# 1 Introducción

El ritmo de vida de los países industrializados crece a un ritmo exponencial, requiriendo cada día de nuevas tecnologías capaces de desarrollar tareas y funciones más complejas. Cada vez es más habitual ver como las máquinas desarrollan el trabajo que anteriormente era realizado por personas: automatización de procesos industriales, fabricaciones a gran escala, desarrollo de nanotecnologías fuera del alcance del ojo humano, robótica e inteligencia artificial son algunos de los grandes avances todavía por explotar y que, sin duda, supondrán el futuro de la tecnología.

Uno de los campos que se está desarrollando es el de la investigación en nuevas tecnologías capaces de aprender del comportamiento humano. Esto, que puede sonar a ciencia ficción, ya es realidad hoy en día.

Esta tecnología puede ser desarrollada con el conjunto de herramientas estadísticas conocido como *Clasificadores*. Un clasificador es un elemento que proporciona una clase etiquetada como salida a partir de un conjunto de características tomadas como entradas.

Grandes cantidades de datos pueden ser tratados por los Clasificadores mediante la técnica de *Aprendizaje automático*. El aprendizaje automático es la parte básica que tienen en común los diferentes tipos de clasificadores que existen. La idea básica del aprendizaje consiste en utilizar las percepciones no sólo para actuar, sino también para mejorar la habilidad de un agente para actuar en el futuro. Existen diferentes tipos de técnicas de aprendizaje:

**Aprendizaje supervisado:**
El aprendizaje supervisado consiste en aprender una función, a partir de ejemplos etiquetados anteriormente, que establezca una correspondencia entre las entradas y las salidas deseadas del sistema. No siempre es posible hacer este tipo de entrenamiento ya que tenemos que disponer de la salida esperada en la función de entrada. El sistema de aprendizaje trata de etiquetar (clasificación) una serie de vectores utilizando una entre varias categorías (clases).

**Aprendizaje no supervisado**

El aprendizaje no supervisado consiste en aprender a partir de patrones de entradas para los que no se especifiquen los valores de sus salidas. El principal problema de esta técnica es la toma de decisiones a la hora de escoger un patrón entre todos los proporcionados. El sistema trata los objetos de entrada como un conjunto de variables aleatorias, construyendo un modelo de densidad para el conjunto de datos.

**Aprendizaje semi-supervisado**

Actualmente existen técnicas que combinan las dos anteriores, ya que en algunos casos puede resultar muy costoso asignar etiquetas o clases a todos los datos. La finalidad es combinar datos etiquetados y no etiquetados para mejorar la construcción de modelos. Aunque no siempre es útil y existen varios métodos para llevarlo a cabo.

**Aprendizaje por refuerzo**

El aprendizaje por refuerzo consiste en aprender observando el mundo que te rodea.
La idea del aprendizaje consiste en construir una función que tenga el comportamiento observado en sus datos de entrada y de salida. Los métodos de aprendizaje se pueden entender como la búsqueda de un espacio de hipótesis para encontrar la función adecuada.

Al mismo tiempo, también existen muchos tipos de clasificadores, que se diferencian entre ellos según el modo de análisis y tratamiento de los datos. Según las funciones que precisemos de ellos nos irá mejor un tipo u otro.

Las aplicaciones de los clasificadores son muy amplias. Su uso se extiende en medicina (análisis de prueba de drogas, análisis de datos de resonancia magnética), teléfonos móviles (descodificación de la señal, corrección de errores), la visión por computador (reconocimiento facial, seguimiento de objetivos), reconocimiento de voz, minería de datos (análisis de compras en supermercados, análisis de clientes al por menor) entre otras áreas diferentes.

Para poder dar algunos pasos más en este campo, en este proyecto se ha desarrollado una plataforma basada en el uso de algunos de estos clasificadores capaces de aprender de la conducta humana, y de automatizar sus movimientos empleando técnicas de clasificación

para decidir de manera autónoma qué movimientos debe realizar para alcanzar su objetivo final. Para ello se emplea una plataforma de juego de coches dónde el usuario debe dirigir un vehículo evitando que éste colisione con los demás coches que circulan por la vía. El juego va evolucionando a lo largo del tiempo, y al final el coche es capaz de evitar más del 90% de los obstáculos sin que el usuario deba dirigirle.

Los juegos en dos dimensiones implementados en JAVA no es nada nuevo. En algunos campos, por ejemplo en las aplicaciones lúdicas para teléfonos móviles o pda's, esta tecnología no tiene competencia alguna. No obstante, si se usaran conjuntamente Java con los clasificadores se podrían llegar a diseñar aplicaciones capaces de evolucionar y cambiar según las decisiones que tomara el usuario.

Este proyecto pretende investigar este campo, y experimentar la utilización de clasificadores para el diseño de nuevas tecnologías inteligentes. El escenario utilizado en el proyecto es sencillo, ya que, además de realizar el diseño y la implementación del aplicativo realizaremos un pequeño estudio experimental sobre la curva de aprendizaje y la imitación de la conducta humana del clasificador J48.

# 2 Objetivos

El primer objetivo es desarrollar la plataforma de juego, es decir, el juego de coches, en el que un usuario debe evitar colisionar conduciendo un vehículo en una autopista.

El segundo objetivo es utilizar la librería estadística Weka para tomar muestras del escenario y entrenar un clasificador con las decisiones que el jugador ha realizado.

El objetivo principal es usar la plataforma de juego para llevar a cabo un estudio sobre la aplicación de clasificadores para resolver problemáticas de decisión, teniendo en cuenta que en la carrera no se ha realizado nada parecido en ninguna asignatura podemos afirmar que es un reto totalmente nuevo.

## 2.1 Objetivos generales

1. Diseñar e implementar la plataforma de juego.
2. Diseñar e implementar el enlace con la librería estadística Weka.
3. Realizar un estudio experimental sobre los clasificadores.

## 2.2 Objetivos funcionales

El software debe estar diseñado e implementado con las herramientas y conocimientos adquiridos a lo largo de la carrera.

El juego de coches debe ser fácil de usar e intuitivo, al finalizar el mismo, se obtendrá un resumen estadístico para su análisis.

La aplicación no se desarrollará para un entorno distribuido.

Las funcionalidades del usuario serán:

1. Iniciar un juego en modo humano, es decir, realizando él los movimientos.
2. Iniciar un juego en modo automático.
3. Configurar la aplicación con el objetivo de hacer su estudio más sencillo.

4. Obtener un resumen estadístico de la partida tras detener el juego, tanto en modo humano como en automático.

## 2.3 Objetivos específicos

El desarrollo de la aplicación se llevará a cabo íntegramente con el leguaje de programación JAVA. El IDE (integrated development environment) elegido para el desarrollo será Netbeans [2].

La librería estadística para clasificar los escenarios y llevar a cabo los experimentos será Weka.

Para la creación y retoque de imágenes de libre uso se utilizarán las herramientas de software libre Gimp e Inkscape.

Tanto para la documentación como para el desarrollo del proyecto se utilizará la plataforma ofimática Microsoft Office 2007, básicamente el procesador de textos y la hoja de cálculo.

La grabación de ficheros persistentes de configuración o externos a los formatos usados por Weka se realizará en formato XML (Extensible Markup Language).

# 3 Tecnología y metodología

## 3.1 Tecnología

La tecnología en la que se va a basar el proyecto viene definida por el título, pero también es recomendable repasar si la elección ha sido adecuada. Para ello analizaremos el lenguaje de programación Java y el .NET Framework de Microsoft.

### 3.1.1 Java

Puntos fuertes de Java:

1. Impleméntalo una vez y ejecútalo en cualquier máquina.
2. Totalmente compatible con UML.
3. Orientado a objetos.
4. Funcionalidades especificas para desarrollar software basado en web y servicios web: foros, tiendas virtuales, encuestas, HTML, etc.
5. Gran facilidad de combinación de aplicaciones y servicios que usan el lenguaje Java para crear servicios o aplicaciones personalizadas.
6. Desarrollar potentes y eficientes aplicaciones para teléfonos móviles, procesadores remotos, productos de consumo de bajo coste y prácticamente cualquier tipo de dispositivo digital.

Realmente, lo que nos importa en el desarrollo son los tres primeros puntos, gracias a la maquina virtual de Java podemos hacer funcionar nuestras aplicaciones en cualquier estación de trabajo en la que este instalada. Es un punto de vista muy bueno si, en un futuro, se amplía el proyecto con nuevas funcionalidad y se lleva a cabo en otra plataforma.
Que sea totalmente compatible con UML es el punto a favor más importante, ya que, es el lenguaje de modelado que hemos aprendido a lo largo de la carrera.
En los últimos años casi todos los lenguajes de programación han migrado o han desarrollado una versión orientada a objetos, pero java nació siendo objetual.

Totalmente compatible con Weka, agrega la librería Weka a tu proyecto y accede rápidamente.

### 3.1.2 Weka

La librería estadística Weka es una colección de algoritmos de aprendizaje automático para minería de datos. Estos algoritmos pueden ser aplicados directamente a un dataset o se puede utilizar la librería Weka.jar para utilizarla directamente en tus aplicaciones Java. A demás en la instalación de Weka encontramos herramientas para pre-procesado, clasificación, regresión, agrupamiento, reglas de asociación y visualización. También nos ofrece la posibilidad de desarrollar nuevos patrones de aprendizaje automático y es software libre.

Weka nos interesa sobre todo la compatibilidad con Java, basta con agregar a nuestro proyecto la referencia a su librería y ya tenemos acceso a la API (Interfaz de Programación de Aplicaciones).

### 3.1.3 .NET de Microsoft

Del resto de lenguajes de programación orientado a objetos el que está de moda es el .Net de Microsoft.
Puntos fuertes de .NET y su entorno de desarrollo integrado Visual Studio:

1. Impleméntalo una vez y ejecútalo en cualquier máquina con .NET Framework instalado.
2. Tiene un editor de modelado pero no es compatible con UML.
3. Orientado a objetos.
4. Funcionalidades especificas para desarrollar software basado en web y servicios web (a partir del .NET 2.0): foros, tiendas virtuales, encuestas, HTML, etc.
5. Crear un web service nunca antes había más fácil y el rendimiento es muy interesante para plataformas distribuidas.
6. Desarrollar potentes aplicaciones de sobremesa y distribuidas.

El .Net es una plataforma de desarrollo de software que soporta varios lenguajes de programación e integra un CLR (Common Language Runtime) que viene a ser una maquina virtual de Microsoft, pudiendo ejecutar el programa en cualquier maquina que tenga instalado un .NET Framework.

A favor de .NET encontramos el Visual Studio .NET, una potentísima herramienta para crear aplicaciones de todo tipo, el IDE es mucho más amigable si lo comparamos con cualquier IDE de J2EE (Java 2 Enterprise Edition) y con una sensación de calidad de software extrema.

### 3.1.4 R

R, es un software gratuito para la computación estadística y de gráficos. Es multiplataforma, pero no tiene una librería directa para poder ser usada con Java, el único lenguaje de programación que puede crear objetos directamente es C.

Tiene varios clasificadores y podríamos llevar a cabo el proyecto con él, pero la incompatibilidad hace que no sea un candidato válido.

### 3.1.5 Entorno de desarrollo

El entorno de desarrollo que utilizaremos es el Netbeans IDE 6.1.

En Junio del 2000 NetBeans nació como un proyecto de software abierto de Sun Microsystems, hoy en día sigue siendo el sponsor del proyecto. La mayoría de los desarrolladores de software en Java reconocen a NetBeans como el primer IDE gratuito, pero desde NetBeans se afirma que no es solo eso, NetBeans también soporta varios lenguajes de programación (C, C++, JavaScript, etc.) y frameworks. El proyecto NetBeans es un proyecto de código abierto dedicado a proveer un entorno de desarrollo sólido para el desarrollo de aplicaciones, ofreciendo todo lo necesario para los desarrolladores. A favor de NetBeans está su gran comunidad donde todo el mundo puede dirigirse si tiene una duda o un problema para recibir soporte técnico, también puede hacerlo vía e-mail o visitando el FAQ. Para hacernos una idea de la popularidad de NetBeans sólo hace falta

ver las descargas que se han realizado del producto, mas de 18 millones, y con un grupo de más de 700.000 desarrolladores participando en el proyecto.

Para poder trabajar con UML en NetBeans es necesario instalar el Plug-in de UML que podemos encontrar en la pestaña de plugins en el mismo entorno de trabajo.

## 3.2 Metodología

Existen números estándares y modelos en los cuales nos podemos basar para desarrollar una aplicación, como Métrica 3, ISO 9000, etc. En nuestro caso podemos asegurar la calidad del proyecto realizando un proceso basado en la programación clásica o secuencial. Utilizaremos este proceso porque la empresa a realizar no está formada por un grupo de trabajo de varias personas ni se debe repartir la faena, al ser un único desarrollador, he creído conveniente usar este tipo de proceso (Figura 1).

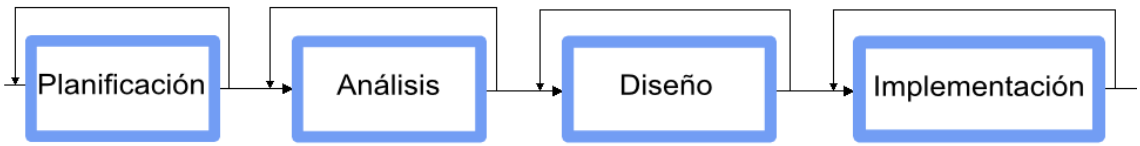



Figura 1: Proceso software clásico.

**Fase de planificación:**

Durante esta fase se llevará a cabo la planificación del proyecto:

Temporal: Se definirá el inicio, el fin y el trabajo a realizar durante cada semana intermedia.
Actividades: Se definirán las actividades a realizar en su acotación temporal.

**Fase de Análisis:**

Durante esta fase se llevará a cabo el estudio de viabilidad, costes, recursos y requisitos.

**Diseño:**

Se diseñará el aplicativo teniendo en cuenta los requisitos de la fase de análisis y los casos de uso.

**Implementación:**

En esta fase se llevará a cabo la construcción del software en el lenguaje de programación Java, será una fase guiada por la planificación, el análisis, el diseño y los casos de uso.

# 4 Planificación

La fase de planificación tiene como objetivo realizar un detallado uso del tiempo, los recursos y las actividades a realizar para evitar problemas organizativos, además pretende definir un marco de acción a seguir para cumplir las especificaciones temporales y de costes.

## 4.1 Planificación del proyecto

En el momento de realizar la planificación del proyecto nos encontramos con tres incertidumbres específicas que nos limitan para empezar la siguiente fase del proyecto. Estas incertidumbres son debidas al nulo conocimiento de la realización de juegos en una plataforma Java, la comunicación y la posibilidad de usar los clasificadores de la librería estadística Weka y si realmente podremos unir ambos conceptos para llegar a la meta de conseguir que el juego lo haga de manera automática.

El proyecto se planificará sobre 16 semanas, a razón de 4 horas al día en un horario establecido de Lunes a Viernes. El inicio del proyecto es el Lunes 14 de Septiembre de 2009 y la finalización el Lunes 18 de Enero de 2010 que se realizará la entrega de la documentación y del proyecto al tutor.

Semana 1 – semana 4

Durante las primeras cuatro semanas se instalará el software necesario para llevar a cabo la aplicación. NetBeans [2] y Weka [3].
Nos centraremos en aprender a crear un juego interactivo en 2D utilizando el lenguaje de programación Java. [4][5]

Semana 4 – semana 8

Durante las siguientes cuatro semanas recopilaremos los requerimientos necesarios para la parte grafica de la aplicación y llevaremos a cabo el diseño y la implementación del juego en una interfaz grafica de usuario de programación preparándolo para la posterior

implementación del modelo de juego automático. Usaremos archivos CSV para obtener la información sobre los escenarios y los movimientos.

Semana 8 – semana 10

En las dos semanas siguientes las destinaremos a entender el funcionamiento de librería estadística Weka [6] y a realizar en la aplicación de Weka pruebas referentes a la clasificación de los archivos CSV creadas.

Semana 10 – semana 13

Durante la decima semana comenzaremos la implementación de la comunicación con la librería Weka y estudiaremos el uso de los archivos ARFF. [7]

Semana 13

En la semana trece diseñaremos e implementaremos la interfaz grafica de usuario final y las configuraciones.

