ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

AMPLIACIÓN Y REINGENIERÍA DE UN SISTEMA EXPERTO BASADO EN REGLAS CON FINES EDUCATIVOS

Pablo Recio Quijano

Febrero de 2010

ESCUELA SUPERIOR DE INGENIERÍA

INGENIERO TÉCNICO EN INFORMÁTICA DE SISTEMAS

AMPLIACIÓN Y REINGENIERÍA DE UN SISTEMA EXPERTO BASADO EN REGLAS CON FINES EDUCATIVOS

- Departamento: Lenguajes y sistemas informáticos
- Director del proyecto: Manuel Palomo Duarte
- Autor del proyecto: Pablo Recio Quijano

Cádiz, Febrero de 2010

Fdo: Pablo Recio Quijano

Agradecimientos

A Rosa, por la ayuda con la interfaz gráfica y las traducciones.
A Noelia, por el diseño del logo, las piezas y los botones.
A las dos por hacer de *beta testers* de la aplicación.
A Manuel Palomo por su labor como tutor y por el núcleo base.
A mi familia por apoyarme durante el transcurso de la carrera y la realización de este proyecto.

Licencia

Notación y formato

Durante todo el desarrollo de este **Proyecto Fin de Carrera** se ha seguido la siguiente notación:

- Para las rutas y nombres de fichero se ha seguido la notación: `fichero.txt`
- Para los nombres de bibliotecas y tecnologías externas se ha seguido el formato: CLIPS
- Para los nombres de las clases se ha seguido el formato *Partida*

Índice general

Indice de figuras

Indice de tablas

Capítulo 1

Introducción

1.1. Contexto y motivación

Innovar y buscar nuevos métodos para la enseñanza es un reto para la comunidad universitaria, sea del campo que sea. Los nuevos *estándares* de enseñanza obligan al profesorado a buscar alternativas mas eficientes y cercanas para transmitir conocimientos a sus alumnos.

En el campo de las enseñanzas técnicas, y concretamente en la ingeniería informática, quizás pueda ser una buena idea cambiar la metodología, no tanto de trabajo, sino del enfoque de desarrollo e intentar motivar al alumnado con problemas más realistas, más aplicables a un entorno real (ó relativamente real).

En el caso que nos ocupa, la enseñanza de sistemas expertos, es común proponerle al estudiante un problema genérico en el que diseñe un sistema que deduzca parentescos familiares, por poner un ejemplo. Sin embargo, a un futuro ingeniero informático, normalmente le gusta hacer cosas más útiles, más visibles, y si puede ser, más entretenidas. Ese es el objetivo de este proyecto.

El desarrollo de este proyecto se centra en la realización de dicho entorno de trabajo, que mejora los sistemas actuales de forma que sea utilizable por los alumnos de la asignatura **Diseño de Videojuegos** de la Ingeniería Técnica en Informática de Sistemas, durante el 2º cuatrimestre del curso 2009/2010 y en futuras ediciones de la asignatura.

Resistencia en Cádiz: 1812[1] pretende aportar un grano de arena en ese aspecto, proporcionando un entorno de desarrollo usable, ameno y productivo. De esta forma, se desarrollará una aplicación para que pueda ser utilizada por el alumnado conocer las características de un **sistema experto basado en reglas**, diseñar el suyo propio y hacerlo competir con el del resto de los compañeros.

1.2. Objetivos

A la hora de definir los objetivos de un sistema, podemos agruparlos en dos tipos de objetivos diferentes: **funcionales** y **transversales**. Los primeros se refieren a lo *que* debe hacer la aplicación a la cual queremos llegar, y que son fundamentalmente lo que terminará viendo el usuario de la aplicación.

[1] Resistencia en Cádiz: 1812 es el nombre de la aplicación resultante del desarrollo de este Proyecto Fin de Carrera.

Por otro lado, los objetivos transversales son aquellos que no son visibles para el usuario, pero que en su mayoría hacen una aplicación mas fácilmente mantenible, ampliable... en definitiva, de una mayor calidad.

Dicho esto, los objetivos (de una forma general, no entrando en una toma de requisitos formal) podrían ser los siguientes:

1.2.1. Funcionales

- Crear un sistema para el apoyo al aprendizaje de programación básica en Sistemas Expertos basados en reglas, basado en el anterior sistema **La Batalla del Guadalete**.
- Permitir al usuario probar cualquier sistema experto creado por él mismo, contra cualquier otro sistema experto incluido en el sistema, o de terceras personas.
- Realizar competiciones de forma sencilla y configurable.
- Proporcionar la posibilidad de hacer automáticamente pruebas masivas por lotes de los sistemas expertos, para poder evaluar la *bondad* de dichas implementaciones.
- Organización intuitiva de los ficheros de *log* de partidas, permitiendo visualizar dichas partidas en cualquier momento.
- Interfaz de usuario clara, amigable e intuitiva, integrada con el entorno de escritorio libre **GNOME**.

1.2.2. Transversales

- Utilizar un enfoque de análisis, diseño y codificación orientado a objetos, de una forma lo más clara y modular posible, para permitir ampliaciones y modificaciones sobre la aplicación por terceras personas.
- En la misma linea del punto anterior, considero importante seguir estándares en la producción del código, para que éste sea entendible por terceras personas ajenas al proyecto.
- Hacer uso de herramientas básicas en el desarrollo de software, como son los **Sistemas de Control de Versiones** para llevar un control realista del desarrollo del software, así como hacer de las veces de sistema de copias de seguridad.
- Utilizar siempre licencias libres, tanto para los contenidos multimedia como para el código. En este sentido, este proyecto es muy estricto, y no deberá hacer uso de ninguna herramienta, recurso ó biblioteca no libre.

  Siempre que nos referimos a software libre nos basamos en la definición dada por Richard M. Stallman en su libro *Software libre para una sociedad libre* [14]:

DEFINICIÓN:

Un programa es software libre siempre que, como usuario particular, tengas:

- La libertad de ejecutar el programa sea cual sea el propósito,

- La libertad de modificar el programa para ajustarlo a tus necesidades. (Para que se trate de una libertad efectiva en la práctica, deberás tener acceso al código fuente, dado que sin él la tarea de incorporar cambios a un programa es extremadamente difícil)
- La libertad de redistribuir copias, ya sea de forma gratuita, ya sea del pago de un precio.
- La libertad de distribuir versiones modificadas del programa, de tal forma que la comunidad pueda aprovechar las mejoras introducidas.

En este sentido, ya que todas las herramientas a utilizar serán libres, tanto la aplicación, como la documentación de la misma, se liberarán con una licencia libre, de forma que se aproveche para hacer retro-alimentación a la comunidad de software libre.

- Manuales y referencias fácilmente entendibles, tanto para la instalación, uso de la aplicación, y generación de sistemas expertos acordes al entorno.

1.3. Versiones anteriores de la aplicación

Como se ha comentado, esta aplicación no es una nueva creación, si no que es una mejora más en el desarrollo del proyecto *Batalla del Guadalete*. A continuación se presenta una visión general de la evolución del proyecto:

1.3.1. Batalla del Guadalete - 2007

La primera versión (realizada por Manuel Palomo Duarte, tutor de este Proyecto Fin de Carrera y profesor coordinador de la asignatura de Diseño de Videojuegos) consistía en una serie de scripts en CLIPS que definían las reglas del juego, así como las funciones accesibles por el usuario a la hora de diseñar su sistema experto.

Esta versión funcionaba solo a nivel de consola, como podemos ver en la captura 1.1, y con unas capacidades muy limitadas. Solamente permitía hacer jugar a dos sistemas expertos entre ellos, y con unas funcionalidades muy recortadas en ese sentido. De todas formas, la idea de esa versión fue la de realizar futuras ampliaciones, como así se ha hecho.

1.3.2. Interfaz gráfica para partidas - 2008

Realmente no es una versión en sí, ya que no añade ninguna funcionalidad nueva. Es *simplemente* una representación gráfica, como observamos en la captura 1.2 de una partida. Está realizada por Roberto García Carvajal, escrita en C++ haciendo uso de la biblioteca gráfica SDL. Podemos descargar esta primera versión (junto con la primera versión del núcleo) desde la forja de software de RedIRIS [2].

Si bien no añade nada nuevo (funcionalmente hablando) a la primera versión, si que facilita la visualización de una partida ya jugada.

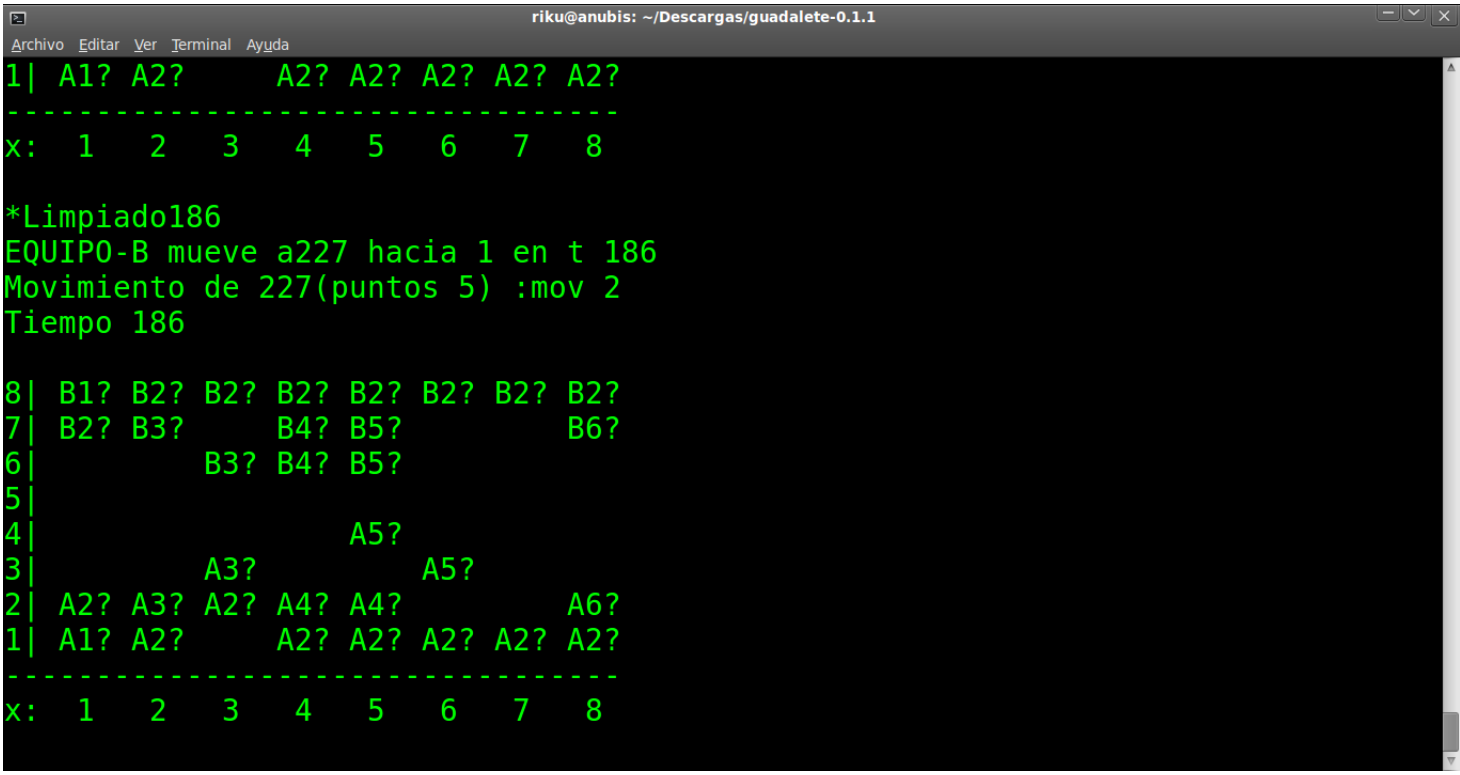


Figura 1.1: Primera versión a consola

1.3.3. La Reconquista - 2009

La reconquista[3] es el intento de ampliar la idea original de **Batalla del Guadalete** para darle una nueva dimensión, siendo realizado por Jesús Soriano Candón como su **Proyecto Fin de Carrera de Ingeniería Técnica en Informática de Sistemas** en el año 2009.

La interfaz incorpora menús, de forma que no es necesario recurrir a la consola para hacer uso de la aplicación. Dicha aplicación está escrita en C++ haciendo uso de SDL.

Con esta aplicación, la idea era poder probar dos sistemas expertos entre si, hacer competiciones entre varios, etcétera. Sin embargo se podría decir que no cumple con unos requisitos mínimos de calidad en varios aspectos:

- Fallos variados en ejecución.
- Rendimiento bastante bajo, consumiendo bastante recursos del sistema, incluso en los menús de espera, especialmente CPU.
- La organización del código no era todo lo clara y coherente que debería así como una ausencia de comentarios explicando el funcionamiento del código.

Estas carencias fueron las que impulsaron la realización de este proyecto.

[2] http://forja.rediris.es/frs/?group_id=497&release_id=1127
[3] https://forja.rediris.es/projects/lareconquista/

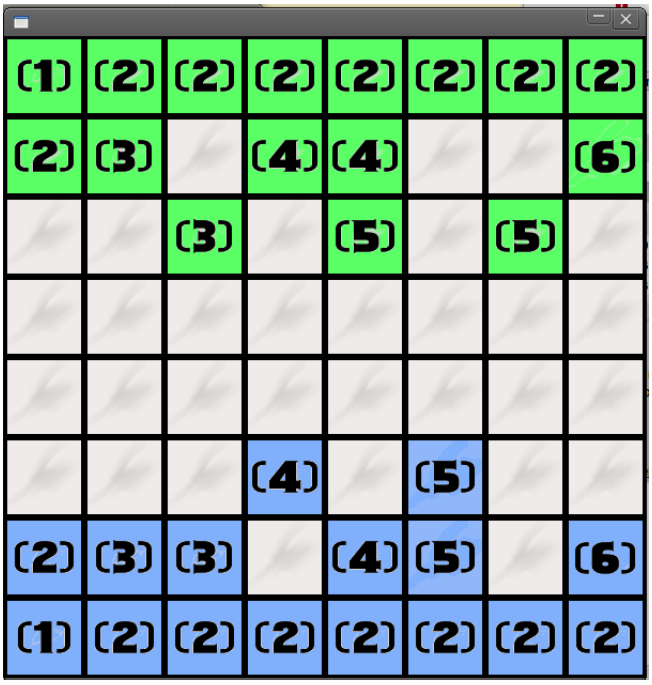


Figura 1.2: Primera versión gráfica

1.4. Punto de partida

Este proyecto que nos ocupa no es tanto un siguiente paso en la iteración del proyecto, si no una rama distinta en dicha evolución, a partir de un enfoque distinto a las anteriores versiones. Gráficamente 1.4 la evolución se podría tomar como un árbol en el que la raíz es el núcleo original, y este proyecto es una rama nueva del árbol, independiente de la anterior linea de desarrollo.
Era por tanto necesario hacer uso del núcleo original de la aplicación, para darle una dimensión similar a la dada en el proyecto **La reconquista** 1.3.3, evitando los fallos y problemas que este tenía, así como incluyendo nuevas funcionalidades y características que hagan de esta aplicación el entorno necesario para ser utilizado en el contexto de una asignatura dedicada (al menos en parte) al aprendizaje de sistemas expertos basados en reglas.

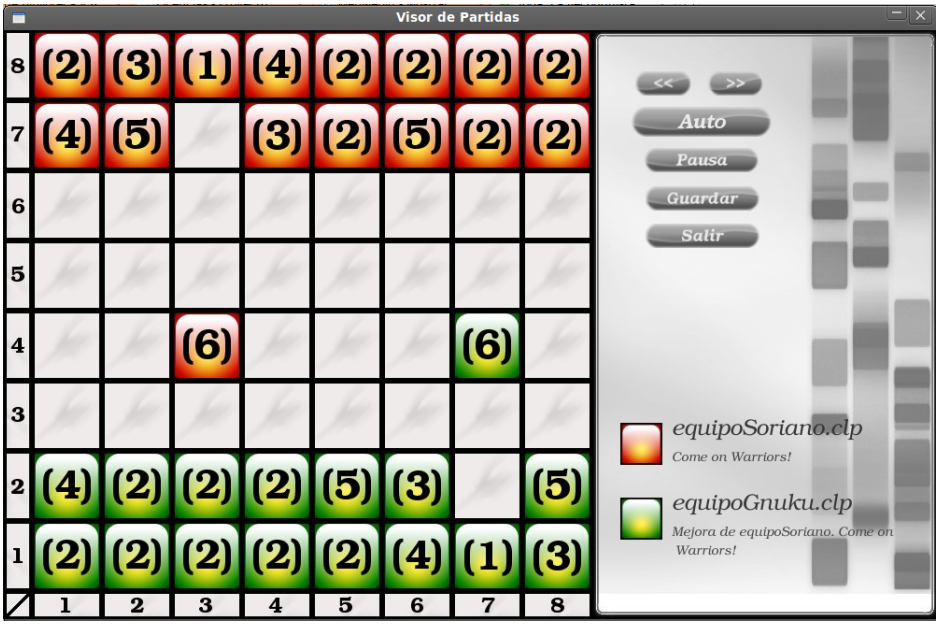


Figura 1.3: La reconquista

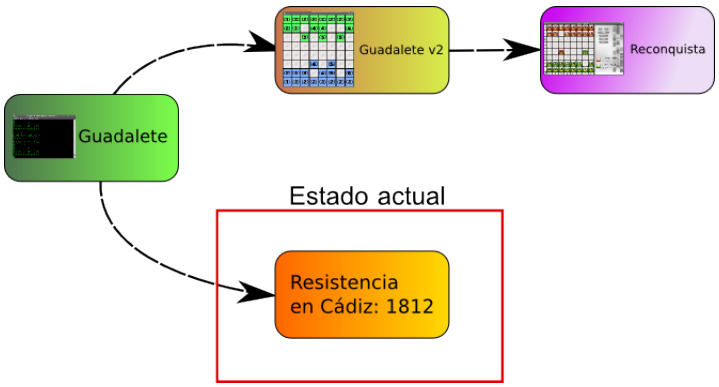


Figura 1.4: Evolución del proyecto Guadalete

Capítulo 2

Conceptos básicos

A lo largo de este capítulo definiremos las bases teóricas necesarias para poder afrontar el desarrollo de este proyecto y la aplicación con la que se culminará el mismo.

Es necesario conocer ciertos conceptos como son los **sistemas expertos**, los cuales son la base de la aplicación. Concretamente, trabajaremos usando **sistemas expertos basados en reglas**, los cuales son una variedad mas especializada de los sistemas expertos genéricos.

Por último, se explicarán las reglas del Stratego, tanto de la versión original del juego de mesa, como de la versión simplificada que utiliza el núcleo de la aplicación.

2.1. Sistemas expertos

La definición de sistema experto dada por el profesor Edward Feigenbahum de la Universidad de Stanford: "un programa de computación inteligente que usa el conocimiento y los procedimientos de inferencia para resolver problemas que son lo suficientemente difíciles como para requerir significativa experiencia humana por su solución"[5].

En palabras más llenas, un sistema experto es un programa que **simula** la habilidad para tomar decisiones de un especialista humano.

Los sistemas expertos son una rama de la Inteligencia Artificial que hace un uso amplio del conocimiento especializado para resolver problemas, tal y como hace un especialista humano. Éste es una persona que tiene una **experiencia** desarrollada en cierta área, permitiéndole utilizar dicha experiencia para resolver problemas que otros no podrían, o en cualquier caso, de una forma más eficiente que el resto de las personas.

En la figura 2.1 podemos ver un diagrama de bloques ilustrando el funcionamiento base de un sistema experto basado en conocimiento. Como podemos ver, el sistema consiste en un usuario aportándole hechos o información y recibe el consejo o la experiencia como respuesta. En su interior, el sistema experto incluye dos componentes principales. La **base de conocimiento** contiene el conocimiento que le permite al **mecanismo de inferencia** sacar conclusiones; éstas son las respuestas del sistema experto

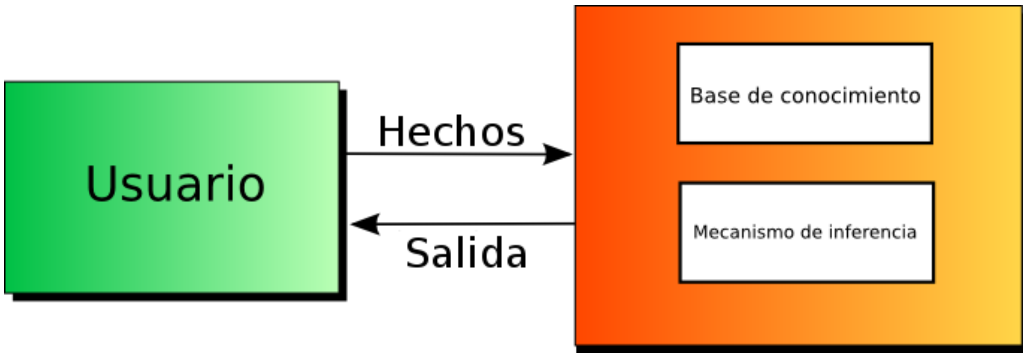


Figura 2.1: Funcionamiento base de un sistema experto

a la consulta especializada del usuario.

En oposición a otras técnicas de solución general de problemas, el conocimiento de un especialista se centra específicamente en el **dominio del problema**. Esto es el área específica en la cual podemos enmarcar el problema: física, medicina, ingeniería, finanzas, etc., en el que un especialista puede resolver problemas con relativa facilidad. Esto se aplica también en la construcción de sistemas expertos, ya que por ejemplo no es posible aplicar la misma metodología en un sistema experto pensado para jugar al ajedrez con otro pensado para ser utilizado en el campo de la medicina. Es decir,**las experiencias en un dominio de problema no llevan automáticamente a otro**.

Al conocimiento del especialista para resolver problemas específico lo denominamos **dominio de conocimiento** del experto. Por ejemplo, un sistema experto médico, diseñado para diagnosticar enfermedades infecciosas debe tener una gran cantidad de conocimiento acerca de los síntomas causados por este tipo de enfermedades. Lógicamente, el dominio de conocimiento de un experto está incluido dentro del dominio de problema, existiendo siempre un área de cierta incertidumbre y desconocimiento.

Dentro de su dominio de conocimiento, un sistema experto razona o hace **inferencias** de la misma forma en que un especialista humano inferiría la solución de un problema. Esto es, dados ciertos hechos, se infiere una solución. Por ejemplo, si nuestro ordenador no funciona correctamente en varios sistemas operativos distintos, podemos inferir que se trata de un fallo físico del equipo.

2.1.1. Ventajas de los sistemas expertos

Los sistemas expertos poseen bastantes características que los hacen atractivos para cierto tipo de problemas:

- Funcionamiento apropiado en entornos de incertidumbre. Esto suele ser uno de los grandes problemas de los métodos clásicos de inteligencia artificial como el recorrido de árboles / grafos. Estos algoritmos clásicos basados en coste (poda alfa-beta, mini-max, ...) no se comportan correctamente en entornos de cierta incertidumbre.

  Supongamos un problema de clásico de camino mínimo en un grafo, es decir buscar el camino mínimo entre dos nodos A y B, pero añadiéndole la condición de que no conocemos realmente que nodo es B hasta que no estemos en él. ¿Cómo podemos realizar el algoritmo de Dijkstra (por ejemplo) si no sabemos cual es el nodo objetivo?

En este problema nos puede ayudar usar un sistema experto. Dicho sistema no tiene un conocimiento total del problema, pero a raíz de su experiencia puede inferir que camino es el que hay que elegir no siendo necesariamente la mejor solución, pero si una suficientemente buena.

- Mayor disponibilidad que un experto real. Un sistema experto tiene su experiencia disponible en cualquier momento. Se podría decir que un sistema experto es un almacén y productor masiva de experiencia.
- Costo reducido. Conseguimos poner la experiencia al servicio del usuario a un coste bastante menor de lo normal.
- Respuestas sólidas, completas y fundamentadas. Esto puede ser importante en tiempo real y en situaciones de emergencia.

2.2. Sistemas expertos basados en reglas

La **base del conocimiento de un sistema experto**, como se ha comentado anteriormente, es el conjunto formado por todo el conocimiento que tiene el sistema en un momento dado. Nuestro sistema experto es un sistema experto basado en reglas, por lo que el conocimiento **permanece**, es decir, a lo largo del tiempo no va asimilando nuevo conocimiento, sino que siempre mantiene el mismo. Concretamente, en este caso el conocimiento se almacenará en forma de *hechos* (afirmaciones sobre el sistema) y *reglas* (que nos permitirá inferir información).

El **motor de inferencia** es el encargado de obtener conclusiones a partir de los datos aplicando reglas. Por *regla* se entiende una proposición lógica que relaciona dos o más objetos e incluye dos partes: *la premisa* y *la conclusión*. Cada una de las partes consiste en una expresión lógica con una o más afirmaciones objeto-valor conectados mediante *operadores lógicos* ('y', 'o', ó 'no')

Un ejemplo de regla puede ser el siguiente:

```
SI la luz es roja ENTONCES detener el vehículo
SI hace frío ENTONCES subir la temperatura
```

Si existe el hecho de que la luz sea roja, esto concuerda con el patrón "la luz es roja", de forma que se cumplió dicha regla, disparándose la acción "detener el vehículo".

Algunos sistemas expertos permiten incluso aprender reglas dinámicamente, a través de la **inducción de reglas**, en la que el sistema crea reglas a partir de tablas de datos. De todas formas no es fácil dar forma de reglas al conocimiento de los especialistas, sobre todo en campos sin explorar de forma sistemática, pudiendo existir inconsistencia, ambigüedades, duplicaciones u otros problemas que pueden no salir a la luz hasta que se trata de representar formalmente el conocimiento del especialista en un sistema experto.

CLIPS[1] es una herramienta que provee un ambiente de desarrollo para la producción y ejecución de sistemas expertos. En términos comunes, su único cometido es aplicar un conjunto de reglas del tipo visto anteriormente a un conjunto de datos.

Las reglas son la parte fundamental del sistema experto. Son de la siguiente forma:

[1] http://clipsrules.sourceforge.net/

```
(defrule biblioteca-regla-1
  (libro (titulo ?X) (estado retraso) (prestado-a ?Y))
  (persona (nombre ?Y) (dirección ?Z))
=>
  (mandar-nota-retraso ?X ?Y ?Z))
```

Traducido a lenguaje natural sería algo como:

```
Biblioteca-Regla-1:
Si
Existe un libro con título X, con retraso y prestado a una persona
  con nombre Y
Y La dirección de esa persona es Z
Entonces
Mandar una nota de retraso a Y en Z del libro X.
```

Los sistemas expertos clásicos suelen tener un conjunto de reglas fijo y un conjunto de hechos en cambio constante. Sin embargo, se comprueba que en la mayoría de sistemas expertos, la mayor parte de este conjunto también permanece constante al aplicar las reglas. Aunque siempre se añaden hechos nuevos y algunos viejos se borran, la proporción de hechos que cambian por unidad de tiempo es más bien pequeña. Por esta razón, la implementación directa del entorno para sistemas experto es muy ineficiente. *CLIPS* utiliza una implementación del algoritmo denominado **Rete**. Este algoritmo se implementa como una *red de nodos*, cada uno de los cuales representa una comprobación de la *parte izquierda de una de las reglas*. Los hechos que se añaden o borran de la base de conocimiento se procesan en esta red de nodos. En la parte inferior de la red están los nodos de reglas individuales. Cuando un conjunto de hechos se filtra por todo el camino hasta la parte inferior quiere decir que ha pasado todas las comprobaciones de una regla en particular y dicha se activa.

El algoritmo que elige la regla que se ejecuta del conjunto de reglas activas en un momento se llama algoritmo de resolución de conflictos, dado depende de la implementación en concreto del algoritmo *Rete* que se haga. Existen implementaciones que emplean técnicas deterministas que dependen del orden en el que se introdujo las reglas en el sistema, y por otro lado, existen implementaciones que usan métodos aleatorios o pseudoaleatorios para decidirlo.

2.3. Stratego

Stratego es un juego de mesa para dos jugadores, originalmente con un tablero de 10x10 y 40 fichas por cada jugador, representando cada ficha un cargo militar, asociado a un valor. Podemos ver una captura de una versión para computadora en la imagen 2.2.

El objetivo del juego es capturar la bandera rival, o dejar al rival sin ninguna ficha con capacidad ofensiva (es decir, que pudiera moverse).

El desarrollo del juego consiste en dos ejércitos, uno de color rojo y otro de color azul. Las piezas tienen dos caras: una se indica el rango de la ficha, y en la otra solo tiene el color del ejército, siendo el mismo para todas las fichas. De esta forma, el rival en principio, solo ve las posiciones iniciales de las otras piezas.

