Manuales de Resistencia en Cádiz: 1812

Pablo Recio Quijano

9 de marzo de 2010

Índice

1. 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/?group_id=680&release_id=1133`. Como siempre, es recomendable bajarse la última versión (a fecha de publicación de este manual la 1.1)

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/

2. 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* 3.

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 1.
Al iniciar nos encontramos con la pantalla principal del entorno siendo ésta el **menú principal** 2 que recoge en ella todas las funcionalidades del sistema.

```
~/.resistencia1812
--| games/
--| teams/
```

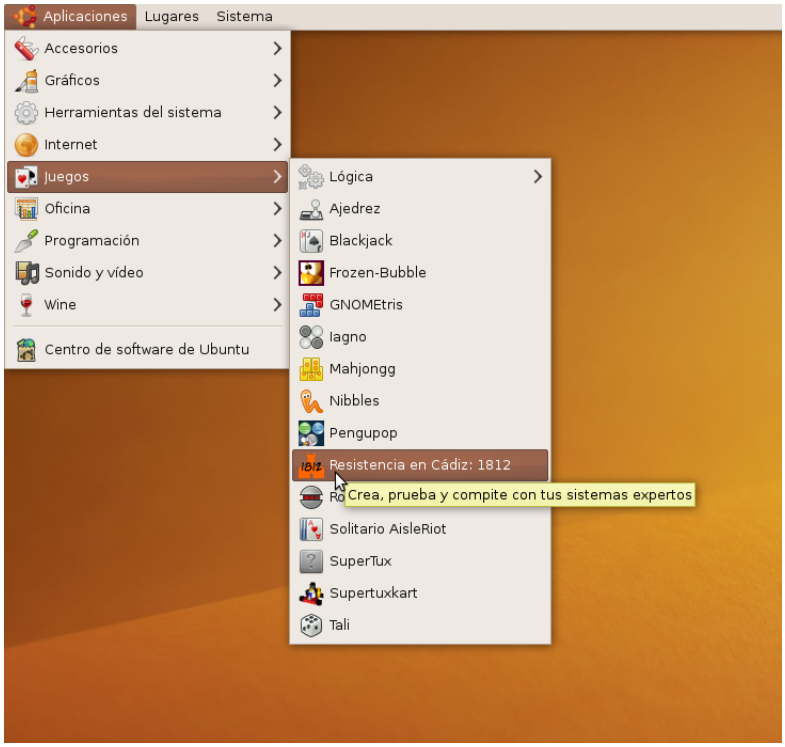

Figura 1: Ruta a seguir para iniciar Resistencia en Cádiz: 1812

```
----| 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.
- **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 3. Con esta funcionalidad podemos ejecutar una partida entre dos sistemas expertos o equipos.

Figura 2: Menú principal de Resistencia en Cádiz: 1812

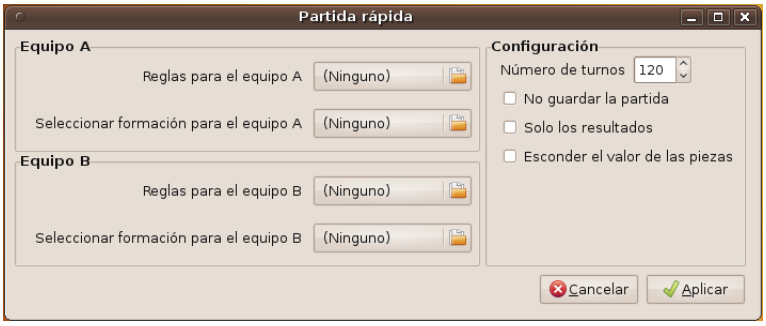

Figura 3: Diálogo de configuración para una partida rápida

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:

- *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 4.