Semana 13-16

Durante las tres últimas semanas se realizaran tareas de afinado y sobretodo de documentación del aplicativo.

# 5 Análisis

Es imprescindible para cualquier proyecto analizar si realmente puede realizarse, pues únicamente nos espera el fracaso si a mediados de un proyecto nos damos cuenta que en realidad no era viable, o que los costes se disparan o que no existe realmente la tecnología para llevarlo a cabo. En nuestro caso realizaremos cuatro análisis: requisitos, recursos, costes, y viabilidad.

## 5.1 Análisis de requisitos

El requisito principal del proyecto es ofrecer al experimentador un entorno de trabajo con varios clasificadores y opciones sobre ellos, y además debe ser la base principal de su experimento y debe satisfacer sus necesidades. El análisis de requisitos estará dividido en dos partes, en la primera analizaremos los requisitos para desarrollar la plataforma básica de juego y en la segunda los de la conducción automática.

**Plataforma básica de juego. Juego de conducción:**

El juego de conducción constará de un escenario de cinco carriles por el que circularán vehículos a gran velocidad por los tres centrales y un vehículo controlado por el usuario que podrá circular por los cinco horizontalmente. Los vehículos que circulan por la autopista lo harán en un rango de velocidades de 6 a 9 m/s, mientras que el controlado por el jugador viajará a 12 m/s. El movimiento de todos los vehículos será vertical y no podrán efectuar ningún tipo de cambio de carril, excepto el del jugador, que sí podrá hacerlo.
El juego se ha de mostrar fluido y será precisa una opción que nos permita ralentizar el juego cuando deseemos.
El sistema debe ofrecer al usuario la opción de seleccionar un escenario creado aleatoriamente y uno creado de forma secuencial; además debe poder iniciar/pausar con la letra “P” y detener el juego en el momento que lo desee.
Para realizar la conducción del vehículo protagonista, el jugador debe utilizar las teclas de flecha derecha y flecha izquierda para poder cambiar de carril. En el caso que no pueda cambiar de carril porque ya ha llegado a un extremo de la autopista, el sistema debe

avisarle de que no puede realizar ese movimiento, mediante un mensaje que aparecerá en pantalla.
Si por alguna circunstancia el coche del jugador colisiona con otro vehículo se informará de la colisión de forma contundente, mediante un mensaje de alerta, y se contabilizará el número de colisiones. Así mismo, el otro vehículo colisionado desaparecerá de la vista. Los vehículos que circulan verticalmente no deben colisionar entre ellos, así pues, si uno de estos vehículos atrapa a otro debe regular su velocidad para evitar la colisión.
Siempre se debe mostrar en pantalla la puntuación, que dependerá de la velocidad del vehículo rebasado, el número de colisiones ocurridas, el tiempo total de juego y los *frames* por segundo.
A lo largo de la partida el juego creará varios archivos que describirán los movimientos realizados por el vehículo. El jugador podrá seleccionar la periodicidad con la que quiere que se crean estos informes, además de poder establecer la ruta dónde se archivarán. Al finalizar el juego también se mostrarán las estadísticas del juego realizado, en las que aparecerá, entre otra información, el tiempo de circulación del jugador por cualquier carril.
**Conducción automática:**

En este modo, el usuario deberá poder realizar todas las acciones descritas en el apartado anterior excepto la de conducir el vehículo, puesto que de esa tarea encarga automáticamente el sistema, mediante la interrogación del clasificador seleccionado.
Los clasificadores que podemos seleccionar son: BayesNet y NaiveBayes, J48, J48graft y NBTree. También tenemos que tener en cuenta la configuración de hacerle caso a un movimiento predicho por el clasificador y en qué medida.

## 5.2 Análisis de recursos

En este apartado realizaremos un estudio de los recursos, tanto tecnológicos como humanos, necesarios para llevara a cabo la empresa.

**Recursos tecnológicos:**

Para el desarrollo del proyecto hemos tenido en cuenta varias herramientas de software libre que realmente nos aporte el valor añadido necesario para realizar el proyecto.

- Para realizar toda la programación y los diseños en UML utilizaremos el IDE gratuito estudiado durante la carrera Netbeans [2].
- La librería estadística Weka [3].
- La librería JDom [10] para la gestión de archivos XML.
- Para los gráficos bidimensionales utilizaremos dos herramientas de software libre que he estudiado este último cuatrimestre. Gimp [8] para la modificación de imágenes e Inkscape [9] para la creación de nuevos gráficos.
- La suite Microsoft Office 2007 [11].
- Un ordenador personal que cumpla los requisitos del software anteriormente mencionado.

**Recursos de personal:**

- Un analista/programador.

## 5.3 Análisis de costes

En el análisis de costes analizaremos los costes de software, hardware y de personal. No tendremos en cuenta ni los costes de mantenimiento ni los de infraestructuras (electricidad, agua, alquiler, impuestos, etc.), es decir, los generales. Tampoco tendremos en cuenta los costes de formación, ya que, no se llevara a cabo ninguna formación ni costes de funcionamiento.

**Costes de hardware nuevo:**

Figura 2: Ordenador Personal Inspiron 545

- Procesador Intel® Celeron® 450 (2,2 GHz, FSB a 800 MHz, caché de 512k)
- Windows® 7 Home Premium original 64bit - Español
- Monitor Dell IN1910N 18.5" HD Widescreen
- Intel® Graphics Media Accelerator 3100 integrada
- Memoria DDR2 de doble canal a 800 MHz de 2.048 MB [2 x 1.024]
- Disco duro SATA de 320 GB (7.200 rpm)
- Unidad DVD +/- RW (lee/escribe CD y DVD)

Total coste hardware: 399 €.

Vida útil del producto: 2 años.

Coste semanal: 3.84 €/Semana.

**Costes de software nuevo:**


Figura 3: Microsoft Office Hogar y Estudiantes.

- Suite Microsoft Office Hogar y Estudiantes (99€).
- NetBeans (gratuito).

- Gimp (gratuito).
- Inkscape (gratuito).
- Weka (gratuito).
- JDom (gratuito).

Total coste hardware: 99 €.
Vida útil del producto: 2 años.
Coste semanal: 0.95€/Semana.
**Costes de personal:**

- Un analista/programador (12€/hora)

Total coste de personal: 3840 €.
Coste semanal: 240 €.

**Total costes:**

| **Tipo de coste** | **Precio** | **Cantidad** | **Subtotal** |
|---|---|---|---|
| **Hardware** | 3.84 € | 16 | 61.44 € |
| **Software** | 0.95 € | 16 | 15.2 € |
| **Personal** | 240 € | 16 | 3840 € |
| | | **Total:** | **3916.64 €** |

Tabla 1: Total costes

Los costes de desarrollo del proyecto ascienden a 3916.64€, si el proyecto fuese la petición de un cliente, deberíamos imputarle el coste mas el margen, además de la parte proporcional de los gastos de la empresa.

## 5.3 Análisis de viabilidad

En este apartado estudiaremos la viabilidad técnica y de fechas. No tendremos en cuenta la viabilidad operativa, ya que, la resolución del problema es el esfuerzo a tratar en este proyecto; ni la viabilidad económica.

**Viabilidad técnica:**

Después de analizar el problema y estudiar las funcionalidades y las restricciones del proyecto en el apartado técnico, llegamos a las siguientes conclusiones:

- Técnicamente es posible crear un juego en dos dimensiones que se comunique con la librería Weka y que realice grabaciones en memoria persistente.
- La plantilla disponible para realizar el proyecto es apta para enfrentarse al desafío.
- Tenemos los recursos de hardware y software disponibles para el desarrollo del proyecto.
- La tecnología actual está capacitada para soportar el software a desarrollar.

El proyecto se declara técnicamente viable.

**Viabilidad de fechas:**

Tras analizar el plazo de entrega se llega a la siguiente conclusión:

- El plazo de entrega es razonable para el desarrollo del proyecto.

El proyecto se declara viable en plazo de entrega.

# 6 Casos de Uso

## 6.1 Especificación de los casos de uso

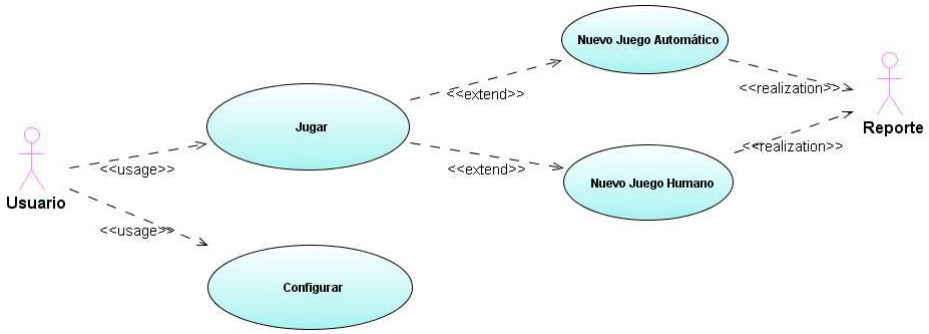



Figura 4: Diagrama de casos de uso

Como podemos observar en la (Figura 4) los casos de uso a tratar son los siguientes:

- Jugar, que nos da acceso a Nuevo Juego Automático y a Nuevo Juego Humano.
- Nuevo Juego Automático que al finalizar realiza un reporte.
- Nuevo Juego Humano que al finalizar realiza un reporte.
- Configurar.

### 6.1.1 Actores principales

El único actor principal es el usuario, a la interrogación del clasificador de la librería Weka no podemos tratarlo como un actor porque realmente no accede a ningún caso de uso ni es la salida del mismo.

El Reporte no debe confundirse con un actor, aunque tenga el mismo icono en la (Figura 4), en realidad es una salida de información.

### 6.1.2 Casos de uso

**Caso de uso Jugar:**

- **Objetivos asociados:** Su función principal es dar acceso al caso de uso *Nuevo Juego Automático* y *Nuevo Juego Humano.*
- **Descripción:** El usuario realizará este caso de uso para acceder al caso de uso *Nuevo Juego Automático* y *Nuevo Juego Humano*, pero podrá salir del mismo cuando lo desee. El usuario puede finalizar el caso de uso *Jugar* en cualquier momento.
- **Actor principal:** Usuario.
- **Precondiciones:** El usuario debe haber iniciado el sistema.
- **Secuencia normal:**
    1. El usuario ejecuta el caso de uso *Jugar*.
    2. El sistema le muestra un formulario desde donde puede acceder al caso de uso *Nuevo Juego Automático* y *Nuevo Juego Humano.*
    3. El usuario finaliza el caso de uso *Jugar*.
- **Secuencia alternativa:**

  3.1 El usuario accede al caso de uso *Nuevo Juego Humano.*

  3.1.1 El sistema ejecuta el caso de uso *Nuevo Juego Humano.*

  3.2 El usuario accede al caso de uso *Nuevo Juego Automático.*

  3.1.1 El sistema ejecuta el caso de uso *Nuevo Juego Automático.*
- **Pos condiciones:**
    - Ninguna.

**Caso de uso Nuevo Juego Humano:**

- **Objetivos asociados:** Su función principal es iniciar la plataforma básica de juego en modo manual.
- **Descripción:** El usuario realizará este caso de uso para iniciar la plataforma básica de juego en modo manual. El usuario puede finalizar el caso de uso *Nuevo Juego Humano* en cualquier momento. El sistema debe utilizar la configuración predeterminada o la guardad por el usuario.
- **Actor principal:** Usuario.
- **Precondiciones:** El usuario debe haber accedido mediante el caso de uso *Jugar*.

- **Secuencia normal:**
    1. El usuario ejecuta el caso de uso *Nuevo Juego Humano*.
    2. El sistema inicia la plataforma básica de juego en modo manual.
- **Secuencia alternativa:**
    - Ninguna.
- **Pos condiciones:**
    - Ninguna.

**Caso de uso Nuevo Juego Automático:**

- **Objetivos asociados:** Su función principal es iniciar la plataforma básica de juego en modo automático.
- **Descripción:** El usuario realizará este caso de uso para iniciar la plataforma básica de juego en modo automático. El usuario puede finalizar el caso de uso *Nuevo Juego Automático* en cualquier momento. El sistema debe utilizar la configuración predeterminada o la guardad por el usuario.
- **Actor principal:** Usuario.
- **Precondiciones:** El usuario debe haber accedido mediante el caso de uso *Jugar*.
- **Secuencia normal:**
    1. El usuario ejecuta el caso de uso *Nuevo Juego Automático*.
    2. El sistema le muestra un formulario para que seleccione el archivo ARFF a utilizar.
    3. El usuario selecciona el archivo ARFF.
    4. El sistema inicia la plataforma básica de juego en modo automático.
- **Secuencia alternativa:**

    3.1 El usuario selecciona un archivo ARFF no válido.

    3.1.1 El sistema le mostrará el mensaje de error y volverá al caso *Jugar*.

    3.2 El usuario selecciona un archivo que no es del tipo ARFF.

    3.2.1 El sistema le mostrará el mensaje de error y volverá al caso *Jugar*.
- **Pos condiciones:** Ninguna.

**Caso de uso Configurar:**

- **Objetivos asociados:** Su función principal es gestionar la configuración.
- **Descripción:** El usuario realizará este caso de uso para configurar la aplicación con los valores deseados.
- **Actor principal:** Usuario.
- **Precondiciones:** El usuario debe haber iniciado el sistema.
- **Secuencia normal:**
    1. El usuario ejecuta el caso de uso *Configurar*.
    2. El sistema le muestra un formulario con los valores de la configuración almacenada para que seleccione e introduzca valores de distintos tipos.
    3. El usuario introduce los valores.
    4. El usuario finaliza el caso de uso *Configurar*.
    5. El sistema graba la configuración actual.
- **Secuencia alternativa:**

  3.1 El usuario introduce un valor no válido en cualquier campo.

  3.1.1 El sistema le mostrará el mensaje de error y volverá al punto 3.
- **Pos condiciones:** Ninguna.

# 7 Guionaje multimedia

El guionaje de la parte multimedia, es muy sencilla en este proyecto, ya que, el objetivo principal no es el de diseñar un aplicativo atractivo para el usuario y que lo sumerja en un juego adictivo, en realidad, la idea de desarrollar el juego es una excusa para poder tener un aspecto visual y entender mejor como se resuelve la problemática decisional, así como, encontrar patrones de comportamiento visualmente.

**Objetivos del producto**

El producto es la parte visual e interactiva del juego, el objetivo es darle al usuario una insterfaz de acción visual para llevar a cabo el experimento decisional sin dejar de lado la atracción e inmersión del mismo.

**Unidad de contenido 1: Juego de coches.**

Una vez iniciado el juego, se mostrará el escenario que consiste en una autopista de tres carriles con dos carriles adicionales que tienen el rol de arcén, por este arcén, solamente podrá circular el vehículo del usuario.

**Análisis media.**

- Texto:

El texto es un componente informativo en la unidad de contenido, nos informa de la puntuación que llevamos, cuántas veces hemos colisionado o del tiempo de juego entre otras cosas.

- Imagen Dinámica:

Las animaciones y las imágenes dinámicas con el medio que se pretende utilizar en esta unidad de contenido para llevar a cabo la escenografía del juego.

- Audio:

Este juego carecerá de sonido o audio.

**Ubicación en el mapa multimedia / Contenidos.**

| | | **Juego de coches** |
|---|---|---|
| **Audio** | **Imagen** | Imágenes en movimiento solamente. |
| | **Sonido** | |
| | **Movimiento** | Animación en dos dimensiones. |
| **Edición** | **Texto** | Información sobre los contenidos. |
| | **Quieto** | El texto será dinámico. |
| | **Mudo** | |

Tabla 2: Ubicación en el mapa multimedia / Contenidos

**Interacción**

| | **Juego de coches** |
|---|---|
| **Acceso** | El acceso es secuencial. |
| **Orientación** | Estructurada |
| **Temporalidad** | Inanimada |
| **Grado de interactividad** | Participativo (3) |
| **Tipo de atención** | Cognitiva |
| **Volatilidad** | Off line |

Tabla 3: Interacción

**Graf Exhaustivo**

La parte multimedia del aplicativo contará de un escenario de cinco carriles por el que circularán vehículos a gran velocidad por los tres centrales y un vehículo controlado externamente que podrá circular por los cinco horizontalmente. El escenario simulará una autopista que da al mar en su parte izquierda y a la montaña en la derecha. Podemos encontrar un boceto visual en la (Figura 5), los vehículos que circulan por la autopista lo harán en un rango de velocidades de 6 a 9 m/s, mientras que el controlado externamente por el jugador viajará a 12 m/s. El movimiento de todos los vehículos será vertical y no

podrán efectuar ningún tipo de cambio de carril excepto el del jugador que será únicamente horizontal.