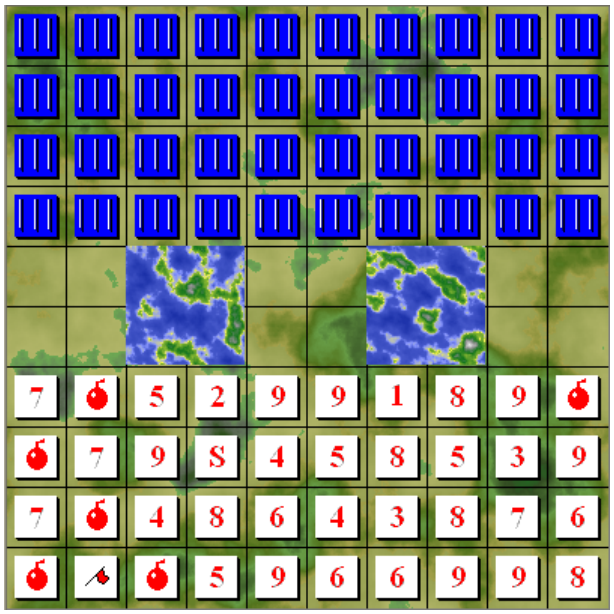


Figura 2.2: Stratego

Cada jugador mueve una única pieza por turno desplazándose solo una casilla de distancia, siendo cuatro las posibilidades: arriba, abajo, izquierda ó derecha. Si la pieza se mueve a una casilla libre, el turno pasa al otro jugador. Si una pieza se mueve a una casilla ocupada por una unidad del ejército rival se considera ataque y ambas descubren su identidad (girándolas). Pueden darse 3 casos:

- La ficha atacante tiene mayor rango que la ficha defensora. En ese caso la ficha defensora es eliminada y la ficha atacante toma su posición.
- La ficha defensora tiene mayor rango que la ficha atacante. En este caso la ficha atacante es eliminada y la ficha defensora mantiene su posición.
- Ambas fichas tienen el mismo valor. En este caso ambas fichas son eliminadas, aunque existen otras versiones.

Sin embargo hay ciertas excepciones en estas reglas:
En la tabla 2.1 podemos ver el valor de las piezas y la cantidad otorgada de cada una al principio de la partida

- **La bomba** es una pieza especial. Esta pieza no puede moverse y solo puede ser eliminada por **el artificiero**, mientras que la bomba elimina a cualquier otra pieza que la toque.
- Otra excepción es **el espía**, que es la única capaz de eliminar a la ficha de mayor rango, **el Mariscal**. Sin embargo el espía pierde cualquier enfrentamiento con otra ficha o si es atacado por el Mariscal.
- Hay casillas que no pueden ser ocupadas por fichas.
- Por último, **el explorador** es la única ficha que puede dar más de un paso en un turno.

Nombre	Valor	Cantidad
Mariscal	10	1
General	9	1
Coronel	8	2
Comandante	7	3
Capitán	6	4
Teniente	5	4
Sargento	4	4
Minador	3	5
Explorador	2	8
Espía	S	1
Bomba	S	6
Bandera	S	1

Tabla 2.1: Valores y cantidades de las piezas del `Stratego`

2.3.1. La batalla del Guadalete

Esta es una versión simplificada del Stratego[4], creada por Manuel Palomo Duarte[2]

Se realiza en un tablero de 8x8, con 16 piezas por jugador, con valores comprendidos entre 1 y 6. El movimiento y ataques de las piezas es igual que en el Stratego original, obviando piezas especiales y excepciones. En esta versión, simplemente las piezas más fuertes ganan a las piezas más débiles, y hay que conquistar al **rey**, que es la pieza de valor 1.

Haciendo una analogía con el ajedrez, podríamos resumir las piezas como se muestra en la tabla 2.2

Nombre	Valor	Cantidad
Reina	6	1
Torre	5	2
Alfil	4	2
Caballo	3	2
Peón	2	8
Rey	1	1

Tabla 2.2: Valores y cantidades de las piezas de `La Batalla del Guadalete`

De todas formas, dada la naturaleza modular del núcleo de la aplicación es posible hacer modificaciones sobre esas reglas, por ejemplo bloqueando alguna posición del tablero, impidiendo que el Rey se mueva y un largo etcétera.

[2]Batalla sucedida en la provincia de Cádiz en el año 711, marcando el comienzo de la ocupación musulmana en la península ibérica. http://es.wikipedia.org/wiki/Batalla_de_Guadalete

Capítulo 3

Calendario

Para la realización de este proyecto se ha seguido un modelo de desarrollo **iterativo incremental**, de forma que hemos ido ocupándonos de problemas más pequeños poco a poco, pudiendo abarcar la totalidad del proyecto de una forma más escalonada.

3.1. Iteraciones

A continuación se presentan brevemente las distintas iteraciones realizadas durante el desarrollo del proyecto. Estas iteraciones se desarrollarán en profundidad más adelante en el capítulo.

3.1.1. Primera iteración: Preliminares

Dada la magnitud del proyecto y la existencia de versiones anteriores, era necesaria una etapa que me permitiera analizar dichas versiones, para ver hasta que punto debía continuar con la linea de desarrollo marcada, o diverger del camino marcado.

En esta iteración también se decide que lenguaje de programación utilizar, herramientas, bibliotecas, etcétera. Así como también servirá para aprender dichas herramientas que conozcamos.

3.1.2. Segunda iteración: Reescritura del núcleo

Una vez elegido el lenguaje de programación **Python**, se vio la necesidad de reescribir el núcleo de la aplicación haciendo uso de la biblioteca **PyCLIPS**, dándole la forma de paquete[1] con una buena API para su uso apropiado y sencillo.

3.1.3. Tercera iteración: Interfaz gráfica

En esta etapa de desarrollo se realizó (de forma general) la interfaz de usuario en base a los requisitos del sistema. A esta interfaz se le incorporaron inicialmente las funcionalidades del proyecto: realizar

[1] En el sentido "Python" del término. Se entiende como paquete un conjunto de módulos.

partidas entre dos sistemas expertos, la generación y almacenamiento de logs, así como un visor de partidas antiguas.

3.1.4. Cuarta iteración: Competiciones y pruebas automáticas

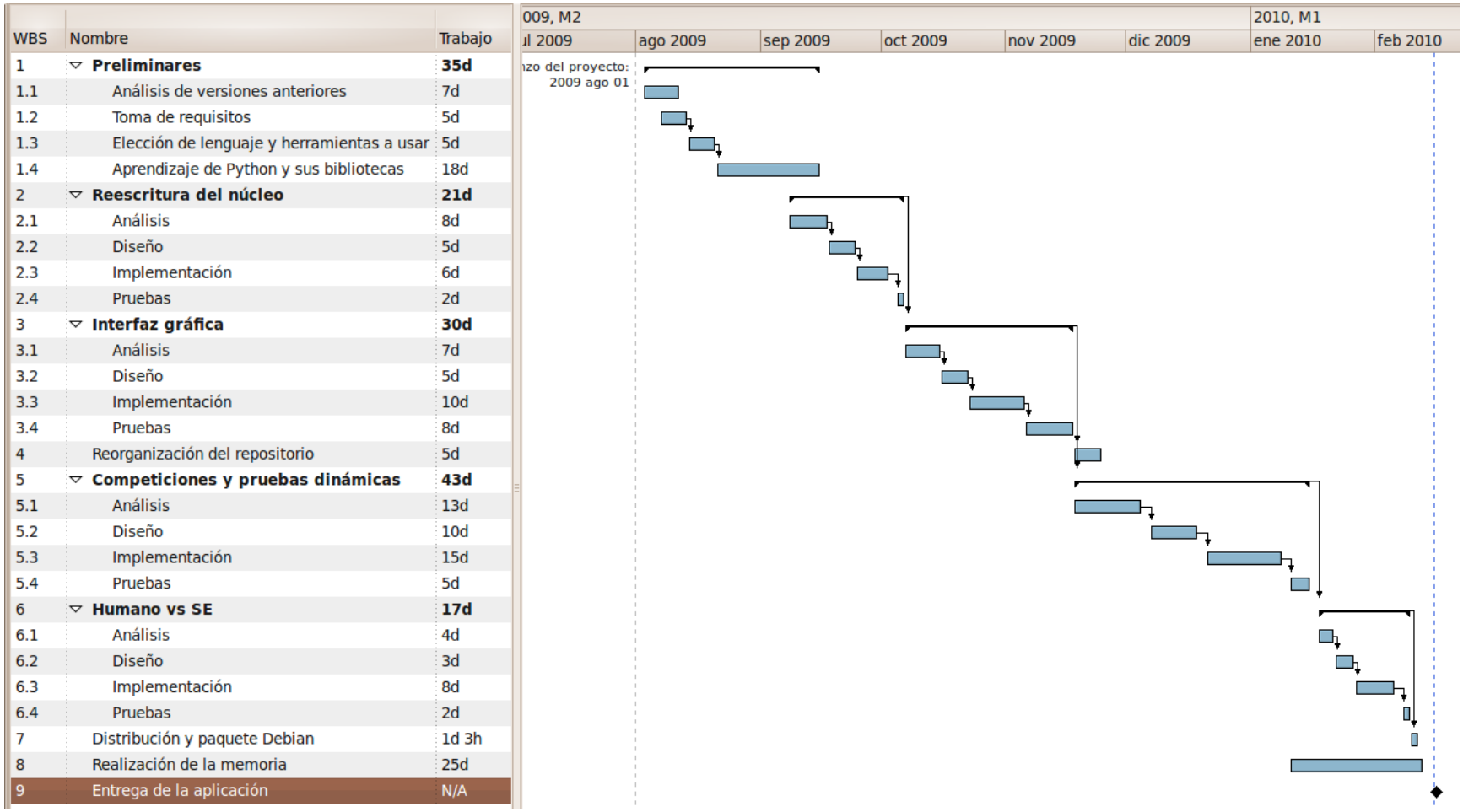
En esta nueva etapa de desarrollo hasta la fecha los objetivos marcados fueron los de implementar un modo de competiciones configurables y ampliables, así como utilizar estos desarrollos para realizar pruebas sobre los sistemas expertos y obtener unas estadísticas que nos puedan ayudar a verificar en cierta medida la bondad del sistema experto.

3.1.5. Quinta iteración: Partidas humano contra Sistema Experto

En esta última etapa de desarrollo, el objetivo fue dotar al sistema de la posibilidad de que un usuario pueda jugar directamente contra un sistema experto. La idea de esto es que el usuario pueda forzar ciertas situaciones para ver como se comporta su sistema experto, y así verificar su corrección.

3.2. Diagrama de Gantt

Podemos ver en la figura 3.1 el diagrama de Gantt con las fechas de realización de las distintas partes de cada una de las iteraciones.



WBS	Nombre	Trabajo
1	**Preliminares**	**35d**
1.1	Análisis de versiones anteriores	7d
1.2	Toma de requisitos	5d
1.3	Elección de lenguaje y herramientas a usar	5d
1.4	Aprendizaje de Python y sus bibliotecas	18d
2	**Reescritura del núcleo**	**21d**
2.1	Análisis	8d
2.2	Diseño	5d
2.3	Implementación	6d
2.4	Pruebas	2d
3	**Interfaz gráfica**	**30d**
3.1	Análisis	7d
3.2	Diseño	5d
3.3	Implementación	10d
3.4	Pruebas	8d
4	Reorganización del repositorio	5d
5	**Competiciones y pruebas dinámicas**	**43d**
5.1	Análisis	13d
5.2	Diseño	10d
5.3	Implementación	15d
5.4	Pruebas	5d
6	**Humano vs SE**	**17d**
6.1	Análisis	4d
6.2	Diseño	3d
6.3	Implementación	8d
6.4	Pruebas	2d
7	Distribución y paquete Debian	1d 3h
8	Realización de la memoria	25d
9	Entrega de la aplicación	N/A

Figura 3.1: Diagrama de Gantt

Capítulo 4

Iteración 1: Preliminares

4.1. Análisis de versiones anteriores

Como vimos en el capítulo introductorio, ese proyecto no es de nueva creación desde cero, si no que parte como continuación de versiones anteriores. En este punto había que definir el siguiente aspecto: **¿desde dónde podíamos partir en el desarrollo?** Es decir: ¿la versión anterior, La Reconquista 1.3.3 tiene una calidad aceptable para ser susceptible a una ampliación? ¿era factible arreglar los fallos para seguir el desarrollo? ¿o por el contrario era necesario hacer borrón y comenzar con una aplicación nueva? Para ello, se realizaron pruebas de ejecución, se analizó el código y la documentación, y se llegaron a las siguientes conclusiones:

- La aplicación era ineficiente en lo que a consumo de recursos se refiere, necesitando una gran cantidad de tiempo de CPU y de memoria principal en el sistema, enlenteciendo el equipo demasiado para el peso de la aplicación.
- La gestión de eventos en los menús tenía varios fallos, ya que en ocasiones no capturaba los eventos de ratón de una forma apropiada, y obliga a pulsar varias veces en un botón para acceder a su funcionalidad.
- Se percibieron fallos en las competiciones, haciendo que a veces en las ligas se sumasen mal los puntos obtenidos por victorias ó derrotas.
- Poco portable entre plataformas y configuraciones de equipos. En equipos de baja resolución, la visualización no era correcta.
- Poca tolerancia a errores del usuario, el sistema se colgaba ante determinadas acciones no contempladas del usuario.
- La documentación de las clases y funciones no estaba completa, dificultando su modificación.
- La organización de los ficheros, así como la nomenclatura de las funciones y variables era poco clara.

Realmente esta versión necesitaba muchas mejoras, y teniendo en cuenta la organización del código y la ausencia de documentación técnica de la aplicación, se optó por hacer una **reescritura** de la aplicación, con el fin de utilizar un enfoque más sistemático durante el proceso de desarrollo para conseguir un

producto de mayor calidad, tanto a nivel de funcionalidades como a nivel de código.

Si nos remontamos a las anteriores versiones, lógicamente la versión original del núcleo sería utilizable, ya que al fin y al cabo es la base de toda la aplicación.

4.2. Toma de requisitos iniciales

Tras diversas reuniones con mi tutor y creador del núcleo original, Manuel Palomo, se fueron desarrollando los requisitos que debería cumplir la aplicación y para ser válida para su correcta utilización y aprovechamiento:

- Permitir **jugar entre dos sistemas expertos** distintos, teniendo en cuenta que los dos han sido diseñados pensando que son el ejército A. Una partida debe ser configurable, permitiendo la elección del número de turnos o poder ocultar los valores de las piezas para dar mayor interés a la partida.
- Dar soporte para la realización de diversos formatos de competiciones, como son **ligas**, **copas** o **eliminatorias(playoffs)**. Estas deben ser fácilmente configurables, permitiendo rondas de ida y vuelta o mostrar solo los resultados.
- Generación de ficheros de logs de todas las partidas jugadas, usando una buena nomenclatura, y almacenándolos de forma ordenada en el sistema.
- El núcleo debe reescribirse de forma que este sea ampliable mediante módulos externos que añadan más reglas y restricciones de uso.
- Poder realizar estadísticas de un sistema experto, evaluando a partir de estas cómo de bueno es un sistema experto.
- Integrarse correctamente en un sistema GNU/Linux, haciendo uso correcto de los directorios del sistema, y respetando la metodología de desarrollo en dichos sistemas.
- La aplicación debe ser internacionalizable para favorecer una posible expansión más allá de la comunidad hispano-hablante.
- Permitir que el usuario pueda jugar directamente contra un sistema experto, con el fin de forzar ciertas situaciones y comprobar como se comporta el sistema.

4.3. Primer análisis informal

Al inicio, hice un pequeño análisis inicial, teniendo en cuenta el objetivo final del proyecto. Esto se hizo con la idea de tener más o menos claro de la forma que podría llegar a tener la aplicación una vez terminado el desarrollo.

En ningún momento se ha tenido en cuenta este análisis para el desarrollo de la aplicación, si no que sirvió para ir investigando que herramientas necesitaría usar y aprender a utilizarlas correctamente.
Podemos ver en el diagrama 4.1 la división inicial que analicé que podría resultar la aplicación. Al final veremos si se cumple, o por el contrario, ha sido totalmente diferente.

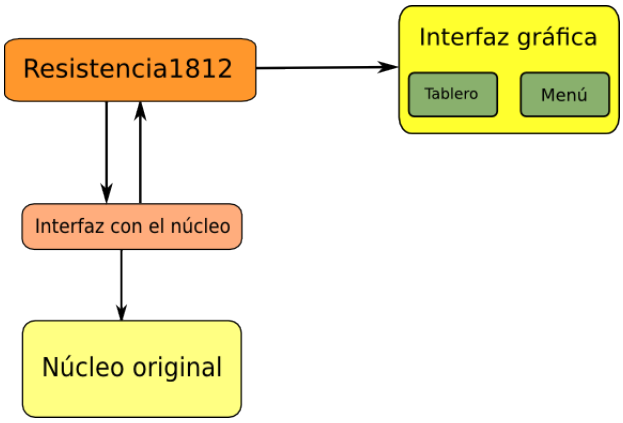


Figura 4.1: Diagrama de bloques inicial

Es importante notar que cada bloque no tiene relación con ninguna entidad de programación al estilo de paquetes, módulos o clases, ya que a estas alturas del proyecto, aun no hemos definido que lenguaje de programación vamos a utilizar.
La idea de cada bloque indicado en el diagrama 4.1 es la siguiente:

Núcleo original: Este bloque representa el núcleo original al cual hacemos referencia en la sección 1.3.1, con algunas modificaciones si fueran necesarias. Será el motor fundamental de la aplicación

Interfaz con el núcleo: En este bloque se realizará una capa de abstracción, que permita acceder a las funcionalidades del núcleo de una forma sencilla y relativamente nativa para el lenguaje de programación utilizado.

Interfaz gráfica: Como su nombre indica, en este bloque se ubicará la interfaz gráfica del usuario. Probablemente sea necesario dividir este módulo en dos, para distinguir de la representación del tablero, con la representación de los menús.

Resistencia1812: Es el bloque que gestionará la comunicación entre el núcleo y la interfaz gráfica, y aportará las funcionalidades principales de la aplicación, como la realización de torneos, pruebas, logs de partidas, etcétera.

4.4. Elecciones iniciales

Si bien es cierto que a la hora de comenzar el análisis y el diseño de un sistema tenemos que abstraernos de la tecnología que vamos a utilizar, me pareció importante definirlo en una etapa temprana de desarrollo para que, en el caso de que las elecciones fueran herramientas desconocidas para mi, paralelizar lo más posible el aprendizaje de las mismas, con el comienzo formal del proyecto.

Dada la intencionalidad ampliable de la aplicación, se debería elegir un lenguaje de programación que soporte **orientación a objetos** para poder encapsular de una forma intuitiva todas las funcionalidades. Además debemos suponer futuras ampliaciones de esta aplicación, y en este aspecto la programación orientada a objetos nos da un entorno apropiado para la reutilización de los módulos generados en el transcurso del desarrollo.

Dado que la idea era la de una aplicación de escritorio, y que se necesitaba que contara con dicha orientación a objetos, inicialmente se propusieron tres lenguajes de programación posibles: **C++**, **Python** y **Java**. A continuación desglosamos las características generales que son relevantes en este caso que nos ocupa de dichos lenguajes, en forma de ventajas y desventajas.

4.4.1. C++

Ventajas

- Lenguaje de propósito general con el cual tendría pocas restricciones si lo controlo bien.
- Es un lenguaje con el que he trabajado en otros proyectos por lo que evitaría el tiempo de aprendizaje del mismo.
- Amplia disponibilidad de bibliotecas tanto para interfaces gráficas, multimedia, etc.
- Lenguaje maduro y con una amplia comunidad.
- Muy eficiente.
- Las versiones anteriores están desarrolladas en C++, permitiéndome reutilizar código si fuera necesario.

Desventajas

- Si se cometen errores de programación del tipo de asignación de memoria, pueden provocar muchos problemas.
- Necesidad de una recompilación cada vez que se cambie de arquitectura de la máquina.
- Aunque existen las bibliotecas Boost, que contiene los nuevos módulos y funcionalidades que serán añadidos al próximo estándar de C++, el estándar del lenguaje está relativamente anticuado.
- El trabajo con ficheros de texto plano es algo arcaico en comparación con otros lenguajes, siendo una necesidad de la aplicación.
- La interacción con el núcleo en CLIPS no es fácil ya que no hay ninguna biblioteca que envuelva bien las funcionalidades de CLIPS.

4.4.2. Python

Ventajas

- Es un lenguaje muy limpio y claro, haciendo que el código sea más fácilmente mantenible y ampliable.
- Gran número de bibliotecas libres utilizables para el desarrollo del proyecto, entre ellas PyCLIPS[1], que es una biblioteca bastante completa para la integración de CLIPS en Python.
- La lectura y procesado de ficheros de texto plano es bastante sencilla.
- A pesar de ser un lenguaje interpretado es bastante rápido.

[1] http://pyclips.sourceforge.net/

- Existen entornos web escritos en Python, permitiendo una posible futura integración de la aplicación en una plataforma web.

Desventajas

- Lenguaje desconocido para mi. Había recibido algún curso de iniciación pero apenas había trabajado con el.
- Aunque es bastante eficiente para ser un lenguaje interpretado, siempre será mas ineficiente que un lenguaje compilado como C++.
- El lenguaje no es estándar ISO, si no que está siendo desarrollado por la comunidad. Eso puede suponer un problema en las actualizaciones de versiones, ya que esas actualizaciones suelen ser bastante grandes.
- Al ser un lenguaje de bastante alto nivel, se pierde el control sobre ciertos conceptos, como la recolección de basura.

4.4.3. Java

Ventajas

- Es el lenguaje que tiene más aplicaciones en Sourceforge, siendo posible encontrar código libre reutilizable para mi aplicación.
- Totalmente multi-plataforma, sin necesidad de recompilación ni tocar nada para que funcionen en un sistema u otro.
- Aunque no al nivel de C++, si que conocía Java lo suficiente para ahorrar mucho tiempo de aprendizaje.

Desventajas

- Es un lenguaje poco eficiente.
- Solo localicé una biblioteca para usar CLIPS[2], pero no es compatible con licencias libres.
- Aunque hay una implementación libre de Java (OpenJDK), su filosofía inicial es poco compatible con la filosofía del software libre.
- Dependiente de los cambios realizados por Sun en las actualizaciones de cada una de las versiones, pues no existe estandar.

Por el simple hecho de la imposibilidad de encontrar una biblioteca libre para Java que trabajase con CLIPS, Java fue descartado como posibilidad, quedando la elección entre Python y C++.

Me decanté por Python por dos motivos fundamentales:

- PyCLIPS es una biblioteca muy completa para el trabajo con CLIPS, y me facilitaría bastante la integración con el núcleo original.
- Aunque no lo conocía, era una buena oportunidad para aprender un nuevo lenguaje.

[2] http://www.jessrules.com/

4.4.4. Bibliotecas y herramientas

Una vez seleccionado el lenguaje de programación, y como gracias al diagrama 4.1 tenemos una idea aproximada de las necesidades que podemos llegar a tener en cuanto a bibliotecas y herramientas.

Uno de los factores en la decisión de Python como lenguaje de programación con el que se escribirá esta aplicación es la existencia de la biblioteca **PyCLIPS**. Dicha biblioteca la podemos utilizar para hacer de interfaz entre el núcleo y la aplicación principal.

La elección de la librería a utilizar para la implementación de la interfaz también es importante. En este aspecto, surgen las siguientes alternativas libres:

PyGTK: *Binding* de **GTK** para Python. GTK son las bibliotecas utilizadas en el entorno de escritorio **GNOME**. Tienen dos ventajas fundamentales: bastante sencillo de programar, y una herramienta llamada **Glade**. Esta herramienta permite diseñar gráficamente las ventanas, diálogos y demás elementos de nuestra interfaz, generando un fichero **XML**, el cual se cargará en el código de la aplicación. Para más información, mirar el Anexo (por incluir anexo sobre programación de PyGTK y Glade) para más información.

wxPython: Otro *binding* para Python, pero esta vez de las librerías **wxWidgets**. Tiene la ventaja que al igual que wxWidgets, es multi-plataforma y se integra bastante bien con el sistema que estemos utilizando.

PyQt: Este otro *binding* para Python se realiza utilizando las librerías Qt. Estas librerías son utilizadas en la implementación del entorno de escritorio KDE.

Sobre estas elegí finalmente **PyGTK** fundamentalmente por la herramienta **Glade**. Es bastante interesante poder tener la interfaz totalmente independiente del código, ya que me evitaría modificarlo cada vez que hiciera cualquier cambio sobre la interfaz. Además, esto permite que dichos diálogos puedan ser utilizables por muchos lenguajes de programación, en el caso que hiciera falta incluir alguna funcionalidad que no pueda aportar Python.

La última decisión a tener en cuenta en este momento es la de la forma de representación del tablero de una partida. Partiendo de la base que ya hemos elegido **Python** como lenguaje de programación y **PyGTK** como librería para la interfaz gráfica, las posibilidades a barajar eran dos:

Cairo: Librería de gráficos vectoriales integrada con PyGTK. El resultado es bueno, pero la complejidad de uso es bastante elevada.

Pygame: Librería multimedia construida a partir de **SDL**[3] ampliando y simplificando sus funcionalidades.

Se optó por utilizar **Pygame** con la intención de tener separada la representación del tablero de la interfaz gráfica de menús. Esta independencia se tuvo en cuenta con el fin de poder reutilizar esos distintos módulos en otros proyectos. Además, Cairo es una herramienta muy potente, pero pensada para proyectos más complejos, pues para definir gráficos solo permitía insertar los vectores a código, siendo un método muy propenso a errores.

[3] Una de las librerías más usadas para videojuegos en C/C++

4.5. Aprendizaje de Python y herramientas

Una vez seleccionado el lenguaje de programación y las herramientas, se invirtió aproximadamente 3 semanas para el aprendizaje de Python y las herramientas asociadas.

Durante este proceso de aprendizaje se tomaron varias medidas:

- Búsqueda y lectura de documentación sobre Python. En este punto recurrí a [10], [9] y [12], así como a la documentación oficial de Python [6].
- Creación de pequeños scripts y aplicaciones para ir conociendo las características del lenguaje.
- Búsqueda de aplicaciones con similitudes a esta, con el fin de tener como patrón de diseño una aplicación ya operativa, así como código accesible para ver ejemplos prácticos.

Para el resto de herramientas, el proceso fue similar: construir pequeños ejemplos para ir familiarizándome con los distintos entornos.

Capítulo 5

Iteración 2: Re-escritura del núcleo

Está iteración en el desarrollo del proyecto consistió en el análisis del núcleo original escrito en **CLIPS**, y portarlo al lenguaje de programación **Python** haciendo uso de la biblioteca **PyCLIPS**.

5.1. Estudio del núcleo original

El núcleo de **La Batalla del Guadalete** se compone de una serie de ficheros escritos en **CLIPS**, en los cuales se definen una serie de plantillas, hechos y módulos que son los que inicializan el entorno del juego, así como aportar las funciones accesibles para el programador que implemente un sistema experto para esta aplicación.

5.1.1. Plantillas

Las plantillas en **CLIPS** se podrían asemejar a una estructura de datos de los lenguajes mas *clásicos* estructurados. Con ellas tenemos unas entidades de programación que nos permitan darle la forma que queramos al programa. En este caso, se han definido las siguientes plantillas:

- `ficha-r`: Esta plantilla contiene los datos de una ficha real del sistema, como su equipo, valor, posición, identificado y si está o no descubierta.
- `ficha`: Abstracción de la plantilla `ficha-r` que nos permite *girar el tablero* a la hora de los cálculos si las fichas son de un equipo u otro. De esta forma, siempre se programa el equipo "A"
- `mueve`: Permite realizar movimientos, indicando que ficha es la que se mueve, en que dirección y en que turno.

5.1.2. Módulos

Para diferenciar las distintas necesidades de la aplicación, se hace una división en módulos para agrupar reglas y hechos de la partida. A continuación se describen brevemente cada uno de esos módulos, con el fin de conocer la estructura interna de la aplicación.

Main

El módulo **MAIN** se encarga de controlar el tiempo de juego y de asignar el control a los módulos de **INFORMAR**, **EQUIPO-A** y **EQUIPO-B** cuando les corresponda.
En este módulo se definen las siguientes reglas, las cuales controlan el flujo de ejecución del programa:

- `inicia-tiempo`: Esta regla es la primera que se carga en la partida, inicializándose los parámetros de `tiempo` y `tiempo-inicial`. Esto nos sirve para saber cuantos turnos llevamos jugados en la partida, y cual es el límite de turnos.
- `borra-fich`: Limpia el fichero que se usará como temporal, por si este existiera.
- `control-y-tiempo`: Se encarga de avanzar un turno en la partida y de registrar el estado actual en el fichero de log.
- `control-sin-tiempo`: Se encarga de llamar a los módulos del **EQUIPO-A** y **EQUIPO-B** para que vayan moviéndose los jugadores.

Mover

El módulo **MOVER** gestiona, como su propio nombre indica, los movimientos de las fichas. Para ello hace uso de la regla `movimiento`, en la cual se comprueba si algún jugador ha realizado el aserto `mueve`. Además controla si alguno de esos movimientos ha repercutido en un ataque, eliminando la ficha oportuna (o las dos).

Informar

El módulo **INFORMAR** tiene dos responsabilidades generales: la primera es la de registrar en el fichero de salida el estado actual del tablero. La segunda es la de comprobar si en el estado actual se ha terminado la partida.

TraducirF

Como vimos en el apartado 5.1.1, en esta aplicación hay dos tipos distintos de fichas, una nos sirve como ficha real, es decir con la que trabajan los sistemas expertos diseñados por los usuarios, y la otra le sirve al sistema para abstraerse del equipo al que pertenece la ficha a la hora de hacer los cálculos oportunos.

Este módulo **TRADUCIRF** se encarga de hacer esa conversión de `ficha-r` a `ficha`, para que el sistema pueda hacer los cálculos necesarios.

TraducirM

Los movimientos de las fichas se identifican por un número entero de 1 a 4, que indican:

- 1 indica que se mueve incrementando el valor de x
- 2 indica que se mueve decrementando el valor de x
- 3 indica que se mueve incrementando el valor de y
- 4 indica que se mueve decrementando el valor de y

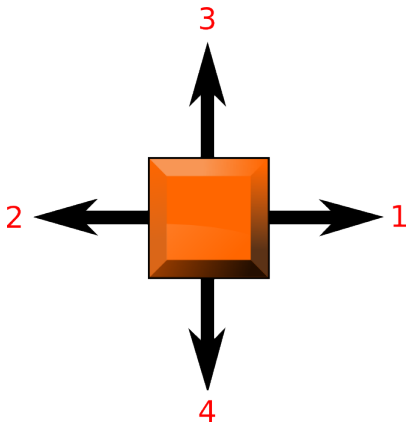


Figura 5.1: Movimientos de una ficha

Esto se puede ver más claramente en el gráfico 5.1. El problema de estos movimientos es que se realizan de esa forma en el supuesto de que la ficha sea del equipo A.

