which are used in the *condition* shall match the *linguistic variable* in the same *condition*. The term used has to be previously defined with the keyword TERM.

Example of subconditions:

```
temp IS hot
temp IS NOT hot
```

It is also possible to use the keyword NOT in front of the subcondition. In this case parentheses shall be used.

```
IF NOT (temp IS hot) THEN ...
```

The *conclusion* can be split into several subconclusions and output variables.

The subconclusion begins with the name of a *linguistic variable* followed by the keyword IS and one *linguistic term* of the given *linguistic variable*.

Subconclusion := *linguisti_ variable* IS *linguisti_ term*

Example with several subconclusions in one or more lines:

```
    IF temp IS cold AND pressure IS low THEN var1, valve1 IS inlet , valve2 IS closed;
or In several lines:
    IF temp IS cold AND pressure IS low
    THEN  var1,
          valve1 IS inlet,
          valve2 IS closed;
```

Optionally it is possible to give each subconclusion a *weighting factor* which is a number of data type REAL with a value between 0.0 and 1.0. This shall be done by the keyword WITH followed by the *weighting factor*.

The *weighting factor* shall reduce the membership degree (membership function) of the subconclusion by multiplication of the result in the subconclusion with the *weighting factor*.

In order to manipulate the Fuzzy Control application parameters externally the *weighting factor* can be a variable. In this case the variable has to be declared in the VAR_INPUT section. This enables the possibility to change the *weighting factor* during runtime in order to adapt the Fuzzy Conrtol program to process needs.

If there is no WITH statement assigned to the subconclusion a default *weighting factor* of 1.0 shall be assumed.

IF *condition* THEN *subconclusion* [WITH *weighting_factor*] *subconclusion* ;

An example of a constant *weighting_factor*:

```
IF temp IS cold AND pressure IS low THEN    valve1 IS inlet WITH 0.5,
                                            valve2 IS closed;
```

An example of a variable *weighting factor*:

```
VAR_INPUT
        w_myrule1 : REAL := 0.8;
END_VAR
RULEBLOCK temp_rule
        RULE 1:     IF temp         IS cold AND pressure IS low
                    THEN valve      IS inlet WITH w_myrule1;
        ..
END_RULEBLOCK
```

5.2.5 Optional parameters

For implementation on different target systems it may be necessary to give additional information to the system in order to allow the best possible conversion of Fuzzy Control applications.

Such additional information may be required in a language element enclosed by OPTIONS and END_OPTIONS.

```
OPTIONS
        application_specific_parameters
END_OPTIONS
```

These language elements shall be used for features in the conformance class of the open level according clause 6.

5.3 FCL example

An example in Fuzzy Control Language is given in Figure 5.3-1.

```
FUNCTION_BLOCK Fuzzy_FB
VAR_INPUT
        temp :      REAL;
        pressure :  REAL;
END_VAR
VAR_OUTPUT
        valve :     REAL;
END_VAR
FUZZIFY temp
        TERM cold := (3, 1)  (27, 0);
        TERM hot  := (3, 0  (27, 1);
END_FUZZIFY
FUZZIFY pressure
        TERM low := (55, 1) (95, 0);
        TERM high:= (55, 0) (95, 1);
END_FUZZIFY
DEFUZZIFY valve
        TERM drainage    := -100;
        TERM closed      := 0;
        TERM inlet       := 100;
        ACCU : MAX;
        METHOD : COGS;
        DEFAULT := 0;
END_DEFUZZIFY
RULEBLOCK No1
        AND : MIN;
        RULE 1 : IF temp IS cold AND pressure IS low THEN valve IS inlet
        RULE 2 : IF temp IS cold AND pressure IS high THEN valve IS closed WITH 0.8;
        RULE 3 : IF temp IS hot AND pressure IS low THEN valve IS closed;
        RULE 4 : IF temp IS hot AND pressure IS high THEN valve IS drainage;
END_RULEBLOCK
END_FUNCTION_BLOCK
```

Figure 5.3-1: Example for fuzzy function block

5.4 Production Rules and Keywords of the Fuzzy Control Language (FCL)

Annex A of Part 3 defines the specification method for textual languages for Programmable Controllers. This specification method is used here for FCL.