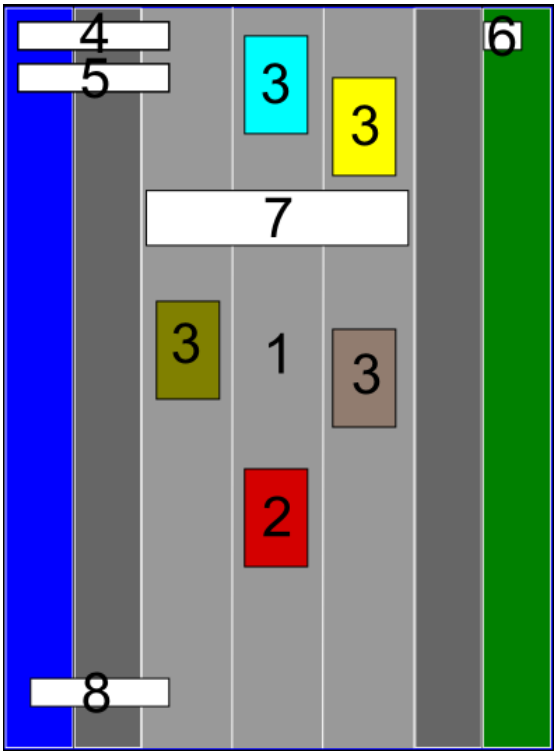



Figura 5: Concepto visual del juego de coches.

| EID | Tipo | Descripción |
|---|---|---|
| **1** | IMG | Imagen de fondo utilizada para dar aspecto de movimiento. |
| **2** | IMG | Imagen del vehículo controlado por el usuario, o en caso de la conducción automática, por el PC. |
| **3** | IMG | Imágenes de los vehículos controlados por el ordenador. |
| **4** | TXT | Texto, número de colisiones, tiene una parte fija: “Número de colisiones:” y otra dinámica, que nos indicará el número en si. |
| **5** | TXT | Texto, tiempo total de juego, tiene una parte fija: “Tiempo de juego:” y otra dinámica que nos indicará el tiempo que llevamos jugando en formato mm:ss. |
| **6** | TXT | Texto dinámico, nos informará del número de “frames” por segundo. |
| **7** | TXT | Texto dinámico, nos informará de los eventos especiales, entiéndase estos como información hacia el usuario de un movimiento no permitido o una colisión. |
| **8** | TXT | Texto, puntuación, tiene una parte fija: “Puntuación:” y otra dinámica que nos indicará los puntos acumulados desde el inicio del juego. |

Tabla 4: Descripción de los objetos en el concepto visual del juego de coches

# 8 Diseño de la aplicación

La aplicación en un principio ha sido diseñada para implementar el patrón Modelo-Vista-Controlador, pero al final decidí que haría una excepción con la clase Escenario, el motivo principal de la excepción es el rendimiento grafico y que no será una aplicación distribuida.

## 8.1 Diseño de la capa Interficie

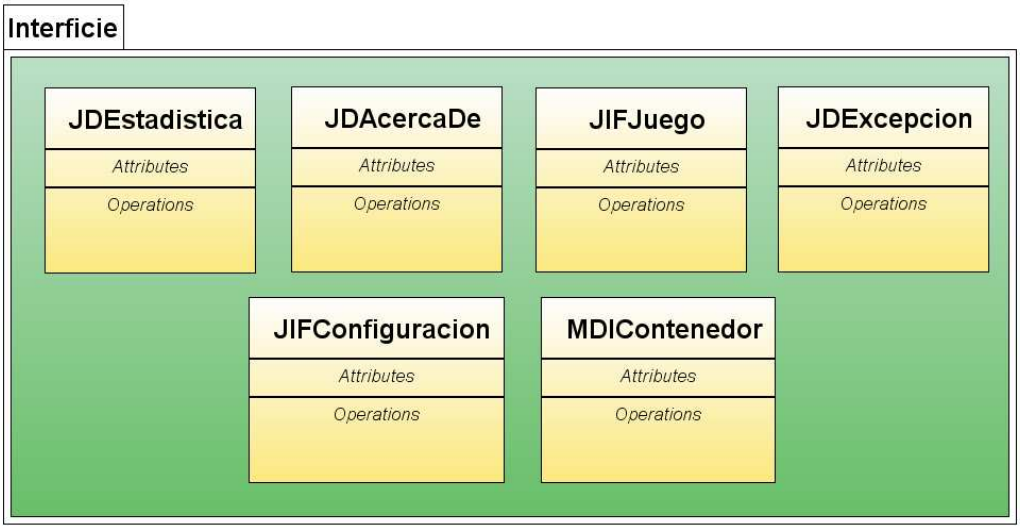



Figura 6: Vista de diseño de la capa Interficie.

**JDAcercaDe:**

La clase JDAcercaDe nos mostrará una breve información sobre el autor del proyecto.

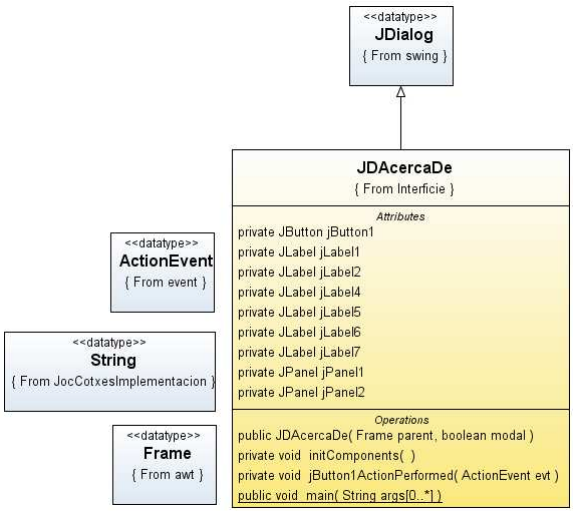



Figura 7: Acoplamientos de la clase JDAcercaDe.

JDAcercaDe no tiene ningún tipo de acoplamiento con el resto del proyecto, únicamente es instanciada por la clase MDIContenedor cuando el usuario accede a la opción de menú AcercaDe.

**JDExcepcion:**

La clase JDExcepcion mostrará al usuario los errores producidos en la aplicación.

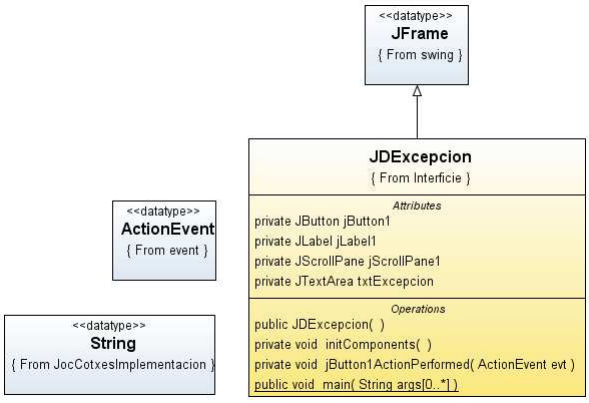



Figura 8: Acoplamientos de la clase JDExcepcion.

JDEExcepcion no se acopla con ninguna otra clase de la capa interficie (Figura 8), únicamente es llamada cuando se ha producido una excepción reportada por el controlador de la aplicación.

**MDIContenedor:**

La clase MDIContenedor mostrará el menú de acceso al usuario y los formularios.

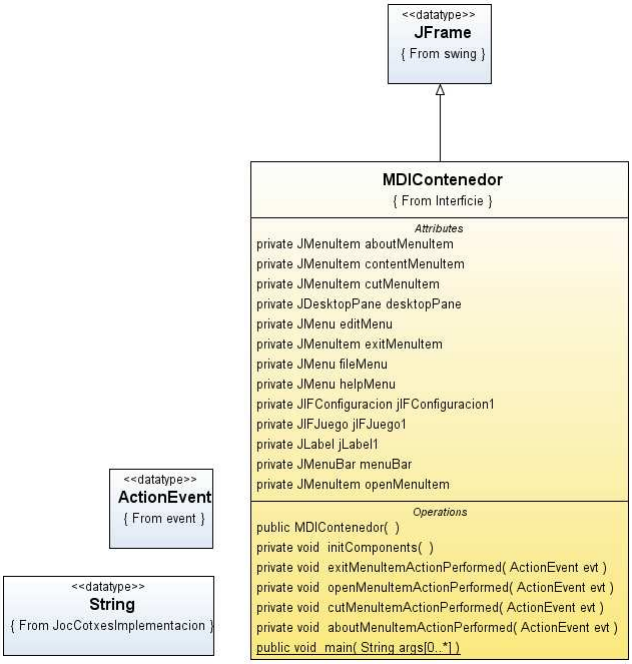



Figura 9: Acoplamientos de la clase MDIContenedor.

MDIContenedor no se acopla con ninguna otra clase de la capa interficie (Figura 9), el proyecto está configurado para que el "main" a ejecutarse al inicio del programa sea esta clase.

**JIFConfiguracion:**

La clase JIFConfiguracion será la encargada de informar al usuario sobre la configuración que se está utilizando en ese instante, así como de informar al controlador de la aplicación sobre cualquier modificación en la misma.

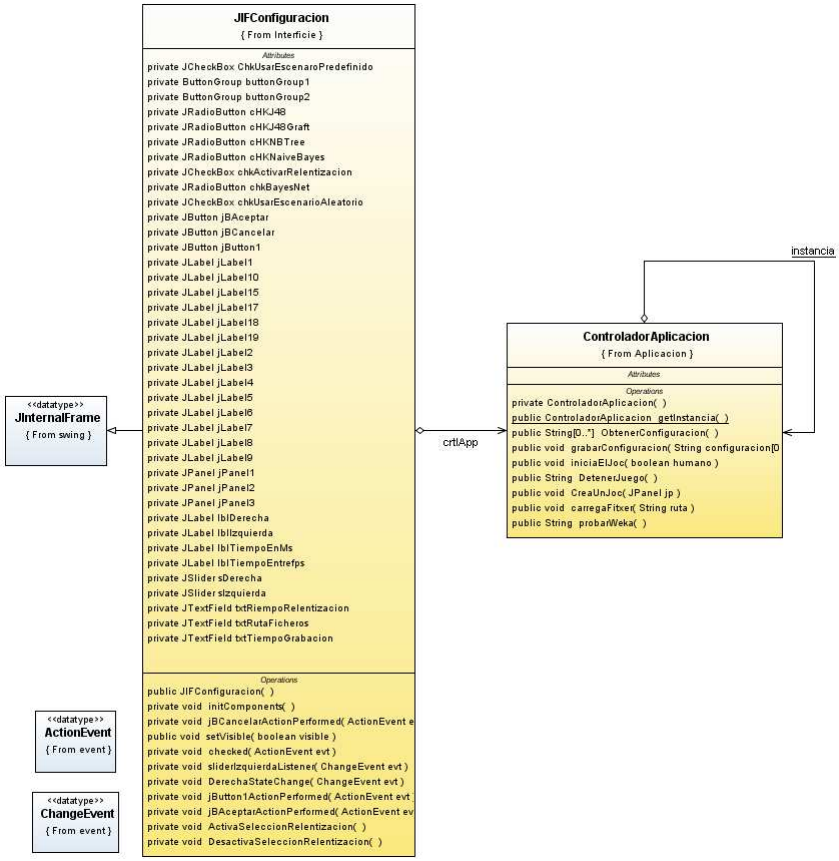



Figura 10: Acoplamientos de la clase MDIConfiguracion.

JIFConfiguracion se acopla con la clase ControladorAplicacion de la capa Aplicación (Figura 10). El atributo artlApp no es realmente necesario, ya que, ControladorAplicacion implememenata el patrón singleton y sólo es accesible mediante la operación getInstnacia(). ObtenerConfiguracion es la operación que suministra a JIFConfiguracion un array de Strings con la configuración de la aplicación.
GrabaConfiguracion es la operación utilizada para grabar la configuración deseada por el usuario, si se produce algún error al revisar los datos, estos no se guardarán.

**JIFJuego:**

La clase JIFJuego mostrará al usuario el panel de juego y las distintas acciones que puede realizar.

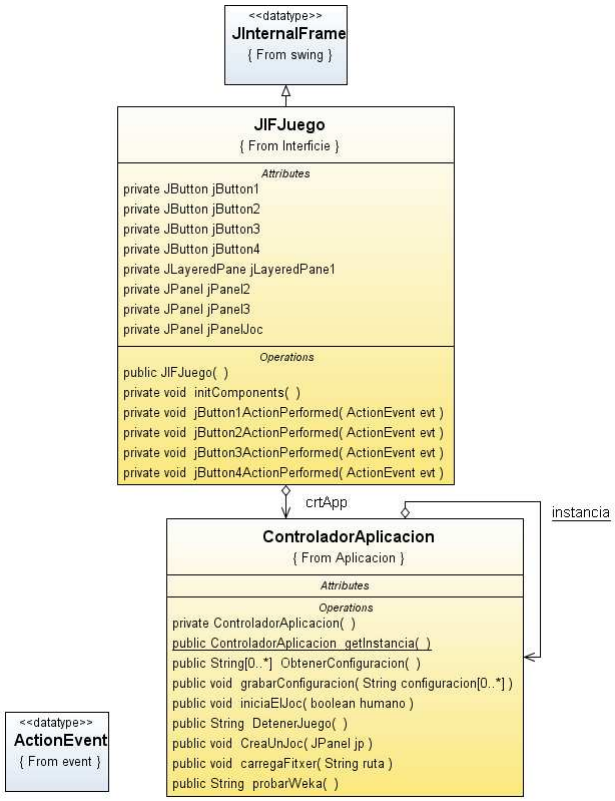



Figura 11: Acoplamientos de la clase JIFJuego.

JIFJuego se acopla con la clase ControladorAplicacion de la capa Aplicación (Figura 11). Es del tipo JInternalFrame para poder ser dibujado dentro de la clase MDIContenedor, con respecto al ControladorAplicacion utiliza las operaciones CreaUnJoc(JPanel jp) pasándole como parámetro el JPanel jPanelJoc, que es donde se dibujará el juego de coches. También usa la operación iniciaElJoc(boolean humano) pasándole por parámetro si el juego se ha de iniciar en modo Manual o automatico para comenzar el juego.

**JDEstadistica:**

La clase JDEstadistica mostrará al usuario información sobre la ejecución de un juego.

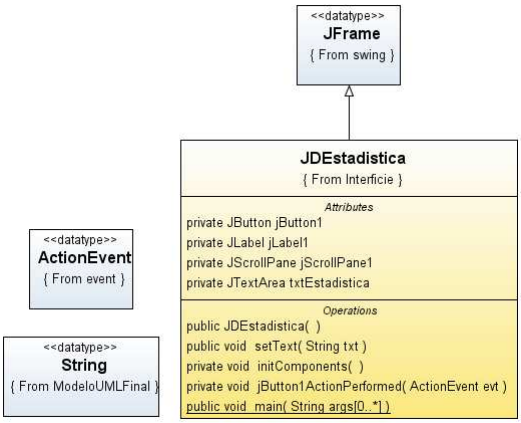



Figura 12: Acoplamientos de la clase JDEstadistica.

JDEstadistica no se acopla con ninguna otra clase (Figura 12), únicamente es instanciada por la clase JIFJuego para mostrar las estadísticas del juego una vez ha concluido.

## 8.2 Diseño de la capa Aplicación


Figura 13: Vista de diseño de la capa Aplicación.

**ControladorAplicacion:**

La clase ControladorAplicacion será la encargada de la comunicación entre la interfaz de usuario, es decir, la capa Interficie, y la aplicación, excepto en el caso de la clase Escenario de la capa Dominio.