El módulo **TRADUCIRM** lo que hace es realizar esa inversión del movimiento para el equipo B. De esta forma, un usuario puede diseñar su sistema experto pensado para el equipo A, pero en el sistema se puede invertir sin ningún problema.

Equipo-A y Equipo-B

Estos dos módulos **EQUIPO-A** y **EQUIPO-B** son realmente iguales, incluyendo cada uno las reglas básicas de cada equipo. Estas reglas están definidas para que en el caso de que no se realize ninguna regla definida por el usuario, al menos se mueva alguna ficha.

Incorporan además una regla `termina` que es la que finaliza el turno del equipo correspondiente, devolviéndole el foco al módulo **MAIN**.

5.2. Ejemplo de ejecución

Para una correcta ejecución del núcleo, deberemos tener (aparte de los ficheros que contienen los módulos y las funciones) dos ficheros por cada equipo: `reglasA.clp`, `equipoA.clp`, `reglasB.clp` y `equipoB.clp`, que son los ficheros creados por los usuarios que creen algún sistema experto. Pero, ¿en qué consiste cada fichero?:

- `equipo*`: En este fichero tenemos la descripción de la formación del equipo correspondiente, es decir, la distribución inicial de las fichas en el tablero. Un ejemplo lo podemos ver en el siguiente código:

```
(deffacts fichas-A
  (ficha-r (equipo "A") (num 111) (puntos 1) (pos-x 1)
           (pos-y 1) (descubierta 0))
  (ficha-r (equipo "A") (num 112) (puntos 2) (pos-x 2)
           (pos-y 1) (descubierta 0))
```

```
(ficha-r (equipo "A") (num 113) (puntos 2) (pos-x 3)
          (pos-y 1) (descubierta 0))
(ficha-r (equipo "A") (num 114) (puntos 2) (pos-x 4)
          (pos-y 1) (descubierta 0))
(ficha-r (equipo "A") (num 115) (puntos 2) (pos-x 5)
          (pos-y 1) (descubierta 0))
(ficha-r (equipo "A") (num 116) (puntos 2) (pos-x 6)
          (pos-y 1) (descubierta 0))
(ficha-r (equipo "A") (num 117) (puntos 2) (pos-x 7)
          (pos-y 1) (descubierta 0))
(ficha-r (equipo "A") (num 118) (puntos 2) (pos-x 8)
          (pos-y 1) (descubierta 0))
(ficha-r (equipo "A") (num 121) (puntos 2) (pos-x 1)
          (pos-y 2) (descubierta 0))
(ficha-r (equipo "A") (num 122) (puntos 3) (pos-x 2)
          (pos-y 2) (descubierta 0))
(ficha-r (equipo "A") (num 123) (puntos 3) (pos-x 3)
          (pos-y 2) (descubierta 0))
(ficha-r (equipo "A") (num 124) (puntos 4) (pos-x 4)
          (pos-y 2) (descubierta 0))
(ficha-r (equipo "A") (num 125) (puntos 4) (pos-x 5)
          (pos-y 2) (descubierta 0))
(ficha-r (equipo "A") (num 126) (puntos 5) (pos-x 6)
          (pos-y 2) (descubierta 0))
(ficha-r (equipo "A") (num 127) (puntos 5) (pos-x 7)
          (pos-y 2) (descubierta 0))
(ficha-r (equipo "A") (num 128) (puntos 6) (pos-x 8)
          (pos-y 2) (descubierta 0)))
```

- `reglas*:` Con el fichero de reglas definimos el comportamiento del equipo en base a reglas. Estas reglas se componen de una prioridad, unas pre-condiciones y unas post-condiciones. De esta forma, se ejecutará la regla de mayor prioridad que cumpla todas sus pre-condiciones.

  Vemos un posible ejemplo de algunas reglas en el siguiente código:

```
(defrule EQUIPO-A::ficha-6-muerta
   (declare (salience 80))
   (not(ficha (equipo "A") (num ?n)
              (pos-x ?x) (pos-y ?y) (puntos 6)))
   =>
   ;(printout t "La ficha " ?n " esta muerta" crlf)
   (assert (ficha_6_muerta))
)

(defrule EQUIPO-A::fichas-5-muertas
   (declare (salience 80))
   (not(ficha (equipo "A") (num ?n)
              (pos-x ?x) (pos-y ?y) (puntos 5)))
```

```
    =>
    (assert (fichas_5_muertas))
)

(defrule EQUIPO-A::adelanta-6-1
  (declare (salience 50))
  (tiempo ?t)
  (ficha (equipo "A") (num ?n)
         (pos-x ?x) (pos-y 2) (puntos 6))
  =>

  (assert (mueve (num ?n) (mov 3) (tiempo ?t)))
)

(defrule EQUIPO-A::adelanta-6-2
  (declare (salience 49))
  (tiempo ?t)
  (ficha (equipo "A") (num ?n)
         (pos-x ?x) (pos-y 3) (puntos 6))
  =>

  (assert (mueve (num ?n) (mov 3) (tiempo ?t)))
)

;;; Y sigue muchas reglas mas
```

Una vez tenemos ambos ficheros, podemos ejecutar la aplicación usando la siguiente orden:

```
clips -f instrucc.bat
```

`instrucc.bat` es el fichero que contiene el orden de carga de los distintos ficheros que contienen los distintos módulos, así como preparar el entorno para la carga de los ficheros del usuario. Una vez lo hemos ejecutado obtenemos **dos salidas distintas**.

La primera salida se produce en **la terminal**, y es bastante larga. En ella se puede ver el proceso de la partida, así como las reglas y módulos que se van activando durante el proceso. Un ejemplo del final de esa salida sería:

```
Tiempo 99

8|             B2? B2? B2?
7|             B4?
6|         B2?     B5?     B2?
5|             B1?
4|         A1? A4  A2? A5?
3|         A2? A3? A2?
2|         A2? A2?
1|             A2?
```

```
---------------------------------
x:  1   2   3   4   5   6   7   8
 Modulo->TRADUCIRF *Limpiado98
 Modulo->EQUIPO-B EQUIPO-B mueve a111 hacia 3 en t 98
 Modulo->TRADUCIRM Traducido mov ficha-r n111de  3 a 4
 Modulo->MOVER Ataque con derrota de 111(puntos 1) : mov 4
Pasamos al modulo INFORMAR.
Tiempo 98

8|             B2? B2? B2?
7|             B4?
6|         B2?     B5?     B2?
5|
4|         A1? A4  A2? A5?
3|         A2? A3? A2?
2|         A2? A2?
1|             A2?
---------------------------------
x:  1   2   3   4   5   6   7   8
Empate, fin del tiempo
Rey del equipo B muerto
CLIPS> ;(facts)
;(exit)
```

La partida por defecto tiene 200 turnos, y en esta se han jugado 102 turnos, y podemos verla durante todo el proceso mediante la consola. Luego se genera otra salida distinta, en un fichero `resultado.txt`, el cual tiene almacenados de una forma más estructurada los distintos estados del tablero en un turno. La misma partida del ejemplo anterior genera el siguiente fichero:

```
tiempo
199
 e:A n:125 p:4 x:5 y:2 d:0
 e:B n:124 p:4 x:4 y:7 d:0
 e:A n:128 p:6 x:8 y:2 d:0
 e:A n:117 p:2 x:7 y:1 d:0
 e:A n:127 p:5 x:7 y:2 d:0
 e:B n:117 p:2 x:7 y:8 d:0
 e:B n:116 p:2 x:6 y:8 d:0
...
...
tiempo
198
 e:A n:121 p:2 x:1 y:2 d:0
 e:A n:124 p:4 x:4 y:2 d:0
 e:A n:117 p:2 x:7 y:1 d:0
 e:B n:118 p:2 x:8 y:8 d:0
 e:A n:118 p:2 x:8 y:1 d:0
...
```

El fichero resultante tiene en este caso 3650 líneas de código, así que se indica solo un ejemplo. Cada linea indica las propiedades de una ficha en el tablero, como son su equipo, identificador, puntuación, posición y si está o no descubierta. A partir de los datos de cada una de las fichas en un tiempo determinado, se podría reconstruir el tablero en ese turno perfectamente.

5.3. Reescritura usando PyCLIPS

Para poder incorporar las funcionalidades del núcleo a una aplicación escrita en Python, había que hacer uso de PyCLIPS para trabajar con el núcleo. Pero hay dos formas de que la biblioteca trabajase con lo que ya estaba hecho.

5.3.1. Opción 1: Capa intermedia

Una opción sería traducir el fichero `instrucc.bat` a PyCLIPS. Recordemos que dicho fichero contiene las directivas que tiene que seguir CLIPS a la hora de cargar los distintos ficheros e inicializar el entorno.

Con esto lo que conseguiríamos sería que el fichero resultante sirviera como capa intermedia entre la aplicación. Como podemos ver un diagrama conceptual en el gráfico 5.2, la idea subyacente del gráfico es la de simplemente sustituir el fichero `instrucc.bat` por otro escrito en Python, que cargue los ficheros de la misma forma.

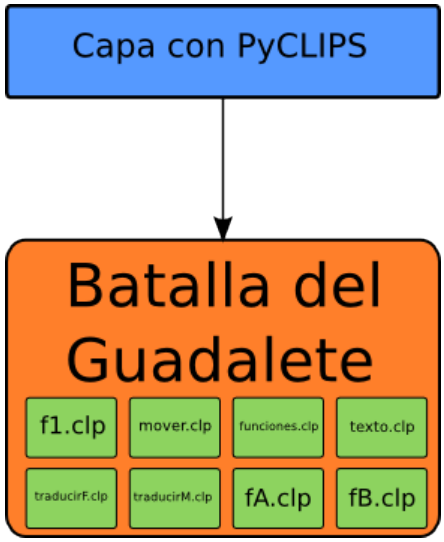


Figura 5.2: Opción 1: Capa intermedia

De esta forma, todos los ficheros que componen **La batalla del Guadalete** permanecerían igual e inalterables.

A continuación se muestra el código del fichero original `instrucc.bat`:

```
;;
;;Por Manuel Palomo Duarte, 2007
;;Disponible bajo los términos de la GNU
;;General Public License (GPL) version 2 o superior
;;
;;Para ejecutar CLIPS y recoger la salida en
;;un fichero de texto se puede hacer:
;;exec clips -f instrucc.bat > salida.log
;;
;;
;;

(clear)
(set-strategy random)
(seed 1234)
(load* "funciones.clp")
(load* "f1.clp")
(load* "mover.clp")
(load* "texto.clp")
(load* "traducirF.clp")
(load* "traducirM.clp")

(load* "./equipoA.clp")
(load* "./equipoB.clp")

(load* "fA.clp")
(load* "./reglasA.clp")
(load* "fB.clp")
(load* "./reglasB.clp")
(reset)
(facts)

(watch facts *)

(run)
;(facts)
;(exit)
```

Podemos ver que realmente es una serie de órdenes de CLIPS para inicializar el entorno y cargar los distintos ficheros con las reglas y funciones, para terminar ejecutando `(run)`, que *dispara* las reglas en el sistema, sucediéndose los *hechos* que definen el avance de la partida.

Ahora, recurriendo a la documentación oficial de PYCLIPS [7] podemos buscar de que forma utilizar esas mismas órdenes de CLIPS para construir un fichero en Python que nos permita la misma funcionalidad. El código resultante es el siguiente:

```
# -*- coding: utf-8 -*-
```

```
import random

import clips

if __name__ == "__main__":
    clips.Clear()
    clips.EngineConfig.Strategy = clips.RANDOM_STRATEGY

    random.seed()
    clips.Eval("(seed " + str(random.randint(0,9999)) + ")")

    clips.Load('funciones.clp')
    clips.Load('f1.clp')
    clips.Load('mover.clp')
    clips.Load('texto.clp')
    clips.Load('traducirF.clp')
    clips.Load('traducirM.clp')

    clips.Load('./equipoA.clp')
    clips.Load('./equipoB.clp')

    clips.Load('fA.clp')
    clips.Load('./reglasA.clp')
    clips.Load('fB.clp')
    clips.Load('./reglasB.clp')

    clips.Reset() #restart the environment

    clips.Run() #start the simulation
    t = clips.StdoutStream.Read()

    print t
```

5.3.2. Opción 2: Traducir cada módulo

La opción vista anteriormente es válida, ya que proporciona las mismas funcionalidades que el núcleo original pero escrito en Python, permitiendo futuras ampliaciones sobre el lenguaje. Sin embargo, ¿existe una forma mejor de hacerlo?

Para ello volvemos a revisar la documentación de PyCLIPS [7] y vemos que no solo se puede cargar ficheros de CLIPS directamente en el entorno, si no que todas las cosas que hacemos en esos ficheros (creación de módulos, reglas, funciones, etcétera) podemos hacerlo usando funciones en Python.

¿Qué significa esto? Por ejemplo, podemos tener la definición de una regla en CLIPS como sigue:

```
(defrule MAIN::borra-fich
  (declare (salience 100))
```

```
  (not (fichero-abierto))
  =>
  (printout t crlf "BORRANDO FICHERO" crlf)
  (assert (fichero-abierto))
  (open "temporal.txt" fich "w")
  (close fich))
```

Que se traduciría en Python de la siguiente forma:

```
# Rule name
rule_name = 'borra-fich'
# Rule precontents
rule_prec  = '(declare (salience 100))' \
             '(not (fichero-abierto))'
# =>
# Rule body
rule_body  = '(printout t crlf "BORRANDO FICHERO" crlf)' \
             '(assert (fichero-abierto))' \
             '(open "temporal.txt" fich "w")' \
             '(close fich)'
# Building the rule
clips.BuildRule(rule_name, rule_prec, rule_body)
```

Si se optara por traducir cada fichero, convirtiendo los módulos, funciones, reglas y hechos CLIPS en funciones. Es una opción más trabajosa pero presenta varias ventajas:

- Al ser cadenas de texto lo que definen las entidades a la hora de construirlas, podemos parametrizar ciertas cosas, por ejemplo:

  ```
  num_turns = 200
  dimension = 8
  turn = 'A'
  base_a = 1
  base_b = dimension
  # Deffacts name
  deffacts_name = "opciones-juego"
  # Deffacts body
  deffacts_body  = "(tiempo-inicial " + str(num_turns) + ")"
  deffacts_body += "(dimension " + str(dimension) + ")"
  deffacts_body += '(turno "' + turn + '")'
  deffacts_body += '(base "A" ' + str(base_a) + ')'
  deffacts_body += '(base "B" ' + str(base_b) + ')'
  deffacts_body += '(modulos INFORMAR TRADUCIRF EQUIPO-A ' \
                   'MOVER INFORMAR TRADUCIRF EQUIPO-B TRADUCIRM MOVER)'
  # Building the Deffact
  opciones_juego = clips.BuildDeffacts(deffacts_name, deffacts_body)
  ```

  Esto permitirá que en un futuro esos valores que definen la partida se puedan generalizar, haciendo el juego ampliable.

- Tener cada regla, cada módulo, cada función, separados y no en un mismo fichero, podrá permitir en un futuro que según se desee, no se carguen ciertas reglas, o añadir otras. Es decir, el núcleo será mas modular.
- Al estar todo escrito en Python, la integración con la aplicación será más natural y probablemente más accesible que si se cargaran solamente los ficheros de CLIPS.

Por tanto, esta etapa se culminó con la reescritura de los distintos ficheros, siguiendo el patrón ilustrado en los códigos anteriores. La idea original de esta etapa de desarrollo que se mostraba en el gráfico 5.2 no ha terminado siendo la realidad implementada. Al final, se ha construido como un único bloque, como podemos ver en la figura 5.3

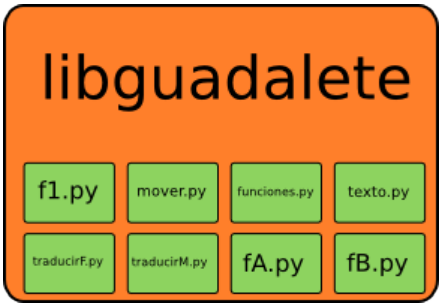


Figura 5.3: Opción 2: Reescritura

Una vez reescritos cada módulo, solamente haría falta reescribir el fichero de ejecución principal `instrucc.txt`, cargando los módulos Python que sustituyen a los ficheros `.clp`.

Durante esta reescritura se detectaron ciertos errores en el núcleo original, en especial en lo que se refiere a la gestión de tiempo y controlar cuando le toca a un equipo u otro. Este fallo se solucionó en una reunión con el director del proyecto, y se pudo solventar sobre la marcha.

5.4. Verificación

Para verificar la corrección de esta iteración en el desarrollo se realizaron ejecuciones del núcleo antiguo y de la nueva reescritura, cotejando ambas salidas.

Evidentemente no tienen por que salir iguales, pues hay un factor de aleatoriedad en la ejecución de una partida, pero sí se verifica que las dos salidas son iguales en forma, así como también generan los ficheros de salida que contienen la información sobre cada estado del tablero.

Siguiendo esta linea de ejecuciones se descubrieron fallos, hasta que finalmente se obtuvo la traducción completamente funcional.

Capítulo 6

Iteración 3: Interfaz gráfica

6.1. Objetivos

Durante la iteración anterior estuvimos realizando una reescritura del núcleo original de **La batalla del Guadalete** para permitir la interacción con una aplicación de escritorio.

A lo largo de esta iteración se comenzará la aplicación de usuario hacia la que está enfocada este proyecto: **Resistencia en Cádiz: 1812**. En principio solo se incluirán las funcionalidades más sencillas, con el fin de que el resto de la aplicación (probablemente) se apoye en éstas. Dichas funcionalidades son básicamente **jugar una partida entre dos sistemas expertos** y poder **visionar una partida antigua**. Pero vamos a hacer el desarrollo apropiado de estas funcionalidades siguiendo las herramientas que nos proporciona la **ingeniería del software**

6.2. Toma de requisitos

Para la toma de requisitos se realizaron varias reuniones con mi director del proyecto, para ir concretando cuales eran las primeras funcionalidades básicas que tenía que tener el sistema. Estas son:

- Permitir jugar dos sistemas expertos entre ellos. Estos sistemas expertos están pensados los dos para ser el equipo A, por tanto deben poderse invertir de una forma correcta, manteniendo las reglas deseadas.
- El número de turnos de una partida es seleccionable por el usuario.
- En una partida, debe poderse seleccionar si ocultar o no el valor de las fichas de una forma real. En el núcleo original esto se representaba poniendo el valor junto a un signo de interrogación `?`, pero se podía ver el valor real de la ficha. Sería interesante poder elegir que no se vean para dar mas interés a una partida.
- En una partida rápida, debe poder elegirse mostrar solo resultado de la partida.
- Mientras que se vea una partida, tiene que ser iterable tanto al incrementar como al decrementar turnos.

- Los ficheros que se leerán para montar los equipos serán de la forma `equipoXXXX.clp` y `equipoYYYY.clp` pudiendo ser `XXXX` e `YYYY` iguales o distintos. En el caso de ser iguales, el sistema deberá interpretarlos como un equipo directamente.
- A no ser que se indique explícitamente, por cada partida jugada se creará un fichero donde se registrará el estado del tablero en cada turno de la partida. Este fichero tendrá el formato `game_AAAA-MM-DD_HH:mm:SS_NombreEquipoA-vs-NombreEquipoB.txt`
- Las rutas en el sistema donde se guarden las partidas antiguas y donde se almacenen por defecto los ficheros de los equipos deben poder ser configurables.
- Durante el desarrollo de una partida habrá una ligera música ambiental. Esta música debería poderse desactivar fácilmente.

Una vez tenemos los requisitos del sistema en esta iteración, podemos pasar a la etapa de análisis.

6.3. Análisis

6.3.1. Casos de uso

Dados los requisitos definidos anteriormente, el diagrama de casos de usos resultante es el indicado en la figura 6.1

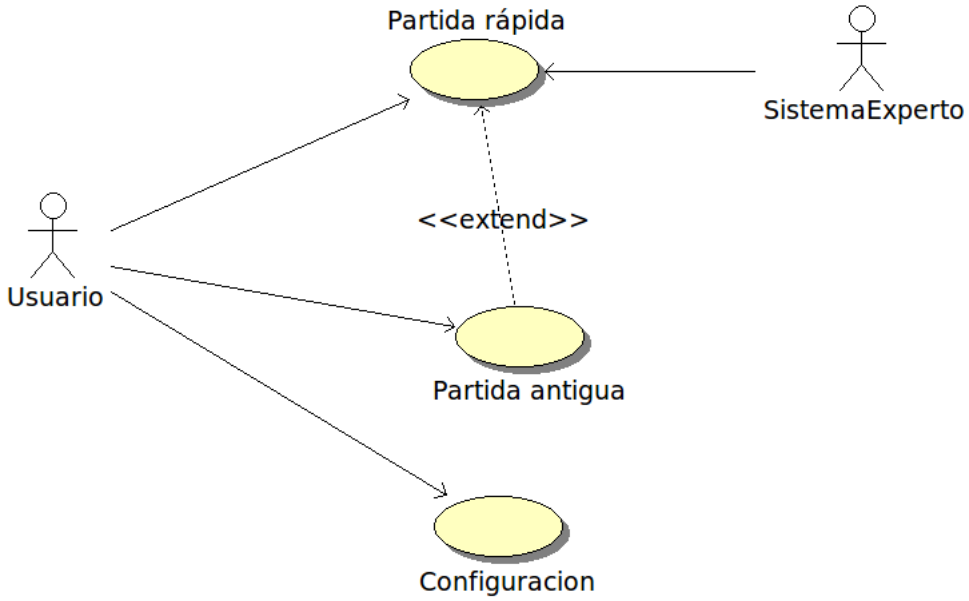


Figura 6.1: Diagrama de casos de usos

Cada caso de uso será una funcionalidad accesible desde el menú principal.

El *extended* entre **Partida rápida** y **Partidas antiguas** se ha considerado necesario por un motivo fundamental: como vimos en la sección 5.2 al simular una partida, se genera un fichero con el estado del tablero en cada uno de los turnos. Es a partir de este fichero sobre el que seremos capaces de representar

gráficamente el tablero. Por tanto en el caso de uso **Partida rápida**, cuando se va a visualizar una partida tras simularla, realmente el comportamiento es el mismo que en el caso el caso de uso **Partidas antiguas**.

A continuación vamos a describir los 3 casos de uso presentes en esta iteración.

Caso de uso: Configuración

Caso de uso: Configuración

Descripción: Permitir al usuario cambiar los parámetros de la configuración en el sistema.

Actores: Usuario

Pre-condiciones: —-

Post-condiciones: Se modifican los parámetros indicados por el usuario, teniendo permanencia en el sistema.

Escenario principal:

1. El sistema solicita qué variable desea modificar.
2. El usuario modifica la ruta predeterminada donde se almacenan las partidas .
3. El sistema verifica el nuevo valor.
4. Vuelta al punto **1.**

Escenarios alternativos:

2a. El usuario modifica la ruta predeterminada que indica donde están almacenados los sistemas expertos.

3a. El usuario activa ó desactiva si quiere escuchar la música durante la partida

4a. El usuario no desea introducir más cambios en la configuración.

1. El sistema valida y almacena los valores modificados.

Caso de uso: Partida rápida

Caso de uso: Partida rápida

Descripción: Permitir al usuario realizar una partida rápida entre dos sistemas expertos.

Actores: Usuario, sistemas expertos

Pre-condiciones: —-

Post-condiciones: Generación de un fichero con el desarrollo de toda la partida.

Escenario principal:

1. El usuario quiere realizar una partida entre dos sistemas expertos
2. El sistema solicita la ruta de los ficheros del equipo A.
3. El usuario introduce la ruta del fichero de reglas del equipo A.

4. El usuario introduce la ruta del fichero de formación del equipo A.
5. El sistema verifica los ficheros del equipo A.
6. El sistema solicita la ruta de los ficheros del equipo B.
7. El usuario introduce la ruta del fichero de reglas del equipo B.
8. El usuario introduce la ruta del fichero de formación del equipo B.
9. El sistema verifica los ficheros del equipo B.
10. El usuario comienza la partida.
11. El sistema simula la partida entre los dos equipos y genera el fichero con la partida resultante.
12. Punto de extensión: Partida antigua.

Escenarios alternativos:

***a.** En cualquier momento el usuario puede cancelar el caso de uso.

1. El sistema cierra el caso de uso.

Caso de uso: Partida antigua

Caso de uso: Partida antigua.

Descripción: Permitir al usuario visualizar una partida ya simulada.

Actores: Usuario.

Pre-condiciones: Debe existir un fichero con la partida a visualizar.

Post-condiciones:

Escenario principal:

1. El sistema solicita la ruta del fichero con la partida a visualizar.
2. El usuario selecciona el fichero de la partida deseada.
3. El sistema muestra la partida.
4. El usuario navega por la partida.
5. El usuario cierra la partida.
6. El sistema muestra el ganador de la partida.

6.3.2. Modelo conceptual de datos

Una vez hemos obtenido los distintos casos de uso de esta primera iteración, podemos realizar el primer diagrama conceptual de datos, haciendo uso de la notación **UML**, que nos ayudará a ir analizando y diseñando el sistema para facilitarnos la tarea a la hora de realizar la implementación. Podemos ver dicho diagrama en la figura 6.2.
Hay que tener en cuenta que para simplificar, se están obviando las clases de interfaz, pues en nuestro caso al utilizar Glade, prácticamente no tendremos que realizar ninguna clase, si no comunicarnos con unas funciones. Por simplificar el diseño, esto se ha obviado en el análisis formal.

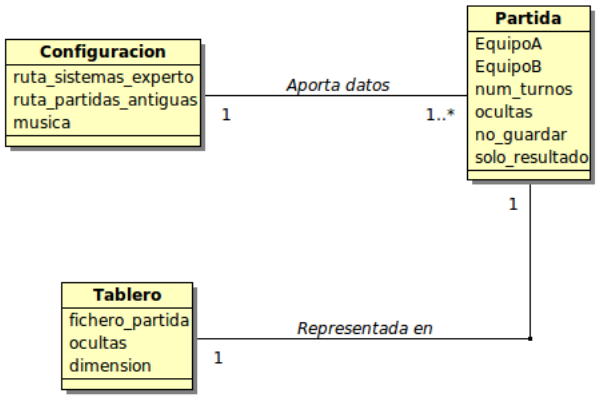


Figura 6.2: Diagrama conceptual de datos

6.3.3. Modelo de comportamiento del sistema

Para realizar el modelo de comportamiento del sistema, tenemos que basarnos en cada caso de uso, para hacer el diagrama de secuencia del sistema que nos permitan saber que operaciones necesitaremos a lo largo del desarrollo.

Una vez realizado dicho diagrama, tendremos que definir el contrato de las operaciones, donde indicamos que atributos del modelo conceptual de datos serán modificados en cada operación.

Caso de uso: Configuración

Figura 6.3: Diagrama de secuencia del sistema

El diagrama de secuencia del sistema para este caso de uso viene reflejado en la figura 6.3.

Caso de uso: Partida rápida

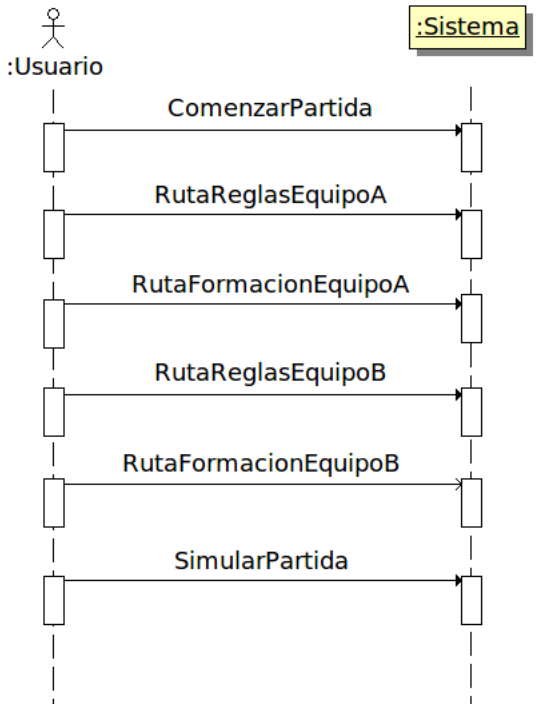


Figura 6.4: Diagrama de secuencia del sistema

El diagrama de secuencia del sistema para este caso de uso viene reflejado en la figura 6.4.

Caso de uso: Partida antigua

El diagrama de secuencia del sistema para este caso de uso viene reflejado en la figura 6.5.

6.3.4. Contratos de las operaciones

Operación: ModificarRutaPartidas

Responsabilidad: Modificar el valor de la ruta donde se almacenan las partidas.

Referencias cruzadas: Caso de uso: Configuración.

Pre-condiciones: –

Post-condiciones:

- Modifica la variable *ruta_partidas_antiguas* del objeto *configuración*.

Operación: ModificarRutaEquipos.

Responsabilidad: Modificar el valor de la ruta donde se almacenan los ficheros de los equipos.

Referencias cruzadas: Caso de uso: Configuración.