Annex B of Part 3 defines the formal specification of language elements for the textual programming languages of Part 3. For FCL a subset of the following language elements of Part 3 shall be used:

B.1.1 Letters, digits, identifers
B.1.2 Constants
B.1.3 Data types
B.1.4 Variables

5.4.1 Production Rules

Additionally to the above listed language elements of Part 3 following language elements shall be used:

function_block_declaration ::= ‘FUNCTION_BLOCK‘ function_block_name
{fb_io_var_declarations}
{other_var_declarations}
function_block_body
‘END_FUNCTION_BLOCK’

fb_io_var_declarations ::= input_declarations | output_declarations

other_var_declarations ::= var_declarations

function_block_body ::= {fuzzify_block}
{defuzzify_block}
{rule_block}
{option_block}

fuzzify_block ::= ‘FUZZIFY’ variable_name
{linguistic_term}
‘END_FUZZIFY

defuzzify_block ::= ‘DEFUZZIFY’ f_variable_name
{linguistic_term}
defuzzification_method
default_value
[range]
‘END_FUZZIFY’

rule_block ::= ‘RULEBLOCK’ rule_block_name
operator_definition
[activation_method]
accumulation_method
{rule}
‘END_RULEBLOCK’

option_block ::= ‘OPTION’
any manufacturere specific parameter
‘END_OPTION’

linguistic_term ::= ‘TERM’ term_name ‘:=‘ membership_function ‘;’

membership_function ::= singleton | points

singleton ::= numeric_literal | variable_name

points ::= {‘(‘ numeric_literal | variable_name ‘,’
numeric_literal ‘)’}

defuzzification_method ::= ‘METHOD’ ‘:’ ‘COG’ | ‘COGS’ | ‘COA’ | ‘LM’ | ‘RM’ ‘;’

default_value ::= ‘DEFAULT’ ‘:=‘ numeric_literal | ‘NC’ ‘;’

range ::= ‘RANGE’ ‘:=‘ ‘(‘numeric_literal ‘..’ numeric_literal‘)’ ‘;’

operator_definition ::= (‘OR’ ‘:’ ‘MAX’ | ‘ASUM’ | ‘BSUM’) |
(‘AND’ ‘:’ ‘MIN’ | ‘PROD’ | ‘BDIF’) ‘;’

activation_method ::= ‘ACT’ ‘:’ ‘PROD’ | ‘MIN’ ‘;’

accumulation_method ::= ‘ACCU’ ‘:’ ‘MAX’ | ‘BSUM’ | ‘NSUM’ ‘;’

rule ::= ‘RULE’ integer_literal ‘:’
‘IF’ condition ‘THEN’ conclusion [WITH weighting_factor] ‘;’

condition ::= (subcondition | variable_name) {AND’ | ‘OR’
(subcondition | variable_name)}

subcondition ::= (‘NOT’ ‘(‘ variable_name ‘IS’ [‘NOT’]) term_name ‘)’) |
(variable_name ‘IS’ [‘NOT’] term_name)

conclusion ::= { (variable_name | (variable_name ‘IS’ term_name)) ‘,’}
(variable_name | variable_name ‘IS’ term_name)