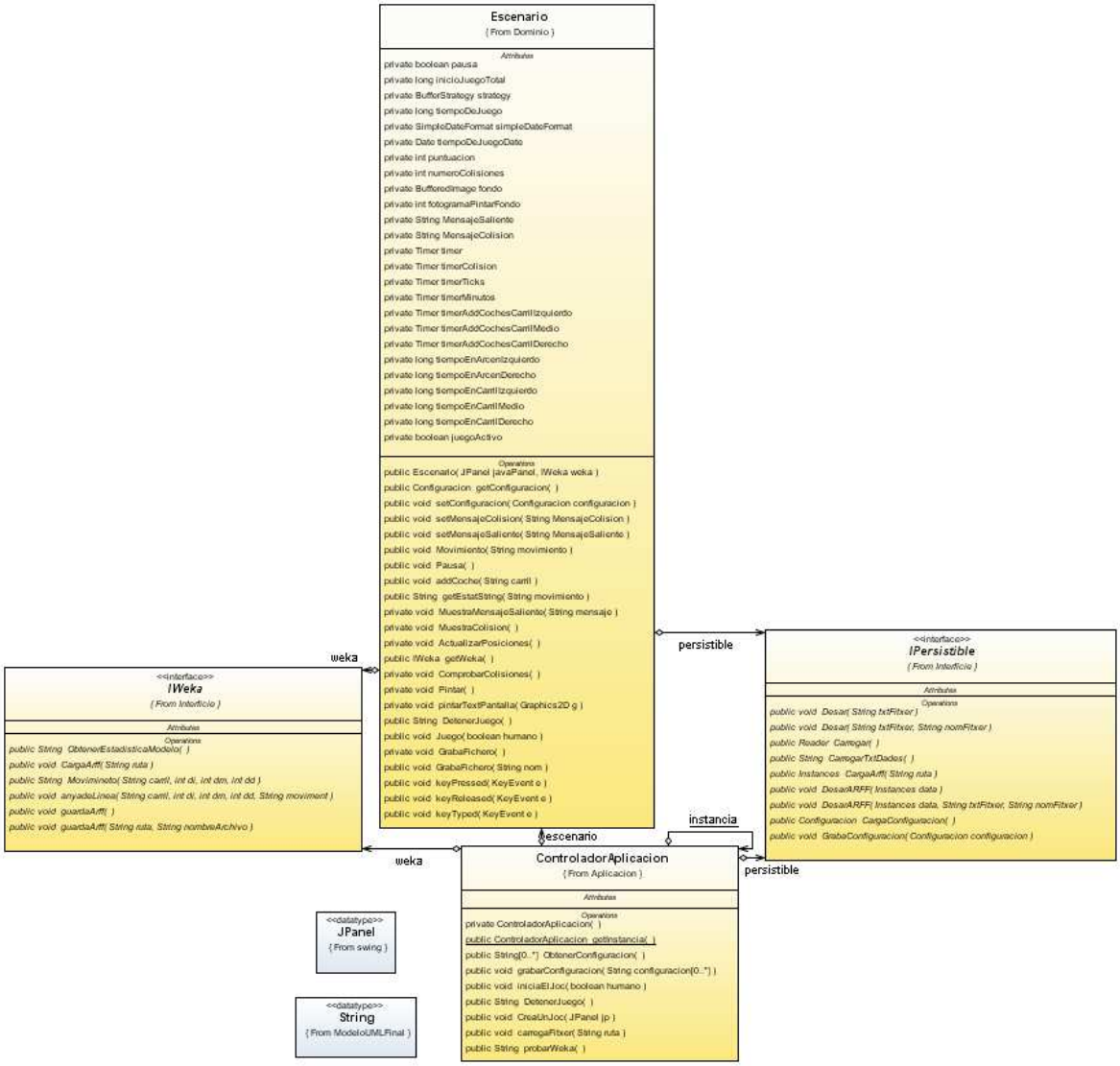



Figura 14: Acoplamientos de la clase ControladorAplicacion.

ControladorAplicacion se acoplará a la clase Escenario de la capa Dominio y al adaptadorWeka de la misma capa a través de la interfaz IWeka, también se acopla a la capa Persistencia mediante la interfaz IPersistible (Figura 14) y será utilizado por la capa Interficie para la comunicación con la aplicación, cabe decir que no es necesario utilizar la interfaz IPersistible pero de ese modo ganamos homogeneidad con el resto del software. Implementará el patrón de diseño singleton para obtener una única instancia.

## 8.3 Diseño de la capa Dominio.Interficie

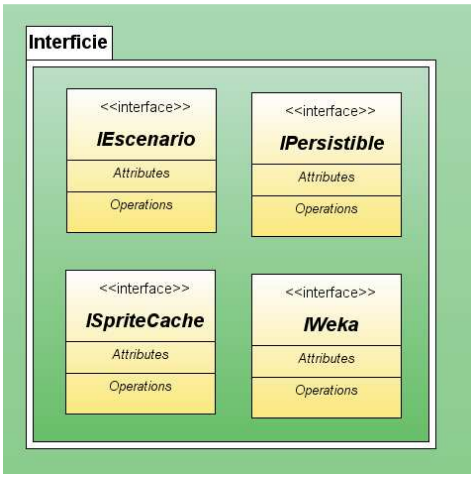



Figura 15: Vista de diseño de la capa Dominio.Interficie.

**IEscenario:**

La interfaz IEscenario está pensada para contener atributos finales con información sobre todo de dimensiones, ratings de actualización y atributos que nos sean de utilidad para la posterior implementación del software.

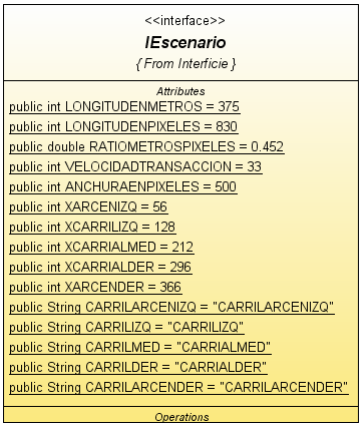



Figura 16: IEscenario.

**IPersistible:**

La interfaz IPersistible definirá aquellas operaciones que tengan que acceder a memoria persistente. La clase Escenario instanciará la clase Persistible de la capa persistencia

declarándola del tipo IPersistible, así conseguiremos que el Dominio no se acople con la persistencia.

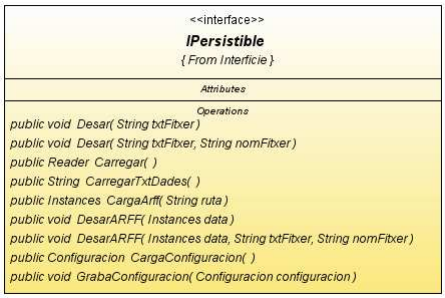



Figura 17: IPersistible.

**ISpriteCache:**

La interfaz ISpriteCache definirá aquellas operaciones que tengan que acceder a memoria persistente única y exclusivamente para acceder a imágenes. La clase Escenario instanciará la clase SpriteCache de la capa persistencia declarándola del tipo ISpriteCache, así conseguiremos que el Dominio no se acople con la persistencia tampoco en este caso.

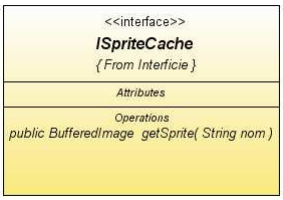



Figura 18: ISpriteCache.

**IWeka:**

La interfaz IWeka definirá aquellas operaciones necesarias para la comunicación con la librería estadística Weka.

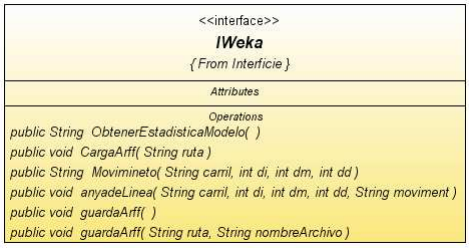



Figura 19: IWeka.

## 8.4 Diseño de la capa Dominio

La capa Dominio albergará toda la lógica objetual de la aplicación, como podemos observar en la Figura 20, nos describe el funcionamiento conceptual de la aplicación. En este caso hemos optado por cohesionar las interfaces dentro de una subcapa.

Para solucionar el problema de la comunicación con un agente externo de funcionamiento desconocido se ha optado por agrupar de forma lógica las operaciones que vamos a utilizar dejando el problema de la implementación para la clase que implementará esta interfaz, como es el caso de IWeka con WekaAdapter. Implementará el patrón adaptador.

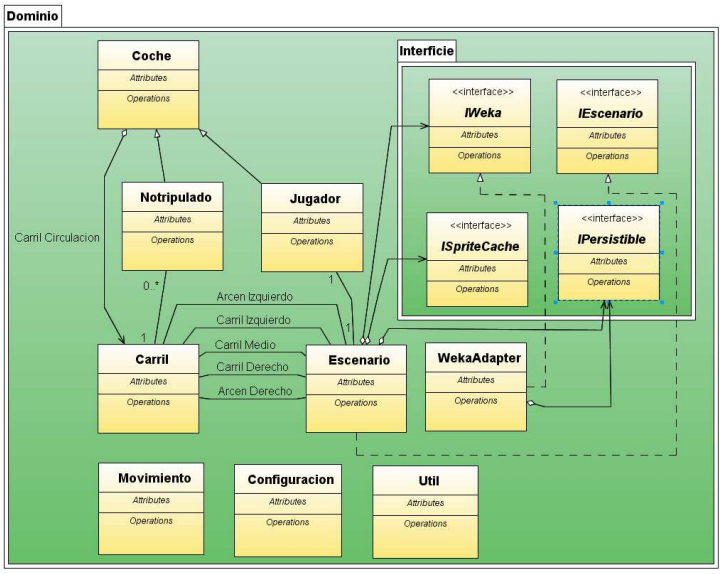



Figura 20: Diseño de la capa Dominio.

**Coche:**

La clase Coche definirá los atributos y operaciones necesarias para crear y mostrar un coche en el escenario. En la (Figura 20) podemos observar que todo coche tiene un carril por donde va a circular.

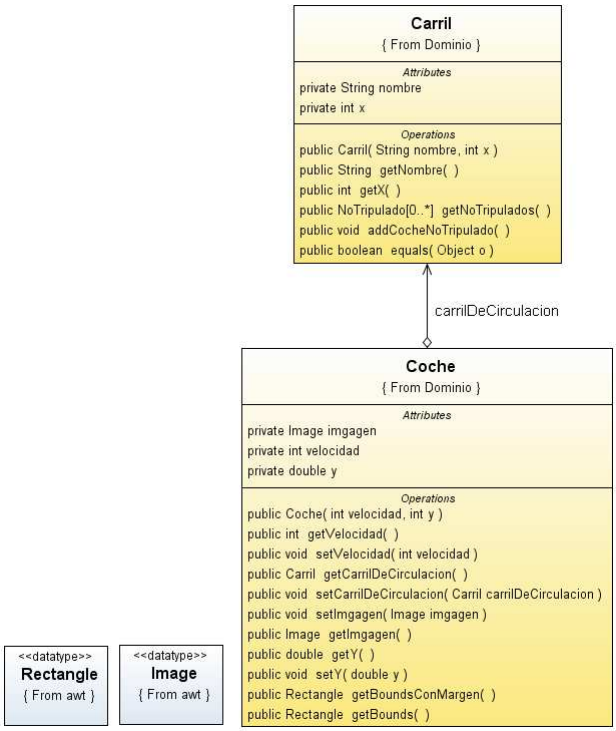



Figura 21: Implementación de la clase Coche.

En la implementación de la clase coche (Figura 21) destacamos los atributos imagen, y, velocidad y carrilDeCirculación. Los atributos de tipo de datos no primitivos que utiliza esta clase son Image y Carril.

Las operaciones getBoundsConMargen() y getBounds() se utilizan para obtener un rectángulo situado en un eje de coordenadas, es decir, la posición, altura y anchura del coche. El tipo de dato retornado es de la clase Rectangle.

**NoTripulado:**

Es una especialización de la clase Coche, esta clase representará a los coches que circularán por la autopista y que el usuario no puede controlar.

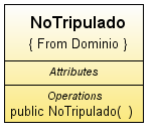



Figura 22: Implementación de la clase NoTripulado.

La única acción realizada en la implementación de la clase NoTripulado ha sido la redefinición de su constructor. (Figura 22)

**Jugador:**

Es la otra especialización de la clase Coche, esta clase representará al coche que controlará el usuario para jugar, en la segunda parte de la aplicación será controlada por la máquina para jugar en modo automático.


Figura 23: Implementación de la clase Jugador.

La única acción realizada en la implementación de la clase Jugador ha sido la redefinición de su constructor. (Figura 23)

**Movimiento:**

La clase Movimiento definirá los tres tipos de movimiento que el Jugador puede realizar: izquierda, derecha o recto.

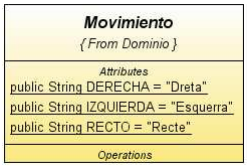



Figura 24: Implementación de la clase Movimiento.

Los atributos finales de la clase Movimiento nos informa con un String del movimiento. Accediendo a este String del modo: Movimiento.IZQUIERDA por ejemeplo.

**Configuración:**

La clase Configuracion mantendrá la configuración de la aplicación personalizable por el usuario.

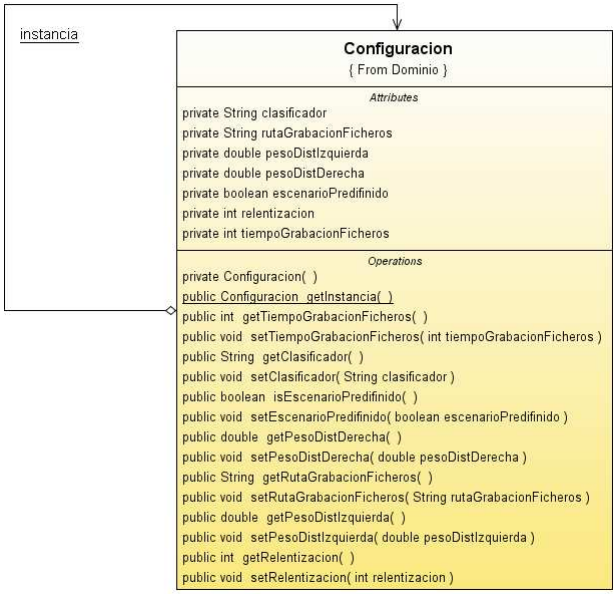



Figura 25: Implementación de la clase Configuracion.

En la implementación de la clase Configuracion (Figura 25) podemos observar los atributos configurables por el aplicativo: clasificador, rutaGrabacionFicheros, pesoDistIzquierda, pesoDistDerecha, escenarioPredefinido, relentizacion y tiempoGrabacionFicheros. Implementa el patrón singleton y el constructor inicializa valores por defecto. Todos los tipos de dato son primitivos.

**Util:**

La clase Util facilitará cualquier implementación basada en una abstracción lógica. Implementará operaciones accesibles desde cualquier punto de la aplicación excepto la interfaz de usuario.

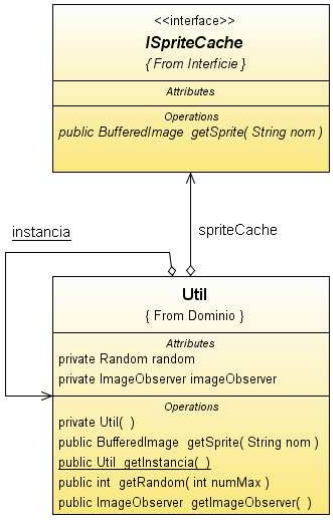



Figura 26: Implementación de la clase Util.

En la implementación de la clase Util (Figura 26) podemos observar los atributos spriteCache, random e imageObserver. Implementa el patrón singleton. Los atributos de tipo de datos no primitivos que utiliza esta clase son ImageObserver, Random e ISpriteCache.

La operación getSprite(String nom) es un punto común de acceso a la clase SpriteCache a través de su interfaz.
La operación getRandom(int numMax) nos devuelve un int primitivo entre 0 y el número pasado por parámetro. La clase Util se encarga de la inicialización aleatoria del objeto y se implementa como un acceso común a la clase Random.
La operación getImageObserver() mantiene un único observador de imágenes para todo el dominio.

**WekaAdapter:**

Implementará el patrón adaptador para que las implementaciones con la librería estadística Weka sea transparente al resto de la aplicación.