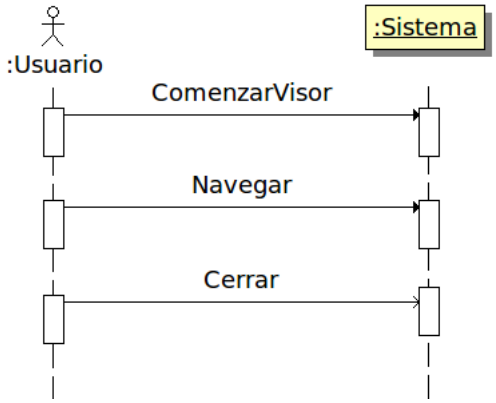


Figura 6.5: Diagrama de secuencia del sistema

Pre-condiciones: –

Post-condiciones:

- Modifica la variable *ruta_sistemas_expertos* del objeto *configuración*.

Operación: ModificarMusicaActiva.

Responsabilidad: Activar ó desactivar la música en el sistema.

Referencias cruzadas: Caso de uso: Configuración.

Pre-condiciones: –

Post-condiciones:

- Modifica la variable *musica* del objeto *configuración*.

Operación: ComenzarPartida(num_turnos, ocultas, no_guardar, solo_resultado).

Responsabilidad: Construye el objeto *Partida*.

Referencias cruzadas: Caso de uso: Partida rápida.

Pre-condiciones: –

Post-condiciones:

- Construye un objeto *partida* de la clase *Partida*, a partir de los parámetros pasados por la función.

Operación: RutaReglasEquipoA(ruta).

Responsabilidad: Inserta la ruta donde se sitúa el fichero de reglas del equipo A.

Referencias cruzadas: Caso de uso: Partida rápida.

Pre-condiciones: –

Post-condiciones:

- Inserta en el parámetro *equipoA* del objeto *partida*, la ruta incluida en el parámetro *ruta* de la función.

Operación: RutaFormacionEquipoA(ruta)

Responsabilidad: Inserta la ruta donde se sitúa el fichero de formacion del equipo A.

Referencias cruzadas: Caso de uso: Partida rápida.

Pre-condiciones: –

Post-condiciones:

- Inserta en el parámetro *equipoA* del objeto *partida*, la ruta incluida en el parámetro *ruta* de la función.

Operación: RutaReglasEquipoB(ruta)

Responsabilidad: Inserta la ruta donde se sitúa el fichero de reglas del equipo B.

Referencias cruzadas: Caso de uso: Partida rápida.

Pre-condiciones: –

Post-condiciones:

- Inserta en el parámetro *equipoB* del objeto *partida*, la ruta incluida en el parámetro *ruta* de la función.

Operación: RutaFormacionEquipoB(ruta)

Responsabilidad: Inserta la ruta donde se sitúa el fichero de formacion del equipo B.

Referencias cruzadas: Caso de uso: Partida rápida.

Pre-condiciones: –

Post-condiciones:

- Inserta en el parámetro *equipoB* del objeto *partida*, la ruta incluida en el parámetro *ruta* de la función.

Operación: SimularPartida

Responsabilidad: Simular la partida para poder visualizarla después.

Referencias cruzadas: Caso de uso: Partida rápida.

Pre-condiciones: Las rutas que contiene el objeto *partida* son válidas.

Post-condiciones:

- Genera un fichero con el registro de toda la partida.
- Llama al caso de uso **partida antigua**.

Operación: ComenzarVisor(fich)

Responsabilidad: Crear el objeto e iniciar el visor de partida.

Referencias cruzadas: Caso de uso: Partida antigua.

Pre-condiciones: El fichero *fich* contiene una partida.

Post-condiciones:

- Construye el objeto *visor* de la clase *partida antigua* a partir del fichero *fich* dado.
- Muestra la partida.

Operación: Navegar(direccion)

Responsabilidad: El usuario puede ver distintos turnos.

Referencias cruzadas: Caso de uso: Partida antigua.

Pre-condiciones:

Post-condiciones:

- Construye realiza el movimiento dado por *direccion*, avanzando o retrocediendo un turno.

Operación: Cerrar

Responsabilidad: Destruir el visor.

Referencias cruzadas: Caso de uso: Partida antigua

Pre-condiciones:

Post-condiciones:

- Destruye el objeto *visor*, y desconectándolo del sistema.

6.4. Diseño

El diagrama de clases y de secuencia del sistema generados durante el análisis son simplemente conceptuales, para entender como funciona básicamente el sistema. En esta fase de diseño, ampliaremos esos conceptos justificando las decisiones tomadas, para llegar a la fase de implementación con unas buenas bases sobre la aplicación que queremos llevar a cabo.

El procedimiento durante esta etapa de desarrollo es el siguiente: tomaremos como base las clases generadas durante el análisis. Luego, respetando las asociaciones entre las clases, ampliaremos las mismas clases si fuera necesario, para poder modular dichos comportamientos y abstraer en clases cuando se requiera.

6.4.1. Configuración

Esta clase es muy sencilla, y se basa sencillamente en tener encapsulados ciertos parámetros generales de la aplicación, como son:

- Ruta donde se sitúan los sistemas expertos.
- Ruta donde se guardan las partidas jugadas.
- Si la música está activa o no.

Por tanto, fundamentalmente esta clase requerirá métodos para obtener y modificar dichos datos.

6.4.2. Tablero

Esta clase es simple en requisitos, pero compleja en cuanto a diseño. El objetivo de esta clase es mostrar por pantalla una partida cualquiera. Para ello toma como base un fichero generado por el núcleo tras simular la partida, mostrando una partida.

Primero, ¿qué definimos como una **partida**? Una partida la podemos considerar como una sucesión de tableros, una serie ordenada de **tableros**, del primer hasta el último turno. Si nos abstraemos un poco más, también podemos ver fácilmente que un tablero está compuesto por una serie de **fichas** que representan una única entidad indivisible en la partida.

Parece claro entonces que tendremos tres clases asociadas de una forma lineal que nos permitan ir de mayor a menor nivel de abstracción. Esto nos servirá para que el código sea más mantenible, y esto es importante dado el desarrollo iterativo que se está ejerciendo sobre este proyecto. Es bastante probable que en futuras etapas sea necesario modificar estas clases, de forma que este diseño *por capas* permitirá que estas tareas de mantenimiento y ampliación sean más sencillas.

6.4.3. Partida

Esta clase se comunica directamente con el núcleo de la aplicación, tomando del usuario que ficheros son los que debe cargar, y hacer uso del núcleo para simular la partida. Una vez simulada, se lo pasará a la clase que se encarga de mostrar esta partida al usuario, como es la clase *tablero* (recordemos que a nivel de análisis, luego en el diseño puede cambiar la cosa).

Se debería encargar también de renombrar y mover el fichero resultante al destino y con el nombre apropiados.

6.4.4. Diagrama de clases

Una vez sabemos de una forma general como debe hacer las cosas el sistema, podemos hacer las divisiones en las clases pertinentes. Dicho diagrama podemos verlo en la figura 6.6. Al igual que pasaba en los diagramas del análisis del sistema, estos diagramas son meramente conceptuales y orientativos. Una vez llegue el momento de la implementación, esto puede variar debido a las peculiaridades de cada lenguaje de programación.
Podemos comparar con el gráfico 6.2, que aunque aumentan el número de clases, la idea en la organización es la misma.

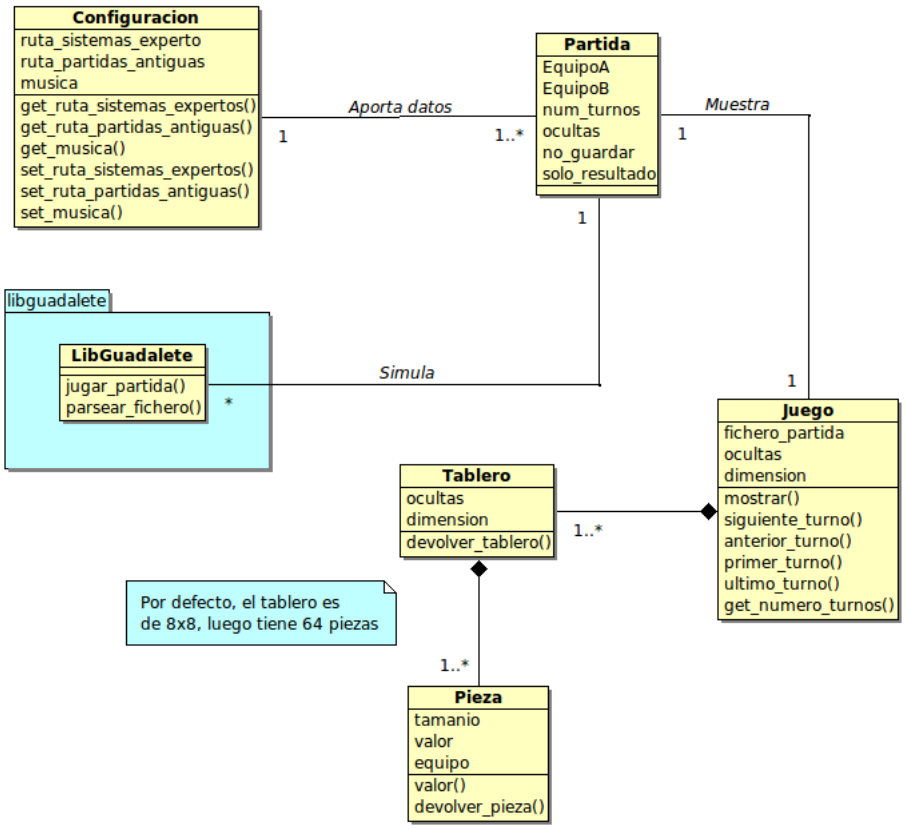


Figura 6.6: Diagrama de clases

Una clase importante, es *LibGudalete*, que es una clase que nos permitirá acceder de forma sencilla al núcleo reescrito en el capítulo anterior. Entre otras, esta clase debería disponer de las funciones:

jugar_partida(): Esta es la función principal, que lanza la simulación.

parsear_fichero(): Con esta función deberemos ser capaz de, a partir de un fichero de salida de la simulación, generar una lista de tableros, que nos facilite la representación por otros módulos.

Hay que notar que se ha realizado una abstracción de las clases de la interfaz de menú.

6.4.5. Diagrama de secuencia del sistema

El gráfico 6.7 muestra el diagrama de secuencia del sistema en una ejecución principal, suponiendo la corrección del proceso.
Si en algún momento falla algunas de las funciones, el sistema avisará del error, recuperándose de él si pudiera.

6.4.6. Diseño de la interfaz de usuario

En principio era necesario una interfaz sencilla, sin demasiada complicación, que permitiera efectuar los distintos casos de uso de una forma simple para el usuario.

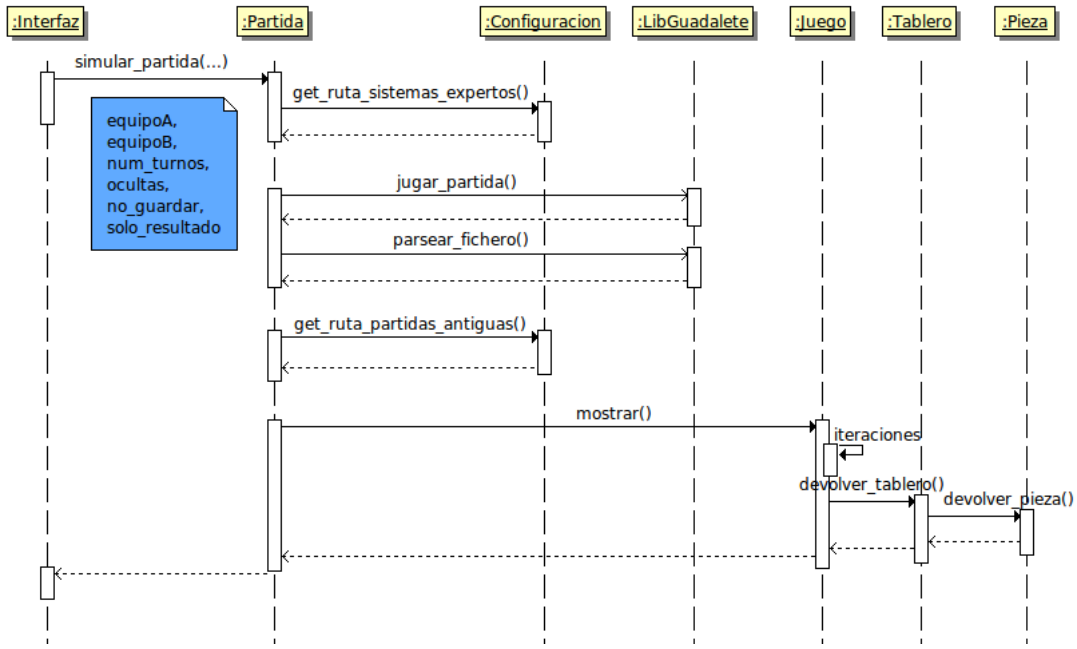


Figura 6.7: Diagrama de secuencia del sistema

A este respecto, recodemos que en la sección 4.4 se valoró utilizar la librería **PyGTK**, y en especial de la herramienta **Glade** para el diseño gráfico de dicha interfaz. Para diseñar estas interfaces se han tenido en cuenta fundamentalmente el diagrama de casos de uso y el diagrama conceptual de datos, para saber exactamente **qué** se necesita del usuario, y no exigir más que esos datos. En concreto hay tres ventanas

- Primero, tenemos la ventana principal en el gráfico 6.8. En ella, son accesibles los 3 casos de uso definidos durante el análisis del sistema.

Figura 6.8: Ventana principal

- En la figura 6.9 podemos ver las opciones para configurar una partida rápida, incluyendo la selec-

ción de ficheros, número de turnos, y algunas otras opciones.

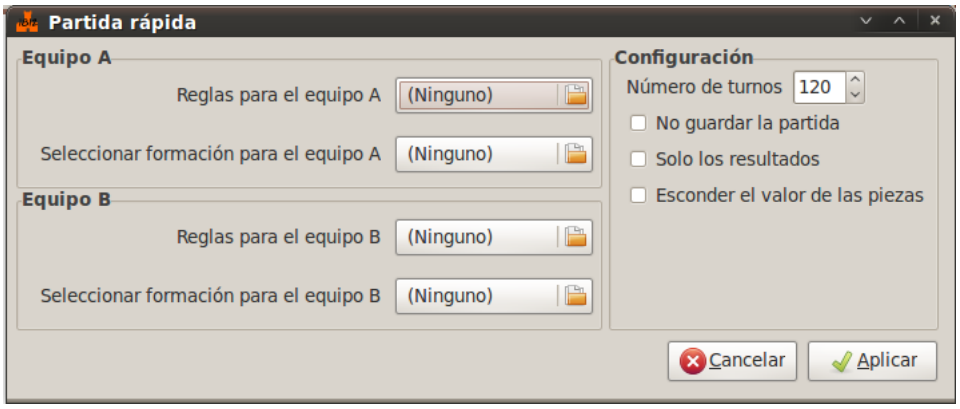


Figura 6.9: Diálogo de configuración de partida rápida

- Como se puede ver en la figura 6.10 para seleccionar una partida antigua hacemos uso del selector de fichero básico del sistema.
- Por último tenemos en la figura 6.11 en la que se muestra la configuración general del sistema, permitiendo modificar las rutas del sistema, o desactivar la música si quisiéramos.

  Estas interfaces aún no son funcionales, pero dadas las características de Glade, se puede usar la propia aplicación para hacer los primeros diseños, sin tener que implementar nada.

6.5. Implementación

Como se comentó en la sección 4.4, el lenguaje de programación empleado durante el desarrollo de la aplicación es Python. Este lenguaje es orientado a objetos, por tanto la traducción desde los diagramas anteriores al propio lenguaje, es relativamente sencillo, al menos el esqueleto de la aplicación. Sin embargo en este punto entran en juego las bibliotecas externas como **PyGTK** ó **Pygame**, las utilizadas para la interfaz gráfica, ya que cada una de ellas tiene sus peculiaridades. Vamos poco a poco explicando el proceso de desarrollo durante la implementación de las distintas clases.

Es importante en este punto indicar que el código está totalmente escrito en inglés. Los nombres de las clases, métodos y atributos durante el análisis y diseño está en español para mayor facilidad para el lector. Sin embargo, se optó por escribir el código (y comentarlo) en inglés, para hacer este proyecto libre más accesible a otros desarrolladores.

6.5.1. Configuración

Este módulo se implementó como una serie de funciones que trabajan sobre un sencillo fichero XML, que es donde se almacenan los datos de configuración de la aplicación, como por ejemplo el siguiente fichero:

```
<?xml version="1.0" ?>
<config>
        <se_path value="/home/pyriku/.resistencia1812/data/teams"/>
```

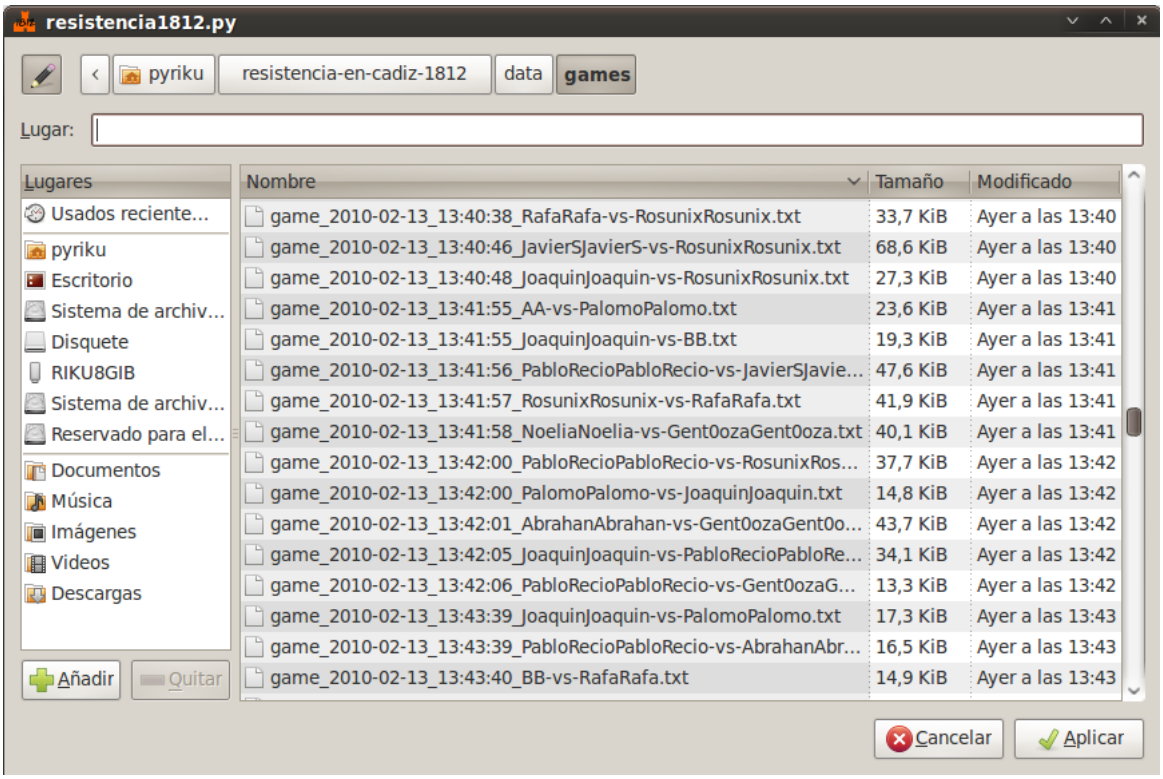


Figura 6.10: Diálogo de selección de fichero

```
        <games_path value="/home/pyriku/.resistencia1812/data/games"/>
        <music_active value="1"/>
</config>
```

Por tanto, este módulo solo aporta funciones para la lectura y escritura de esos parámetros.

6.5.2. Guadaboard

Guadaboard es el nombre al paquete Python que se le ha dado al conjunto de módulos y clases que permiten dibujar una partida haciendo uso de Pygame.

Este grupo de módulos tiene bastante trabajo realizado, y un nivel de abstracción bastante bueno. Se podría destacar del módulo **XML Layout**. Este módulo de desarrollo propio, tiene un objetivo general bastante simple: separar representación gráfica de código.

Esto es lo que hacemos al hacer uso de Glade en las interfaces gráficas de los menús. Se genera un fichero que contiene los datos de las ventanas, diálogos, widgets, etcétera. Sin embargo, usando **Pygame** para representar el tablero, esto no era posible. Entonces, para poder indicar en que coordenadas tienen que ubicarse cada elemento en la ventana que se muestra el tablero, como la que vemos en la figura 6.10. En esta ventana, podemos distinguir 4 zonas principales:

- El tablero en sí
- Los nombres de los jugadores

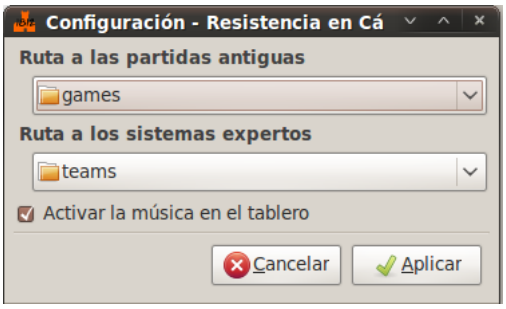


Figura 6.11: Diálogo de configuración de la aplicación

- Los botones de acción
- El botón de salir

Si bien es posible indicar esas posiciones a nivel de código, es más elegante y cómodo, poder leer esos datos de un fichero, permitiendo además, modificar el aspecto solo al cambiar un fichero. Un ejemplo del fichero puede ser el siguiente:

```
<?xml version="1.0" ?>
<window-board>
  <propierty name="window_title" type="title_string">Resistencia en Cadiz: 181
  <propierty name="background" type="image">./images/fondo.png</propierty>
  <propierty name="favicon" type="image">./images/favicon.png</propierty>
  <propierty name="font_type" type="font">./fonts/zektonbi.ttf</propierty>
  <propierty name="font_size" type="int">22</propierty>
  <propierty name="font_color" type="color">
    <value id="r">0</value>
    <value id="g">0</value>
    <value id="b">0</value>
  </propierty>
  <propierty name="window_size" type="size">
    <value id="weight">760</value>
    <value id="height">560</value>
  </propierty>

  <label name="board" type="dynamic_surface">
    <propierty name="board_size" type="size">
      <value id="weight">480</value>
      <value id="height">480</value>
    </propierty>
    <propierty name="board_position" type="position">
      <value id="x">10</value>
      <value id="y">10</value>
    </propierty>
  </label>
  <!--
```

```
     sigue
    -->
</window-board>
```

¿Cómo leemos estos datos? Haciendo uso de la biblioteca `xml.dom.minidom` podemos parsear de una forma sencilla un fichero XML. De forma independiente, podemos crear un fichero que procese cada tipo de etiqueta del XML.

De esta forma, en ese fichero tendremos funciones de la forma `parse_tag_attr(element)`, que permitirán devolver la estructura apropiada. En el código siguiente, se puede ver como se sigue el siguiente proceso:

1. Para cada nodo del fichero XML:
2. Se lee la etiqueta y los atributos *tipo* y *nombre*.
3. Se llama a la función `parse_etiqueta_tipo(element)`
4. Se almacena en un diccionario con la clave *nombre*

```
import os
import sys
from xml.dom import minidom

import pygame
import pygame.font

import main_layout_functions as functions

class Layout(object):
    def __init__(self, xml_layout_document):
        self.elements = {}
        docxml = minidom.parse(xml_layout_document)

        window_board = docxml.firstChild
        window_board_childs = window_board.childNodes

        functions.erase_childs_end_of_line(window_board_childs)

        for element in window_board_childs:
            tag = element.tagName
            attr = element.getAttribute('type')

            x = eval('functions.parse_' + tag + '_' + attr + '(element)')
            self.elements[element.getAttribute('name')] = x

        self.window_size = self.elements['window_size']
        self.window_title = self.elements['window_title']
        self.state = {'button_exit': 0, 'button_left_2': 0, 'button_left_1': 0,
                      'button_right_1': 0, 'button_right_2': 0}
```

De esta forma, obtenemos un diccionario con las coordenadas, tamaños, imágenes etcétera, necesarios para *levantar* una interfaz de Pygame que nos permita mostrar el tablero.

6.6. Pruebas

Para la comprobación del cumplimiento de los objetivos retomamos los requisitos del sistema que se propusieron al comienzo de la iteración, para poder verificar si se han cumplido dichos objetivos:

- *¿Dos sistemas expertos pueden jugar entre ellos correctamente?. ¿Estos sistemas expertos están diseñados para ser el equipo A, y se comportan bien como el equipo B?*

  Este objetivo, fundamental en el sistema, se ha cumplido correctamente. Es posible hacer jugar a dos sistemas expertos situándolos en cualquiera de los equipos.

- *¿El número de turnos de una partida es seleccionable por el usuario?*

  En el diálogo de configuración de la partida se puede seleccionar el número de turnos. Por tanto este objetivo se cumple.

- *¿Se pueden esconder el valor de las fichas?*

  También es algo configurable en el diálogo de configuración de la partida.

- *En una partida rápida, ¿puede elegirse mostrar solo resultado de la partida?*

  Es una opción seleccionable en el diálogo de configuración de la partida.

- *Durante la visión de la partida, ¿se puede iterar en los turnos en ambos sentidos?*

  Podemos avanzar y retroceder turnos sin problema.

- *¿Los ficheros de reglas tienen el formato especificado en los requisitos?*

  Si.

- *¿Los ficheros de log se almacenan con el nombre* `game_AAAA-MM-DD_HH:mm:SS_NombreEquipoA-vs-NombreEquipoB.txt`*?*

  Si, en una carpeta especial para ello, los juegos se almacenan siguiendo el formato dado.

- *Las rutas en el sistema donde se guarden las partidas antiguas y donde se almacenen por defecto los ficheros de los equipos deben poder ser configurables.*

  En el diálogo de configuración estos parámetros se pueden cambiar a gusto del usuario.

- *Durante el desarrollo de una partida habrá una ligera música ambiental. Esta música debería poderse desactivar fácilmente.*

  En el mismo diálogo de configuración podemos activar o desactivar la música.

Capítulo 7

Iteración 4: Torneos y pruebas

7.1. Objetivos

En esta iteración el objetivo es realizar competiciones en el sistema de una forma sencilla e intuitiva para el usuario. Aprovechando la implementación de estas competiciones, podremos añadir también un formato de pruebas automáticas, que nos permitan jugar muchas partidas de forma repetida, con el fin de evaluar la bondad de nuestro sistema experto.

7.2. Toma de requisitos

De la misma forma que en la anterior iteración, antes de comenzar con el análisis y el diseño de esta iteración, hay que tener claro cuales son los objetivos a los cuales queremos llegar para considerar que hemos terminado el desarrollo de esta iteración.

Los requisitos a cumplir para esta iteración, son los siguientes:

- Poder jugar distintos formatos de competición distintos.
- Se deben poder seleccionar el número de turnos que se quieren jugar por partida en la competición.
- Si es aplicable, el usuario debería poder elegir si quiere una segunda vuelta en la competición.
- Sería interesante una opción que seleccionase para participar en la competición todos los sistemas instalados.
- Botones para añadir un sistema experto, con sus dos ficheros, así como para quitarlos.
- Poder ver cada una de las partidas, o solo los resultados entre las rondas a gusto del usuario.
- En las pruebas, poder seleccionar cuantas rondas se quieren jugar.
- Las pruebas deberían enseñar algunos parámetros que sirvan para evaluar la bondad de un sistema experto, así como generar un fichero CSV[1] con los resultados más específicos de esas pruebas.

[1] Comma Separated Values

7.3. Análisis

7.3.1. Casos de uso

Completando el diagrama 6.1, podemos añadir las nuevas funcionalidades en dos casos de uso distintos, como podemos ver en la ampliación del diagrama 7.1

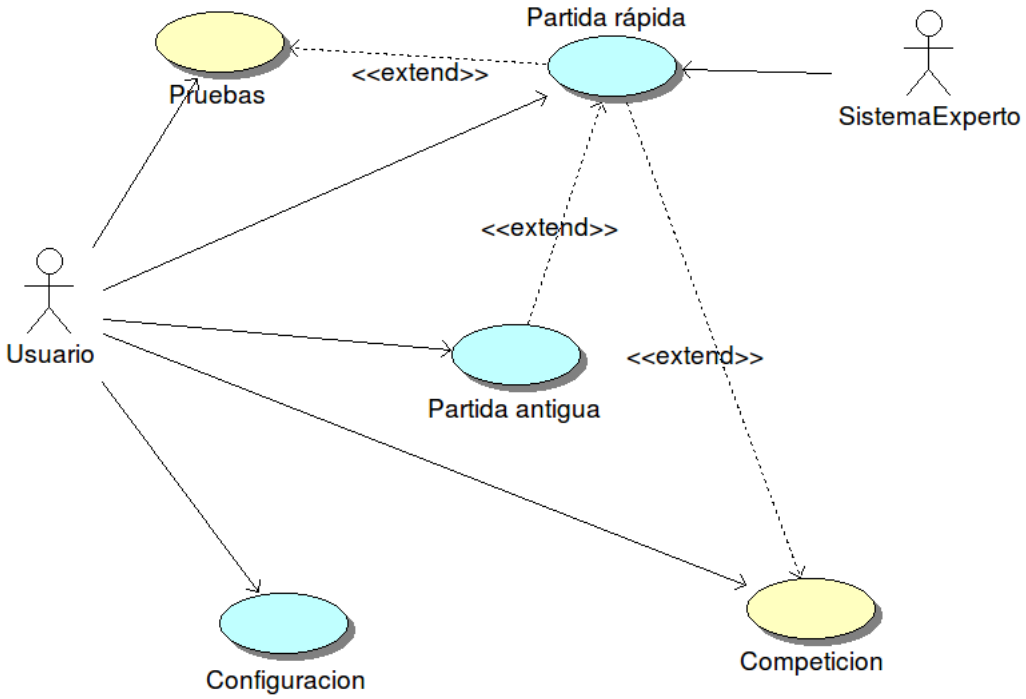


Figura 7.1: Diagrama de casos de usos

A continuación, procedemos a describir los 2 nuevos casos de uso:

Caso de uso: Competición

Caso de uso: Competición

Descripción: Permitir al usuario generar una competición entre varios sistemas expertos.