weighting_factor ::= variable | numeric_literal

function_block_name ::= identifier

rule_block_name ::= identifier

term_name ::= identifier

f_variable_name ::= identifier

variable_name ::= identifier

numeric_literal ::= integer_literal | real_literal

input_declarations ::= see IEC 1131-3 Annex B

output_declarations ::= see IEC 1131-3 Annex B

var_declarations ::= see IEC 1131-3 Annex B

identifier ::= see IEC 1131-3 Annex B

5.4.2 Keywords

Table 5.4.2-1: Reserved keywords

Keyword	Meaning	Clause
( )	Parentheses in condition, term, range	5.2.4
ACCU	Accumulation method	5.2.4
ACT	Actuation method	5.2.4
AND	AND operator	5.2.4
ASUM	OR operator, Algebraic sum	5.2.4
BDIF	AND operator, Bounded difference	5.2.4
BSUM	Accumulation method, Bounded sum	5.2.4
COA	Center of area defuzzification method	5.2.3
COG	Center of gravity defuzzification method	5.2.3
COGS	Center of gravity defuzzification of singletons	5.2.4
DEFAULT	Default output value in case no rule has fired	5.2.3
DEFUZZIFY	Defuzzification of output variable	5.2.3
END_DEFUZZIFY	End of defuzzification specifications	5.2.3
END_FUNCTION_BLOCK	End of function block specifications	5.2.1
END_FUZZIFY	End of fuzzification specifications	5.2.3
END_OPTIONS	End of options specifications	5.2.5
END_RULEBLOCK	End of rule block specifications	5.2.4
END_VAR	End of input/output variable definitions	5.2.1
FUNCTION_BLOCK	End of function block specifications	5.2.1
FUZZIFY	Fuzzification of input variable	5.2.2
IF	Begin of rule which is followed by the condition	5.2.4
IS	Follows linguistic variable in condition and conclusion	5.2.4
LM	Left Most Maximun defuzzification method	5.2.3
MAX	Maximum accumulation method	5.2.3
METHOD	Method of defuzzification	5.2.3
MIN	Minimum as AND operator	5.2.4
NC	No Change of output variable in case no rule has fired	5.2.3
NOT	NOT operator	5.2.4
NSUM	Normalised sum accumulation method	5.2.4

OPTIONS	Definition of optional parameters	5.2.5
OR	OR operator	5.2.4
PROD	Product as AND operator	5.2.4
RANGE	Range of variable for scaling of membership function	5.2.3
RM	Right Most Maximum defuzzification method	5.2.3
RULE	Begin of specification of fuzzy rule	5.2.4
RULEBLOCK	Begin of specification of rule block	5.2.4
TERM	Definition of a linguistic term (membership function) for a linguistic variable	5.2.2
THEN	Separates condition from conclusion	5.2.4
VAR	Definition of local variable (s)	5.2.1
VAR_INPUT	Definition of input variable(s)	5.2.1
VAR_OUTPUT	Definition of output variable(s)	5.2.1
WITH	Definition of weighting factor	5.2.4

6 Compliance

6.1 Conformance Classes of Fuzzy Control Language FCL

The levels of conformance for Control System using the Fuzzy Control Language (FCL) are shown in Figure 6.1-1. The hierarchy consists of the following three levels:

Basic Level including the definitions of the function block and the data types of Part 3.

Extension level with optional features.

Open Level encompassing additional features.

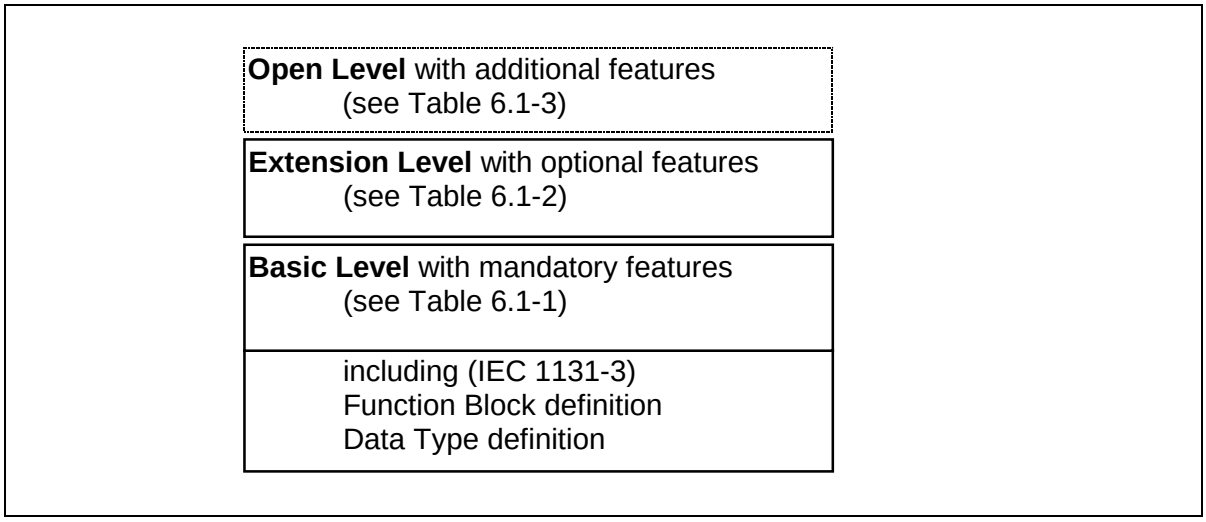


Figure 6.1-1: Levels of conformance

A Control System using the Fuzzy Control Language FCL claiming to conform with this part of the Standard shall achieve the following rules:

1. It shall be able to use the Function Block features according to Part 3 of this Standard in order to realise the Fuzzy Control functionality. Therefore the definition of the Function Blocks and the Data Types required for the input and output parameters of the function block shall be in accordance with Part 3.