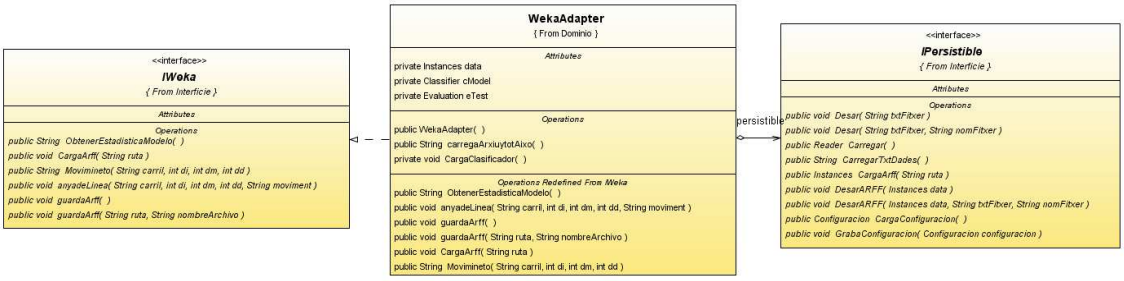



Figura 27: Implementación de la clase WekaAdapter.

En la Implementación de la clase WekaAdapter (Figura 27) podemos observar los atributos data, cModel y eTest. Los atributos de tipo de datos no primitivos que utiliza esta clase son Instances, Classifier y Evaluation. Estos tipos de datos están definidos en la librería estadística Weka.

Las operaciones públicas que no están definidas por la interfaz IWeka son operaciones de pruebas con un ejemplo de todos los pasos necesarios para tratar con la librería.

La operación ObtenerEstadisticaModel() nos devuelve datos estadísticos del modelo utilizado por el adaptador, así como, estadísticas del entrenamiento realizado.
La operación anyadeLinea(String carril, int di, int dm, int dd, String moviment) añade una nueva Instance al atributo data, para que posteriormente se tenga en cuenta en el entrenamiento.
La operaciones guardaArff(), guardaArff(String ruta, String nombreArchivo) y CargaArff(String ruta) son las implementadas para guardar y cargar de memoria persistente la información del atributo data.
La operación Movimiento(String carril, int di, int dm, int dd) devuelve un String con el movimiento a realizar según los datos pasados por parámetro. El parámetro di se refiere a la distancia del vechículo mas cercano del carril izquierdo, dm del carril central y dd del derecho.

**Escenario:**

La clase escenario se acoplará como componente del panel donde se dibujará, además implementará la interfaz IEscenario y KeyListener.

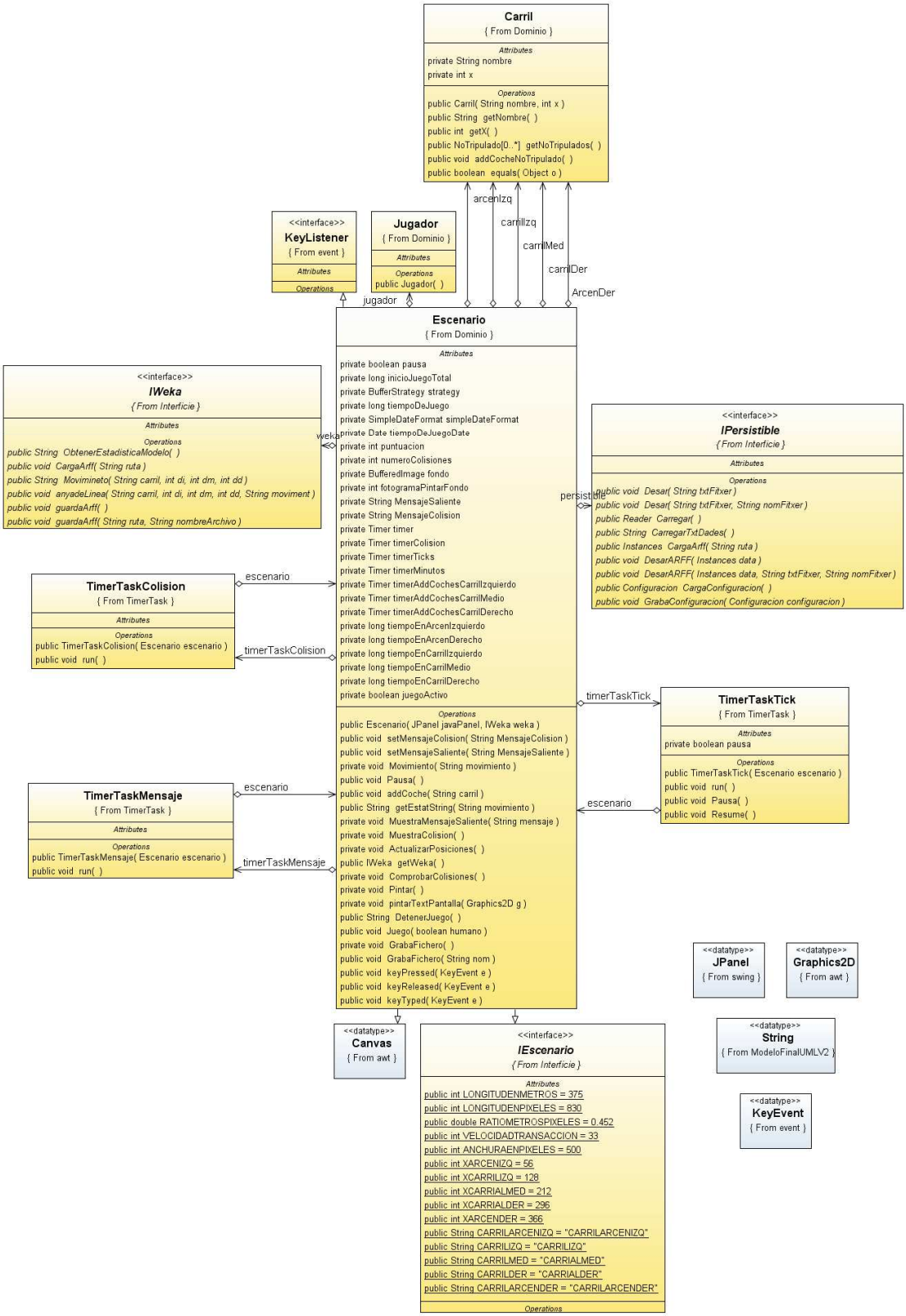



Figura 28: Implementación de la clase Escenario.

En la implementación de la clase Escenario (Figura 28) podemos observar un gran número de atributos, esto es debido a que, al diseñar la aplicación para que el Dominio pueda

pintarse sobre un Jpanel, la clase Escenario también implementa la problemática grafica. En la (Tabla 5) podemos observar una descripción mas detallada de los atributos.

| Nombre del atributo | Tipo de datos | Utilización |
|---|---|---|
| jugador | Jugador | El coche de rol jugador |
| arcenIzq | Carril | Arcen izquierdo |
| carrilIzq | Carril | Carril izquierdo |
| carrilMed | Carril | Carril del medio |
| carrilDer | Carril | Carril derecho |
| ArcenDer | Carril | Arcen derecho |
| pausa | boolean | Indica si el juego esta pausado |
| inicioJuegoTotal | long | Instante del inicio del juego tiempo expresado en milisegundos |
| strategy | BufferStrategy | Objeto utilizado para la técnica del doble buffer |
| tiempoDeJuego | long | Tiempo de juego acumulado en milisegundos |
| simpleDateFormat | SimpleDateFormat | Objeto utilizado para dar formato a las fechas o horas |
| tiempoDeJuegoDate | Date | Tiempo de juego acumulado en formato Fecha |
| puntuacion | int | Puntuación actual de la partida |
| numeroColisiones | int | Número de colisiones de la partida |
| fondo | BufferedImage | Imagen que hace de fondo |
| fotogramaPintarFondo | int | Numero de fotograma para pintar el fonde de nuevo creando un efecto persiana |
| MensajeSaliente | String | Mensaje saliente de información para el usuario |
| MensajeColision | String | Mensaje de colisión |
| timer | Timer | Timer para controlar el tiempo de emisión de los mensajes en pantalla |
| timerColision | Timer | Timer para controlar el tiempo de emisión del mensaje de colisión en pantalla |
| timerTicks | Timer | Timer para controlar el tiempo de ejecución de la tarea TimerTaskTick |
| timerMinutos | Timer | Timer para controlar el tiempo de ejecución de la tarea TimerTaskGrabacionFicheros |
| timerAddCochesCarrilIzquierdo | Timer | Timer para controlar el tiempo de ejecución de la tarea TimerTaskAddCochesCarrilIzquierdo |
| timerAddCochesCarrilMedio | Timer | Timer para controlar el tiempo de ejecución de la tarea TimerTaskAddCochesCarrilMedio |
| timerAddCochesCarrilDerecho | Timer | Timer para controlar el tiempo de ejecución de la tarea TimerTaskAddCochesCarrilDerecho |
| persistible | IPersistible | Instancia de la clase Persistible declarada como Ipersistible para grabar los ficheros ARFF |
| timerTaskMensaje | TimerTaskMensaje | Tarea que reinicia el mensaje saliente |
| timerTaskColision | TimerTaskColision | Tarea que reinicia el mensaje de colisión |
| timerTaskTick | TimerTaskTick | Añade una nueva linia al fichero con movimiento recto |
| weka | IWeka | Instancia de la clase WekaAdapter declarada como IWeka para tratar con la librería estadistica Weka |
| tiempoEnArcenIzquierdo | long | Tiempo en frames de la circulación del coche jugador en el arcen izquierdo |
| tiempoEnArcenDerecho | long | Tiempo en frames de la circulación del coche jugador en el carril izquierdo |
| tiempoEnCarrilIzquierdo | long | Tiempo en frames de la circulación del coche jugador en el carril del medio |
| tiempoEnCarrilMedio | long | Tiempo en frames de la circulación del coche jugador en el carril derecho |
| tiempoEnCarrilDerecho | long | Tiempo en frames de la circulación del coche jugador en el arcen derecho |
| juegoActivo | boolean | Indica si el juego ha comenzado |

Tabla 5: Relación de atributos de la clase Escenario, tipo de datos y utilización.

La única operación que nos interesa de las redefinidas de la interfaz KeyListener es la del evento keyPressed(KeyEvent e). Esta operación gestiona la comunicación del teclado con el juego.

La operación Juego(Boolean humano) inicia el juego en un bucle mientras el juego este activo. Conceptualmente, en dicho bucle, lo primero que se hace es actualizar las posiciones, se comprueban las colisiones y en ese momento se vuelve a pintar todo actualizado. Si el juego ha sido iniciado en modo automatico acto seguido se preguntará sobre el escenario actual, es decir, las posiciones actuales de los vehículos más cercanos al coche del jugador y el carril por el que circula, que decisión debe tomar.

La operación DetenerJuego() recoge la estadística a mostrar y finaliza los Timers.

La operación getEstatString(String movimiento) se encarga de crear una instantánea de la posición de los vehículos más cercanos al Jugador y del movimiento que ha realizado el mismo.

En este momento han aperecido varias clases que no teníamos previstas en el diseño y que están orientadas a crear efectos en una línea temporal, todas ellas las hemos agrupado bajo la subcapa TimerTask en la capa Dominio.

**Capa Dominio.TimerTask**

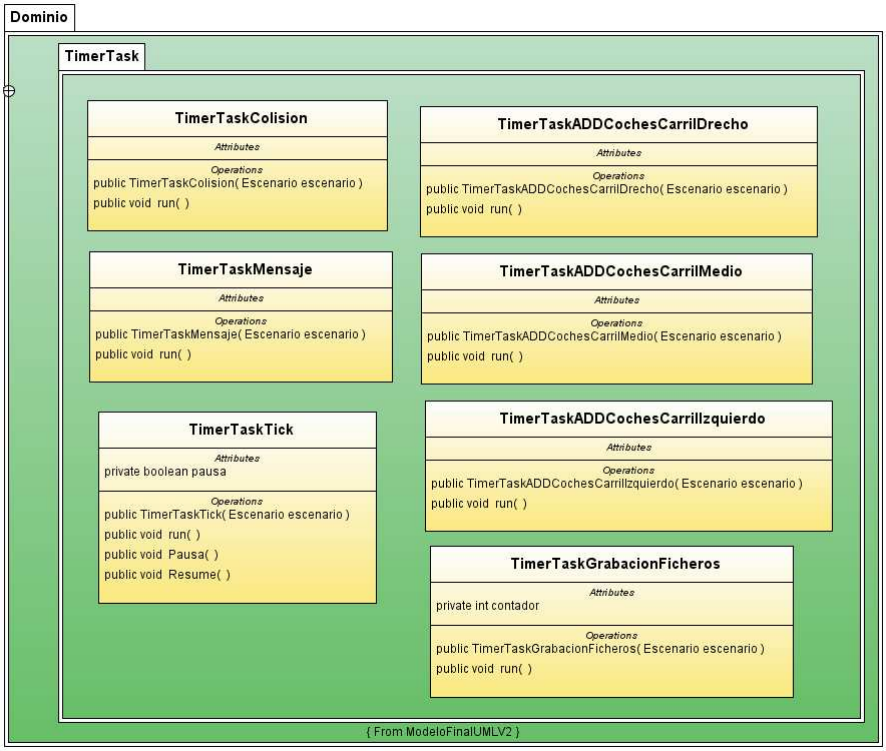
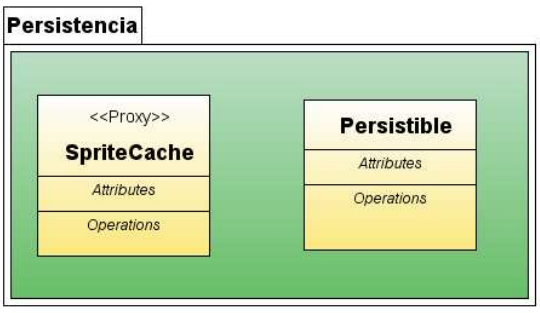



Figura 29: Capa Dominio.TimerTask

Los TimerTask resuelven, en nuestro caso, problemas que tienen que ver con el tiempo, como por ejemplo la aparición de un texto en la pantalla y que desaparezca un segundo y medio después, o la programación de que cada treinta segundos se ejecute la operación que graba en memoria persistente el archivo con el historial de movimientos.

## 8.5 Diseño de la capa Persistencia



**SpriteCache:**

La clase StripeCache implementará la interfaz ISpriteCache y el patrón de diseño Proxy, la primera vez que se acceda a la memoria persistente se guardarán los recursos en memoria para evitar los accesos a disco innecesarios y aumentar el rendimiento.

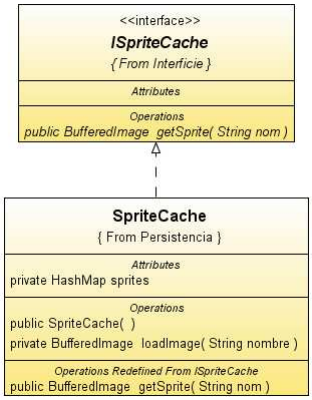



Figura 30: Implementación de la clase SpriteCache

En la implementación de la clase SpriteCache (Figura 30) podemos observar el atributo sprites del tipo de dato no primitivo HashMap, en esta colección guardaremos las referencias de las imágenes conforme las vayamos cargando de memoria persistente. Implementa el patrón proxy.

La operación getSprite(String nom) mira si tiene la imagen en la colección, si la tiene, la devuelve, sino, la carga de memoria persistente, la guarda en su colección y al final la retorna.

**Persistible:**

Implementará la interfaz IPersistible y será la encargada de guardar y leer los archivos generados por la aplicación en memoria persistente.

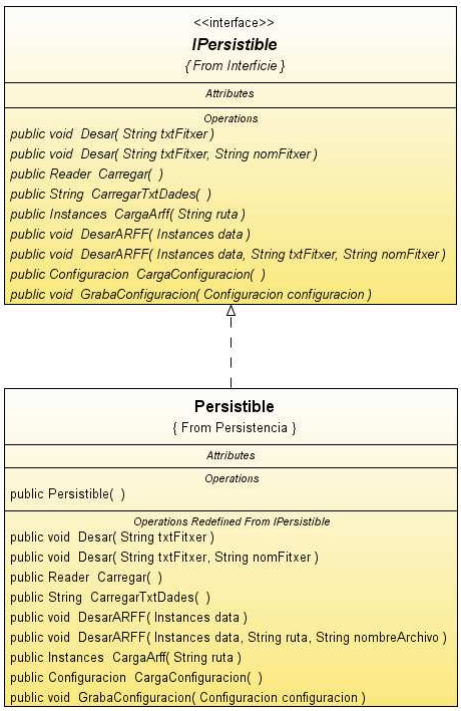



Figura 31: Implementación de la clase Persistible

En la implementación de la clase Persistible (Figura 31) observamos la redefinición de las operaciones de la clase IPersistible del Dominio, Persistible se acopla con la librería estadística Weka para obtener un objeto que le ayuda a guardar y a cargar de memoria persistente los objetos del tipo Instances y se acopla con la librería jDom.jar para el tratamiento de los archivos en XML, es decir, para grabar y cargar la configuración.