Actores: Usuario

Pre-condiciones: —-

Post-condiciones: Se generan los ficheros de registro de todas las partidas y uno de la competición.

Escenario principal:

1. El usuario desea preparar una competición.
2. El sistema solicita que formato de competición desea.
3. El usuario indica que desea una *liga*.

4. El sistema solicita inserción de los sistemas expertos participantes.
5. El usuario inserta los sistemas expertos.
6. El usuario comienza la competición.
7. Hasta la última ronda:
   a) El sistema simula cada partida de la ronda.
   b) El sistema muestra las puntuaciones intermedias.
   c) El usuario avanza hacia la siguiente ronda

Escenarios alternativos: **3a.** El usuario indica que desea una *copa*

3b. El usuario indica que desea una *playoff*

***a.** En cualquier momento, el usuario cancela la ejecución del torneo.

Caso de uso: Pruebas

Caso de uso: Pruebas

Descripción: Permitir al usuario generar una prueba dinámica entre varios sistemas expertos con el fin de comprobar la bondad de un sistema experto.

Actores: Usuario

Pre-condiciones: —-

Post-condiciones: Se genera un fichero CSV con los datos de la partida.

Escenario principal:

1. El usuario desea preparar una prueba.
2. El sistema solicita inserción de los sistemas expertos participantes.
3. El usuario inserta los sistemas expertos.
4. El usuario indica el número de rondas que desea.
5. El sistema verifica el valor aportado.
6. El usuario comienza la competición.
7. El sistema devuelve las estadísticas generales de la prueba

Escenarios alternativos: ***a.** En cualquier momento, el usuario cancela la ejecución de la prueba.

7.3.2. Modelo conceptual de datos

En esta iteración tomamos como base sólida los módulos desarrollados antes. Haciendo uso de la clase *Partida*, podemos definir de forma sencilla las competiciones. Podemos ver dicho diagrama en la figura 7.2. Este modelo complementa al presentado en el capítulo anterior, aumentando la funcionalidad del sistema.

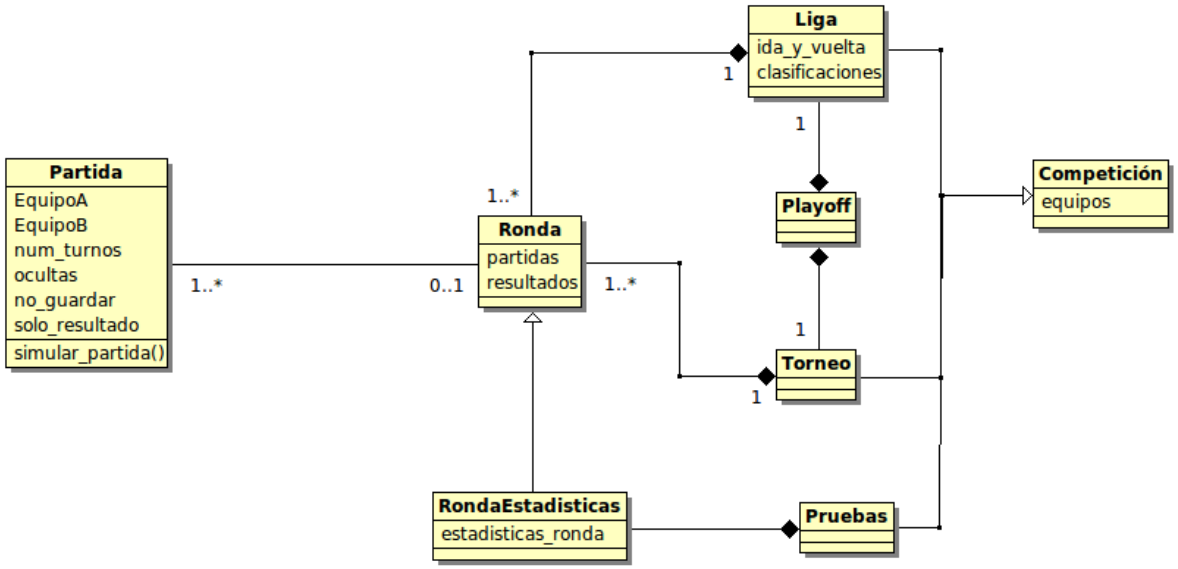


Figura 7.2: Diagrama conceptual de datos

7.3.3. Modelo de comportamiento del sistema

Para realizar el modelo de comportamiento del sistema, tenemos que basarnos en cada caso de uso, para hacer el diagrama de secuencia del sistema que nos permitan saber que operaciones necesitaremos a lo largo del desarrollo.

Una vez realizado dicho diagrama, tendremos que definir el contrato de las operaciones, donde indicamos qué atributos del modelo conceptual de datos serán modificados en cada operación.

Caso de uso: Competición

El diagrama de secuencia del sistema para este caso de uso viene reflejado en la figura 7.3.

Caso de uso: Pruebas

El diagrama de secuencia del sistema para este caso de uso viene reflejado en la figura 7.4.

7.3.4. Contratos de las operaciones

Operación: NuevaCompeticion()

Responsabilidad: Comenzar la ejecucion de una competición.

Referencias cruzadas: Caso de uso: Competición.

Pre-condiciones:

Post-condiciones:

- Construye un objeto de la clase *Competición*

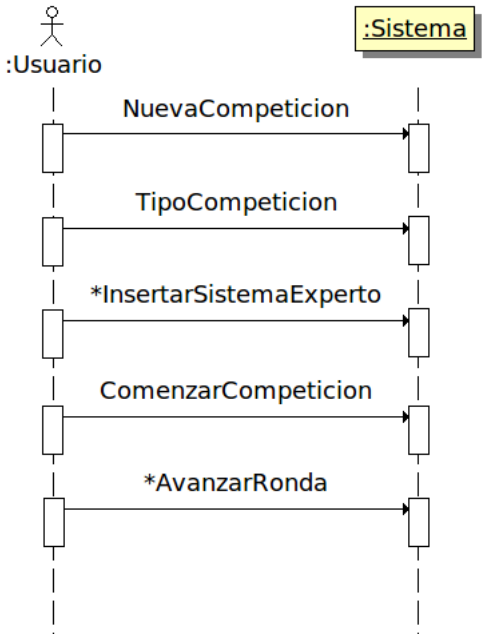


Figura 7.3: Diagrama de secuencia del sistema

Operación: TipoCompeticion(tipo)

Responsabilidad: Especializa la competición.

Referencias cruzadas: Caso de uso: Competición.

Pre-condiciones:

Post-condiciones:

- Usando el objeto anterior, especializa el tipo de competición en *liga*, *copa* o *playoff*.

Operación: InsertarSistemaExperto(equipo)

Responsabilidad: Especializa la competición.

Referencias cruzadas: Caso de uso: Competición, Pruebas.

Pre-condiciones:

Post-condiciones:

- Añade a la competición un sistema experto para que participe en ella, añadiéndolo a la lista de *equipos*.

Operación: ComenzarCompeticion()

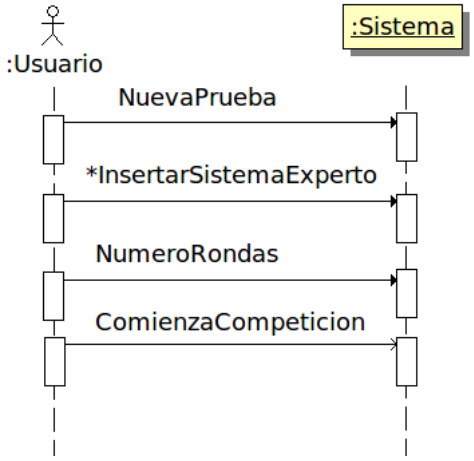


Figura 7.4: Diagrama de secuencia del sistema

Responsabilidad: Genera las rondas intermedias para jugar la competición.

Referencias cruzadas: Caso de uso: Competición, Pruebas.

Pre-condiciones: Competición creada.

Post-condiciones:

- Genera los emparejamientos en forma de ronda.
- Haciendo uso del algoritmo de emparejamiento, genera las distintas rondas de la competición.

Operación: AvanzarRonda()

Responsabilidad: Juega la ronda completa, generando los resultados.

Referencias cruzadas: Caso de uso: Competición.

Pre-condiciones: Competición creada y emparejamientos hechos.

Post-condiciones:

- Juega cada una de las partidas.
- Apunta el resultado de cada partida.
- Si es una liga, al terminar suma las puntuaciones con las anteriores.
- Si es una copa, calcula la siguiente ronda.

Operación: NuevaPrueba()

Responsabilidad: Comenzar la ejecucion de una prueba automática.

Referencias cruzadas: Caso de uso: Pruebas.

Pre-condiciones:

Post-condiciones:

- Construye un objeto de la clase *Pruebas*.

Operación: NumeroRondas(num)

Responsabilidad: Indica cuantas rondas se quieren iterar.

Referencias cruzadas: Caso de uso: Pruebas.

Pre-condiciones:

Post-condiciones:

- Construye *num* número de rondas asociadas a las pruebas.

7.4. Diseño

Se sigue el mismo patrón de desarrollo que en la etapa anterior: tomar las clases generadas en el análisis como una base, y a partir de ellas ampliar las clases si fuera necesario para poder cumplir los objetivos de la fase de diseño.

7.4.1. Competiciones

Esta clase de análisis es una clase general que nos permite agrupar las funcionalidades con las que cuentan las competiciones del tipo *Liga* y del tipo *Playoff*. Se podría decir que *Torneo* se escapa un poco de esa herencia, pues un *Torneo* no es más que una liga y un torneo después de esa liga.

7.4.2. Liga

Una liga se define como una competición entre varios equipos, de forma que todos los equipos se enfrentan entre ellos un número fijo de veces. Por ejemplo, si tenemos 4 equipos en una liga, una liga de solo una vuelta, podría ser la mostrada en la tabla 7.1

EquipoA	EquipoD
EquipoC	EquipoB

(a) Jornada 1

EquipoA	EquipoC
EquipoB	EquipoD

(b) Jornada 2

EquipoA	EquipoB
EquipoD	EquipoC

(c) Jornada 3

Tabla 7.1: Emparejamientos en una liga de 4 equipos

Por tanto en una liga necesitaremos básicamente poder montar los emparejamientos, iterar en las rondas e ir obteniendo los resultados de cada una de ellas.

EquipoA	EquipoD
EquipoC	**EquipoB**
EquipoE	**EquipoF**
EquipoH	EquipoG

(a) Ronda 1

EquipoA	EquipoB
EquipoF	**EquipoH**

(b) Ronda 2

EquipoA	EquipoH

(c) Ronda 3

Tabla 7.2: Emparejamientos en un torneo de 8 equipos

7.4.3. Torneo

El comportamiento del torneo es bastante similar al de una liga, pero difiere sustancialmente en la organización de las rondas. Supongamos un torneo de 8 equipos, como el que podemos ver en la tabla 7.2 Aunque la idea es similar (una sucesión de rondas), el planteamiento es distinto: en este caso no nos interesan los puntos resultantes, simplemente quien ha ganado. Además, en una *liga* podemos (y debemos) hacer los emparejamientos con anterioridad, pero en una copa eso no es posible ya que dependemos de quien haya ganado la anterior ronda para montar la actual.

7.4.4. Rondas

Fijándonos un poco en el diagrama conceptual de datos 7.2 podemos ver que prácticamente el centro en el desarrollo de esta iteración es la clase *Ronda*. Con una buena clase que modele el comportamiento de una ronda de una competición, podremos apoyar el resto del desarrollo de esta iteración en dicha clase.

¿Por qué es tan importante el desarrollo de esta clase? Viendo las ideas que hemos extraído en las clases *liga* y *torneo*, queda claro que realmente lo importante son las rondas de cada competición. Independientemente de como se realicen luego los emparejamientos, tener una buena clase para las rondas nos permitirá que el resto del esquema sea más simple.

Obviamente, esta clase necesitará funciones como por ejemplo son

- *construir_ronda*
- *jugar_ronda*, que iterará en todas las partidas de la ronda generando el resultado.
- *devolver_ganadores*
- *devolver_resultados*

7.4.5. Pruebas

La parte de pruebas es bastante similar a la de competiciones, con la salvedad de que las rondas no serán de equipos distintos, si no de uno contra todos. Además, tendrán que devolver los valores estadísticos de dicha ronda.

7.4.6. Diagrama de clases

Una vez hemos visto que tiene que hacer cada parte del sistema, modificamos el diagrama conceptual de datos visto en el análisis 7.2 para que se amolde a las necesidades del diseño. Este resultado lo podemos ver en la figura 7.5

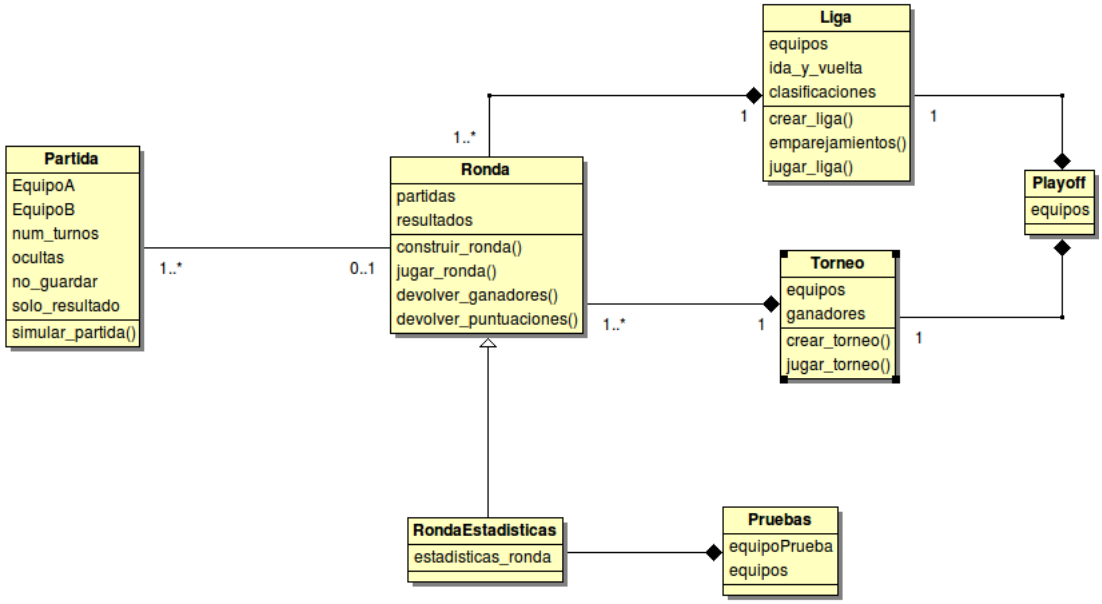


Figura 7.5: Diagrama de clases

Podemos ver que la generalización inicial de la clase *Competición* la hemos eliminado. Esta decisión se ha tomado por un motivo funcional:

A pesar de que comparten un atributo común (la lista de equipos participantes), las funciones son distintas. Por ejemplo, la construcción de un objeto difiere en la relación con la clase *Ronda*, ya que *Liga* por ejemplo es capaz de generar todas las rondas en el constructor, mientras con una *Copa* no es capaz de hacer eso. Por ese mismo motivo, la función que hace que se juegue la competición también difiere bastante ya que *Copa* tiene que ir generando las rondas.

En este contexto no tenía demasiado sentido hacer una generalización, ya que en ningún momento vamos a aprovechar las ventajas de esta característica de la orientación a objetos, ya que aunque conceptualmente son clases similares, se comportan de una forma bastante distinta y eso impide dicha generalización.

Por otro lado, sí es interesante especializar la clase *Ronda* en otra que se dedique a las rondas de las pruebas dinámicas. Esta especialización tendrá por objetivo obtener (aparte de los ganadores) los datos estadísticos de cada partida.

7.4.7. Diagrama de secuencia del sistema

En el diagrama 7.6 podemos ver la secuencia del sistema simplificado de la ejecución del torneo. Esto enlazaría con el diagrama hecho en la iteración anterior a la hora de jugar una única partida.

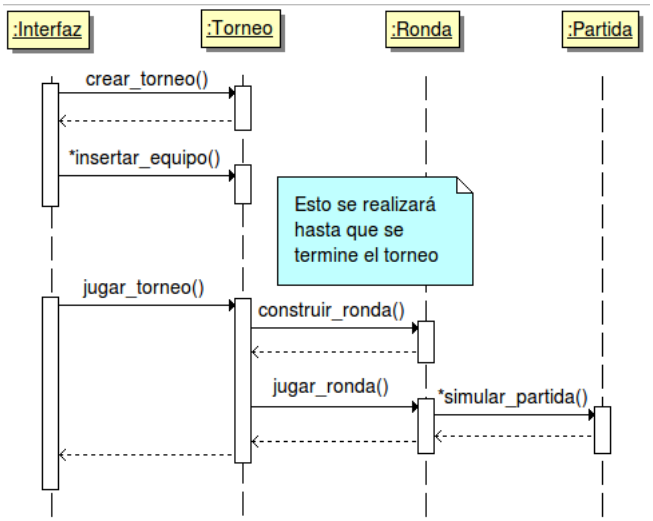


Figura 7.6: Diagrama de secuencia del sistema

La secuencia de una liga o las pruebas es similar, cambiando simplemente el detalle de que en una liga o en las pruebas, no es necesario construir la ronda en cada iteración de la competición, si no que esas rondas se pueden construir a la hora de crear dicha competición.

7.4.8. Diseño de la interfaz

La adición de los dos nuevos casos de uso suponen nuevas opciones en el menú principal, como podemos ver en la captura 7.7.

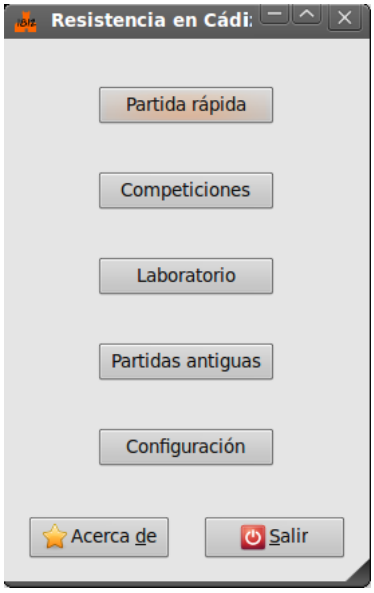


Figura 7.7: Ventana principal v2.0

Para la configuración de las competiciones, se ha optado por un diálogo con dos pestañas. La primera es

para las opciones generales 7.8, y la segunda es para la adición de participantes a la competición 7.9.

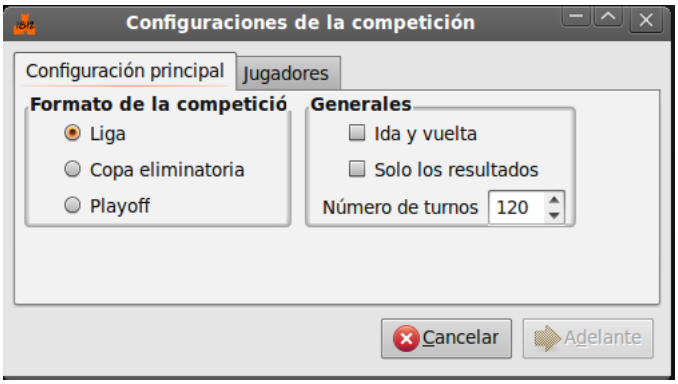


Figura 7.8: Diálogo de competiciones. Primera pestaña

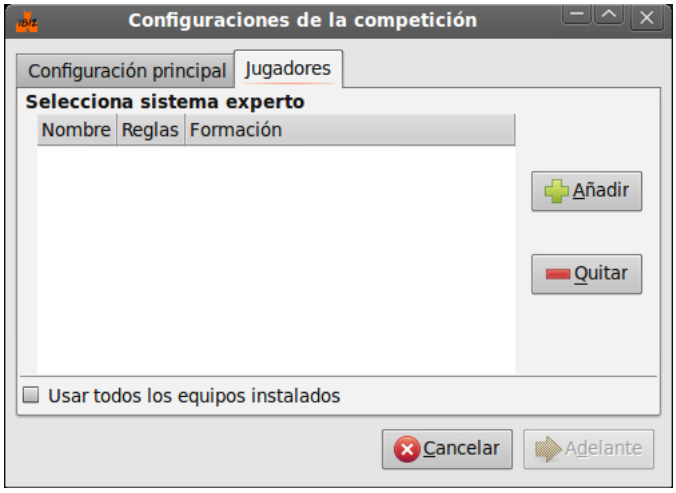


Figura 7.9: Diálogo de competiciones. Segunda pestaña

Por otro lado, para las pruebas hemos diseñado un diálogo que agrupa tanto las opciones generales, como la selección de equipos con los que probar nuestro sistema experto 7.10. El resultado de las pruebas consiste en un diagrama de sectores, así como algunos valores numéricos 7.11

7.5. Implementación

Continuando en la línea de la anterior etapa de implementación, ampliamos las funcionalidades de la aplicación siguiendo el diseño que hemos realizado anteriormente.

Al igual que antes, gracias al diseño, el esqueleto de la aplicación es relativamente sencillo si conocemos la estructura del lenguaje, lo que realmente nos complica es el desarrollo es el trabajo con las librerías externas. Sobre el modelo anterior no hay que hacer modificaciones, así que solo vamos añadiendo las clases definidas en el diseño, e integrándolas en el sistema. A continuación comentamos los eventos más importantes en dicho desarrollo:

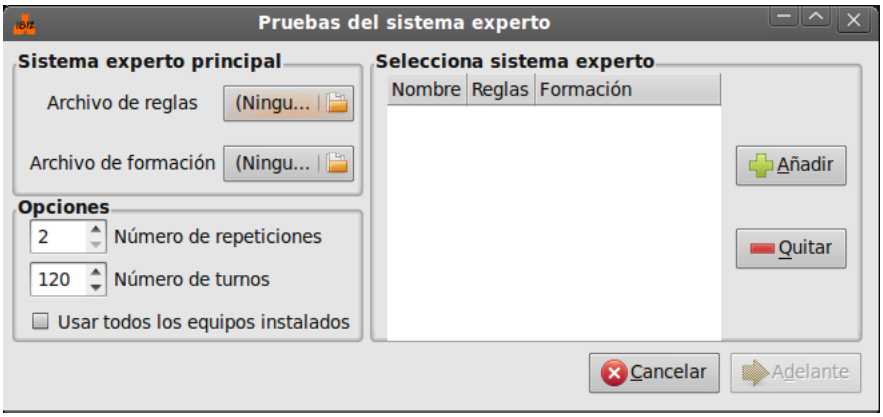


Figura 7.10: Diálogo de pruebas

7.5.1. Ligas y emparejamientos

Para poder realizar correctamente los emparejamientos, se utilizó el algoritmo de planificación **Round-Robin**. Este algoritmo consiste en enfrentar a todos contra todos, haciendo los emparejamientos a partir de una raíz.

El algoritmo consiste en lo siguiente:

Supongamos una lista no ordenada de equipos participantes a una liga:

$$e_1, e_2, e_3, \ldots, e_{n-1}, e_n$$

Para emparejarlos, *doblamos* la lista por la mitad, haciendo que se enfrenten el primero contra el último, el segundo contra el penúltimo, etcétera. Quedaría unos emparejamientos así:

$$e_1 \longleftrightarrow e_n$$

$$e_2 \longleftrightarrow e_{n-1}$$

$$e_3 \longleftrightarrow e_{n-2}$$

$$\ldots$$

Y así sucesivamente, habremos obtenido los emparejamientos de la 1ª ronda. Ahora, para hacer los cruces de la siguiente, hacemos rotar a todos los equipos menos a uno. De esta forma, nunca se volverán a cruzar ningún equipo hasta el final de la vuelta. La lista quedaría:

$$e_1, e_n, e_2, e_3, \ldots, e_{n-2}, e_{n-1}$$

Repitiendo el proceso de doblar la lista, obtenemos los emparejamientos de la segunda jornada:

$$e_1 \longleftrightarrow e_{n-1}$$

$$e_n \longleftrightarrow e_{n-2}$$

$$e_2 \longleftrightarrow e_3$$

$$\ldots$$

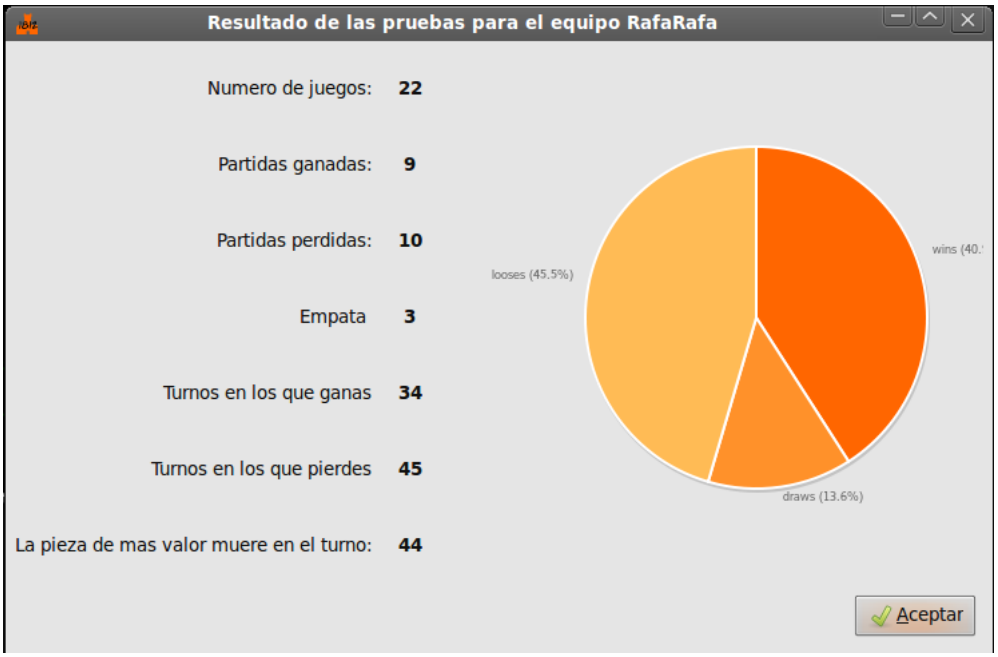


Figura 7.11: Resultado de las pruebas

Este proceso debe continuarse hasta que obtenemos la lista en el orden original. Si quisiéramos una segunda vuelta con el orden invertido, simplemente tendríamos que repetir el proceso, pero invirtiendo el orden de los participantes.
En el caso de que $n \mod 2 \neq 0$, hay que añadir un equipo *fantasma* a la competición para cuadrar los emparejamientos, y en cada ronda, el equipo al que le toque el equipo fantasma descansará. Una versión simplificada del algoritmo escrito en Python sería la que sigue:

```
def make_pairings(teams, back_round=False):
    """This functions do the pairings using round-robin algorithm.

    Keywords arguments:
    teams -- list of teams names that participate on the contest
    back_round -- Boolean value that indicates if it's a two round contest

    Returns a list that contains lists of matchs, something like that:
    [[(a,d), (b,c)], [(a,c), (d,b)], [(a,b), (c,d)]
    """
    random.shuffle(teams)

    #If has an odd number of teams, we have to add a ghost team
    if len(teams) % 2 == 1:
        teams.append('aux_ghost_team')

    first_order = teams #to check if we finish pairing
    elements = teams
    matchs = []
```

```
    finish = False #flag to control if we make the entire round

    while not finish:
        matchs.append(_get_pairing(elements))
        elements = _rotate_less_first(elements)
        if elements == first_order:
            finish = True

    return matchs
```

Repito, es una simplificación, además de obviar algunas funciones que utiliza, como por ejemplo `_rotate_less_first()` ó `_get_pairing()`

7.5.2. Interacción entre PyGTK y Pygame

Uno de los principales problemas que aparecieron a lo largo del desarrollo de la aplicación fue la gestión de eventos entre PyGTK y Pygame. Ambas son unas bibliotecas gráficas que tienen sus propios eventos, y los gestionan a su forma, por tanto en las ocasiones que hay que alternar entre las dos, pueden llegar a causar problemas. Ver la siguiente página con el código.

Dicho código nos muestra la función que inicializa una liga. Tenemos que tener en cuenta que en la liga se muestran (por defecto) cada una de las partidas usando Pygame. Al terminar cada ronda, se lanzará un diálogo con PyGTK que mostrará los resultados intermedios de esa ronda.

```
def _init_league(teams, fast, num_turns, back_round):
    l = league.League(teams, num_turns, back_round)

    band = False

    while not l.league_completed and not band:
        i = l.get_round_number()
        l.play_round(fast)
        r = l.get_round(i)

        classifications = l.get_actual_puntuations()
        results = r.get_round_results()

        R = round_results.roundResults(classifications, results,
                                       l.get_prev_round_number() + 1,
                                       l.get_number_of_rounds())
        button_pressed = R.result_dialog.run()

        while gtk.events_pending():
            gtk.main_iteration(False)

        if button_pressed == -4 or button_pressed == 0:
```

```
            band = True

def _init_tournament(teams, num_turns, fast):
    t = tournament.Tournament(teams, num_turns)
    band = False

    while not t.tournament_completed and not band:
        i = t.get_round_number()
        t.play_round(fast)
        r = t.get_round(i)

        classifications = []
        results = r.get_round_results()

        R = round_results.roundResults(classifications, results,
                                       t.get_prev_round_number() + 1,
                                       t.get_number_of_rounds(),
                                       show_classifications=False)

        button_pressed = R.result_dialog.run()

        while gtk.events_pending(): #very important
            gtk.main_iteration(False)

        if button_pressed == -4 or button_pressed == 0:
            band = True
```

El problema de eso es que, cuando se cierra el diálogo, ya sea por que se quiera cancelar o bien continuar con el desarrollo del torneo, pueden quedarse eventos de GTK por procesar, pero el control de la aplicación pasa a Pygame. De esta forma pueden darse anomalías como diálogos que no se cierran, ventanas inactivas, etcétera.