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 5.

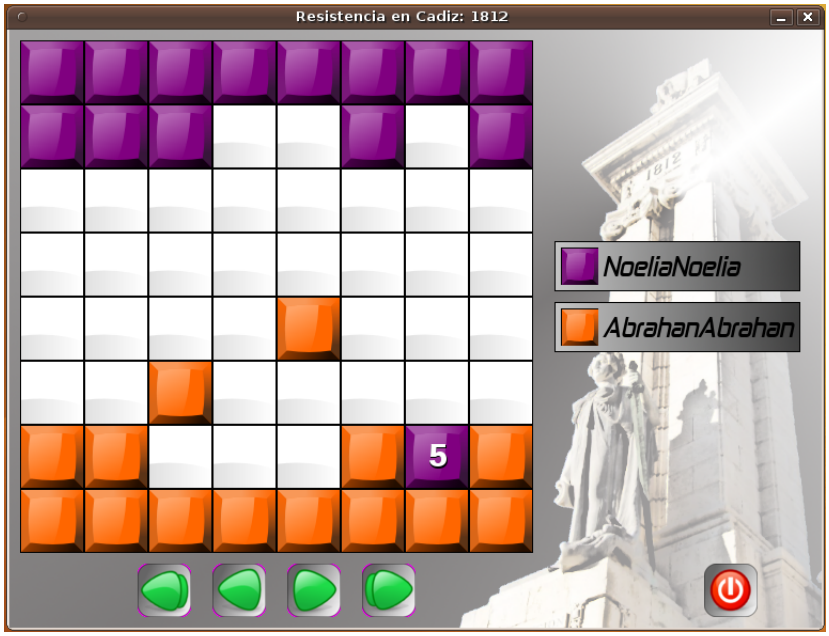

Figura 4: Partida rápida sin valor en las piezas

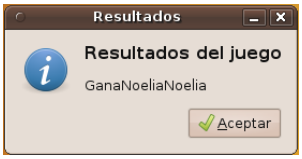

Figura 5: Finalización y muestra del resultado de la partida rápida

Competiciones

En esta opción podemos elegir entre varios tipos de competiciones, como se nos muestra en la imagen 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.
Si cambiamos de pestaña, podremos ver esta imagen 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.

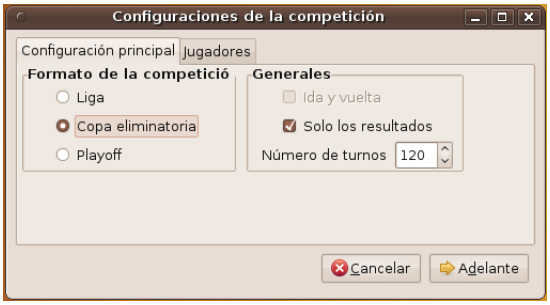

Figura 6: Primera parte del diálogo de configuración para competiciones.

Figura 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 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:

Figura 8: Resultados parciales de un playoff en el que participan todos los jugadores.

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

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.

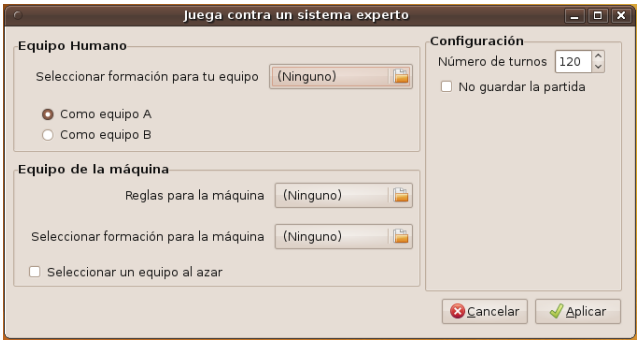

Figura 9: Diálogo de configuración para jugar contra el sistema experto.

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 10.

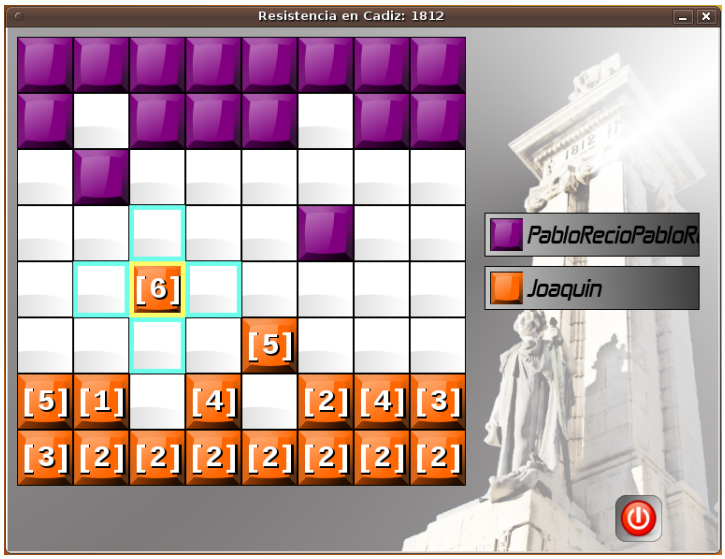

Figura 10: Tablero que muestra la interacción del usuario con la partida.