2. All features of Fuzzy Control functionality defined in Table 6.1-1 shall be implemented according to the definitions of this Part. This table defines the set of Basic Level elements which all standard systems shall have in common.

3. A subset of the Extension Level elements defined in table 6.1-2 are additional elements which can be implemented optionally. The implementation shall be exactly according to the definitions of this Part. These features shall be marked as Standard Extensions and a list of realised features in form of Table 6.1-2 shall be part of the system documentation.

4. Further features exceeding the Basic Level and the Extended Level may be realised provided these features do not have the same or similar functionality or representation of the standard features thereby avoiding any possible confusion. These features shall be marked as Open Level features and a list in form of Table 6.1-3 shall be part of the system documentation.

5. The exchange of application programs among different Fuzzy Control systems shall be done in the textual form of the Fuzzy Control Language FCL according to the definitions in this part. This

format shall be made available by the standard comformant systems as input and output formats.

6. In order to achieve the most comfortable and suitable user interface and to not hinder future progress, the external representations for the design, input, testing etc. of Fuzzy Control application programs can be realised by any graphical or textual means.

The elements in the Table 6.1-1 are the basic set of features, which shall be realised in all standard Fuzzy Conrol systems.

Table 6.1-1: FCL Basic Level language elements (mandatory)

Language Element	**Keyword**	**Details**
function block declaration	VAR_INPUT, VAR_OUTPUT	contains input and output variables
membership function	input variable: TERM	maximum of three points (degree of membership co-ordinate = 0 or 1)
	output variable: TERM	only singletons
conditional aggregation	operator: AND	algorithm: MIN
activation	-	Not relevant because singletons are used only
accumulation (result aggregation)	operator: ACCU	algorithm: MAX
defuzzification	METHOD	algorithm: COGS
default		Note - Provisions shall be made to handle this feature in the Basic Class
condition	IF ... IS ...	n subconditions
conclusion	THEN	only one subconclusion
weighting factor	WITH	value only

The elements in the Table 6.1-2 are the extended set of features, which can be optionally realised in a standard Fuzzy Control system (e.g. for the AND operator the algorithm PROD or BDIF or both might be chosen).

Table 6.1-2: FCL Extension Level language elements (optional)

Language Element	Keyword	Details
function block declaration	VAR	contains local variables
membership function:	input variable:TERM	maximum of four points (degree of membership co-ordinate = 0 or 1)
	output variable:TERM	maximum of four points (degree of membership co-ordinate = 0 or 1)
conditional aggregation	operator: AND	algorithm: PROD , BDIF
	operator: OR	algorithm: ASUM , BSUM
	operator: NOT	1 - {argument}
	parentheses	( )
activation	operator: ACT	algorithm: MIN, PROD
accumulation	operator: ACCU	algorithm: BSUM , NSUM
defuzzification method	operator: METHOD	algorithm: COG , COA , LM , RM
default value	DEFAULT	NC, value
condition	IF	n subconditions, n input variables
conclusion	THEN	n subconclusions, n output variables
weighting factor	WITH	value assigned to variable in the declaration part VAR_INPUT.....END_VAR

The Table 6.1-3 defines an example of a list of language elements in the Open Level. This list shall be a part of the system documentation.

Table 6.1-3: Examples of List with Open Level language elements

free input/output membership functions (e.g. Gaussian, exponential)
more than four membership function points
degree of membership co-ordinate values from 0 to 1
please dream on .....

6.2 Data check list

This data check list shall be delivered within the technical documentation. In this list a manufacturer of programmable controllers, Fuzzy programming tools and application software shall describe specific performance features of their Fuzzy Control system. In order to facilitate the transfer of Fuzzy Control applications among different manufacturer's systems the following Data check list is the means of verifying a possible program transfer.

Table 6.2-1: Data Check List

Technical data	Manufacturer statement (*examples*)
data types of function block inputs and outputs	*REAL, INT*
line comments in the FCL program	*YES, NO*
length of identifiers (e.g. name of variables, ruleblocks, terms)	6, 8
max. number of input variables for fuzzification	6, 8
max. number of membership function terms per input variable	5, 7
max. total number of membership function terms for all input variables	30, 56
max. number of points for the membership function associated with each input variable term	3, 4, 10
max total number of points for membership functions associated with all input variable terms	90, 224
max. number of output variables for defuzzification	6, 8
max. number of membership function terms per output variable	5, 7
max. total number of membership function terms for all output variables	30, 56
max. number of points for the membership function associated with each output variable term	1, 4, 10
max total number of points for membership functions associated with the all output variable terms	90, 224
max. number of rule blocks	1, 10
max. number of rules per block	10
max. number of subconditions per rule	4, 10
max. number of all rules	15
max. number of subconclusions per rule	4
Nesting depth of ( )	1, 3

Bibliografía

[AMHH04] T. Akenine-Möller, E. Haines, y N. Hoffman. *Real Time Rendering. Third Edition*. 2004.

[ASN+12] H. Akiyama, H. Shimora, T. Nakashima, Y. Narimoto, y K. Yamashita. HELIOS2012 Team Description Paper. 2012.

[BCvKO08] M. Berg, O. Cheong, M. van Kreveld, y M. Overmars. *Computational Geometry: Algorithms and Applications*. 2008.

[BGB+07] M. Beetz, S. Gedikli, J. Bandouch, B. Kirchlechnerand, y A. Perzylo. Visually Tracking Football Games Based on TV Broadcasts. 2007.

[BMM+07] R. Barros, M. Misuta, R. Menezes, P. Figueroa, F. Moura, S. A Cunha, R. Anido, y N. Leite. Analysis of the distances covered by first division Brazilian soccer players obtained with an automatic tracking method. 2007.

[BV01] P. Buhler y José M. Vidal. Biter: A Platform for the Teaching and Research of Multiagent Systems' Design using Robocup. 2001.

[BZL+12] A. Bai, H. Zhang, G. Lu, M. Jiang, y X. Chen. WrightEagle 2D Soccer Simulation Team Description 2012. 2012.

[Fou08] LA84 Foundation. *Manual de entrenamiento de fútbol*. 2008.

[Fun99] S. Fung. *Basic coaching manual*. 1999.

[GHJV96] E. Gamma, R. Helm, R. Johnson, y J. Vlissides. *Dessign Patterns*. Addison-Wesley, 1996.

[Has09] M. Hasic. The most common soccer formations. 2009.

[HMN05] E. Huerta, A. Mangiaterra, y G. Noguera. *GPS: Posicionamiento satelital*. 2005.

[IPLS09] J. Iglesias, M. Palancín, A. Ledezma, y A. Sanchís. Estudio del Comportamiento de los Jugadores de la RoboCup Utilizando VIENA. 2009.

[KAK+97] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, y H. Matsubara. RoboCup: A Challenge Problem for AI. 1997.

[Mam74] E. H. Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. 1974.

[MTMH11] M. Montañes, E. Torres, J. Martínez, y J. Herrero. Real-time GPU color-based segmentation of football players. 2011.

[Nam01] Namahn. Using eye tracking for usability testing. 2001.

[Nú08] J. Núñez. Introducción a Robocup. 2008.

[PL02] E. Pazera y A. LaMothe. *Focus On SDL*. 2002.

[Ree03] T. Reenskaug. The Model-View-Controller (MVC). Its Past and Present. 2003.

[RN04] Stuart J. Russell y Peter Norvig. *Inteligencia Artificial: Un enfoque moderno (segunda edición)*. 2004.

[Sau10] T. Sauder. *Soccer systems to play*. 2010.

[Sha75] R. Shannon. *Systems simulation: The art and science*. Prentice-Hall, 1975.

[TS85] T. Takagi y M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. 1985.

[Tur01] Tom Turner. Coaching team tactics for U-11's and U-12's. 2001.

[Vli07] H. Vliet. *Software Engineering: Principles and Practice*. 2007.

[XLO04] M. Xu, L. Lowey, y J. Orwell. Architecture and algorithms for tracking footbal players with multiple cameras. 2004.

[Zad65] L. A. Zadeh. Fuzzy Sets. 1965.

[Zad76] L. A. Zadeh. A Fuzzy Algorithm Approach. 1976.

Este documento fue editado y tipografiado con LaTeX
empleando la clase **arco-pfc** que se puede encontrar en:
`https://bitbucket.org/arco_group/arco-pfc`