Para evitar ese problema está la función `gtk.main_iteration(False)`. Esta función dentro de ese bucle dispara todos los eventos de GTK que estén pendientes en el sistema, limpiando la cola de eventos del sistema. Así nos evitamos que cuando Pygame tome el control de la aplicación se queden eventos sin activar en GTK.

7.6. Pruebas

De la misma forma que verificamos la corrección de la anterior etapa, vamos a retomar los requisitos iniciales de esta etapa para comprobar que se han cumplido todos correctamente. Así podremos verificar que esta etapa se ha concluido de una forma apropiada,

- *¿Se pueden jugar competiciones en distintos formatos?*

  Esto se ha completado correctamente, de forma que en el diálogo de configuración de una competición, podemos elegir tres formatos distintos.

- *Se puede seleccionar el número de turnos que se quieren jugar por partida en la competición?*

  En el diálogo de configuración de una competición, podemos seleccionar de cuantos turnos se van a realizar en cada partida.

- *¿El usuario puede elegir que haya una segunda vuelta en la competición.*

  Si el usuario decide jugar una liga, puede habilitar una ronda de vuelta.

- *¿Se pueden seleccionar automáticamente los equipos instalados en el sistema?*

  Hay una opción en el diálogo para seleccionar todos los equipos.

- *¿Se pueden añadir manualmente los ficheros de un sistema experto? ¿Se pueden quitar en caso de error?*

  Si, sin ningún problema. Se usan botones especialmente para ello en la lista de participantes.

- *¿Se puede elegir ver todas las partidas, o por el contrario ver solo los resultados entre rondas?*

  Si, también tenemos opción en el menú para ello.

- *En las pruebas, ¿se seleccionar cuantas rondas se quieren jugar?*

  Si, en un botón habilitado para ello.

- *¿Se muestran de una forma apropiada los resultados de las pruebas? ¿Se pueden observar con más detalle en un fichero CSV?*

  Si, se muestra un diálogo con los resultados de las pruebas, así como se genera un fichero con las partidas más desglosadas.

Capítulo 8

Iteración 5: Partidas humano contra Sistema Experto

8.1. Objetivos

En esta última iteración del desarrollo de la aplicación solo tenemos un objetivo, pero fundamental para la usabilidad del producto que estamos desarrollando. En este punto del desarrollo, un usuario puede hacer jugar a dos sistemas expertos, puede montar competiciones entre ellos e incluso puede probar su sistema experto contra otros muchos. Sin embargo, no tenemos ninguna herramienta para probar nuestro sistema con una mejor granularidad.

Supongamos que nuestro sistema experto tiene una serie de reglas que consisten en hacer que el rey huya. Sin embargo, mientras lo estamos probando contra otro equipo distinto observamos que esas reglas de huida, no funcionan si nos atacan por el lado derecho. ¿Qué podemos hacer si queremos depurar ese problema? Tendremos que aplicar cambios sobre el sistema experto, y volver a someterlo a dicha situación para comprobar que la reacción ahora es más adecuada. Ahora surge otro problema: ¿cómo volvemos a poner al sistema experto en otra situación similar?

Es por eso necesario incluir un modo de juego que nos permita jugar contra un sistema experto, con el fin de poder *forzar* ciertas situaciones para comprobar como se comporta nuestro diseño.

8.2. Toma de requisitos

Como siempre, el primer paso para poder comenzar a desarrollar una nueva iteración de nuestro producto, es tener claro que es lo que tenemos que hacer concretamente. En esta iteración, serían los que siguen:

- Dar soporte al usuario para jugar contra cualquier sistema experto.
- El usuario debe elegir una formación.
- El usuario debe poder elegir el número de turnos.
- El usuario puede elegir un sistema experto al azar contra el que jugar.
- Se debe controlar los movimientos erróneos del usuario, y actuar correctamente frente a ellos.

8.3. Análisis

8.3.1. Casos de uso

Completando el diagrama 7.1, podemos añadir la nueva funcionalidades en un nuevo caso de uso, como podemos ver en la ampliación del diagrama 8.1

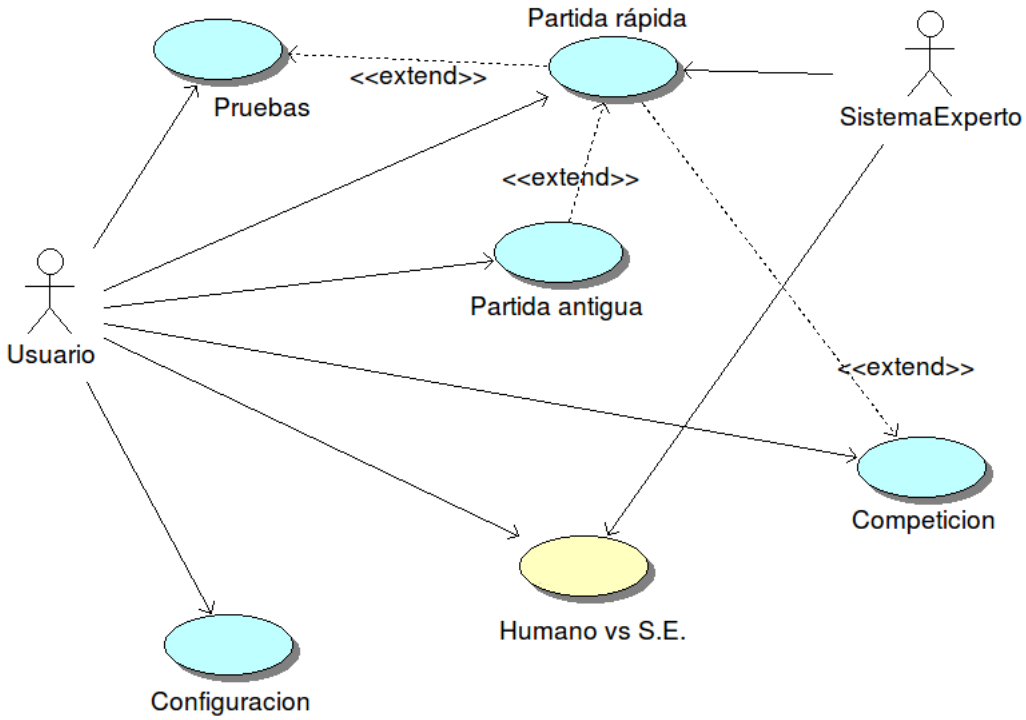


Figura 8.1: Diagrama de casos de usos

A continuación, procedemos a describir el nuevo caso de uso:

8.3.2. Caso de uso: Humano contra Sistema Experto

Caso de uso: Humano contra Sistema Experto

Descripción: Permitir al usuario realizar una partida directamente contra un sistema experto

Actores: Usuario, Sistema experto

Pre-condiciones: —-

Post-condiciones: Se generan los ficheros de registro de la partida

Escenario principal:

1. El usuario desea jugar contra un S.E.
2. El sistema solicita el equipo contra el que desea jugar
3. El usuario selecciona el equipo

4. El sistema solicita la formación con la que desea jugar el usuario
5. El usuario selecciona el fichero de formación
6. El sistema comienza la partida
7. Hasta la última ronda:
   a) El sistema simula un movimiento
   b) El usuario hace un movimiento
   c) El sistema valida y realiza el movimiento

Escenarios alternativos: ***a.** En cualquier momento, el usuario cancela la ejecución de la partida.

8.3.3. Modelo conceptual de datos

En esta iteración, si recurrimos a las dos iteraciones anteriores, con sus diagramas conceptuales 6.2 y 7.2, vemos dados los requisitos de esta iteración no es necesario construir un nuevo modelo conceptual, pues con los datos de las anteriores iteraciones podemos construir esta funcionalidad.

8.3.4. Modelo de comportamiento del sistema

Para realizar el modelo de comportamiento del sistema, tendremos que basarnos en la descripción del caso de uso. A partir de dicha descripción podemos construir el diagrama de secuencia.

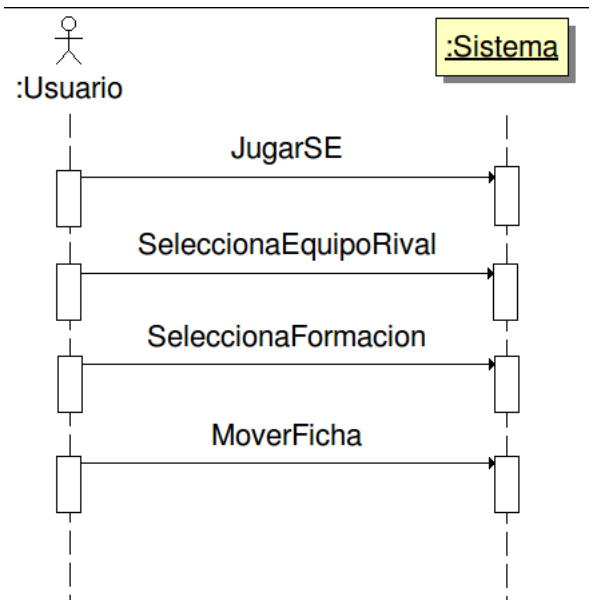


Figura 8.2: Comportamiento del sistema

Con el diagrama 8.2 podemos ver la secuencia en el sistema que permita a un usuario jugar contra un sistema experto directamente.

8.3.5. Contratos de las operaciones

Operación: JugarSE()

Responsabilidad: Comienza la realización de una partida contra un SE.

Referencias cruzadas: Caso de uso: Humano contra Sistema Experto.

Pre-condiciones:

Post-condiciones:

- Muestra al usuario las opciones disponibles.

SeleccionarEquipoRival() y *SeleccionarFormacion()* son idénticas a las funciones de elección de ficheros en el caso de uso *Partida rápida* de la tercera iteración 6.3.4, por tanto no es necesario volver a definirlas.

Operación: MoverFicha()

Responsabilidad: El usuario realiza el movimiento de una de sus fichas.

Referencias cruzadas: Caso de uso: Humano contra Sistema Experto.

Pre-condiciones: La partida aún no se ha terminado. Es el turno del usuario.

Post-condiciones:

- Si el movimiento es válido, la ficha se mueve a la posición indicada por el usuario.

8.4. Diseño

Hemos visto que esta etapa se sustenta fundamentalmente en modificar algunos supuestos que se tomaron inicialmente en la aplicación. Al comenzar este desarrollo, no se consideró esta posibilidad y por tanto, la estructura en la que se ha ido desarrollando el proyecto no está pensada para la implantación de esta nueva funcionalidad.

Recapitulando un poco lo que tenemos hasta ahora, el resumen de clases podemos verlo en el diagrama 8.5. Tenemos que recordar que este diagrama es un **simplificación**, y están obviadas ciertas clases como las clases de interfaz gráfica, o alguna clase para facilitar el acceso a rutas del sistema, etcétera.

Podemos ver claramente que para conseguir la funcionalidad que perseguimos, "solo" es necesario modificar dos clases: *Partida* y *LibGuadalete*. Basándonos en el diagrama de secuencia del sistema de una partida normal 6.7 no es demasiado complicado ver los cambios que son necesarios para que este caso de uso soporte esta funcionalidad. El diagrama de secuencia resultante lo podemos ver en el gráfico 8.3.

Realmente hay cambiar el comportamiento completo de la interacción entre los dos módulos comentados, ya que hasta ahora el comportamiento era el siguiente:

1. El usuario aportaba los datos (ficheros de formación y reglas)
2. El núcleo simulaba la partida, almacenándola en un fichero

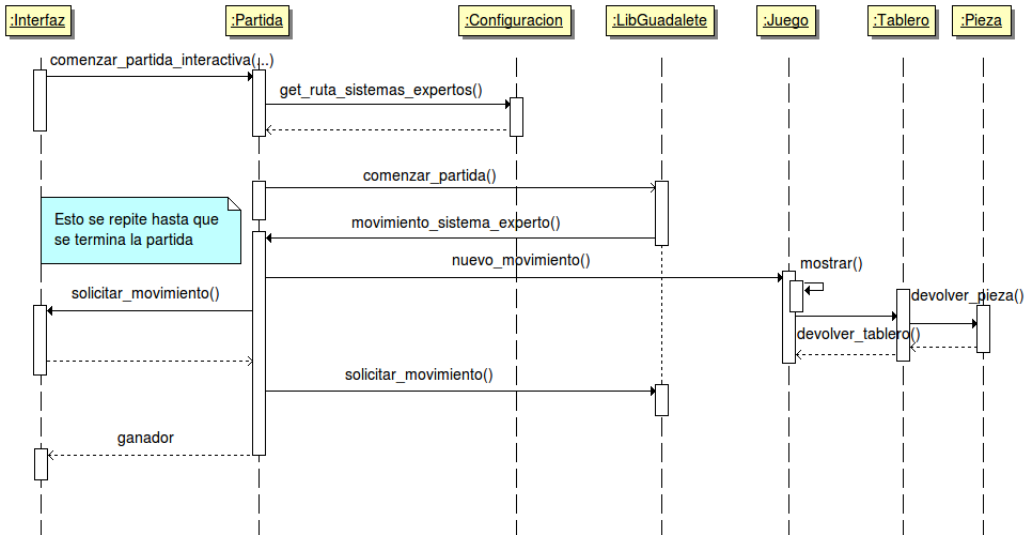


Figura 8.3: Diagrama de secuencia del sistema

3. El "representador" de la partida analiza el fichero, convirtiéndolo a un formato más fácilmente procesable, como una lista de matrices, siendo cada matriz un turno de la partida.

4. Los módulos que trabajan con Pygame pintan la partida, permitiendo la iteración por los turnos.

Como se puede apreciar, el comportamiento que se busca ahora es sustancialmente distinto. El objetivo que vamos a perseguir en la próxima etapa de implementación es introducir la interacción del usuario en medio de la etapa de simulación de la partida. Evidentemente, para ello tendremos que incluir también en medio de ese proceso la visualización de la partida para que el usuario sepa que movimientos realiza el sistema experto contra el que está jugando.

8.5. Implementación

Esta etapa, a pesar de ser la que ha tenido el análisis y el diseño más sencillo de todas las anteriores, es la que mas complicada ha sido de programar. Evidentemente era necesario modificar el núcleo de la aplicación para que las reglas de CLIPS permitieran esta funcionalidad.

Esta implementación se llevo a cabo en 3 pasos:

1. Integrar dicha funcionalidad en el núcleo original de CLIPS, sin usar Python.

2. Una vez se comprobó que eso era posible, implementar lo mismo en **LibGuadalete**.

3. Cuando se implementó correctamente en **LibGuadalete** se integró en la interfaz gráfica.

8.5.1. Paso 1: Núcleo original

Probablemente esta etapa fue la más sencilla de todas. Si recordamos como funciona CLIPS, la implantación de solicitar una entrada al usuario es sencillo. En lugar de cargar el fichero normal con las reglas básicas del equipo, se cargará el siguiente fichero:

```
(defmodule EQUIPO-A
  (import MAIN deftemplate initial-fact ficha
            dimension tiempo mueve tiempo-inicial)
  (import MAIN deffunction ?ALL))

(deffun EQUIPO-A::read-mov(?n ?t)
        (bind ?mov (read))
        (printout t "EQUIPO-B mueve a" ?n "
                    hacia " ?mov " en t " ?t crlf))

(defrule EQUIPO-A::lecturaA
  (declare (salience 100))
  (tiempo ?t)
  =>
  (bind ?n (read))
  (read-mov ?n ?t))

(defrule EQUIPO-A::termina
  (declare (salience 100))
  (tiempo ?t)
  (dimension ?dim)
  (mueve (num ?n) (mov ?m) (tiempo ?t))
  (ficha (equipo "A") (num ?n) (pos-x ?x) (pos-y ?y))
  (test (mov-valido ?dim ?m ?x ?y))
  (not (ficha (equipo "A")  (pos-x ?x2&:(= (+ ?x
       (mov-x ?m)) ?x2)) (pos-y ?y2&:
       (= (+ ?y (mov-y ?m)) ?y2))))
  =>
  (pop-focus))
```

La idea es la siguiente: el equipo A (en este caso) solo tendrá una regla de máxima prioridad que solicita que ficha se quiere mover, y en que dirección, realizando dicho movimiento.

Esto hace, que en cada turno del equipo A, lo único que ha CLIPS sea solicitar la entrada y mover la ficha. Con esto hemos solucionado el problema.

8.5.2. Paso 2: Integración en PyCLIPS

Este paso no fue tan sencillo por culpa de la librería PyCLIPS. Esta no utiliza el mismo flujo de entrada ó salida que utiliza CLIPS, es decir que la entrada de datos que utiliza CLIPS se simula mediante funciones. ¿Qué problema tiene esto? Recordemos que para ejecutar una simulación en PyCLIPS tenemos que hacer la función `clips.Run()`, que es equivalente a la función `(clips)`

Pero debido a que la entrada de datos en PyCLIPS se simula, la ejecución de `clips.Run()` no permite usar la entrada de datos de forma nativa, bloqueando la posibilidad de que el usuario mueva sus fichas.

Para solventar este problema se recurrió a una idea alternativa. Se puede integrar en el entorno PyCLIPS una función **Python** que lea de alguna forma esos datos y los inserte en vez de la función `(read)`:

```
import clips

def clips_piece_input():
    #get the value someway
    return clips.Integer(val)

def clips_move_input():
    #get the value someway
    return clips.Integer(val)

clips.RegisterPythonFunction(clips_piece_input, "piece-input")
clips.RegisterPythonFunction(clips_move_input, "move-input")

def LoadFunctions(clips, team):
    # Generates the CLIPS module
    mod_name = "EQUIPO-" + team
    # Module body
    mod_body  = "(import MAIN deftemplate initial-fact ficha dimension "\
                "tiempo mueve tiempo-inicial)" \
                "(import MAIN deffunction ?ALL)"
    mod_equipo = clips.BuildModule(mod_name, mod_body)

    # Function that reads the movement
    fun_name = 'read-mov'
    fun_para = '?n ?t'
    fun_body = '(bind ?mov (python-call move-input))' \
               '(printout t "EQUIPO-B mueve a" ?n "' \
               ' hacia " ?mov " en t " ?t crlf)' \
               '(assert (mueve (num ?n) (mov ?mov) (tiempo ?t)))'

    fun_lectura_mov = mod_equipo.BuildFunction(fun_name, fun_para, fun_body)

    # rule with max salience that will be trigged on every turn
    rule_name = 'lectura' + team

    rule_prec = '(declare (salience 100))' \
                '(tiempo ?t)'

    rule_body  = '(bind ?n (python-call piece-input))' \
                 '(read-mov ?n ?t)'

    lectura = mod_equipo.BuildRule(rule_name, rule_prec, rule_body)

    # Rule name
    rule_name = 'termina'
    # Rule precontents
    rule_prec  = '(declare (salience 100))'\
                 '(tiempo ?t)' \
```

```
                '(dimension ?dim)' \
                '(mueve (num ?n) (mov ?m) (tiempo ?t))' \
                '(ficha (equipo "A") (num ?n) (pos-x ?x) (pos-y ?y))' \
                '(test (mov-valido ?dim ?m ?x ?y))' \
                '(not (ficha (equipo "A")  (pos-x ?x2&:' \
                '(= (+ ?x (mov-x ?m)) ?x2)) (pos-y ?y2&:' \
                '(= (+ ?y (mov-y ?m)) ?y2))))'
# =>
# Rule body
rule_body  = '(pop-focus)'
# Building the rule
termina = mod_equipo.BuildRule(rule_name, rule_prec, rule_body)
# --------------------------------
```

De esta forma, el proceso es el siguiente:

1. Se ejecuta `clips.Run()`
2. Cada vez que le toca el turno al jugador, se le pasa el control a las funciones Python implementadas en el módulo de interacción.
3. Estas funciones se comunicarán de algunas forma con el usuario, obteniendo la ficha que se quiere mover, así como el movimiento.

8.5.3. Paso 3: Integración con la interfaz gráfica

Una vez hecho el paso 2, este es un poco más trivial. Se desarrolló una clase que se comunicaba entre la clase *LibGuadalete* y *Partida*. Esta clase captura los movimientos que indica el usuario, para que luego *LibGuadalete* pueda realizar esos movimientos.

Podemos ver un ejemplo de esta interfaz en la imagen 8.4

8.6. Pruebas

¿Hemos cumplido los objetivos que nos propusimos al comienzo de esta iteración? Como siempre, comprobaremos la corrección de los requisitos iniciales:

- *¿Le hemos proporcionado soporte al usuario para poder jugar contra cualquier sistema experto?*

  Si, existe una nueva opción en el menú principal para ello.

- *¿El usuario puede elegir una formación?*

  Si, es una de las opciones del diálogo de configuración de partida.

- *¿El usuario puede elegir el número de turnos de la partida?*

  Si, en las opciones del diálogo de configuración es una de las opciones disponibles.

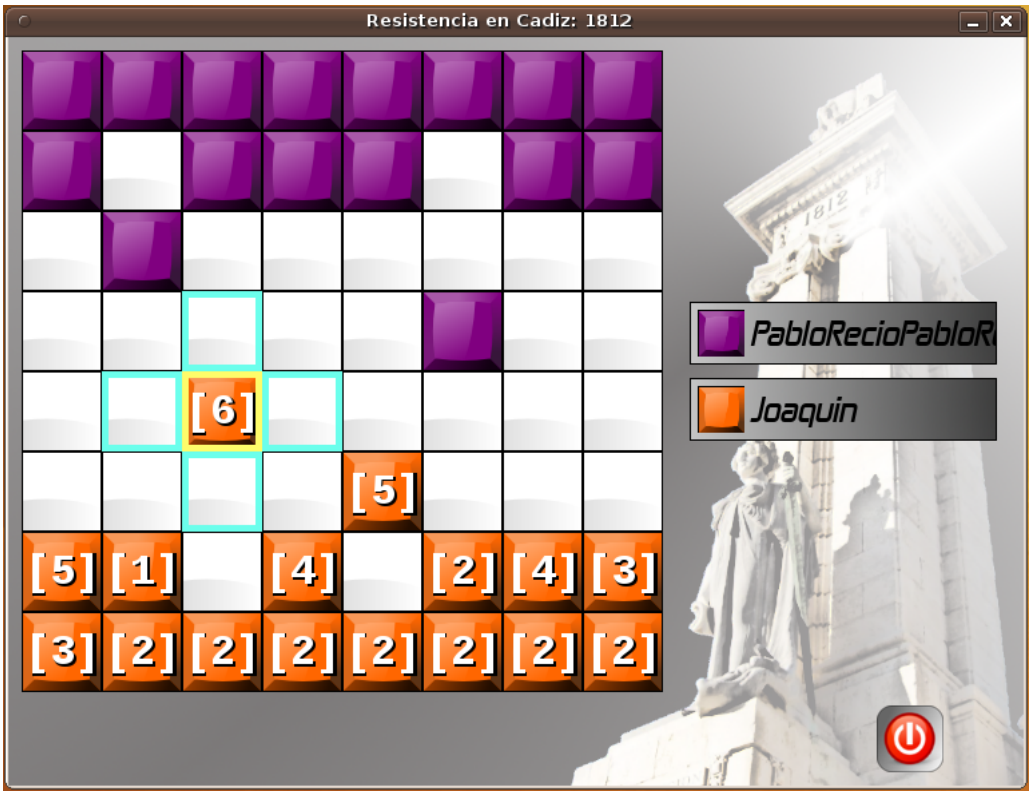


Figura 8.4: Interacción del usuario en una partida

- *¿El usuario puede elegir jugar contra un sistema experto al azar?*

  Si, en lugar de seleccionar los ficheros como se hace normalmente, es posible marcar esa opción.

- *¿El sistema controla correctamente los movimiento erróneos sobre el tablero?*

  Cuando se marca una ficha para moverla, se marcan a que posiciones puede moverse. En caso que el usuario no pulse en esas casillas, no se realizará ningún movimiento.

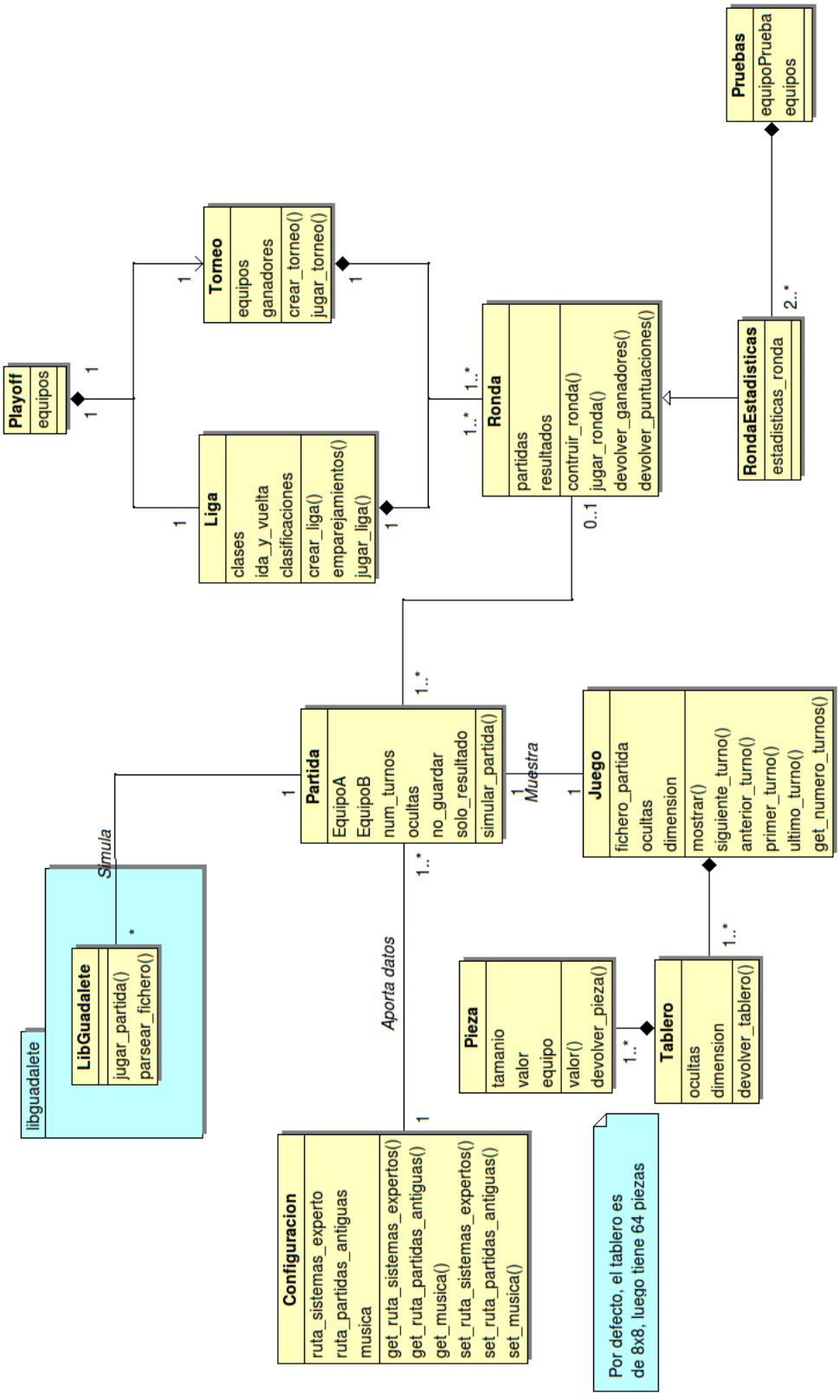


Figura 8.5: Diagrama general de clases

Capítulo 9

Conclusiones finales

9.1. Impresiones finales

Tras dedicarle varios meses a este **Proyecto Fin de Carrera**, y una vez terminado el desarrollo del mismo llega el momento de sacar las conclusiones, y analizar que tal se ha producido este desarrollo.

Personalmente me hubiera gustado terminar y presentar este proyecto antes de la fecha en lo que lo he hecho. Sin embargo, por motivos laborales esto no ha sido posible. De todas formas, el calendario inicial se hizo pensando en esta circunstancia y por tanto este calendario se ha cumplido en su mayoría.

Si bien es cierto que en alguna etapa se pudo provocar algún retraso por complicaciones no previstas, la planificación fue bastante realista. Aun así, no estimé todo lo bien que debería la duración de ciertas etapas, pues fueron decisiones que tomé en base a mi experiencia programando durante el transcurso de la carrera, y efectivamente en algún momento erré mis cálculos.

En lineas generales estoy bastante contento, tanto por el trabajo realizado como por la metodología de desarrollo que he seguido para construir esta aplicación. En la medida que ha sido posible, esta metodología ha seguido un desarrollo orientado a la filosofía de desarrollo del **Software Libre**. Siguiendo estas directivas, el código está **internacionalizado**, de forma que simplemente añadiendo los correspondientes ficheros de traducción, la aplicación soportaría cualquier lenguaje.

Así mismo, todo el código está escrito y comentado en **inglés**, con el fin de facilitar la reutilización de este código por terceras personas, sean de donde sea.

9.2. Conocimientos adquiridos

El desarrollo de un **Proyecto Fin de Carrera** se puede enfocar de dos formas:

- El último paso para conseguir el título de Ingeniero Técnico.
- Una oportunidad para demostrar que el alumno es capaz de desenvolverse en un proyecto, aplicando los conocimientos adquiridos, así como aprender a ser relativamente autónomo en el desarrollo de un proyecto.