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.

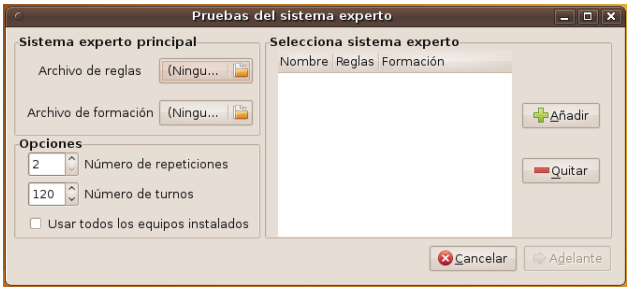

Figura 11: Diálogo de configuración para las pruebas de un sistema experto concreto.

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 estadística resultante sea más fiable. El ejemplo de este diálogo está visible en la imagen 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 12 agrupadas en un gráfico y en una especie de tabla.

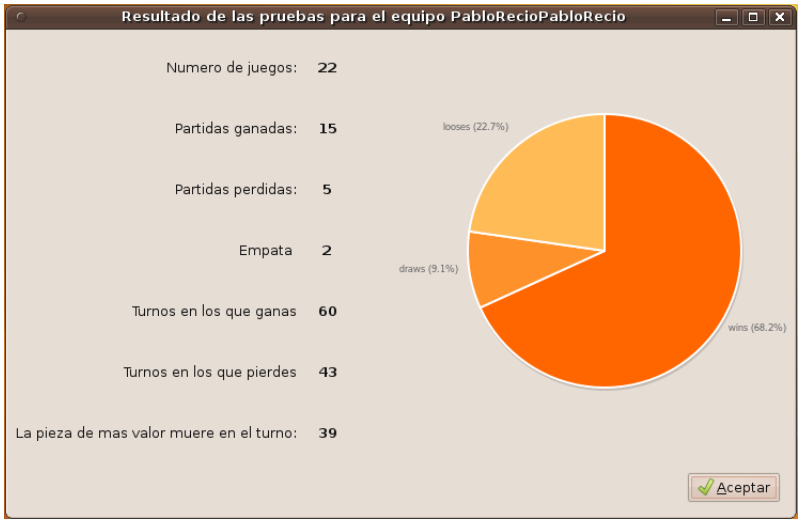

Figura 12: Estadísticas que ofrece la funcionalidad después de realizar las pruebas.

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 componen la partida

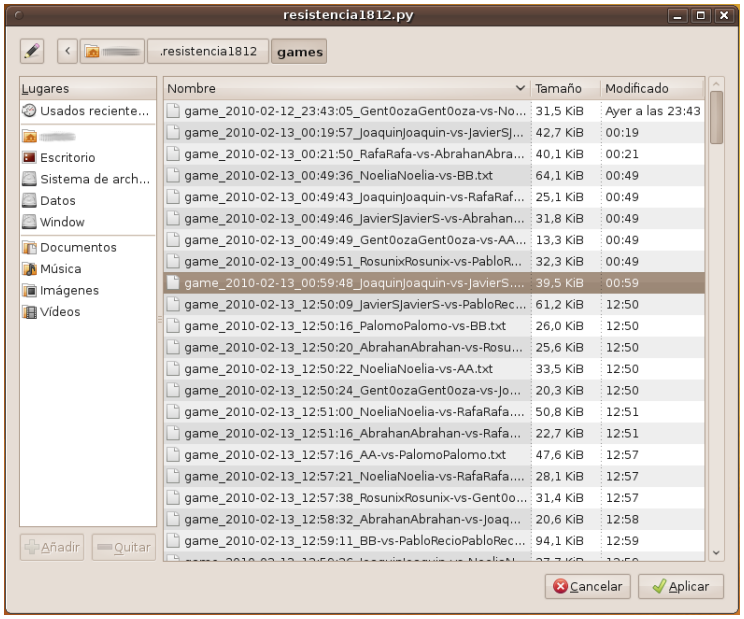

Figura 13: Elección del fichero de la partida a visualizar.

En esta opción podemos elegir cualquier fichero de partidas con el selector de fichero predeterminado de GNOME 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 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.

3. 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.

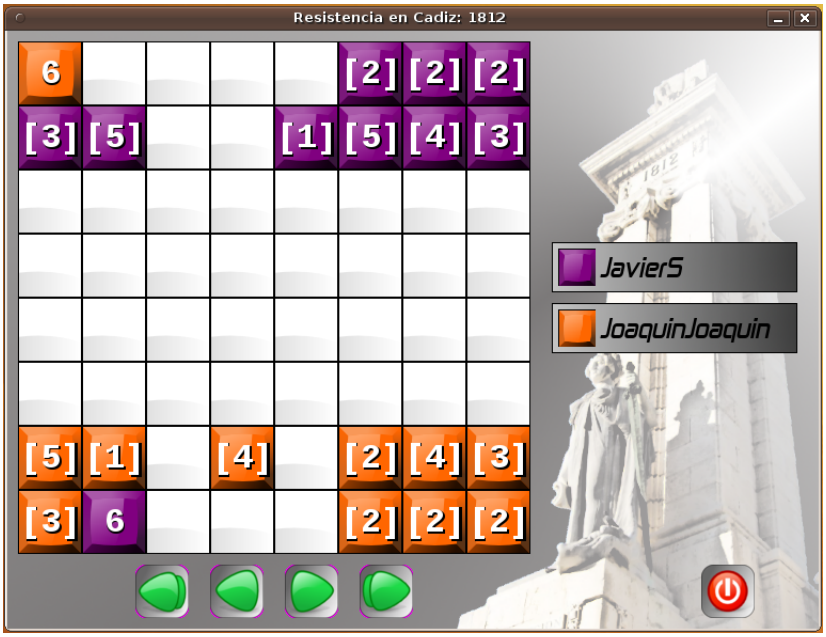

Figura 14: Visualización de una partida antigua.

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 15.

Figura 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)))
)
```