# 9 Implementación

En este capítulo entraremos profundamente en la implementación de algunas operaciones y de cómo funcionan las comunicaciones mediante librerías ajenas a las propias que nos ofrece Java.

## 9.1 JDom.jar

JDom es una librería que nos ofrece una solución completa y basada en java para la creación, manipulación y guardado de ficheros XML.

### 9.1.1 Configuración XML

Vamos a mostrar la implementación de la grabación y la carga de la configuración en formato XML.

Como podemos observar en la (Figura 32) lo primero es crear una instancia de la clase XMLOutputter, esta clase se encargará de realizar la grabación a memoria persistente, siendo totalmente transparente a nosotros cómo se guardan los archivos.

La clase FileOutputStream es necesaria para saber dónde vamos a guardar el archivo, a su constructor hay que pasarle por parámetro la ruta donde se creará o sobrescribirá el archivo físico. En este caso se guarda en una ruta relativa a la raíz del aplicativo.
La forma de funcionar de los XML es ir encapsulando nodos, unos dentro de otros, con atributos y valores. En nuestro caso hemos decidido que la configuración tendrá la estructura que podemos observar en la ( Figura 34).

Creamos un elemento raíz del tipo Element, el parámetro del constructor será el titulo del nodo, a este nodo le añadiremos los demás.

Después se crea un nuevo elemento “Clasificicador” y se llama a la operación setText para guardar la información en ese elemento.

```
public void GrabaConfiguracion(Configuracion configuracion) throws Exception {

    XMLOutputter xMLOutputter = new XMLOutputter();
    FileOutputStream fileOutputStream = new FileOutputStream("configuracion.xml");

    Element raiz = new Element("configuracion");

    Element clasificador = new Element("Clasificador");
    clasificador.setText(configuracion.getClasificador());
```

Figura 32: Primera parte del código fuente de la operación GrabaConfiguracion

A partir de aquí se pueden crear tantos Element como queramos. En este caso solo hemos creado un elemento para el ejemplo

```
    raiz.addContent(clasificador);

    Document document = new Document(raiz);

    xMLOutputter.output(document, fileOutputStream);
    fileOutputStream.flush();
    fileOutputStream.close();

}
```

Figura 33: Segunda parte del código fuente de la operación GrabaConfiguracion

Una vez creados todos los Elements se tienen que encapsular con la estructura que definamos llamando a la operación addContent y pasadole como parámetro un elemento, éste último, quedara encapsulado entre el tag de apertura y de clausura del anterior. Una vez llegado a este punto, creamos el documento XML pasándole el elemento raíz y realizamos la grabación tal y como podemos observar en la (Figura 33).

```
<?xml version="1.0" encoding="UTF-8" ?>
- <configuracion>
        <Clasificador>j48</Clasificador>
        <rutaGrabacionFicheros>C:/ProyectoFinalCarreraDani/</rutaGrabacionFicheros>
        <pesoDistIzquierda>0.0</pesoDistIzquierda>
        <pesoDistDerecha>0.0</pesoDistDerecha>
        <escenarioPredifinido>false</escenarioPredifinido>
        <relentizacion>0</relentizacion>
        <tiempoGrabacionFicheros>20</tiempoGrabacionFicheros>
 </configuracion>
```

Figura 34: Ejemplo del contenido del archivo de configuración en formato XML

## 9.2 Weka.jar

Del proyecto Weka hemos hablado anteriormente, pero la librería que nos ofrece la posibilidad de utilizar las prestaciones de ese proyecto la podemos encontrar en la ruta de instalación del propio WEKA.

### 9.2.1 Estructura de los archivos ARFF

La estructura de un fichero con formato ARFF es muy sencilla (Figura 35), por lo que seguro que no tendremos ningún problema a la hora de crearlo.

```
@relation Atributos

@attribute Carril
{CARRILARCENIZQ,CARRILIZQ,CARRIALMED,CARRIALDER,CARRILARCENDER}
@attribute di numeric
@attribute dm numeric
@attribute dd numeric
@attribute class {Recte,Esquerra,Dreta}

@data
CARRIALMED,227,227,227,Recte
CARRIALMED,154,154,154,Esquerra
```

Figura 35: Ejemplo de formato de archivo ARFF

@relation <nombre>

Todo fichero ARFF empieza con una declación "@relation" y no podemos dejarlo en blanco, si queremos ponerle espacios debe ir entre comillas

@attribute <nombre> <Tipo de dato>

En esta sección incluiremos una línea por cada atributo que vayamos a incluir en nuestro conjunto de datos, indicando su nombre y el tipo de dato.
Los tipos de datos son los siguientes:

- Numeric para numéricos.
- String para el texto.

- Date [<formato de fecha>] para fechas. En <formato de fecha> indicaremos el formato de la fecha.
- <nominal-specification> para los tipos de datos definidos por nosotros mismos y que pueden tomar una serie de valores que indicamos, como el atribute class de la (Figura 35).

@data

En esta sección incluiremos los datos propiamente dichos. Separaremos cada columna por comas y todas filas deberán tener el mismo número de columnas, número que coincide con el de declaraciones @attribute.

### 9.2.2 Comunicación Weka

Weka ya se encarga mediante sus objetos de la creación, lectura, modificación y grabación de los tipos de archivo ARFF.
Únicamente hemos de utilizar una instancia de la clase ArffSaver para guardar a disco, utilizando las operaciones setInstances(Instances instances) para indicarle que objeto del tipo Instances ha de guardar y la operación setFile(File file) para indicarle físicamente donde se va a guardar. Una vez hemos realizado estos pasos sólo tenemos que utilizar la operación writeBatch() para crear o remplazar el archivo de manera persistente.
Para la lectura podemos ver un ejemplo en la (Figura 36) del uso del ArffLoader.

```
public Instances CargaArff(String ruta) {
    try {

        ArffLoader loader = new ArffLoader();
        if (!ruta.endsWith(".arff")) {
            ruta = ruta + ".arff";
        }
        File f = new File(ruta);
        loader.setFile(f);
        return loader.getDataSet();

    } catch (Exception e) {
        System.out.print(e.getMessage());
    }
    return null;
}
```

Figura 36: Código fuente de ejemplo para la lectura de un archivo con formato ARFF

Ahora ya sabemos crear y leer archivos ARFF, pero aun no sabemos cómo funciona la clase Instances, y realmente es lo que estamos leyendo y guardando. En la (Figura 37) podemos ver como se crea una instancia del objeto Instances y como se relaciona con el tipos de archivo ARFF.

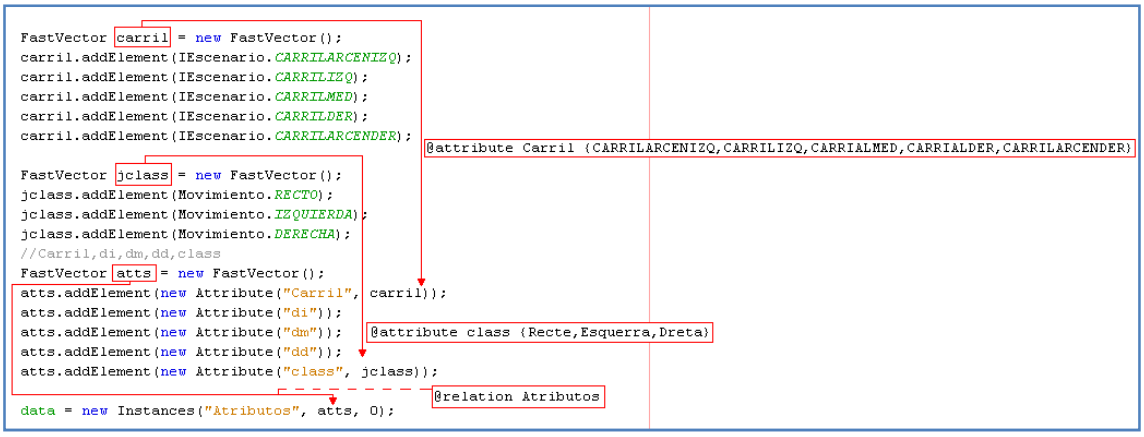



Figura 37: Relación del código fuente para la creación de Instances y el formato de archivos ARFF

Hemos creado la estructura básica pero ahora hemos de introducir datos en la sección @data, en la (Figura 38) podemos ver como se crea una instancia del objeto Instance (No confundir con el plural) y se añade al atributo del tipo Instances.

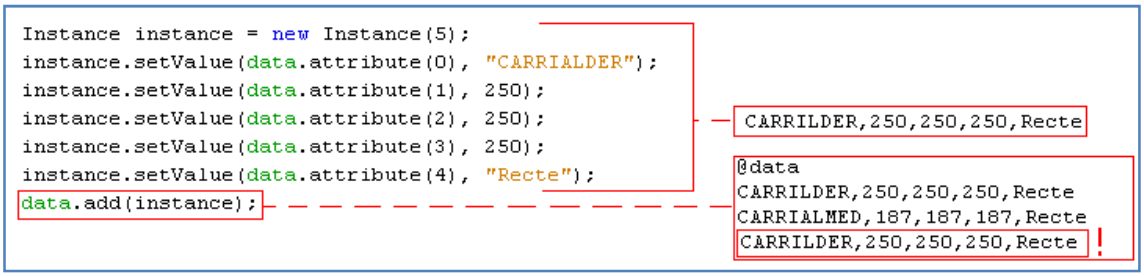



Figura 38: Relación del código fuente para añadir una nueva Instance y el formato de archivos ARFF

Aun nos queda por explicar que es el atributo class, para la clasificación hace falta indicarle al objeto Instances cuál de sus atributos va a ser el class (Figura 39), nos referimos a cuál va ser el atributo destacado, una vez clasifiquemos los datos sobre el atributo class podremos evaluar una nueva instancia del tipo Instance del cual no sabemos su class y obtener una ponderación de su valor.

```
data.setClass(data.attribute(4));
```

Figura 39: Código fuente de la selección del atributo class

En este punto ya sabemos crear instancias del tipo Instances bien formateadas y añadir nuevos datos del tipo Instance. Así que pasaremos a explicar el funcionamiento de la clase Classifier y Evaluation. Gracias a estas dos clases podemos clasificar nuestro conjunto de datos y evaluar una instancia del tipo Instance.

Lo primero que hemos de hacer es crear el clasificador, Weka nos permite acceder a múltiples clasificadores, pero el aplicativo contempla los que hemos decidido que se amoldan mejor a nuestra problemática: BayesNet y NaiveBayes por parte de clasificadores bayesianos y J48, J48graft y NBTree por parte de clasificadores en árbol.

En la (Figura 40) podemos observar la creación del clasificador de árbol J48 y seguidamente la construcción del clasificador con la instancia de la clase Instances. Acto seguido se puede observar la creación de la instancia de la clase Evaluation pasándole por parámetro el atributo data y la evaluación del modelo, es decir, el entrenamiento.

```
cModel = (Classifier) new J48();
cModel.buildClassifier(data);

Evaluation eTest = new Evaluation(data);
eTest.evaluateModel(cModel, data);
```

Figura 40: Código fuente de ejemplo de uso de las clases Classifier y Evaluation

Para clasificar un escenario, tenemos que crear una instancia de la clase Instance como podemos observar en la (Figura 41), establecer los valores a los atributos que no sean del tipo class y establecer el conjunto de datos con la operación setDataset(data). Acto seguido usaremos la operación distributionForInstance(instance) para obtener un array de double[] con las probabilidades entre 0 y 1 del conjunto de datos class. Siendo cada posición de la array uno de los atributos, es decir, fDistribution[0] para Recto, fDistribution[1] para izquierda y fDistribution[2] para derecha.

```
Instance instance = new Instance(4);
instance.setValue(data.attribute(0), carril);
instance.setValue(data.attribute(1), di);
instance.setValue(data.attribute(2), dm);
instance.setValue(data.attribute(3), dd);

instance.setDataset(data);

double[] fDistribution = cModel.distributionForInstance(instance);
```

Figura 41: Código fuente de ejemplo de la evaluación de una nueva Instance.

# 10 Estudio experimental

Una vez desarrollado el aplicativo realizaremos un estudio experimental para verificar si realmente la tecnología desarrollada puede realizar la tarea de conducir automáticamente y si en algún caso imita la conducta humana.

## 10.1 Metodología.

La realización del estudio se llevará a cabo con el escenario creado de manera secuencial, en el que realizaremos tres partidas imitando un tipo de conducción distinta en cada una de ellas, por lo tanto, obtendremos tres conjuntos de entrenamiento distintos, el Conjunto de entrenamiento uno, el dos y el tres.
En la primera partida conduciremos por el carril izquierdo y siempre que podamos giraremos a la derecha para adelantar, evitando el arcén izquierdo el máximo tiempo posible.
En la segunda conduciremos siempre que podamos por el carril central y efectuaremos los adelantamientos por la izquierda siempre que nos sea posible.
En la tercera conduciremos por los tres carriles de manera indiferente evitando los arcenes y realizando los adelantamientos como podamos.

El clasificador utilizado en el experimento será el J48 y no usaremos las ponderaciones de peso para los movimientos, es decir, las dejaremos a cero.

A lo largo de la partida el sistema grabará, cada 30 segundos, un archivo que describirá los movimientos realizado por el vehículo. Cada una de las partidas está estipulada en 3 minutos, por lo que tendremos que entrenar 6 clasificadores, usando únicamente los 30, 60, 90, 120, 150 y 180 primeros segundos de cada una de las partidas, para analizar la curva de aprendizaje del clasificador J48, así como la tendencia a circular por el mismo carril que el usuario.

Después de cada uno de los conjuntos de entrenamiento ejecutaremos el juego en modo conducción automática por cada uno de los clasificadores a estudiar, esta prueba tendrá una

durada de 2 minutos y al finalizar cada una ellas analizaremos las estadísticas mostradas por el programa y lo compararemos con las obtenidas por la partida manual.

El formato de Intances que utilizará el clasificador sigue el formato descrito en la (Figura 35), las instancias de ir recto se graban automáticamente cada segundo, en cambio la creación de Instance con el movimiento de ir a izquierda o a derecha se graba siempre que se realiza el movimiento.

Para entender mejor los arboles que mostraremos más adelante tendremos en cuenta que "di" quiere decir distancia del vehículo más cercano que circula por el carril izquierdo y con el que aun podemos colisionar, "dm" se refiere a la distancia del que circula por el del medio y "dd" por el derecho. Estas distancias varían de 250 como distancia máxima delante de nosotros a 45 sin colisionar, las distancias de colisión horizontal son de [46,-39). Esto es debido a la longitud de los vehículos y a la longitud en pixeles de sus imágenes.

## 10.2 Resultados

**Partida 1:** Siempre que podamos circularemos por el carril izquierdo y adelantaremos por la derecha. Evitando en medida de lo posible el arcén.

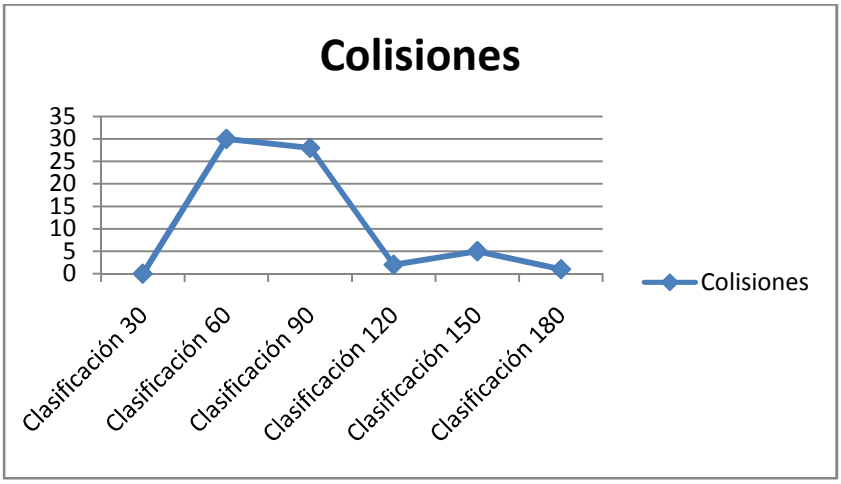



Gráfico 1: Número de colisiones de la partida uno

En el (Gráfico 1) no podemos observar una clara tendencia de reducción de colisiones, ya que la Clasifiación 60 y 90 podemos ignorarlas, conforme aumentamos el número de instancias y el tiempo de juego.

Hay que destacar que en la última muestra, la de 180 segundos sólo ha habido una colisión sobre 85 vehículos adelantados. Lo que significa un 1.18% de colisiones.

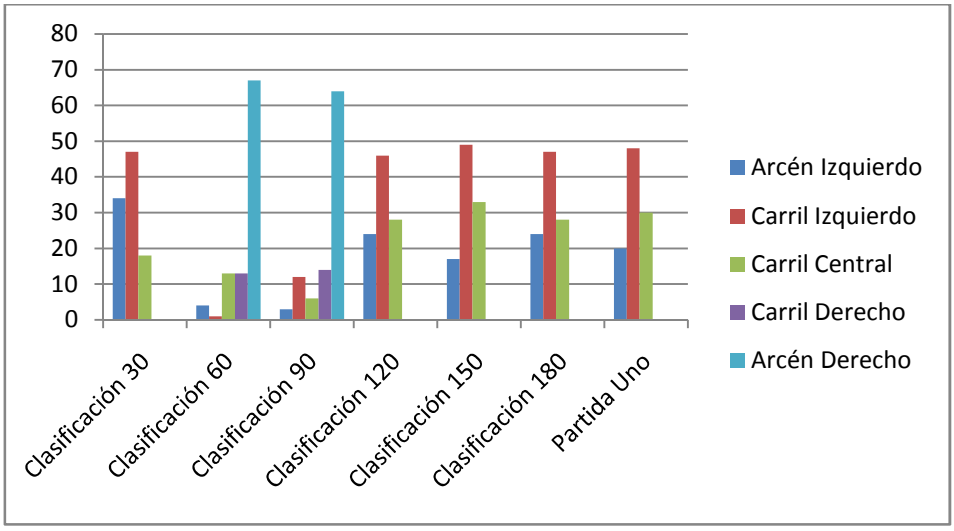



Gráfico 2: Tiempo de ocupación de los carriles de la partida uno

Las barras verticales del (Gráfico 2) nos indican el porcentaje de conducción en cada carril. La figura que debemos tomar como base es la creada por las barras de la Partida Uno, en el eje horizontal, para que el clasificador imite el comportamiento humano las demás barras tratándolas como cada uno de los subconjuntos debe ser lo más parecida posible.

La Clasificación 60 y la 90 han generado una clasificación extraña y se ha disparado tanto el número de colisiones (Gráfico 1) como la ocupación de los carriles (Gráfico 2). A parte de estos dos, tanto la Clasificación 30, 120, 150 y 180 tienen una similitud asombrosa con la ocupación de los carriles de la partida uno.

```
Carril = CARRILARCENIZQ
|   di <= -44: Dreta (20.0)
|   di > -44
|   |   di <= 133: Recte (17.0)
|   |   di > 133: Dreta (2.0)
Carril = CARRILIZQ
|   di <= 59
|   |   dm <= 149
|   |   |   dm <= -36: Dreta (7.0/1.0)
|   |   |   dm > -36: Esquerra (22.0/1.0)
|   |   dm > 149: Dreta (16.0)
|   di > 59: Recte (33.0)
Carril = CARRIALMED
|   dm <= 59
|   |   dm <= -2: Recte (3.0)
|   |   dm > -2: Esquerra (22.0)
|   dm > 59: Recte (17.0)
Carril = CARRIALDER: Recte (0.0)
Carril = CARRILARCENDER: Recte (0.0)
```

Figura 42: Poda del árbol de decisiones de la clasificación 180 (J48) de la partida uno

Analizando el árbol de la (Figura 42) podemos observar que ha realizado la primera división, es decir, los nodos principales, con los carriles por los cuales circulamos, después, dependiendo de la distancia del vehículo más cercano por los distintos carriles se decide la acción a realizar. Si analizamos la rama del CARRILARCENIZQ encontramos que sólo realiza dos acciones: “Dreta” o “Recte”, esto tiene muchísimo sentido, ya que en la partida uno, cuando circulábamos por el arcén izquierdo nunca hemos intentado ir mas a la izquierda. Otro dato importante es que únicamente controla la distancia del coche que circula por el carril izquierdo para decidir realizar la incorporación. La rama del CARRILIZQ es sin duda la más compleja, esto es debido a que ha sido el carril por el que más tiempo hemos conducido y también por el que más acciones hemos realizado, tanto de salida como de entrada. La rama CARRIALMED únicamente observa los vehículos que circulan por su carril, por lo tanto si hay algún vehículo en el izquierdo colisionaremos. En esta parte del árbol de decisión han sido donde ha ocurrido la colisión. En el CARRIALDER y CARRILARCENDER no hay ninguna hoja porque no hemos circulado por ellos. La punta final de cada rama es un movimiento, es realmente el mismo formato de árbol que hubiésemos diseñado si tuviéramos que resolverlo manualmente.

**Partida 2:** Siempre que podamos circularemos por el carril central y adelantaremos por la izquierda. Evitando en medida de lo posible el arcén.

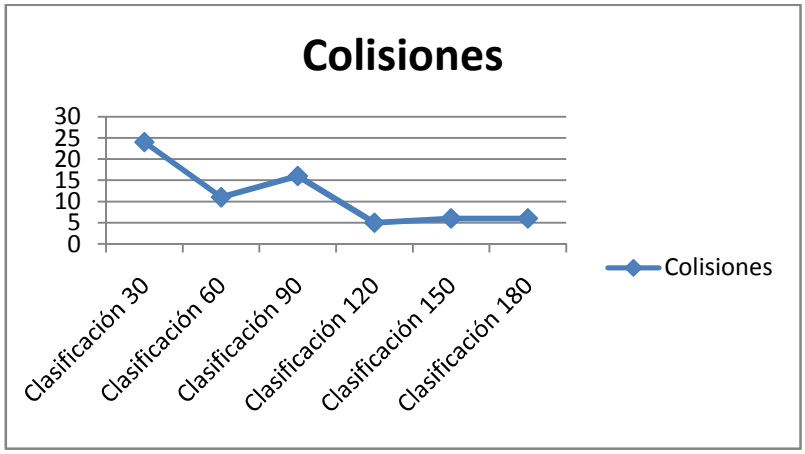



Gráfico 3: Número de colisiones de la partida dos

En el (Gráfico 3) podemos observar la clara tendencia de reducción de colisiones conforme aumentamos el número de instancias y el tiempo de juego.

Hay que destacar que en la última muestra, la de 180 segundos sólo ha habido seis colisiones sobre 85 vehículos adelantados. Lo que significa un 7.06% de colisiones. Esto es siete veces más colisiones que en la partida 1, pero también puede ser debido a la política de conducción.

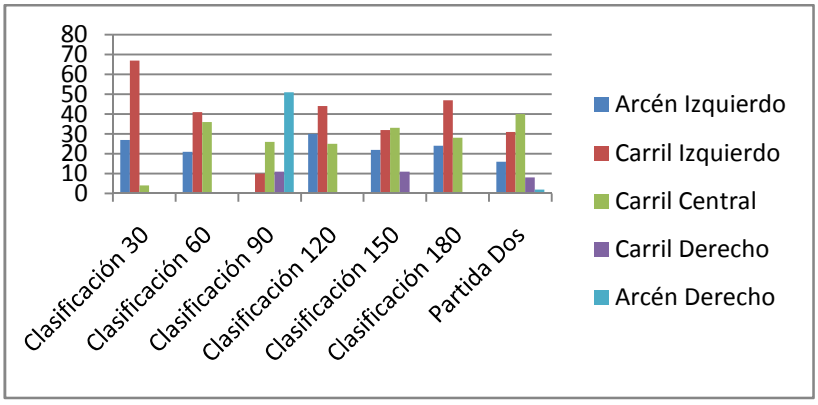



Gráfico 4: Tiempo de ocupación de los carriles de la partida dos

Las barras verticales del (Gráfico 4) nos indican el porcentaje de conducción en cada carril. La figura que debemos tomar como base es la creada por las barras de la Partida Dos, en el

eje horizontal, para que el clasificador imite el comportamiento humano las demás barras tratándolas como cada uno de los subconjuntos debe ser lo más parecida posible.

Esta vez únicamente la clasificación 90 ha generado una clasificación extraña y se ha disparado tanto el número de colisiones (Gráfico 3) como la ocupación de los carriles. A parte de esto, y aunque no es tan clara la semejanza como en la partida uno, podemos observar que la Clasificación 150 si mantiene cierta similitud.

```
Carril = CARRILARCENIZQ
|   di <= -50: Dreta (10.0)
|   di > -50
|   |   di <= 81: Recte (30.0)
|   |   di > 81: Dreta (5.0/1.0)
Carril = CARRILIZQ
|   dm <= -41
|   |   di <= 94: Dreta (11.0)
|   |   di > 94
|   |   |   dd <= 75: Recte (3.0)
|   |   |   dd > 75: Dreta (3.0)
|   dm > -41
|   |   dm <= 154
|   |   |   di <= 67
|   |   |   |   di <= 2: Recte (5.0)
|   |   |   |   di > 2: Esquerra (16.0/2.0)
|   |   |   di > 67: Recte (47.0)
|   |   dm > 154: Dreta (6.0)
Carril = CARRIALMED
|   dm <= 65
|   |   di <= 138
|   |   |   di <= -41: Esquerra (10.0/1.0)
|   |   |   di > -41
|   |   |   |   dm <= 48: Recte (9.0/1.0)
|   |   |   |   dm > 48
|   |   |   |   |   dd <= -18: Dreta (4.0)
|   |   |   |   |   dd > -18
|   |   |   |   |   |   dd <= 138: Recte (4.0/2.0)
|   |   |   |   |   |   dd > 138: Dreta (3.0)
|   |   di > 138: Esquerra (8.0)
|   dm > 65: Recte (61.0/1.0)
Carril = CARRIALDER
|   dm <= -41: Esquerra (5.0)
|   dm > -41
|   |   dm <= 94: Recte (17.0/1.0)
|   |   dm > 94: Esquerra (4.0/1.0)
Carril = CARRILARCENDER
|   dm <= 39: Esquerra (3.0/1.0)
|   dm > 39: Recte (4.0)
```

Figura 43: Poda del árbol de decisiones de la clasificación 180 (J48) de la partida dos

Analizando el árbol de la (Figura 43) lo primero que nos llama la atención es que es mucho más grande que el generado en el apartado anterior pero el formato es el mismo, es decir, los nodos principales siguen siendo los carriles y después dependiendo de la posición del coche más cercano en uno u otro carril decide qué movimiento realizar. Como curiosidad, observando la rama del CARRIALDER vemos que no realiza nunca un movimiento a derecha, pero en cambio, si tiene decisiones que tomar en caso de estar en el CARRILARCENDER.

**Partida 3:** Conduciremos por los tres carriles de manera indiferente realizando los adelantamientos al azar.

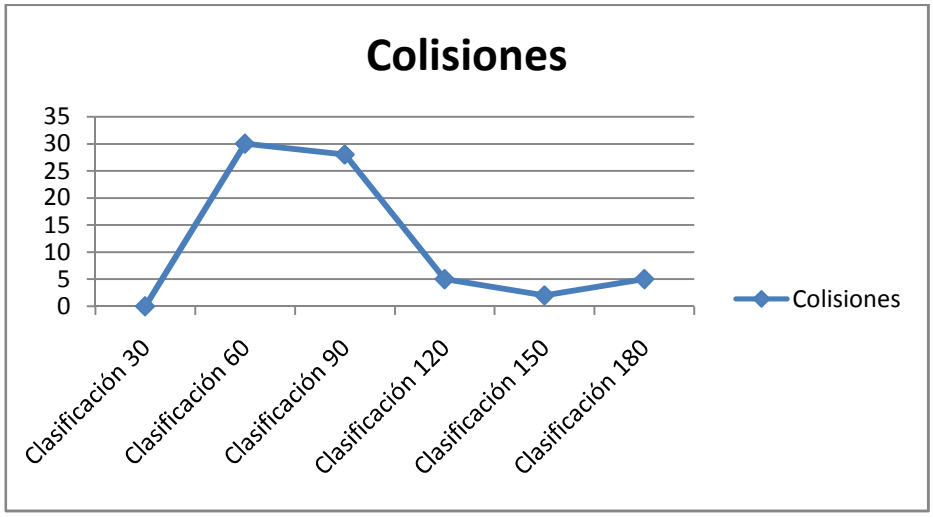



Gráfico 5: Número de colisiones de la partida tres

En el (Gráfico 5) no podemos observar una clara tendencia de reducción de colisiones, ignorando la Clasificación 60 y 90, conforme aumentamos el número de instancias y el tiempo de juego.

Hay que destacar que en la última muestra, la de 180 segundos sólo ha habido cinco colisiones sobre 80 vehículos adelantados. Lo que significa un 6.25% de colisiones. Esto es seis veces más colisiones que en la partida 1, pero mejora en 1% las colisiones de la partida dos.

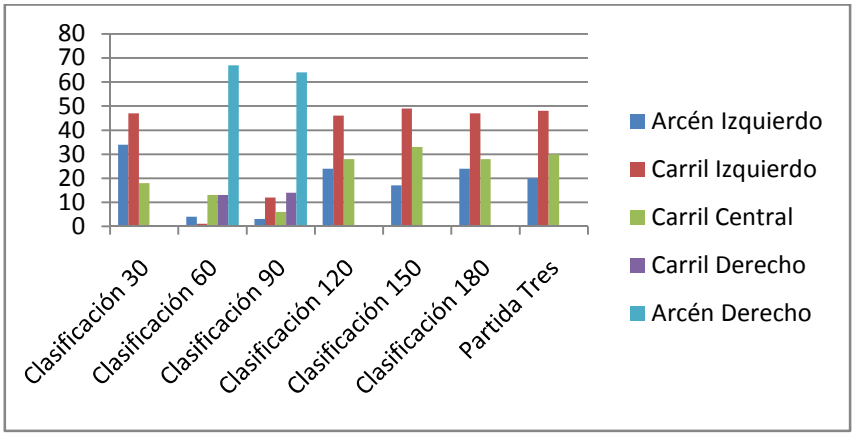



Gráfico 6: Tiempo de ocupación de los carriles de la partida tres

Las barras verticales del (Gráfico 6) nos indican el porcentaje de conducción en cada carril. La figura que debemos tomar como base es la creada por las barras de la Partida Tres, en el

eje horizontal, para que el clasificador imite el comportamiento humano las demás barras tratándolas como cada uno de los subconjuntos debe ser lo más parecida posible.

Esta vez se repite el mismo caso que en la partida uno y la clasificación 60 y 90 han generado una clasificación extraña y se ha disparado tanto el número de colisiones como la ocupación de los carriles. En esta partida, podemos observar claramente una gran similitud entre la Clasificación 120, 150, 180 y la Partida Tres.

```
Carril = CARRILARCENIZQ
|  di <= 59
|  |  di <= -46: Dreta (4.0)
|  |  di > -46: Recte (17.0)
|  di > 59: Dreta (5.0)
Carril = CARRILIZQ
|  di <= 57
|  |  dm <= 65
|  |  |  dm <= -41: Dreta (2.0)
|  |  |  dm > -41: Esquerra (10.0/1.0)
|  |  dm > 65: Dreta (10.0/1.0)
|  di > 57: Recte (37.0)
Carril = CARRIALMED
|  dm <= 62
|  |  dm <= 2: Recte (3.0)
|  |  dm > 2
|  |  |  dm <= 53
|  |  |  |  dd <= -32: Recte (2.0/1.0)
|  |  |  |  dd > -32: Esquerra (3.0)
|  |  |  dm > 53
|  |  |  |  dd <= 138
|  |  |  |  |  di <= 126: Dreta (8.0/3.0)
|  |  |  |  |  di > 126: Esquerra (5.0)
|  |  |  |  dd > 138: Dreta (6.0)
|  dm > 62: Recte (49.0/1.0)
Carril = CARRIALDER
|  dd <= 59
|  |  dm <= 138
|  |  |  dd <= 51
|  |  |  |  dm <= 2: Esquerra (4.0)
|  |  |  |  dm > 2: Recte (3.0)
|  |  |  dd > 51
|  |  |  |  dm <= -38: Recte (3.0)
|  |  |  |  dm > -38: Dreta (11.0)
|  |  dm > 138: Esquerra (10.0/1.0)
|  dd > 59: Recte (40.0)
Carril = CARRILARCENDER
|  dd <= 75
|  |  dd <= -51: Esquerra (7.0/1.0)
|  |  dd > -51: Recte (24.0)
|  dd > 75: Esquerra (5.0)
```

Figura 44: Poda del árbol de decisiones de la clasificación 180 (J48) de la partida tres

Analizando el árbol de la (Figura 44) lo primero que nos llama la atención es que tiene un tamaño considerable también pero el formato es el mismo, es decir, los nodos principales siguen siendo los carriles y después dependiendo de la posición del coche más cercano en uno u otro carril decide qué movimiento realizar.

## 10.3 Conclusiones

Las conclusiones más relevantes son:

- Los resultados anómalos de la clasificación 60 y 90 de las partidas 1 y 3, como también la clasificación 90 de la partida 2, son debidos a la inmadurez de la clasificación.

- En todos los casos de clasificación de más de 120 segundos se obtenido resultados esplendidos tanto en el numero de colisiones, por debajo del 8% y con la imitación de la conducción humana.

- El margen de mejora en la clasificación es aún muy grande, por lo tanto, con un entrenamiento más extenso, podríamos rebajar el número de colisiones a casi ninguna y cumpliremos el requisito de imitar la conducta humana.

- En la partida 1 y 3 se observa que el clasificador 30 obtiene un índice de colisión muy bajo, pero realmente no imita el modo de conducción del jugador como podemos ver en el (Gráfico 2) y (Gráfico 6).

# 11 Manual de usuario

### Pantalla principal

La pantalla principal (Figura 45) nos da acceso al menú y es la primera pantalla que nos muestra la aplicación. Desde aquí podemos acceder a comenzar un juego nuevo o a configurar la aplicación.


Figura 45: Pantalla principal

En el menú de la aplicación encontramos la siguiente estructura:

- Juego
  - Nuevo
  - Salir

- Herramientas
    - Configuración
- Ayuda
    - Manual de usuario
    - Acerca de

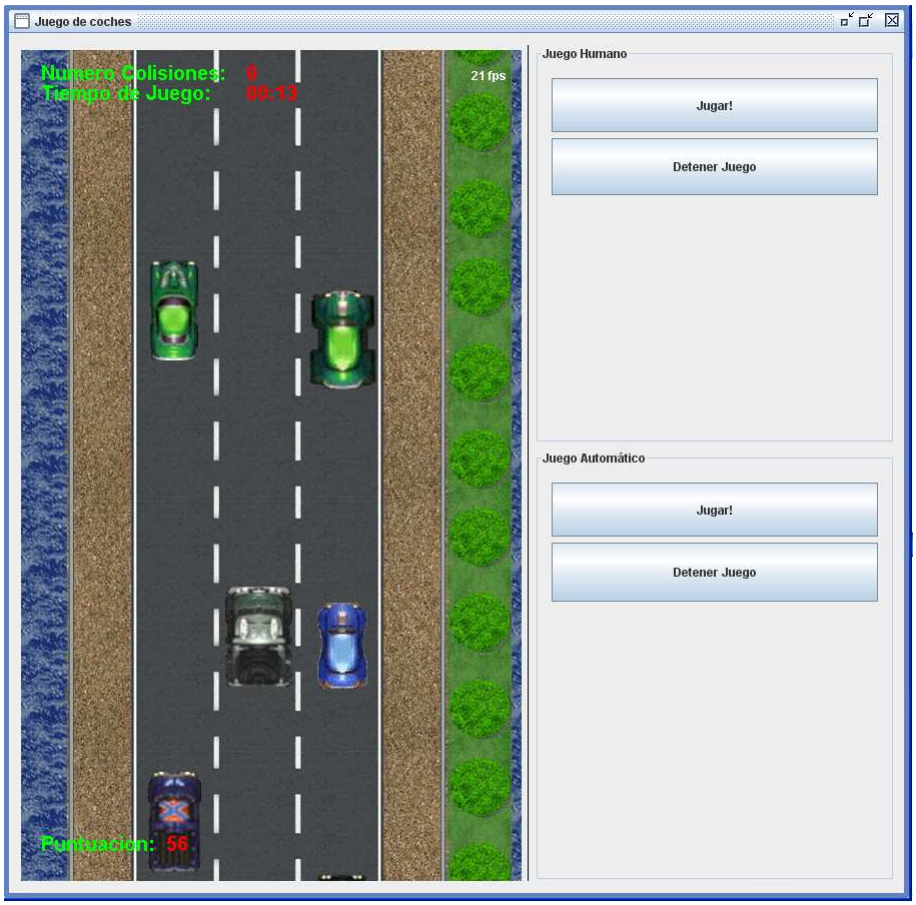



Figura 46: Pantalla de nuevo juego

**Pantalla de nuevo juego:**

La pantalla de nuevo juego esta divida en tres partes, toda la vertical izquierda esta asignada para mostrar el juego de coches, a la derecha encontramos dos divisiones mas, juego manual y juego automático. Si usamos el botón de jugar! Del modo manual controlaremos el vehículo y jugaremos una partida. Si utilizamos el modo Automático se abrirá un formulario de selección de archivo que queremos utilizar (Figura 47).

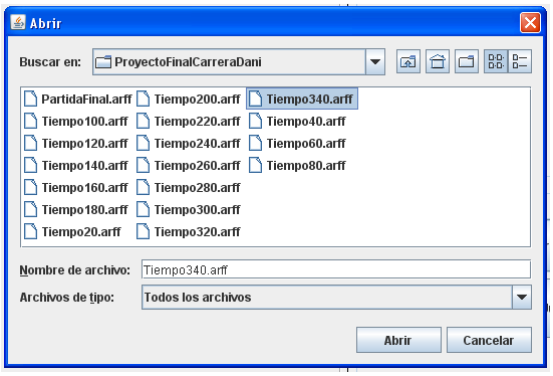



Figura 47: Selección de archivo.

## Controles de usuario:

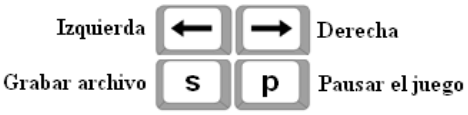



Figura 48: Controles de usuario

## Configuración:

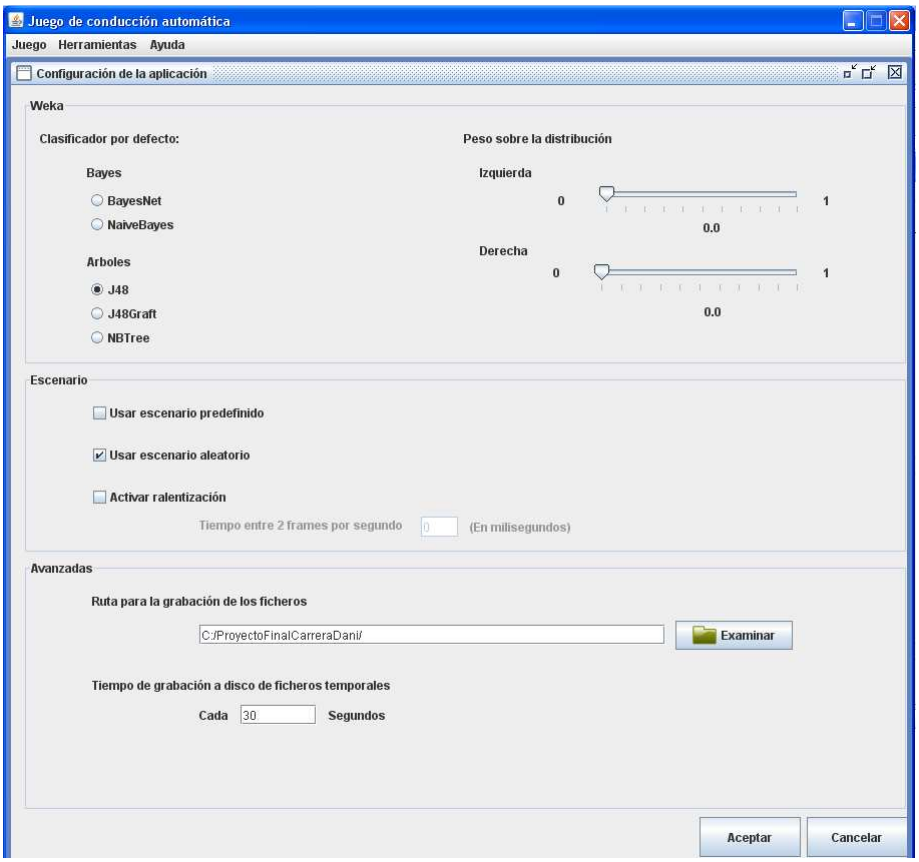



Figura 49: Pantalla de configuración

En la pantalla de configuración (Figura 49) encontramos tres divisiones: Weka, Escenario y avanzadas.

**Weka:**

En el apartado Weka a la izquierda encontramos la selección del clasificador por defecto, los bayesianos BayesNet y NaiveBayes, y debajo los basados en estructuras de árbol J48, J48Graft y NBTree. Al realizar una partida en modo automático se utilizará para la clasificación y la toma de decisiones el clasificador que hayamos seleccionado previamente.

En la parte derecha encontramos dos barras que van de cero a uno, estas barras configuran el porcentaje mínimo que tiene que tener una decisión para hacerle caso. La barra superior se encarga de la decisión de ir a la izquierda y la inferior a derechas.

**Escenario:**

En el apartado escenario encontramos tres seleccionables, los dos superiores se comportan como una selección única, por lo tanto, sólo puede haber uno seleccionado a la vez. El seleccionable “Usar escenario predefinido” utiliza un escenario creado regularmente y siempre es el mismo. En cambio la selección “Usar escenario aleatorio” creará un escenario distinto cada vez que se inicie un juego.

La tercera opción, “Activar ralentización” nos permite configurar un tiempo de espera en milisegundos entre dos frames por segundo. Esto se realiza para afinar la velocidad del juego en ordenadores de distinta potencia.

**Avanzadas:**

En avanzadas encontramos la ruta de creación de archivos ARFF y debajo el tiempo que esperará la aplicación para crear un archivo en la ruta seleccionada del archivo con los datos actuales.

# 12 Conclusiones y trabajo futuro

**Las conclusiones más relevantes son:**

- El desarrollo de un videojuego bidimensional no es nada que hayamos desarrollado durante la carrera y aunque no se ha profundizado mucho en este aspecto, es una problemática tan diferente que he tenido que rehusar un diseño Modelo-Vista-Controlador.
- Las diferentes potencias de hardware hacen que todo el aplicativo funcione más deprisa en algunas maquinas que en otras, en la parte de gestión, esto no es un problema, pero en la parte gráfica es un punto vital a tener en cuenta, ha sido un acierto implementar un modo configurable de estandarizar la velocidad del juego manualmente.
- El aspecto visual del juego y su sencillez ha sido un acierto, ya que, varias personas me han comentado que les recuerda a su juventud cuando jugaban a juegos muy parecidos en las recreativas.
- En el proyecto se han aplicado varios patrones de diseño de software para solventar las distintas problemáticas que me he encontrado:
    - El patrón proxy para la obtención de imágenes ha sido una buena decisión, ya que, durante las pruebas de rendimiento del juego el consumo de memoria se mantenía estable.
    - El patrón Fachada nos ha ayudado a solventar la problemática de los acoplamientos de la clase Escenario con la Interfaz de usuario y poder realizar la gestión de las grabaciones a memoria persistente de un modo muy entendible y fácil de ampliar o modificar.
    - El uso del patrón adaptador para aislar la librería estadística Weka y centrarnos en lo que el escenario requería de Weka sin preocuparnos de la implementación ha sido sin duda la mejor decisión tomada en el proyecto a lo que a patrones se refiere.
- La implementación ha sido fácil dentro de la dificultad de desarrollar un tipo de aplicación nuevo por primera vez y no ha surgido ningún problema, esto es debido sin duda a que nos hemos enfrentado a los desafíos realizando un diseño robusto.

- El uso de la librería JDom para solucionar la gestión en memoria persistente de la configuración no estaba planificada en la fase de diseño y ha sido la única cosa que se ha improvisado un poco, yo tenía claro que necesitaba tratar con XML para guardarla pero no sabía que Java no trae ninguna clase para realizar la faena. Su implementación ha sido rápida y eficaz.
- Me he desviado un poco de la planificación establecida al haber de viajar dos semanas en Diciembre sin posibilidad de seguir realizando el proyecto y me he dado cuenta de que debería haber planificado el proyecto con más ambición, es decir, intentando acabarlo una semana o dos antes.
- Del desarrollo del proyecto lo más costoso ha sido la documentación, le planifiqué tres semanas, pero al final sólo he podido dedicarle dos y aunque he aumentado el número de horas de trabajo al día, pienso que debería haberle dedicado más tiempo, sin dudad en un futuro planificaré mas cuidadosamente los proyectos a desarrollar.
- Weka te ofrece muchos clasificadores para realizar tus estudios, pero en nuestro caso, con nuestro escenario, los que mejor resultado nos han ofrecido han sido los basados en árboles.
- Se han empleado clasificadores capaces de aprender, tomando como ejemplo las decisiones que el jugador ha efectuado, y decidir sobre el mejor camino para alcanzar la meta.
- El software diseñado es capaz de imitar la conducta humana después de aprender de ella durante un mínimo de 120 segundos. Pasado este tiempo, los resultados son cada vez más precisos.
- He solucionado los objetivos que me había planteado al iniciar el proyecto, pero ahora creo que con la experiencia que he obtenido y los desafíos a los que me he enfrentado me ayudarán seguro en un futuro para tener un punto referencia cuando tenga que planificar algún proyecto.

**Trabajo futuro:**

- Dar un paso más en el desarrollo del entorno grafico e implementar un controlador que nos permita tener la certeza de que obtendremos el mismo comportamiento sin importar la potencia de la maquina sería un paso muy importante.

- Implementar el uso de nuevos clasificadores para poder experimentar con ellos las políticas de conducción y la curva de aprendizaje realizado en la parte de proyectos.
- Experimentar con políticas de conducción más complejas y adaptar la clasificación a políticas también más complejas.
- Incluir en la clasificación las velocidades a la que circulan los vehículos, ya que, en muchos casos no podemos basarnos únicamente en la posición de estos, sobretodo en escenarios aleatorios.
- Si desarrolláramos un escenario mucho más complejo podríamos, en vez de utilizar la librería Weka para la creación del archivo, utilizar un sistema gestor de base de datos para almacenar un gran volumen de información, ya que, Weka permite trabajar también directamente con una base de datos para realizar el entrenamiento.

# 14 Bibliografía

[1] Confidence-Based Policy Learning from Demonstration using Gaussian Mixture Models. Sonia Chernova and Manuela Veloso. (2007)

[2] http://www.netbeans.org/ (Enero 2010)

[3] http://www.cs.waikato.ac.nz/ml/weka/ (Enero 2010)

[4] http://java.sun.com/javase/7/docs/api/ (Enero 2010)

[5] http://casidiablo.net/marcianos-java/ (Enero 2010)

[6] http://www.metaemotion.com/diego.garcia.morate/download/weka.pdf - Manual de Weka - Diego Gracía Morate (Enero 2010)

[7] http://weka.wikispaces.com/Programmatic+Use (Enero 2010)

[8] http://www.gimp.org.es/ (Enero 2010)

[9] http://www.inkscape.org/?lang=es (Enero 2010)

[10] http://www.jdom.org/ (Enero 2010)

[11] http://office.microsoft.com/es-es/suites/FX101674083082.aspx (Enero 2010)

# 15 Anexo

1. Carpeta con la documentación.
    - Artículo.
    - Documentación.
    - Experimento (Ficheros Excel).
2. Carpeta con la aplicación.
    - Código fuente.
3. Carpeta con el software necesario.
    - NetBeans 6.1.
    - Weka.
4. Carpeta con el software utilizado.
    - Gimp.
    - Inkscape.
5. Carpeta con las librerías.
    - Weka.
    - Jdom.
    - Absolute Layout.