Personalmente me he tomado este desarrollo como comento en el segundo punto. Es cierto que al fin y al cabo es el último paso para terminar este nivel de estudios, pero es la primera oportunidad que tenemos los alumnos de tomar decisiones con una importancia relativamente grande y que pueden influir de una forma u otra a la correcta realización de un proyecto.

Resumiendo un poco, durante el desarrollo de este proyecto he aprendido:

- Un nuevo lenguaje de programación, Python.
- A diseñar e implementar interfaces de usuario.
- Aunque ya había trabajado con Subversion, en este proyecto he utilizado Git, por lo que he aprendido dicho sistema de control de versiones.
- Las ideas fundamentales de como funcionan los sistemas expertos basados en reglas. Es un campo bastante interesante, que se debería explorar en más profundidad durante el desarrollo de la carrera.
- A intentar llevar y mantener un producto software de una dimensión relativamente grande.
- Para realizar una correcta planificación hay que ser realista con las capacidades de los trabajadores.
- Es importantísimo un código claro y bien documentado, para que alguien sea capaz de seguir con el trabajo que has realizado.

9.3. Futuras mejoras de la aplicación

Este proyecto lógicamente no es perfecto, y siempre hay funcionalidades que se pueden incluir para hacerlo un producto más atractivo. Mi idea como desarrollador del mismo es que esta aplicación no se quede parada tras la entrega y presentación de este Proyecto Fin de Carrera. Es más, como **alumno colaborador** del departamento de Lenguajes y Sistemas Informáticos, una de mis labores asignadas es este mismo proyecto por lo que no es algo que vaya a parar de desarrollar a corto plazo.

Algunas posibles mejores que se pueden realizar son las siguientes:

- Dar la posibilidad de modificar las formaciones de los equipos mediante una interfaz gráfica. Esto puede resultar interesante, para incluso permitir cambios antes de una partida, sin tener que modificar el fichero.
- Hacer que el núcleo de la aplicación sea ampliable. Por ejemplo, si añadimos un módulo, podemos modificar el tamaño del tablero, o bloquear el acceso a ciertas casillas, que el 1 no se mueva, etcétera. Esto le daría mucho juego para hacer variantes del juego mas complejas.
- Hacer problemas, al estilo de los problemas de Ajedrez ó los Tsumegos de Go[1]
- Realizar estudios durante la duración de una partida. Por ejemplo, si en un equipo quedan dos piezas, el 1 y el 2, y al rival le queda el 1 y el 5, según la posición de estas, definir si la partida está acabada o no.

[1] Juego de estrategia originado en China hace más de 2500 años http://en.wikipedia.org/wiki/Go_(game)

9.4. Resistencia en Cádiz: 1812 en cifras

Gracias a haber utilizado un sistema de control de versiones como Git, es muy sencillo sacar estadísticas a "posteriori" de lo que ha dado de sí el desarrollo del proyecto. Vamos a ver alguna de esas estadísticas:

- 122 revisiones de código.
- 9237 líneas de código totales.
- 5 ramas distintas utilizadas durante el desarrollo.

Se puede ver que con 122 revisiones, el trabajo ha quedado bien registrado, por lo que si fuera necesario volver a alguna versión antigua por algún error, ese cambio está bien registrado en estas revisiones.

9.5. Licencias libres

En este documento está liberado bajo licencia GFDL, y la aplicación **Resistencia en Cádiz:1812** está liberada bajo licencia GPLv3. Así mismo, todos los gráficos y músicas utilizados durante la aplicación usan licencias Creative Commons en varias de sus modalidades.

Las bibliotecas utilizadas para la aplicación, como son PyGTK, Pygame, Pycha o PyCLIPS, están licenciadas usando LGPL, permitiendo la reutilización de las mismas. De esta forma hemos logrado que toda la aplicación pueda liberarse sin ningún tipo de problema legal por un uso indebido de algún componente.

Como desarrollador me parece importante esto ya que no me gustaría que este proyecto se usase 1 ó 2 años en la asignatura de **Diseño de Videojuegos**, si no que me gustaría que cualquier persona que quiera adaptarlo a sus necesidades y seguir dándole uso y aprovechando el trabajo que he realizado. Es por eso que creo que es necesario utilizar **licencias libres** en el desarrollo de software, sobre todo de este tipo (educativo).

Manual de instalación

Requisitos

- Sistema de tipo GNU/Linux
- Python 2.6
- PyGTK versión 2.16 ó superior
- Pygame versión 1.8.1 ó superior
- PyCLIPS versión 1.0.7 ó superior

Esta aplicación no tiene una carga grande en lo que a hardware se refiere, por lo que con los requisitos mínimos de un sistema Debian, esta aplicación debería funcionar bien.

Sistemas basados en Debian

Para los sistemas basados en Debian se han realizado los respectivos paquetes `.deb` para instalar la aplicación de una forma sencilla, hay que descargarse el fichero situados en la forja de RedIRIS: http://forja.rediris.es/frs/download.php/1664/resistencia1812_1.0-1_all.deb

Dependencias

El paquete `.deb` comprueba las dependencias, de forma que si no están instaladas en tu sistema, te solicitará dicha instalación. Sin embargo, la biblioteca **PyClips** no está disponible aun en ningún repositorio Debian ó derivados. Es necesario descargarse dicha librería manualmente desde la página del proyecto: http://sourceforge.net/projects/pyclips/files/

Otros sistemas

Para otros sistemas, se puede compilar e instalar los fuentes directamente. Primer descargamos la última versión del software: http://forja.rediris.es/frs/download.php/1665/resistencia1812-1.0.tar.gz.
Lo descomprimimos, y dentro de la carpeta de los fuentes ejecutamos:

```
\item $ make
\item $ make locale
\item $ sudo make install
```

Y ya tendremos nuestra aplicación instalada.

Dependencias

Para resolver las dependencias, os recomiendo usar el gestor de paquetes de vuestra distribución. Por si alguna de las bibliotecas que comento no están disponibles en vuestro gestor, aquí indico las direcciones oficiales de dichos proyectos, para seguir sus instrucciones para la instalación:

PyGTK: http://www.pygtk.org/

Pygame: http://www.pygame.org/

PyCLIPS: http://pyclips.sourceforge.net/web/

Pycha: http://www.lorenzogil.com/projects/pycha/

Manual de usuario

Conceptos Básicos

Resistencia en Cádiz: 1812 es una versión simplificada del juego *Stratego* utilizada para el diseño de *sistemas expertos*. Dicho de otra forma, es un entorno de desarrollo en el cual el usuario podrá comprobar el funcionamiento de un sistema experto.

En nuestro caso, el sistema experto se identifica con un equipo el cual se compone de **reglas** y una **formación inicial**. Tanto las reglas como la formación inicial están codificadas mediante **CLIPS** de la forma en que se explica en el *Manual de programador* 9.5.

Un aspecto a tener en cuenta a lo largo del manual de usuario de la intefaz es la organización de los ficheros de la aplicación. Una vez instalada la aplicación, encontramos en nuestro directorio `home` una carpeta con nombre `.resistencia1812`. Dentro de ésta se encuentra un directorio llamado `games` donde se almacena todos los ficheros de *partidas jugadas*, los registros de los *torneos realizados* y los ficheros de las *estadísticas* para un equipo. Además, nos encontramos con la carpeta `teams` donde se almacenan los equipos distribuyendo las reglas en el directorio `teams/rules` y la formación inicial en `teams/formations`.

Menú principal

Una vez completado la instalación de nuestra aplicación podemos iniciarla accediendo mediante **Aplicaciones** → **Juegos** → **Resistencia en Cádiz: 1812** 9.1.
Al iniciar nos encontramos con la pantalla principal del entorno siendo ésta el **menú principal** 9.2 que recoge en ella todas las funcionalidades del sistema.

```
~/.resistencia1812
--| games/
--| teams/
----| formations/
----| rules/
--| configuration.xml
```

Las funcionalidades que ofrece son:

- **Partida rápida**: Esta opción se utiliza para jugar una partida con dos sistemas expertos incluidos en la aplicación o codificados por el usuario.
- **Competición**: En competición podremos realizar *torneos*, *ligas* y *playoff* entre varios equipos.

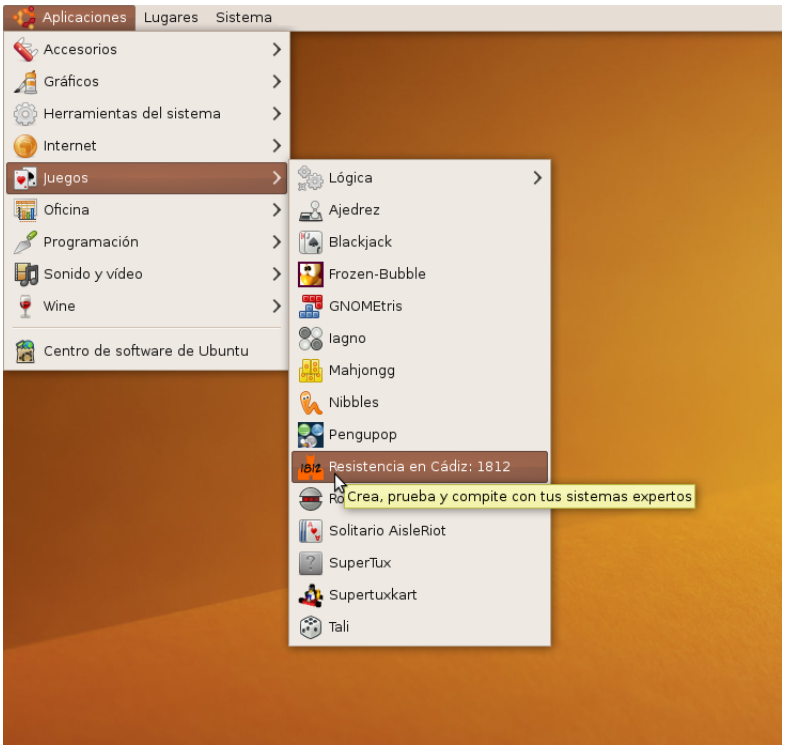


Figura 9.1: Ruta a seguir para iniciar Resistencia en Cádiz: 1812

- **Jugar contra un sistema experto**: Lugar donde podremos probar nuestro sistema experto contra nosotros mismos, es decir, es la misma persona la que juega contra un equipo.
- **Laboratorio**: Zona de pruebas estadísticas para un equipo concreto.
- **Partidas antiguas**: En esta opción podremos visualizar las partidas jugadas anteriormente mediante la carga del fichero de dicha partida guardado en `/home/usuario/.resistencia1812/games`
- **Configuración**: También incluye la funcionalidad de poder configurar aspectos de la aplicación: música y rutas de ficheros.

A continuación se explicará la navegación por las distintas funcionalidades del sistema.

Funcionalidades

Partida rápida

Podemos ver el diálogo en la imagen 9.3. Con esta funcionalidad podemos ejecutar una partida entre dos sistemas expertos o equipos.

Para ello, tenemos que elegir dichos equipos a partir del fichero de reglas y del fichero de formación de cada uno facilitándonos la tarea con un selector de archivos por cada uno.

Además el diálogo ofrece otras opciones que puede configurar el usuario:

Figura 9.2: Menú principal de Resistencia en Cádiz: 1812

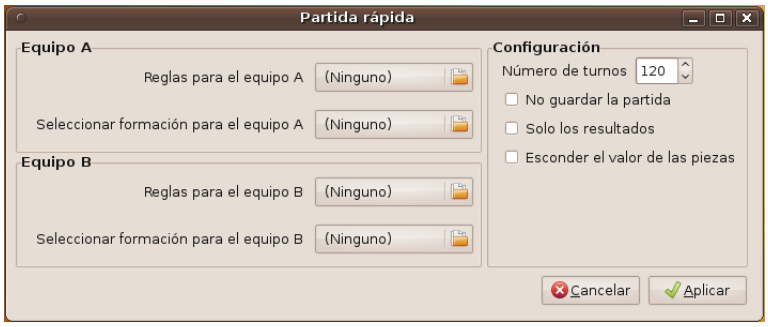


Figura 9.3: Diálogo de configuración para una partida rápida

- *Número de turnos*: Define el número máximo de turnos que puede llegar a tener la partida. Hay que tener en cuenta que los turnos se cuentan por movimiento realizado en la partida, es decir, configurando unos 200 turnos, cada jugador realizará 100 movimientos.
- *No guardar la partida*: Con esta opción podemos deshabilitar el guardado automático de la partida que se va a realizar.
- *Solo los resultados*: Si marcamos esta opción no se visualizará el tablero con la partida completa sino que sólo se mostrará el ganador.
- *Esconder el valor de las piezas*: También es posible el darle más interés a la partida escondiendo el valor de las piezas descubríendose solamente durante la partida con las acciones pertinentes.

Como es de esperar, siempre podremos cancelar para volver al menú principal si la opción elegida no es la deseada.

Como ejemplo, hemos tomado una partida rápida en la cual se esconde inicialmente el valor de las piezas nos dará un resultado como el de la figura 9.4.

Figura 9.4: Partida rápida sin valor en las piezas

Al finalizar la partida y al configurar la partida marcando la opción de mostrar solo resultado, nos muestra el resultado final de la partida, en la imagen 9.5.

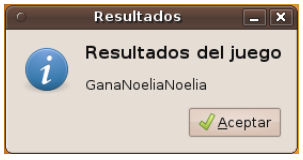


Figura 9.5: Finalización y muestra del resultado de la partida rápida

Competiciones

En esta opción podemos elegir entre varios tipos de competiciones, como se nos muestra en la imagen 9.6:

- **Liga**: una liga se compone de partidas en las cuales todos los participantes se han enfrentado con todos. Si el usuario quisiera, puede elegir la opción de realizar una liga con *ida y vuelta* para que cada partida se repita invirtiendo la posición de los equipos.
- **Copa eliminatoria**: definimos esta competición como un torneo en el cual los equipos se emparejan aleatoriamente, avanzando por árbol de jugadas los ganadores de las partidas y se eliminan los perdedores hasta que queden dos, juegen su partida y se obtenga el equipo ganador de la copa. En este tipo de competición eliminamos la opción *ida y vuelta* ya que es incompatible con esta clase de competición.
- **Playoff**: Los playoff se componen de una liga entre todos los participantes y una vez obtenidos los resultados finales, la mitad de los participantes en mejor posición pasarán a realizar una copa

eliminatoria obteniendose así el ganador del playoff. Como es obvio, en la parte de liga dentro del playoff podemos elegir si realizar *ida y vuelta* o no, sin embargo en la parte del torneo no se realiza.

En este diálogo tenemos las opciones de configuración de *turnos por partida* e incluso ver *sólo resultados* y no tener porqué visualizar la pantalla de la partida.

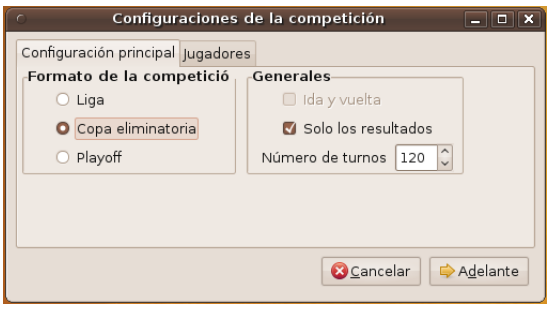


Figura 9.6: Primera parte del diálogo de configuración para competiciones.

Si cambiamos de pestaña, podremos ver esta imagen 9.7, para poder elegir los participantes. En este diálogo podemos utilizar dos medios:

1. Mediante la elección de un equipo por su formación y reglas (uno a uno).
2. Mediante la opción de utilizar como participantes todos los equipos instalados.

Figura 9.7: Segunda parte del diálogo de configuración para competiciones.

Una vez que terminan de ejecutarse todas las partidas de una ronda, muestra los resultados parciales. Tenemos un ejemplo en la imagen 9.8 Hay que distinguir que entre los tipos de competiciones ya que cada resultado se mostrará de distinta forma. Por ejemplo, en las ligas se muestran todas las partidas jugadas siendo el jugador resaltado en azul el ganador de dicha partida y los que estén en verde quiere decir que han empatado. Sin embargo, en los playoff se muestra lo anterior y además la clasificación de jugadores para saber en cada momento quienes pasan a la segunda fase:
Finalmente, el sistema mostrará los resultados finales en el que podemos visualizar el ganador de la competición.

Jugar contra un Sistema Experto

En este caso, para uno de los equipos se utilizará un fichero con la formación inicial definida y por reglas no se utilizará un fichero sino el propio usuario. Podemos ver una captura en la imagen 9.9

Figura 9.8: Resultados parciales de un playoff en el que participan todos los jugadores.

También nos da la elección de quién se posiciona como equipo A o como equipo B para comprobar las estrategias implementadas en ambos vandos. Además podemos elegir jugar contra un sistema experto al azar y comprobar si dicho sistema es capaz de ganar al usuario.

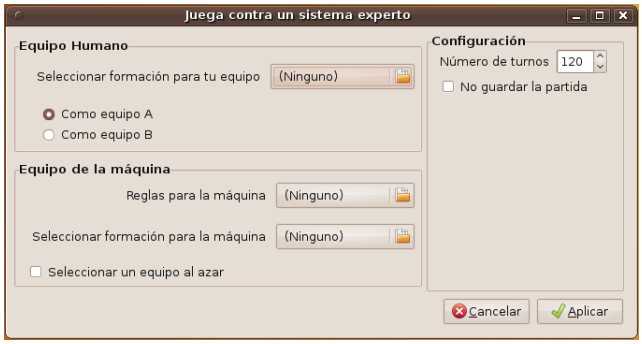


Figura 9.9: Diálogo de configuración para jugar contra el sistema experto.

Se podrá interactuar con el tablero por medio del ratón seleccionando la pieza que se quiera mover. Una vez seleccionada se marcan los movimientos válidos que se pueden realizar con esa pieza y seguidamente se marca el movimiento deseado, como vemos en la imagen 9.10.

Finalmente, una vez que termine la partida se cierra la ventana de dicha partida mostrando el ganador al igual que hemos visto en otras funcionalidades.

Laboratorio

Esta opción es una de las más interesantes para el programador de un sistema experto. Es aquí donde evaluar la estrategia que esté diseñando.
Una vez accedido a la funcionalidad, ésta nos muestra un diálogo de configuración para seleccionar el sistema experto a probar y con qué sistemas expertos probarlos. Además, ofrece la oportunidad de ampliar las pruebas realizando más repeticiones de todas las partidas que se pueden generar para que la

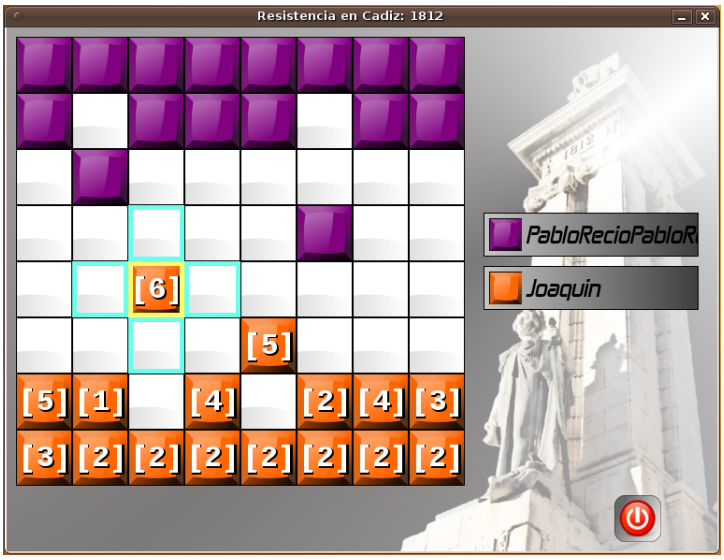


Figura 9.10: Tablero que muestra la interacción del usuario con la partida.

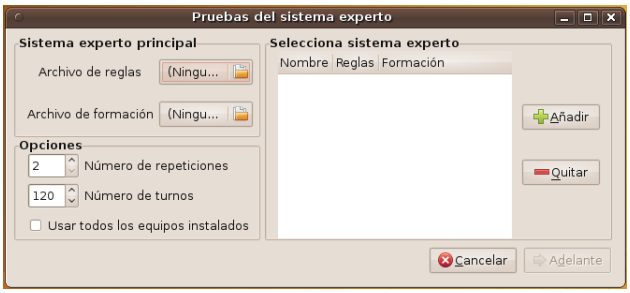


Figura 9.11: Diálogo de configuración para las pruebas de un sistema experto concreto.

estadística resultante sea más fiable. El ejemplo de este diálogo está visible en la imagen 9.11

Una vez calculadas todas las pruebas la aplicación muestra las estadísticas resultantes en base a las siguientes características:

- Número de partidas realizadas totales.
- Número de partidas ganadas, número de partidas perdidas y número de partidas empatadas.
- Media de turnos de las partidas en las que ganas y media de turnos de las partidas en las que pierdes.
- Media de turnos que sobrevive tu pieza de mayor valor.

Estas estadísticas se muestran en el diálogo 9.12 agrupadas en un gráfico y en una especie de tabla.

Partidas antiguas

Como se ha comentado anteriormente, la aplicación guarda las partidas jugadas en un fichero .txt dentro de la carpeta especificada y con un formato en el cual se informa de la fecha, hora, y equipos que

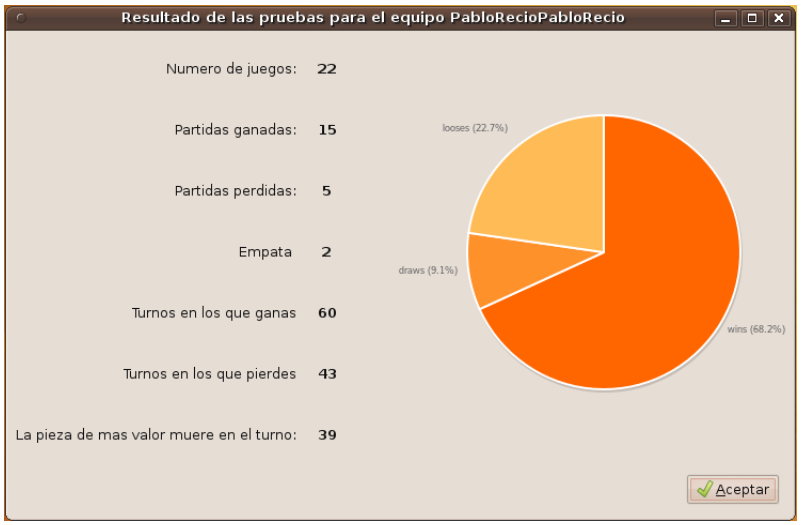


Figura 9.12: Estadísticas que ofrece la funcionalidad después de realizar las pruebas.

componen la partida
En esta opción podemos elegir cualquier fichero de partidas con el selector de fichero predeterminado de GNOME 9.13 para poder reproducirla y volverla a visualizar. Una vez elegido el archivo la aplicación muestra la pantalla con el tablero como hemos estado viendo hasta ahora en casi todas las funcionalidades. La peculiaridad de ésta es que siempre se verán las puntuaciones de las piezas, como demuestra la imagen 9.14.
Como es de esperar, una vez que cerremos esta pantalla se nos notificará el ganador de la partida.

Configuración

En la zona de configuración general de la aplicación podemos cambiar varios aspectos de la misma: las rutas a los directorios que utiliza la aplicación y el activar o no la música.

Existe una peculiaridad en la elección de la ruta de los equipos. Imaginemos que creamos un directorio vacío el cual va a ser el futuro directorio donde se almacenarán los equipos. Si no se expecifica nada más, todos los ficheros que componen un equipo se almacenarán conjuntamente, con lo que puede dificultar la elección de éstos en otras funcionalidades. Sin embargo, si al directorio anterior le especificamos internamente los directorios de `formations` y `rules` el usuario almacenará los archivos de los equipos en sus respectivos directorios y la elección de los ficheros se realizará de la misma forma que ofrece la aplicación por defecto, reduciendo el número de fallos al seleccionar los archivos en otras funcionalidades.

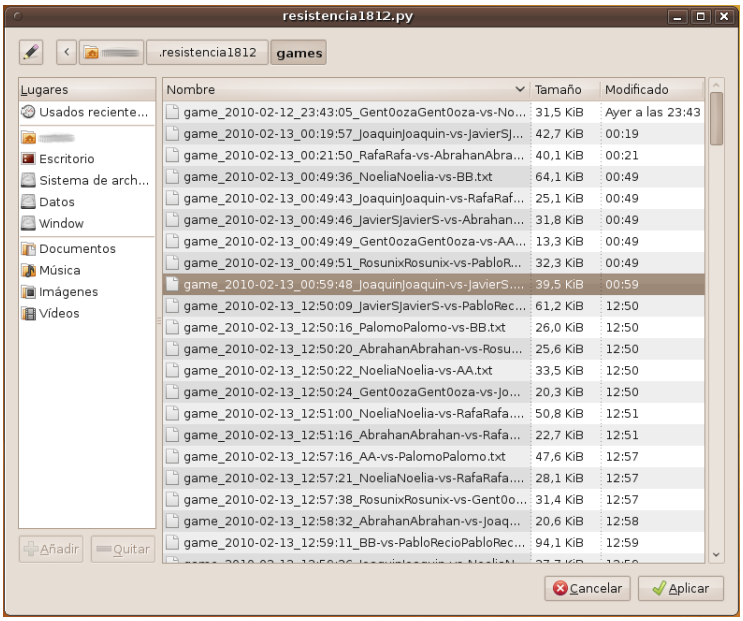


Figura 9.13: Elección del fichero de la partida a visualizar.

Figura 9.14: Visualización de una partida antigua.

Manual de usuario programador

Introducción

En este manual se desarrollará la creación y codificación de un sistema experto usable en la aplicación **Resistencia en Cádiz:1812**. En este caso, el sistema experto se identifica como un equipo compuesto por una formación inicial y unas reglas y con el objetivo de ganar y ser el mejor equipo posible ejecutado en la aplicación.

Los componentes del equipo se representan mediante ficheros nombrados de la forma `equipoXXX.clp` para la formación inicial y `reglasXXX.clp` para las reglas de dicho equipo. La nomenclatura `XXX` corresponde al nombre específico que se le haya asignado al equipo, por ejemplo, `equipoPablo.clp | reglasPablo.clp`.

Tanto la formación como las reglas están codificadas bajo **CLIPS** pero, ¿qué es?. Es una herramienta que provee un ambiente de desarrollo para la producción y ejecución de *sistemas expertos*. Está basada fundamentalmente en la **programación lógica de sistemas expertos basados en reglas** que permite que el *conocimiento* sea representado como **reglas heurísticas** que especifican las acciones al ser ejecutadas dada una situación.

Como cualquier lenguaje basado en reglas para sistemas expertos, CLIPS trabaja con **reglas** y **hechos**. Los hechos se corresponde con el conocimiento contenido en el sistema experto. Para declarar hechos se utiliza la palabra reservada **deffacts** especificando el nombre del hecho y el conocimiento. En nuestro caso, esta va a ser la manera de definición de la formación inicial, siendo ésto, la única base de conocimiento que se necesita en nuestro sistema experto.

Las reglas se corresponden con el motor de inferencia definido en el sistema experto. Declaramos reglas a partir de la palabra reservada **defrule** añadidendo las precondiciones necesarias para que la regla se active y pueda llegar a ejecutarse dependiendo de la prioridad dada y de si no existe una regla activa con mayor prioridad (ya que sino se ejecutaría ésta última). Las reglas a definir en nuestro equipo se reducirán a la realización de un movimiento de una pieza dependiendo de la situación en la que se encuentre.

Para añadir comentarios a los ficheros de hechos y reglas utilizaremos el símbolo **;** antes del comentario a insertar.

Además, comentar que CLIPS tiene una sintaxis basada en **paréntesis balanceados**. Esto quiere decir que todo ámbito, definición, o inicialización de valores debe estar rodeado de sus correspondientes paréntesis para evitar errores. Un ejemplo de regla puede ser al siguiente:

```
(defrule regla-ejemplo
```

```
      (objeto1 (atributo1 "valor") (atributo2 ?x))
      (test (> ?x 20))
=>
      (assert (objeto2 (atributo1 "valor")))
)
```

Formación de un equipo

Como hemos comentado anteriormente, la formación del equipo corresponde con los hechos asociados a un sistema experto, es decir, **el conocimiento**. Con lo cual se tienen que definir el *estado inicial de cada pieza* en el tablero representándose así todo el conocimiento del sistema experto en este caso.

Para la representación de las piezas del equipo utilizaremos el fichero `equipoXXX.clp`. Seguidamente redactamos todos los hechos dentro del ámbito **deffacts** y por cada hecho tenemos que definir:

- **ficha-r** Con esto creamos el hecho que existe una **ficha real** definida mediante las siguientes características.
- **equipo** Definimos de que tipo de equipo son las fichas. Para evitar fallos de codificación hay que tener en cuenta que **siempre** deben ser las piezas del equipo **A** y nunca del **B**.
- **num** Representa el identificador exclusivo de una pieza concreta.
- **puntos** Representa la puntuación de la pieza. Hay que tener en cuenta que deben existir:
  - Una única pieza de 1 punto.
  - Ocho piezas de 2 puntos.
  - Dos piezas de 3 puntos.
  - Dos piezas de 4 puntos.
  - Dos piezas de 5 puntos.
  - Una única pieza de 6 puntos.

  De esta forma representamos la puntuación de las 16 piezas que compone el equipo.
- **pos-x** Indica la posición en el eje *X* de coordenadas de la pieza. Se define en el rango **[1,8]**.
- **pos-y** Indica la posición en el eje *Y* de coordenadas de la pieza. Al estar definiendo piezas para el equipo A, el rango donde se deben de posicionar es **[1,2]**.
- **descubierta** Esta opción debe permanecer siempre a *cero* cuando estamos definiendo los hechos ya que al inicio de la partida todas las piezas deben tener su puntuación oculta.

Veamos ahora un ejemplo real de la descripción de la formación del equipo:

```
(deffacts fichas-A

   (ficha-r  (equipo "A") (num 111) (puntos 1)
             (pos-x 1) (pos-y 1) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 112) (puntos 2)
```

```
            (pos-x 2) (pos-y 1) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 113) (puntos 3)
            (pos-x 3) (pos-y 1) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 114) (puntos 2)
            (pos-x 4) (pos-y 1) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 115) (puntos 2)
            (pos-x 5) (pos-y 1) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 116) (puntos 2)
            (pos-x 6) (pos-y 1) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 117) (puntos 2)
            (pos-x 7) (pos-y 1) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 118) (puntos 2)
            (pos-x 8) (pos-y 1) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 121) (puntos 6)
            (pos-x 1) (pos-y 2) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 122) (puntos 3)
            (pos-x 2) (pos-y 2) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 123) (puntos 5)
            (pos-x 3) (pos-y 2) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 124) (puntos 4)
            (pos-x 4) (pos-y 2) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 125) (puntos 2)
            (pos-x 5) (pos-y 2) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 126) (puntos 2)
            (pos-x 6) (pos-y 2) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 127) (puntos 5)
            (pos-x 7) (pos-y 2) (descubierta 0) ) ;; Cierra ficha-r

   (ficha-r  (equipo "A") (num 128) (puntos 4)
            (pos-x 8) (pos-y 2) (descubierta 0) ) ;; Cierra ficha-r

) ;; Cierra deffacts.
```

A partir de esta formación, el resultado lo podemos ver en la siguiente figura 9.15.

Figura 9.15: Formación para los hechos definidas anteriormente.

Reglas de un equipo

Las reglas de un sistema experto se definen mediante las precondiciones que deben ser cumplidas para activar la regla y mediante las postcondiciones que se ejecutarán si es ésta la elegida por el algoritmo de CLIPS. Por tanto, se componen de tres parte:

1. **Nombre y características de la regla** Con esto definimos el nombre de la regla (siempre debe ser única) y las características de la misma como puede ser la **prioridad** en cuanto a las demás.

2. **Precondiciones** Define la situación que se debe cumplir para que la regla se active y pueda ser ejecutada en un momento determinado.

3. **Postcondicones** Define, generalmente y en nuestro caso, el movimiento que realizará la pieza concreta que entra en el perfil de la situación, cumpliendo con todas las precondiciones. Aparte, puede ser posible la activación de **flags** para tener en cuenta otras reglas en futuras activaciones.

Dentro del entorno **defrule** de una regla, podemos utilizar lo siguiente:

- **declare** Aquí se declara la prioridad de esta regla siendo un valor perteneciente a **[1,80]**. Esta declaración pertenece a la sección de características de una regla.

- **ficha** En una precondición que utilice este aserto se define que la ficha especificada por sus atributos (**equipo** -A o B-, **num** -identificador-, **pos-x** -[1,8] o cualquiera-, **pos-y** -[1,8] o cualquiera-, **puntos** -[1,6] o cualquiera- y/o **descubierta** -[0,1] o cualquiera-) existe todavía en la partida o no. Un ejemplo puede ser:

  ```
  (ficha (equipo "B") (num ?n) (pos-x ?x) (pos-y ?y)
         (puntos 6) (descubierta 1) )
  ```

  donde se comprueba si existe la ficha del equipo contrario (B) en cualquier posición, con puntuación 6 y descubierta. Al saber que es única, si en la partida existe todavía dicha pieza pero no está descubierta esta precondición sería falta y no activaría la regla que la contiene. También podemos utilizar **(not (ficha (...)))** para comprobar que ya no existe la ficha deseada.

- **tiempo** Podemos utilizar también el numero de turnos para poder activar reglas en un determinado tiempo dentro de la partida. Pertenece al grupo de las precondiciones.

- **test** Con este aserto podemos aplicar operadores matemáticos con los atributos definidos en las fichas. Para explicarlo mejor veamos las siguientes precondiciones:

```
(ficha (equipo "A") (num ?n1) (pos-x ?x1) (pos-y ?y1)
       (puntos ?p1) (descubierta ?d1) )  ;; Ficha equipo A
(ficha (equipo "B") (num ?n2) (pos-x ?x2) (pos-y ?y2)
       (puntos ?p2) (descubierta ?d2) )  ;; Ficha equipo B
(test (= ?x1 ?x2)) ;; Si están en la misma posición en el eje X.
(test (> ?y1 ?y2)) ;; Si la pieza A está en mayor posición que
                      la pieza B en el eje Y.
(test (= (- ?y1 ?y2) 1)) ;; Si la diferencia entre la posición
                            de las piezas en el eje Y es de 1 unidad.
(test (> ?p1 ?p2)) ;; La puntuación de la pieza A es mayor que de
                      la pieza B
```

Estas precondiciones describen la situación de: en cualquier posición del tablero existe una pieza del equipo A y otra del equipo B adyacentes en el eje Y donde la pieza del equipo A está situada encima de la pieza del equipo B y la primera con mayor puntuación que la segunda. Como es de esperar, esta sentencia forma parte de la precondición.

- **assert** Esta sentencia es la que realiza el movimiento de la pieza dada, es decir, completa la regla con la postcondición. Si seguimos el ejemplo anterior, su postcondición será:

```
=>  ;; Comienzo de las postcondiciones en una regla.
(assert (mueve(num ?n1) (mov 4) (tiempo ?t))
```

De esta forma completamos la regla con la acción que vaya a ejercer en la partida. La sentencia **mueve** sólo necesita el identificador de la pieza y el movimiento siendo **1-Derecha**, **2-Izquierda**, **3-Arriba** y **4-Abajo**. Además también podemos encontrarnos **assert** de este estilo:

```
=>
(assert (pieza_p6_muerta))
```

Cuando realizamos las comprobaciones pertinentes y concluimos con que la pieza de mayor valor está muerta. Además esto define una **flag** que podemos utilizar en la precondición de otras reglas de la forma:

```
(pieza_p6_muerta)
```

Si añadimos esa precondición comprobará si la flag existe (porque se haya ejecutado una regla anterior que la crea) continuando con el análisis de la regla si es cierto o pasando a analizar la siguiente si no.

Veamos ahora, en una regla codificada de un equipo las tres partes indicadas anteriormente:

```
;;;;;;;;;;;;;;;; FICHAS DE NIVEL 5 ;;;;;;;;;;;;;;;;
;;; las fichas de nivel 5 sirven de "barrera" ;;;

(defrule EQUIPO-A::rastreando-izq
         ;;; Controla que no se acerque ninguna ficha enemiga
    (declare (salience 77))
    (tiempo ?t)
```

```
    (ficha (equipo "A") (num ?n) (pos-x ?x) (pos-y ?y) (puntos 5))
    (ficha (equipo "B") (num ?n2) (pos-x ?x2) (pos-y ?y2))
    (test(> ?x ?x2)) ; Se acerca por la izquierda
    (test(< ?y ?y2))
    (test(< (- ?y2 ?y) 2)) ;Esta a dos en vertical
    (test(< (- ?x ?x2) 2))
    =>
    (printout t ?n ": Enemy spotted (rastrando-izq)" crlf)
    (assert (mueve(num ?n) (mov 2) (tiempo ?t)))
)

(defrule EQUIPO-A::ataca-barrido-izq
    (declare (salience 78))
    (tiempo ?t)
    (ficha (equipo "A") (num ?n) (pos-x ?x) (pos-y ?y) (puntos 5))
    (ficha (equipo "B") (num ?n2) (pos-x ?x2) (pos-y ?y)(puntos ?p))
    (test (= (- ?x ?x2) 1) )
    (test (>= ?p 5))
    =>
    (printout t ?n ": NO PASARAS!!! (izquierda)" crlf)
    (assert (mueve(num ?n) (mov 2) (tiempo ?t)))
)

(defrule EQUIPO-A::rastreando-dch
        ;;; Controla que no se acerque ninguna ficha enemiga
    (declare (salience 77))
    (tiempo ?t)
    (ficha (equipo "A") (num ?n) (pos-x ?x) (pos-y ?y) (puntos 5))
    (ficha (equipo "B") (num ?n2) (pos-x ?x2) (pos-y ?y2))
    (test(< ?x ?x2)) ; Se acerca por la derecha
    (test(< ?y ?y2))
    (test(< (- ?y2 ?y) 2)) ;Esta a dos en vertical
    (test(< (- ?x2 ?x) 2))
    =>
    (printout t ?n ": Enemy spotted (rastrando-dch)" crlf)
    (assert (mueve(num ?n) (mov 1) (tiempo ?t)))
)

(defrule EQUIPO-A::ataca-barrido-dcha
    (declare (salience 77))
    (tiempo ?t)
    (ficha (equipo "A") (num ?n) (pos-x ?x) (pos-y ?y) (puntos 5))
    (ficha (equipo "B") (num ?n2) (pos-x ?x2) (pos-y ?y)(puntos ?p))
    (test (= (- ?x2 ?x) 1) )
    (test (>= ?p 5))
    =>
    (printout t ?n ": NO PASARAS!!! (derecha)" crlf)
    (assert (mueve(num ?n) (mov 1) (tiempo ?t)))
```

```
)

(defrule EQUIPO-A::pos-original-izq-1
    (declare (salience 65))
    (tiempo ?t)
    (ficha (equipo "A") (num A123) (pos-x ?x) (pos-y ?y) (puntos 5))
    (test (= ?y 4))
    (test (< ?x 3))
    =>
    (printout t "A123: Pos original (pos-original-izq-1)" crlf)
    (assert (mueve(num A123) (mov 1) (tiempo ?t)))
)

(defrule EQUIPO-A::pos-original-izq-2
    (declare (salience 65))
    (tiempo ?t)
    (ficha (equipo "A") (num A123) (pos-x ?x) (pos-y ?y) (puntos 5))
    (test (= ?y 4))
    (test (> ?x 3))
    =>
    (printout t "A123: Pos original (pos-original-izq-2)" crlf)
    (assert (mueve(num A123) (mov 2) (tiempo ?t)))
)

(defrule EQUIPO-A::pos-original-dch-1
    (declare (salience 65))
    (tiempo ?t)
    (ficha (equipo "A") (num A127) (pos-x ?x) (pos-y ?y) (puntos 5))
    (test (= ?y 4))
    (test (> ?x 7))
    =>
    (printout t "A127: Pos original (pos-original-dch-1)" crlf)
    (assert (mueve(num A127) (mov 2) (tiempo ?t)))
)

(defrule EQUIPO-A::pos-original-dch-2
    (declare (salience 65))
    (tiempo ?t)
    (ficha (equipo "A") (num A127) (pos-x ?x) (pos-y ?y) (puntos 5))
    (test (= ?y 4))
    (test (> ?x 7))
    =>
    (printout t "A127: Pos original (pos-original-dch-2)" crlf)
    (assert (mueve(num A127) (mov 1) (tiempo ?t)))
)
```

Programación con PyGTK y Glade

Breve introducción

Vamos a realizar un pequeño tutorial introductorio a **PyGTK** y **Glade**. Estas dos tecnologías conjuntas es una de las formas más comunes de trabajar para desarrollar aplicaciones para el escritorio libre **GNOME**. Pero, ¿en qué consisten?

Glade es una herramienta para diseñar interfaces de usuario de forma gráfica. Podemos ver un ejemplo en la imagen 9.16.

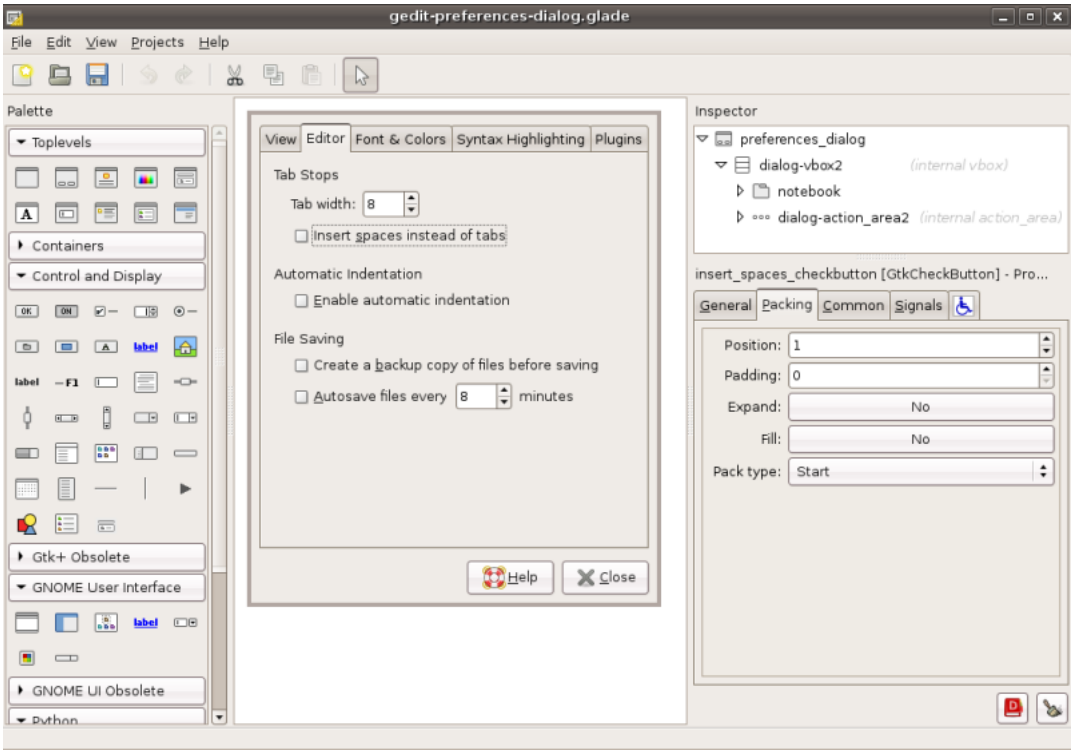


Figura 9.16: Interfaz de Glade

Esta herramienta nos permite diseñar la interfaz de cualquier aplicación, y luego a base de código podemos añadirle las funcionalidades. Es importante que Glade genera código XML para poder montar la interfaz. Esto implica que es independiente del lenguaje, si bien es cierto que se usa normalmente C ó Python, como en este caso que usaremos Python para montar nuestro pequeñó ejemplo.

Interfaz

Podemos comenzar con algo simple: un pequeño contador con dos botones que incremente y decremente un valor. Para ello, abrimos Glade y creamos una ventana desde la botonera de la izquierda como vemos en la imagen 9.17.

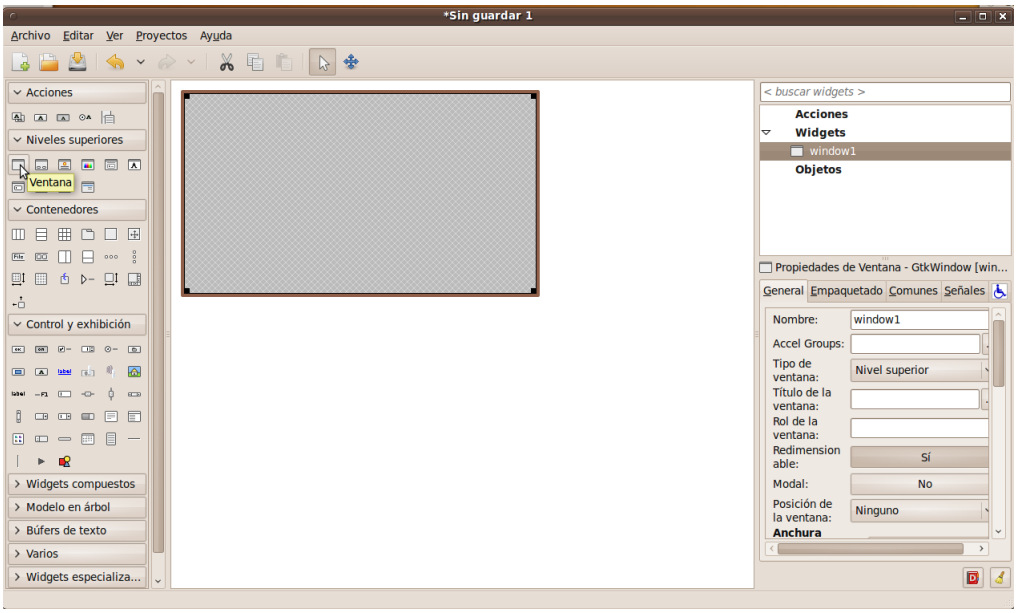


Figura 9.17: Interfaz de Glade

Para poder organizar los widgets que montemos dentro de la ventana, tenemos que montar cuadrículas o separadores para poder colocarlo todo de una forma ordenada. Por ejemplo en la imagen 9.18 vemos que hemos creado un separador horizontal de dos elementos.
De esta misma forma, podemos crear diversos elementos para nuestra ventana. Por ejemplo, vamos a crear una botonera horizontal con dos elementos para los dos botones, y encima vamos a poner un *label* donde pondremos el texto. El resultado final debería quedar como el de la imagen 9.19.
Podéis ver a la derecha, en el árbol de widgets, que hemos cambiando los nombres a algunos de los elementos. Esto es importante para acceder a ellos de una forma clara y simple, como veremos en el código.

Como última tarea antes poder terminar con Glade, tenemos que **asignar los eventos**. Si seleccionamos uno de los dos botones y nos vamos a las propiedades de la ventana (abajo a la derecha) y marcamos **señales**. Ahí podemos ver un desplegable con todas las señales asociadas a ese elemento, que en este caso es un botón. Como podemos ver en la imagen 9.20, solo hace falta asignar la señal *on-click*.
Recordamos que este paso es genérico tanto para C como para Python.

Código

Veamos primero el código completo en Python para luego ir desgranándolo:

```
import gtk
import gtk.glade
```

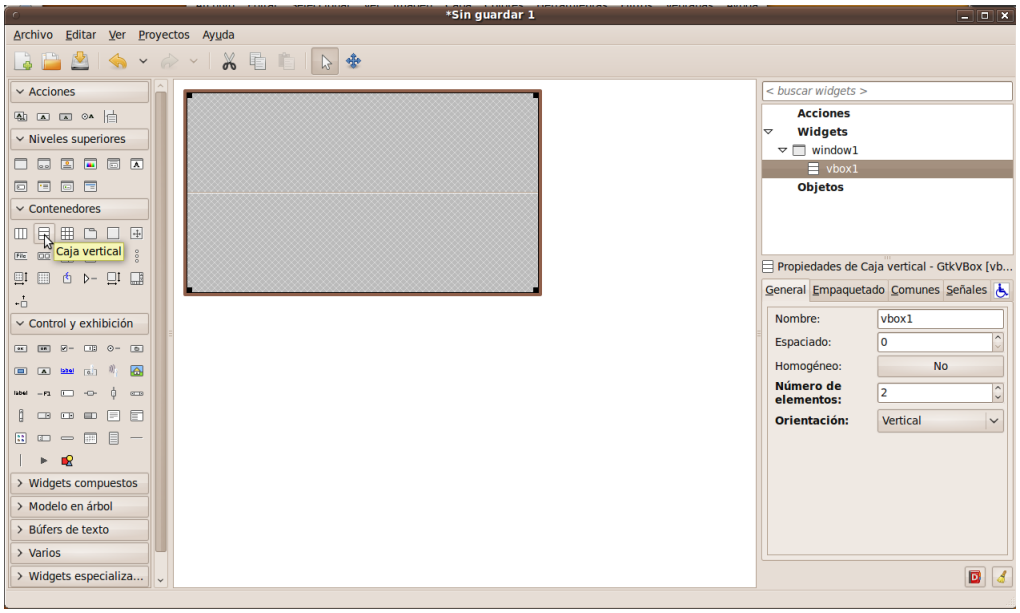


Figura 9.18: Interfaz de Glade

```
class VentanaEjemplo:
    def __init__(self):
        builder = gtk.Builder()
        builder.add_from_file('ejemplo.glade')

        self.window = builder.get_object('window')
        self.contador = builder.get_object('lbl_contador')

        builder.connect_signals(self)

    def on_btn_sumar_clicked(self, widget, data=None):
        num = int(self.contador.get_text())
        self.contador.set_text(str(num + 1))

    def on_btn_restar_clicked(self, widget, data=None):
        num = int(self.contador.get_text())
        self.contador.set_text(str(num - 1))

if __name__ == "__main__":
    ventana = VentanaEjemplo()
    ventana.window.show()
    gtk.main()
```

Vayamos por partes. Primero, necesitamos las librerías `gtk` y `gtk.glade`, por tanto las tenemos que importar.

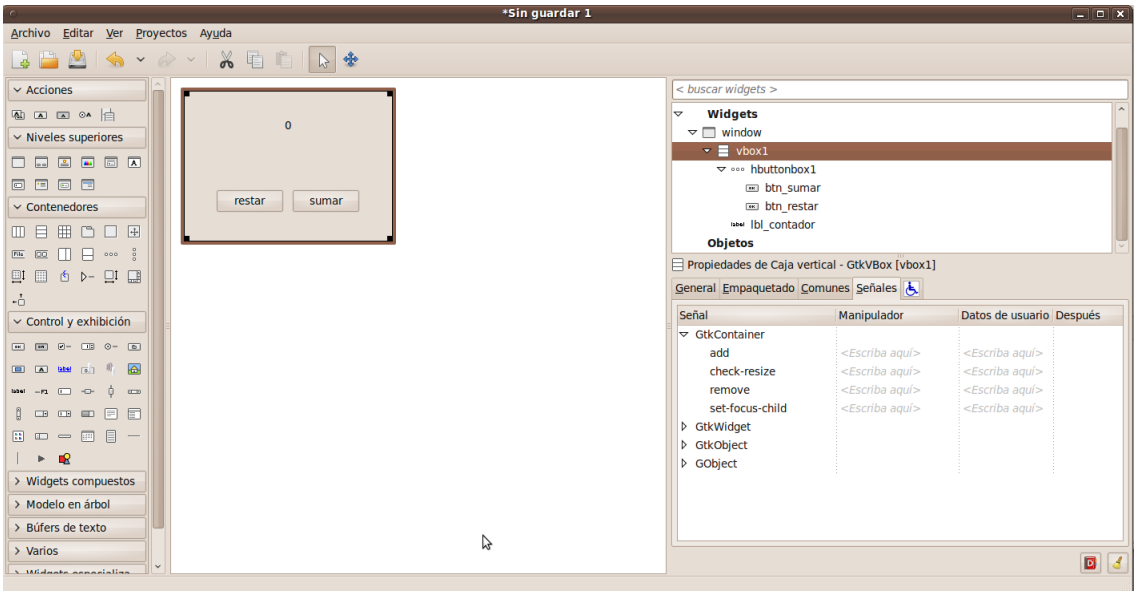


Figura 9.19: Interfaz de Glade

Cada elemento de PyGTK es recomendable que lo tratemos como una clase. Esto nos facilitará su modularidad y capacidad de abstracción. Veamos el constructor:

```
def __init__(self):
    builder = gtk.Builder()
    builder.add_from_file('ejemplo.glade')

    self.window = builder.get_object('window')
    self.contador = builder.get_object('lbl_contador')

    builder.connect_signals(self)
```

Lo primero que hacemos es crear un objeto de tipo `Builder`. A ese objeto le añadimos el fichero .glade que hemos creado antes. Con la función `builder.get_object()` obtenemos el widget con el nombre que le hayamos pasado. Esto quiere decir que si le pasamos `'lbl_contador'`, la variable `self.contador` tendrá ese widget. Por último conectamos todas las señales, con el fin de poder definir las funciones asociadas a dichas señales. Veamos ahora dichas funciones:

```
def on_btn_sumar_clicked(self, widget, data=None):
    num = int(self.contador.get_text())
    self.contador.set_text(str(num + 1))

def on_btn_restar_clicked(self, widget, data=None):
    num = int(self.contador.get_text())
    self.contador.set_text(str(num - 1))
```

Estos son las dos señales que asignamos antes en Glade. Cada vez que esos eventos se activen, se llamarán a estas funciones. Vemos que su comportamiento es simple: extraen el texto que tiene la etiqueta y

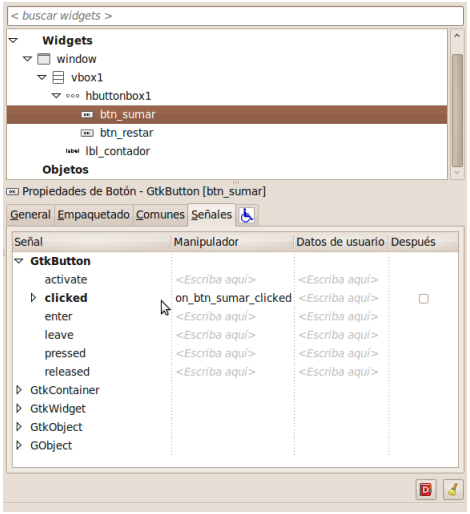


Figura 9.20: Señales

lo incrementan o decrementan según sea el caso.

Por último:

```
if __name__ == "__main__":
    ventana = VentanaEjemplo()
    ventana.window.show()
    gtk.main()
```

Creamos el objeto de la nueva clase que hemos definido, mostramos la ventana, y lanzamos el entorno gtk para su correcta ejecución. Podemos comprobar el resultado en la imagen 9.21.

Figura 9.21: Resultado final

Actividades de difusión

Concurso Universitario de Software Libre

La aplicación resultante del desarrollo de este Proyecto Fin de Carrera se ha presentado a la **4ª edición del Concurso Universitario de Software Libre**, en la categoría de **Educación y ocio**. De forma automática también es participante a la edición local del CUSL, cuya edición está organizada por la **Oficina de Software Libre de la Universidad de Cádiz**.

Las ediciones de estos concursos se cierran en Mayo y Marzo respectivamente, así que es un buen motivo para seguir desarrollando y mejorando esta aplicación.

Creación de comunidad

Con el fin de crear una comunidad alrededor de esta aplicación, se han tomado varias lineas de acción generales:

- Alojar el proyecto en **forjas** de proyectos libres. Así será más accesible para otras personas en un determinado momento.
- Creación de un **blog**[2] en el que se ha intentado llevar un diario de desarrollo.
- Una cuenta de **Twitter**[3] donde se hace un seguimiento más regular del avance del proyecto, además de intentar crear una comunidad más cercana.
- Construcción de una página web[4] de resumen a los distintos enlaces disponibles en relación al proyecto. Podemos ver un enlace en la imagen 9.22

[2] http://sumergiendose.wordpress.com
[3] http://twitter.com/resistencia1812
[4] http://cusl4-res-cadiz.forja.rediris.es/

Figura 9.22: Web de Resistencia en Cádiz: 1812

Bibliografía

[1] GNOME development team. GNOME Human Interface Guidelines 2.2. http://library.gnome.org/devel/hig-book/stable/.

[2] Pygame development team. Pygame Documentation. http://www.pygame.org/docs/.

[3] PyGTK development team. PyGTK 2.0 Reference Manual. http://library.gnome.org/devel/pygtk/stable/.

[4] Manuel Palomo Duarte. La competitividad como un factor motivante para el aprendizaje de sistemas experto. *Actas de las II Jornadas Nacionales de Metodologías ECTS*, 2007.

[5] Edward A. Feigenbaum. Knowledge engineering in the 1980s. *Dept. of Computer Science, Stanford University*, 1982.

[6] Python Software Foundation. Python documentation. http://docs.python.org/.

[7] Francesco Garosi. PyCLIPS Manual. http://pyclips.sourceforge.net/manual/.

[8] Andrew M. St. Laurent. *Understanding Open Source & Free Software Licensing*. O'Reilly, 2004. ISBN 978-0-596-00581-8.

[9] Mark Lutz. *Programming Python*. O'Reilly, 2006. ISBN 978-0-596-00925-0.

[10] Mark Lutz. *Learning Python*. O'Reilly, 2009. ISBN 978-0-596-51398-6.

[11] Joaquín Ataz López. Creación de ficheros LaTeX con GNU Emacs. ftp://ftp.dante.de/tex-archive/info/spanish/guia-atx/guia-atx.pdf.

[12] Mark Pilgrim. *Dive into Python*. Apress, 2004. ISBN 1-59059-356-.

[13] Eric S. Raymond. *The cathedral and the bazaar*. O'Reilly, 2001. ISBN 978-0-596-00108-7.

[14] Richard M. Stallman. *Software libre para una sociedad libre*. Traficantes de sueños: Mapas, 2004. ISBN 978-84-933555-1-7.

[15] Guido van Rossum and Barry Warsaw. Style Guide for Python Code. http://www.python.org/dev/peps/pep-0008/.

[16] Joseph Giarratano y Gary Riley. *Sistemas expertos. Principios y programación*. International Thomson, 1998. ISBN 970-686-059-2.

[17] Frank Mittelbach y Michel Goossens. *The LaTeX Companion*. Addison-Wesley, 2004. ISBN 0-201-36299-6.

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.
This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.
We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.
A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.
A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall

subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.
The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.
The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.
A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".
Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.
The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.
The "**publisher**" means any person or entity that distributes copies of the Document to the public.
A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.
The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying

of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.
You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.
If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.
If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.
It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.
You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.
You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.
The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.
The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.
In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.
You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.
If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers.

In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.
If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.
However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.
Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.
Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.
Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.
“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.
“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.
The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.
If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

GNU General Public License

PREAMBLE

The GNU General Public License is a free, copyleft license for software and other kinds of works.
The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.
When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.
To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.
For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.
Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.
For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.
Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.
The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component

(kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright

notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

   b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

   c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

   d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

   a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

   b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the

public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all.

For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

    Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE

PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy`