
UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

DSIC
DEPARTAMENTO DE SISTEMAS
INFORMÁTICOS Y COMPUTACIÓN

# VIDEOJUEGO CON REDES DE PETRI INTERACTIVAS

AUTOR: ALEJANDRO PÉREZ SEGURA

DIRECTOR: FRANCISCO JAVIER OLIVER VILLARROYA

## CONTENIDO

## AGRADECIMIENTOS

Me gustaría dar las gracias a todas las personas que me han apoyado en el desarrollo de esta tesis de máster, en especial a mi director y supervisor de la Universidad Politécnica de Valencia, Francisco Javier Oliver Villarroya, que me ha enseñado innumerables cosas, por su dirección, por su paciencia y por sus profundas revisiones sin las cuales la redacción de este documento no habría sido posible.

Me gustaría expresar una especial gratitud a todas las personas que se han involucrado en el desarrollo del proyecto: a Paula, Jorge, Pablo, Mauro, Juan, Jordi y Carlos por su colaboración en el testeo de la aplicación y sus valiosos comentarios; y a Christian, por sus excelentes gráficos.

Gracias a todos mis compañeros del trabajo, amigos y familiares que siempre han estado ahí para apoyarme en el desarrollo del proyecto.

Por último me gustaría agradecer su ayuda a mi novia Patricia, por su apoyo incondicional en todos los aspectos de esta tesis y en cada cosa que hago.

Gracias.

## RESUMEN

Se pretende diseñar y crear una herramienta con la que el público general pueda aprender fácilmente los conceptos básicos de las Redes de Petri (RdP), a la vez que se muestre la utilidad práctica de su utilización en entornos reales.

Se pretende facilitar el aprendizaje haciendo la experiencia de utilizar la herramienta entretenida, a la vez que se fomenta la creatividad e imaginación de los usuarios. Para ello crearemos un juego, o entorno interactivo, en el que podremos experimentar con las RdP, y en el que será necesario ir comprendiendo y asimilando bien los conceptos y la mecánica de las RdP para seguir avanzando.

El proyecto incluye implantar el juego en un entorno web real, de manera que cualquier persona con conexión a internet interesada en las RdP pueda aprender sobre ellas visitando una página web y jugar y aprender de forma gratuita.

# 1. INTRODUCCIÓN

Los Sistemas Dinámicos de Eventos Discretos (DEDS) se basan en los sistemas dinámicos (sistemas que evolucionan con el tiempo), donde los estados y eventos discretos tienen un rol fundamental. Los DEDS han crecido en interés, en la medida que la importancia de la automatización y control lo hacen en la tecnología moderna. Los ámbitos de aplicación como sistemas de fabricación flexibles, sistemas de transporte, sistemas distribuidos, y redes de telecomunicaciones usan intensivamente los conceptos y técnicas de los DEDS.

Los DEDS típicos exhiben evoluciones paralelas, las cuales conducen a conductas complejas debido a la presencia de fenómenos de sincronización y recursos compartidos. Las Redes de Petri (RdP) son un formalismo matemático apropiado para modelación de DEDS concurrentes. Los modelos de red son frecuentemente observados como especificaciones autodocumentadas, a causa de que su naturaleza gráfica facilita la comunicación entre diseñadores y usuarios. Los fundamentos matemáticos del formalismo permiten corrección lógica y análisis de eficiencia.

Más que un simple formalismo, las RdP son una familia de ellos, desde bajo a alto nivel, cada una de ellos más apropiados para diferentes propósitos. En cualquier caso, con ellos se pueden representar conductas muy complejas desde la simplicidad del modelo real, constituyéndose de unos pocos objetos, relaciones y reglas.

Las RdP son un modelo formal (matemático) para describir estados y acciones. Disponen de una representación grafica, como el caso de un grafo dirigido, en el que intervienen dos clases de nodos: lugares (representados por circunferencias) y transiciones (representadas por segmentos rectilíneos o rectángulos negros) y arcos que los unen.

Las características fundamentales de las RdP son su simplicidad (intervienen pocas y simples entidades matemáticas), su generalidad (capacidad de modelar secuencias, decisiones, concurrencia, sincronizaciones, etc.), su adecuación (capacidad de expresar diferentes estructuras de concurrencia) y su flexibilidad (permite un modelado progresivo, refinamientos sucesivos y una estructura modular).

Son valores adicionales de las RdP su representación gráfica, la existencia de técnicas de validación (como la vivacidad, limitaciones, bloqueos, etc.), la interpretación estocástica (para la evaluación de rendimientos) y la existencia de herramientas de software de diseño y análisis, que implementan las técnicas conocidas.

# 2. ASPECTOS TEÓRICOS

En este capítulo introducimos todas las definiciones, conceptos y métodos de análisis relacionados con este proyecto. Los temas de este capítulo han sido elegidos para presentar de manera formal todas las definiciones y propiedades que serán usadas después, sin intentar ser completos a la literatura de las Redes de Petri.

## 2.1. REDES DE PETRI

Una Red de Petri (RdP) es una representación matemática o gráfica de un sistema de eventos discretos en el cual se puede describir la topología de un sistema distribuido, paralelo o concurrente.

Son una generalización de la teoría de autómatas que permite expresar un sistema de evolución en paralelo o eventos concurrentes compuesto de varios procesos que cooperan para la realización de un objetivo común.

Una red de Petri está formada por lugares, transiciones, arcos dirigidos y marcas que ocupan posiciones dentro de los lugares. La presencia de marcas en los lugares se interpreta habitualmente como presencia de recursos.

Las reglas son:

- Los arcos conectan un lugar a una transición así como una transición a un lugar, con un peso asociado (comúnmente uno).

- No puede haber arcos entre lugares ni entre transiciones.

- Los lugares contienen un número finito o infinito contable de marcas.

- Una transición está habilitada si tiene marcas en todos sus lugares de entrada.

- Las transiciones habilitadas se disparan, consumiendo marcas de sus lugares de entrada iguales al peso de los arcos que los unen con la transición y producen marcas en sus lugares de salida.

En la representación gráfica formal se representan los lugares como circunferencias; las transiciones como segmentos rectilíneos; las marcas como pequeños círculos negros en el interior de los lugares y los arcos como flechas que unen lugares y transiciones.

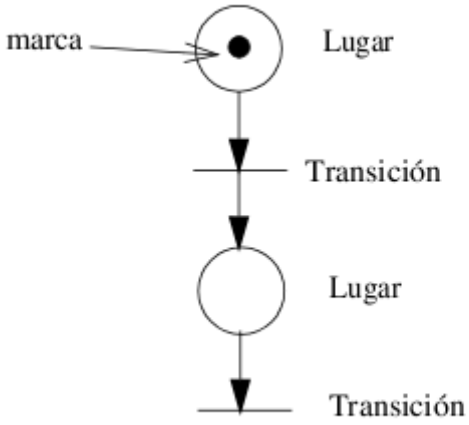



Ilustración 1: Partes de una red de Petri

### 2.1.1. DEFINICIÓN FORMAL Y TERMINOLOGÍA BÁSICA

Las Redes de Petri (RdP) son sistemas con estados y transiciones que extienden las redes elementales.

Definición 1. Una red es una tripleta $N = (P, T, F)$ donde:

$P$ y $T$ son conjuntos finitos disjuntos de lugares y transiciones, respectivamente.

$F \subset (P \times T) \cup (T \times P)$ es un conjunto de arcos.

Definición 2. Dada una red N = (P, T, F), una configuración es un conjunto C tal que C $\subseteq$ P.

Definición 3. Una red elemental es una red de la forma EN = (N, C) donde:

N = (P, T, F) es una red.

C es tal que C $\subseteq$ P es una configuración.

Definición 4. Una red de Petri es una red de la forma PN = (N, M, W), la cual extiende la red elemental de tal manera que:

N = (P, T, F) es una red.

M : P → Z es un multiconjunto de lugares, en el que Z es un conjunto contable. M extiende el concepto de configuración y es comúnmente descrita en referencia a los diagramas de RdP como "marcado".

W : F → Z es un multiconjunto de arcos, de modo que el recuento de cada arco es una medida de multiplicidad de arco.

Si una RdP es equivalente a una red elemental, entonces Z puede ser el conjunto contable {0,1} y esos elementos en P que tienen asignado 1 a M forman una configuración. Del mismo modo, si una RdP no es una red elemental, entonces el multiconjunto M puede ser representado como un conjunto no único de configuraciones. En ese sentido, M extiende el concepto de configuración en la redes elementales para las redes de Petri.

La siguiente definición formal se inspira en la de Peterson, en 1981, aunque existen muchas definiciones alternativas.

### 2.1.2. SINTAXIS

Un grafo de RdP es una 3-tupla $(S, T, W)$, donde:

- S es un conjunto finito de lugares
- T es un conjunto finito de transiciones
- S y T son disjuntos, es decir, que ningún elemento puede ser a la vez lugar y transición.
- $W : (S \times T) \cup (T \times S) \to \mathbb{N}$ es un multiconjunto de arcos, a cada arco se le asigna un entero no negativo como "multiplicidad"; nótese que ningún arco conecta dos lugares o dos transiciones.

La relación de flujo es un conjunto de arcos: $F = \{(x, y) \mid W(x, y) > 0\}$. En muchos libros de texto, los arcos sólo pueden tener multiplicidad 1. En estos libros a menudo se definen las RdP utilizando F en lugar de W. Cuando se usa esta convención, un grafo de RdP es un multigrafo bipartido $(S \cup T, F)$ con partición de nodos S y T.

El pre-conjunto de una transición t es el conjunto de los lugares de entrada: ${}^\bullet t = \{s \in S \mid W(s, t) > 0\}$; su post-conjunto es el conjunto de lugares de salida: $t^\bullet = \{s \in S \mid W(t, s) > 0\}$.

Las definiciones de pre- y post-conjuntos de los lugares son análogos.

Un **marcado** de una RdP (grafo) es un multiconjunto de sus lugares $M : S \to \mathbb{N}$. Decimos que un marcado asigna a cada lugar un número de marcas.

Una RdP es una 4-tupla $(S, T, W, M_0)$, donde:

- $(S, T, W)$ es un grafo de RdP;
- $M_0$ es el marcado inicial, un marcado del grafo de RdP.

### 2.1.3. SEMÁNTICA DE DISPARO

El comportamiento de una RdP está definido con relación a su marcado de la siguiente manera.

Nótese que los marcados pueden ser añadidos como cualquier multiconjunto: $M + M' \stackrel{D}{=} \{s \to M(s) + M'(s) \mid s \in S\}$

La ejecución de un grafo de RdP $G = (S, T, W)$ puede ser definido como la relación de transición $\to_G$ entre sus marcados, como sigue:

- para cualquier t en T:

$$M \to_{G,t} M' \stackrel{D}{\Leftrightarrow} \exists M'' : S \to \mathbb{N} : M = M'' + \sum_{s \in S} W(s,t) \wedge M' = M'' + \sum_{s \in S} W(t,s)$$

- $M \to_G M' \stackrel{D}{\Leftrightarrow} \exists t \in T : M \to_{G,t} M'$

En palabras:

- disparando una transición t en un marcado M consume $W(s,t)$ marcas de cada uno de sus lugares de entrada s, y produce $W(t,s)$ marcas en cada uno de sus lugares de salida s
- una transición está **habilitada** para dispararse (puede dispararse) en M si hay suficientes marcas en los lugares de entrada para que se puedan consumir todas las marcas requeridas: iff . $\forall s : M(s) \geq W(s,t)$

Normalmente, estamos interesados en lo que puede suceder cuando las transiciones pueden dispararse continuamente en orden aleatorio.

Decimos que un marcado $M'$ es alcanzable en un sólo paso si $M \to_G M'$; decimos que es alcanzable desde M si $M \to_G{}^* M'$, donde $\to_G{}^*$ es la clausura transitiva reflexiva de $\to_G$; es decir, si es alcanzable en 0 o más pasos.

Para una RdP (marcada) $N = (S, T, W, M_0)$, nos interesan los disparos que pueden realizarse empezando con el marcado inicial $M_0$. Su conjunto de marcados alcanzables es el conjunto $R(N) \stackrel{D}{=} \{M' \mid M_0 \to_{(S,T,W)}{}^* M'\}$

El grafo de alcanzabilidad de N es la relación de transición $\to_G$ restringida a sus marcados alcanzables $R(N)$. Es el espacio de estados de la red.

Una secuencia de disparos para una RdP con grafo G, marcado inicial $M_0$ es una secuencia de transiciones $\vec{\sigma} = \langle t_{i_1} \dots t_{i_n} \rangle$ tal que: $M_0 \to_{G,t_{i_1}} M_1 \wedge \dots \wedge M_{n-1} \to_{G,t_{i_n}} M_n$.

El conjunto de secuencias de disparo se denomina $L(N)$.

## 2.2. SITUACIONES TÍPICAS EN LA EVOLUCIÓN DE UN SISTEMA DE REDES DE PETRI

Las Redes de Petri (RdP) se ajustan perfectamente para la descripción de conflictos y concurrencia. De hecho, un conflicto implica una elección que puede ser naturalmente descrita gráficamente con una RdP como en la Ilustración 2. Cuando la transición *t* se dispare, una marca será depositada en el lugar *p*, y ambas $t_1$ y $t_2$ se habilitarán al mismo tiempo. Sin embargo, tan pronto como alguna de ella se dispare, se consumirá la marca de *p*, deshabilitando e imposibilitando el disparo de la otra.

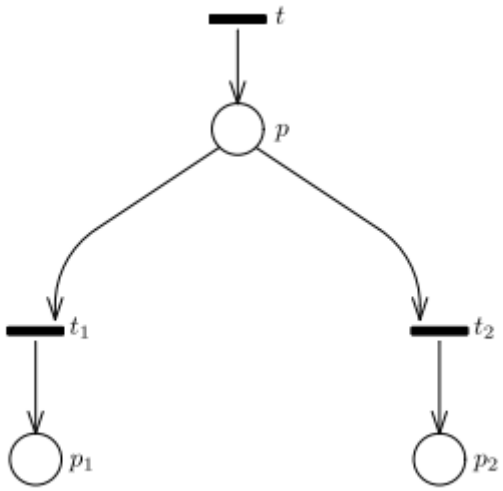



**Ilustración 2: Conflicto en el modelado de una Red de Petri**

Considerando ahora el caso en el que hubieran 2 marcas en *p.* Si $t_1$ se dispara primero entonces $t_2$ todavía estaría habilitada; no obstante algo ha cambiado ya que ahora $t_2$ sólo podría dispararse una vez sin que *t* se dispare. Queremos que nuestra definición de conflicto incluya todas estas situaciones, y por consiguiente distinguir cuántas veces está una transición habilitada en un marcado dado.

Para cualquier RdP $(S, T, W, M_0)$ el grado en el que está habilitada una transición es una función GH : T × [S → N] → N tal que ∀t ∈ T, ∀M : P → N,

ED(t,M) =k iff

• ∀p∈ •t, M(p)≥k·I(p,t), y

• ∃p∈ •t: M(p)< (k+1)·I(p,t)

Así que indicamos con GH(t,M) el número de veces que la transición *t* está habilitada en el marcado *M.* Indicamos con E(M) el multiconjunto de transiciones habilitadas en

el marcado M. Denotamos por t ∈ E(M) la condición "t habilitada en M", y con T' ⊆ E(M), dónde T' ⊆ T, un conjunto de transiciones habilitadas en M.

En general, un **conflicto** es cualquier situación en la que el disparo de una transición reduces el grado en el que está habilitada de otra transición (nótese que se excluyen los conflictos de transiciones con ellas mismas). Si el grado en el que está habilitada una transición se reduce a 0, entonces el disparo de una transición ha deshabilitado otra transición que estaba habilitada antes del disparo.

Para una RdP, ∀tl,tm ∈ T

Tal que tl = tm, ∀M : P → IN, transición tl está en **conflicto efectivo** con tm en el marcado M (denotado como tl EC(M) tm) sii

M[tl>M' y ED(tm,M) < ED(tm,M')

De manera contraria a la situación de conflicto, tenemos la **concurrencia**, que es caracterizada por el paralelismo de actividades; así pues dos transiciones se dice que son concurrentes cuando están habilitadas en un marcado M y no están en conflicto.

Para un RdP, las transiciones t1 y t2 son concurrentes en el marcado M sii:

t1,t2 ∈ E(M) ⇒ no (t1 EC(M)t2) y no (t2 EC(M)t1)

Una situación interesante aparece si se mezclan la concurrencia y el conflicto, invalidando así la ingenua idea de que la concurrencia es sinónimo de independencia.

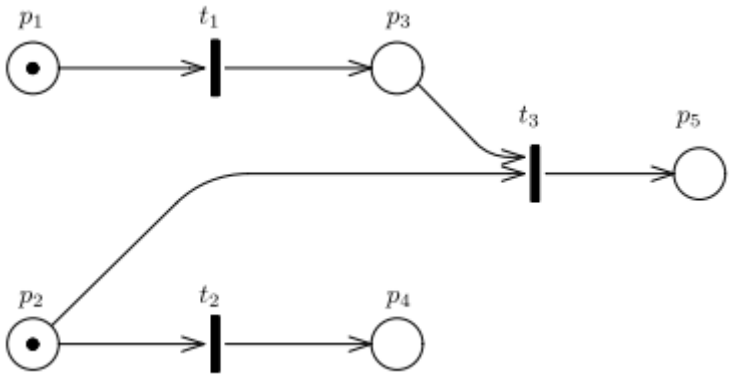



Ilustración 3: Confusión

En la ilustración 3, considerando el marcado en el que sólo tienen marcas p1 y p2, t1 y t2 son concurrentes. Sin embargo si t1 se dispara primero, entonces t2 entraría en conflicto con t3; mientras que si se dispara primero t2, no se genera ningún conflicto. Esta situación en la que el orden de disparo de transiciones que no están en conflicto crea conflictos y determina el futuro de la evolución de la RdP es conocida como

**confusión,** y puede ocasionar bastantes problemas en sistemas de modelado cuyos conflictos no se resuelvan de forma indeterminista, sobretodo en RdP estocásticas.

Otra posible relación entre dos transiciones es la causalidad**.** Se dice que dos transiciones tl y tm están en **relación causal** en un marcado M si tl está habilitada en M, tm está habilitada k veces en M (posiblemente 0, lo que significaría que no está habilitada en M), y el disparo de tl genera un marcado en el que el número de veces que está habilitada tm es estrictamente mayor a k. Formalmente:

tl y tm están en conexión causal, expresada como tlCC(M)tm, sii

M[tl > M' $\Rightarrow$ ED(tm,M')>ED(tm,M)

Cuando los modelos de RdP se utilizan para validar y verificar sistemas, es importante saber cuando dos transiciones nunca pueden estar habilitadas en el mismo marcado, o lo que es lo mismo, son **mutuamente excluyentes**. En fórmula:

tl y tm son mutuamente exclusivas sii

$$\nexists M \in RS(M_0) : t_l \in E(M) \text{ and } t_m \in E(M)$$

Nótese que mientras el conflicto, la concurrencia, la confusión y la causalidad son propiedades de un solo marcado, la exclusión mutua es una propiedad de dos transiciones en todo un sistema, lo que requiere, en principio, comprobar todos los marcados alcanzables.

## 2.3. PROPIEDADES DE LAS REDES DE PETRI

Hay muchas propiedades importantes de los modelos de Redes de Petri(RdP), los sistemas de RdP y las RdP que pueden ser definidas y comprobadas con las herramientas de análisis adecuadas. Demostrar estas propiedades es un paso clave en el análisis de los modelos de RdP, ya que permite hacer declaraciones del modelo acerca de su comportamiento de una manera objetiva.

Vamos a presentar algunas de las propiedades más útiles, y comentaremos su utilidad e importancia respecto al proceso de modelado y análisis, así como discutir qué métodos de análisis podemos usar para probar estas propiedades.

### 2.3.1. ALCANZABILIDAD Y REVERSIBILIDAD

Como ya hemos definido en el punto 2.1.3, un marcado M' es alcanzable desde M si existe una secuencia de disparos σM tal que M[σM > M'. La alcanzabilidad puede ser usada para responder a preguntas sobre la posibilidad del sistema modelado de llegar a un marcado M.

Una propiedad importante de la alcanzabilidad es la reversibilidad. Un sistema de RdP se define como reversible sii desde cualquier marcado alcanzable desde el marcado inicial, es posible volver al marcado inicial. En términos formales un sistema de RdP con marcado inicial $M_0$ es reversible sii $\forall M \in RS(M_0), M_0 \in RS(M)$.

La reversibilidad expresa la posibilidad de un sistema de RdP para volver de forma infinita a su marcado inicial.

### 2.3.2. AUSENCIA DE PUNTO MUERTO

Un sistema de RdP contiene un punto muerto (en ocasiones descrito también como **bloqueo mutuo**) si puede alcanzar un estado en el que ninguna transición pueda ser disparada. Un modelo de RdP está libre de punto muerto si todos los sistema de RdP obtenidos parametrizando el modelo con distintos marcados iniciales están libres punto muerto.

No hay una definición de ausencia de punto muerto en las RdP. Como mucho es posible determinar si una RdP puede potencialmente tener un punto muerto, dependiendo de la estructura de su grafo.

### 2.3.3. VIVACIDAD

Una transición t decimos que está viva en un sistema de RdP sii para cada marcado M alcanzable desde el marcado inicial $M_0$, existe un marcado M' alcanzable desde M tal que $t \in E(M')$. Es decir, que pueda volver a estar habilitada. Una transición t está viva en un modelo de RdP si lo está en todos los sistemas obtenidos del modelo instanciándolo con un marcado inicial. Un sistema de RdP está vivo si todas sus transiciones están vivan en él. Un modelo de RdP está vivo si cada uno de los sistemas de RdP obtenidos instanciando el marcado inicial de forma paramétrica, están vivos. Finalmente, una RdP está viva si existe un sistema de RdP vivo obtenido de la red.

La vivacidad de una RdP es una propiedad existencial ya que en la mayoría de los casos es posible construir un sistema de RdP correspondiente que no está vivo: por ejemplo usando un marcado inicial en el que ninguna transición esté habilitada.

Una transición que no está viva, se dice que está muerta. Para cada transición t muerta, es posible encontrar un marcado M tal que ninguno de sus marcados alcanzables habilita t.

Una consecuencia importante de la vivacidad es que, si al menos hay una transición viva, entonces el sistema de RdP no puede tener punto muerto. Sin embargo la ausencia de punto muerto no es condición suficiente para que un sistema de RdP esté vivo: un sistema sin puntos muertos y con algunas transiciones muertas es un sistema parcialmente vivo.

La vivacidad define la posibilidad de una transición de estar habilitada (y dispararse) de forma infinita. Si queremos información más detallada, como si la transición puede estar habilitada un número k de veces de forma infinita, entonces estamos interesados en lo que llamamos k-vivacidad. Una transición t en un sistema de RdP está k-vivo sii para cada marcado alcanzable M es posible alcanzar un marcado M' en el que: $ED(t,M') = k$.

### 2.3.4. ACOTACIÓN

Un lugar p de un sistema de RdP, decimos que está k-acotado sii para cada marcado alcanzable M, el número de marcas en ese lugar es menor o igual a k. Un sistema de RdP se dice que está k-acotado sii todos sus lugares $p \in P$ están k-acotados.

Un modelo de RdP se dice que está k-acotado si cada sistema de RdP obtenido dando un marcado inicial al modelo, está k-acotado.

Los modelos de RdP y los sistema de RdP que están 1-acotados se dice que son **seguros**.

Los modelos y sistemas de RdP que están k-acotados para ciertos k, se dice que están acotados.

Una consecuencia muy importante de la acotación es que implica la finitud del espacio de estados. En particular, si un modelo de RdP con N lugares está k-acotado, el número de estados o marcados posibles no podrá exceder $(k+1)^N$.

La acotación se puede definir al nivel de red (acotación estructural), determinando que una RdP está acotada sii para cada marcado inicial finito $M_0$ el resultado es un sistema de RdP acotado.

Es interesante fijarse en que la acotación, la vivacidad y la reversibilidad son propiedades completamente independientes.

### 2.3.5. EXCLUSIÓN MUTUA

Hay dos propiedades interesantes de la exclusión mutua: una entre lugares y otra entre transiciones. Dos lugares p y q estás excluidos mutuamente en un sistema de RdP si sus marcados no pueden ser positivos en ambos en el mismo marcado, $\forall M \in RS\ M(p) \cdot M(q) = 0$. Dos transiciones en un sistema de RdP están mutuamente excluidas si no pueden estar habilitadas a la vez en cualquier marcado.

Dos lugares (o dos transiciones) son mutuamente exclusivos en un modelo de RdP si están excluidos mutuamente en cada uno de los sistemas obtenidos instanciando los parámetros del modelo (el marcado inicial).

Las propiedades que se han citado son bastante genéricas. Su interpretación depende enormemente del sistema específico que estemos estudiando. Por ejemplo, si una RdP describe el comportamiento de un programa distribuido, es muy importante para asegurar que el programa es correcto, que el sistema de RdP no tenga punto muerto o bloqueos mutuos. O al revés, si el modelo describe un programa que se supone que debe terminar, lo que haría que no sólo fuera necesario que el sistema tenga un punto muerto, sino que además debería ser alcanzable desde todos los marcados, asegurando así que el programa terminará el algún momento.

## 2.4. TÉCNICAS DE ANÁLISIS

Lo modelos y sistemas de Red de Petri(RdP) en los que se ignora el marcado inicial, se convierten en grafos bipartidos que definen el componente estático o estructural de los modelos. Este componente estructural es lo que llamamos una RdP.

Muchos estudios se han dedicado a la investigación de las propiedades de las RdP que pueden ser comprobadas directamente de su estructura. El interés en este tema es debido al hecho de que cualquier propiedad probada estructuralmente es válida para cualquier sistema (o modelo) posible obtenido de una RdP superponiendo un marcado.

Las técnicas que investigan las propiedades estructurales de las RdP se denominan **técnicas estructurales**, que se dividen en dos clases:

- Técnicas algebraicas lineales: que trabajan con la definición en forma de matriz de la RdP, la matriz de incidencia.
- Técnicas de análisis de grafo: que trabajan directamente con el grafo bipartido de la red.

Como hemos mencionamos antes, existen propiedades de las RdP que dependen del marcado inicial, y que sólo pueden ser estudiadas y comprobadas teniendo esta información en cuenta. Las técnicas usabas para comprobar esta propiedades se denominan **técnicas de análisis del espacio de estados** (o de alcanzabilidad). Las técnicas de análisis del espacio de estados son obviamente no paramétricas con respecto al marcado inicial, ya que requieren la instanciación completa. Estas técnicas siempre se refieren a sistemas de RdP, y no a modelos de RdP o a RdP.

El análisis de alcanzabilidad se basa en la construcción del grafo de alcanzabilidad del sistema de RdP, y sólo es factible cuando el grafo de alcanzabilidad y el conjunto de estados alcanzables son finitos. El tamaño del grafo y el conjunto de marcados alcanzables crece de forma exponencial según el número de lugares y el número de marcas en el marcado inicial $M_0$.

Las técnicas de análisis de alcanzabilidad son muy potentes, ya que permiten probar muchas de las propiedades de interés, ya que el conjunto y grafo de alcanzabilidad contienen todas las evoluciones posibles del sistema de RdP. Sin embargo, normalmente la complejidad espacial y temporal para construir el grafo de alcanzabilidad excede cualquier límite aceptable.

## 2.5. CLASES DE REDES DE PETRI

- RdP limitadas: Fuerzan un límite en el número de marcas que puede contener cada lugar (capacidad). Modificando las reglas de disparo de transiciones para que estas sólo se disparen si disparándose ninguno de sus lugares de salida exceda su capacidad.
- RdP normales: RdP limitadas, dónde todos los lugares tienen {capacidad = 1}.
- RdP inhibidoras: añaden conexiones entre lugares y transiciones (arcos inhibidores) que impiden que se dispare la transición cuando en el lugar haya marcas.
- RdP priorizadas: permiten asignar prioridad a las transiciones. Cuando más de una transición está habilitada, se dispara la que tiene mayor prioridad. **Esto puede aplicarse para eliminar el indeterminismo de las RdP.**
- RdP coloreadas: asignan atributos a las marcas, y las transiciones se disparan en base al tipo específico de marcas que hay en sus lugares de entrada.
- RdP temporizadas: asignan una duración al disparo de las transiciones. Cuando se dispara una transiciones, se consumen las marcas de los lugares de entrada y tras transcurrir el tiempo determinado por la duración, se ponen las marcas en los lugares de salida.
- RdP estocásticas: son aquellas con transiciones con tiempo, disparo atómico y en la que los retrasos de transición son variables aleatorias con distribución exponencial negativa.
- RdP estocásticas generalizadas: son RdP estocásticas en las que se permite que las transiciones pertenezcan a dos clases diferentes: inmediatas y temporizadas, es decir, que existan transiciones cuyo retraso sea siempre 0.

## 2.6. REDES DE PETRI CON PRIORIDAD

La disponibilidad de diferentes niveles de prioridad en los modelos de RdP incrementa el poder descriptivo, dando más libertad y flexibilidad al modelar la red, ya que podemos separar las transiciones por clases de diferentes niveles lógicos, por ejemplo: acciones que requieren tiempo contra acciones que corresponden a elecciones lógicas que ocurren de forma instantánea.

En este capítulo extenderemos el formalismo de las Redes de Petri (RdP) clásicas del punto 2.1, para incluir niveles de prioridad a las transiciones y definir las dinámicas de este nuevo formalismo.

### 2.6.1. DEFINICIÓN FORMAL

Una RdP con prioridad es una 5-tupla (S, T, W, $M_0$, P), donde:

- $(S, T, W, M_0)$ define una RdP y
- P : T $\rightarrow \mathbb{N}$ es la función de prioridad que asigna a cada transición un número natural que representa su prioridad.

La definición de prioridad que asumimos es: las transiciones habilitadas con una prioridad dada *k* siempre se disparan antes que otra transición habilitada con prioridad *j* < *k*.

Este tipo de definición de prioridad nos permite separar las transiciones por clases de diferentes niveles lógicos, por ejemplo: acciones que requieren tiempo contra acciones que corresponden a elecciones lógicas que ocurren de forma instantánea; y además nos da la posibilidad de especificar de forma determinista el criterio de resolución en un conflicto.

### 2.6.2. SEMÁNTICA DE DISPARO

Para las reglas de disparo de los modelos de RdP con prioridad requiere las siguientes definiciones:

- una transición $t_j$ se dice que tiene **concesión** en el marcado M si verifica las condiciones para estar habilitada en una RdP sin prioridad
- una transición $t_j$ está habilitada en un marcado M si tiene concesión en ese mismo marcado, y no hay ninguna otra transición $t_k \in T$ con prioridad P(k) > P(j) que tenga concesión en M. Como consecuencia, dos transiciones sólo pueden estar habilitadas simultáneamente en un marcado dado sólo si tienen el mismo nivel de prioridad.

- Una transición $t_j$ puede dispararse sólo si está habilitada. El efecto de disparo de la transición es idéntico al caso de los modelos de RdP sin prioridad.

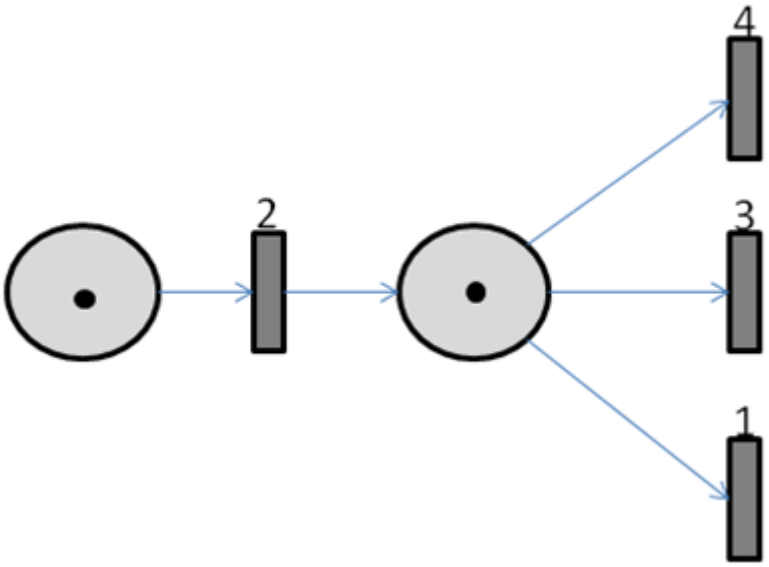



Ilustración 4: Conflicto efectivo indirecto

Debemos tener en cuenta el conflicto efectivo indirecto, ya que si a la hora de disparar transiciones vamos comprobando por orden de prioridad las transiciones para dispararlas, podemos caer en este conflicto, en el que se disparan transiciones menos prioritarias antes que otras con las que estén en conflicto.

Para evitar este conflicto, iniciaremos la comprobación de todas las transiciones por orden descendente de prioridad tras cada disparo, para saber cuál debe dispararse.

Para nuestro principal objetivo, asociaremos las Redes de Petri(RdP) a mecanismos que simulen el comportamiento de máquinas reales, como una máquina expendedora o una cadena de montaje.

Debemos acabar con el indeterminismo asociado a las RdP, pues queremos centrarnos en mecanismos deterministas, como la mayoría de las máquinas reales. Para ello podemos utilizar las RdP con prioridad, que asignan un valor único a cada transición para que en el caso de que hayan varias transiciones habilitadas dado un marcado, sea una y sólo una (la más prioritaria) la que se dispare.

Normalmente haremos referencia a las transiciones habilitadas como aquellas que tengan concesión en el marcado actual, es decir, que estarían habilitadas en una RdP sin prioridad, y que se disparará si sigue teniendo concesión cuando las transiciones con mayor prioridad que ella dejen de tener concesión.

# 3. ENTORNO DE DESARROLLO Y METODOLOGÍA

Para el entorno de desarrollo se ha elegido la tecnología FLASH®, por la facilidad para incorporar herramientas interactivas en páginas webs; y por su expansión, ya que debido a plataformas como Youtube, el reproductor específico de Flash (Flash Player) está ampliamente extendido por todo el mundo.

Adobe Flash (formalmente Macromedia Flash) es una plataforma multimedia usada para añadir animaciones, video e interactividad a las páginas web.

Flash manipula gráficos rasterizados y vectoriales para proveer a las animaciones de texto, dibujos o imágenes fijas. Soporta streaming bidireccional de audio y video, y puede capturar entrada del usuario a través del ratón, el teclado, el micrófono y la cámara. Flash utiliza un lenguaje orientado a objetos llamado ActionScript.

El contenido Flash puede ser visualizado en cantidad de ordenadores y dispositivos usando el Adobe Flash Player, disponible de forma gratuita para navegadores web y teléfonos móviles.

Para nuestro proyecto hemos creado los gráficos con Adobe Flash, y los publicamos o exportamos en forma de bibliotecas accesibles como archivos flash (.swf) para su utilización desde el código.

Utilizamos Flash Develop como IDE (Entorno de Desarrollo Integrado) para la implementación y gestión del código fuente de la aplicación en ActionScript3.

Hemos elegido Webnode como entorno de desarrollo para crear la página web en la que incluiremos el contenido Flash, porque es una herramienta gratuita que permite crear rápidamente una web y alojarla de manera gratuita, sin publicidad ni anuncios; y así poder centrar nuestros esfuerzos en la aplicación, minimizando el coste de ponerla en producción, pero permitiéndonos acceder a la aplicación desde casi cualquier dispositivo conectado a internet y probar la herramienta en un entorno real.

La metodología elegida es la de prototipado incremental, es decir, para cada fase del proyecto, se realizará una mini-cascada (requerimientos, diseño, implementación, verificación, mantenimiento) que culminará en un prototipo que constará de la funcionalidad acumulada de su fase del proyecto junto con las anteriores. Este tipo de metodología nos permitrá adaptarnos rápida y fácilmente a los cambios tanto de requisitos como de diseño, además de ir dándonos una idea del aspecto final de la aplicación.

Se ha elegido un diseño ascendente (bottom-up), es decir, que primero se definirán e implementarán los componentes más pequeños, reutilizables e independientes y en cada iteración posterior se definirán e implementarán componentes más complejos que utilizarán los componentes inferiores. De esta manera se facilitarán las pruebas de los componentes más simples y se podrá obtener el primer prototipo en las fases tempranas del desarrollo.

La documentación de la aplicación se ha incluído en el código en la misma fase de implementación. Esto facilita enormemente su mantenimiento y flexibilidad a la hora de realizar las etapas de desarrollo posteriores.

## 4. DESARROLLO DEL PROYECTO

Para llevar a cabo el desarrollo del proyecto ha sido de vital importancia crear una planificación a largo plazo, así como hacer comprobaciones mensuales sobre el estado real del proyecto y compararlo con la planificación inicial para realizar los ajustes oportunos. La planificación inicial fue algo tan sencillo como esto:

Se documentará todos los meses el desarrollo del proyecto, incluyendo los diseños. La última semana de cada mes se validará el avance de los objetivos y el estado de la memoria.

PLAN DE PROYECTO

septiembre:
- diseñar loader ligero para el programa principal
- diseñar el componente básico (clase lógica con gráfico asociado)
- diseñar librería de assets para trabajar con componentes

octubre:
- configurar Framework para Flash (10.1? 11?) con FlashDevelop
- implementar loader ligero para cargar el programa principal
- diseñar los botones
- diseñar los componentes arrastrables

noviembre:
- implementar el componente básico
- implementar librería de assets para trabajar con componentes
- diseñar puzzles
- diseñar sistema de juego

diciembre:
- crear gráficos de los componentes básicos del juego
- implementar componentes básicos del juego
- implementar botones

enero:
- implementar componentes arrastrables
- crear sistema de conexión entre componentes

febrero:
- crear gráficos del sistema de juego
- implementar sistema de juego

marzo:
- crear puzzles

abril:
- testeo e implantación

mayo:
- terminar documentación/memoria

junio:
- defensa de la tesina

Fue descrita a mediados de septiembre, cuando comenzó el proyecto, y pese a que ha sufrido numerosos cambios y no había tenido en cuenta muchas las tareas que eran necesarias para el desarrollo de la aplicación, fue un buen primer paso para determinar tanto el alcance como el objetivo del proyecto.

Para empezar con la primera fase de diseño, en la que se definirían las bases para poder crear el juego se redactaron los siguientes puntos:

## 4.1. CONCEPTOS PREVIOS

**ASSET**

Llamamos asset a cualquier objeto de texto, imagen o sonido con formato binario que podemos utilizar.

**STAGE**

La Stage (o escena), representa el área principal de dibujo.

Sobre ella se colocarán los gráficos que se deseen visualizar en el Flash® Player.

**RELACIÓN DE PADRE-HIJO**

Cuando un componente gráfico se compone de otros componentes gráficos, decimos que el gráfico compuesto es el padre de todos los gráficos de los que se compone.

El gráfico padre es el que decide la profundidad de sus hijos, decidiendo qué hijo se visualizará encima de qué otro, mediante una lista de profundidad.

Todas las propiedades de los hijos toman como referencia las propiedades del padre, de tal manera que si movemos o escalamos el gráfico padre, se reflejará a la hora de dibujar los hijos, aunque las propiedades de los hijos permanezcan intactas.

**SPRITE**

El Sprite es la unidad básica de construcción de elementos gráficos: es una lista de elementos gráficos que pueden contener a su vez hijos.

Herencia en la jerarquía de clases de Flash:
Sprite → DisplayObjectContainer → InteractiveObject → DisplayObject →

EventDispatcher → Object

Avanzando desde la raíz podemos definirlo como un objeto capaz de enviar eventos, que puede contener gráficos y textos, con el que se puede interactuar, y que puede tener hijos.

## 4.2. DISEÑO DEL SISTEMA DE GRÁFICOS

**Componente Básico**

El Componente Básico (Component) será la unidad básica para cualquier componente gráfico que utilicemos en el proyecto.

Constará de un Sprite, y lo envolverá para ofrecer los métodos y atributos que nos interesen de este(altura, anchura, posición, padre, visible, escena, ...).

Se podrá crear con un Sprite, de forma directa, o con un nombre de clase (className) para instanciar un Sprite de una clase gráfica exportada, utilizando la librería de assets.

**Librería de Assets**

La Librería de Assets se encargará de gestionar todos los assets requeridos para el proyecto.

Gestionará la carga de los swf (Shockwave Flash), pasándole la ruta del archivo.

Permitirá obtener instancias de Sprites exportados pasándole el nombre de exportación de la clase gráfica.

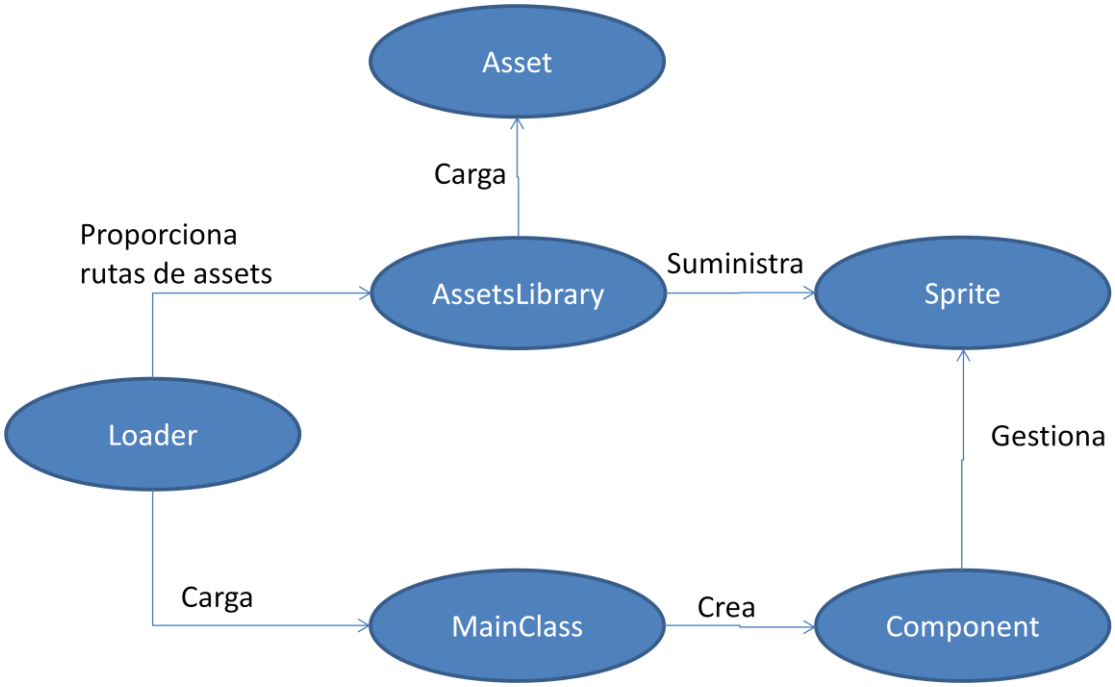



Ilustración 5: Sistema de Gráficos

**Loader ligero**

Con el fin de mostrar el progreso de carga de los assets necesarios para el proyecto, se incluye un loader ligero que muestre esa información.

Constará de un fondo, una barra de progreso y una etiqueta textual dinámica, con la fuente correspondiente embebida para cambiar el texto de forma dinámica en ejecución.

Una vez cargados todos los assets necesarios, retiraremos el loader de la escena y pondremos en su lugar la clase principal del proyecto.

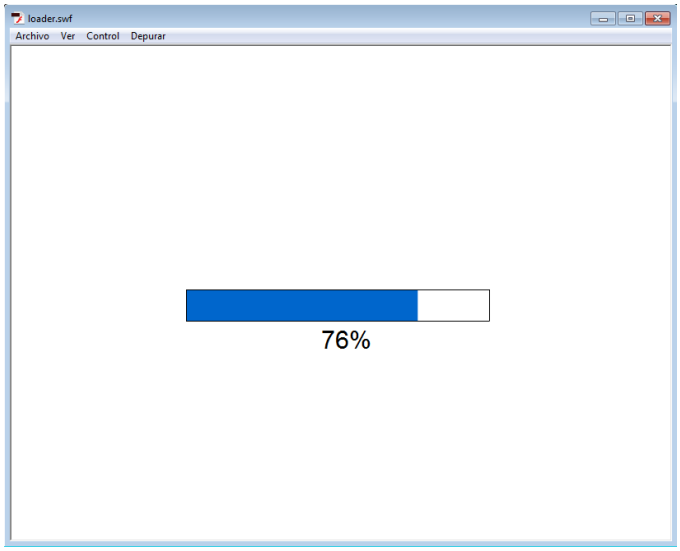



Ilustración 6: Loader ligero

**Botón**

Los botones serán componentes gráficos con diversos estados: up, over, down, disabled.

Estos estados determinan el aspecto gráfico del botón así como la funcionalidad en cada uno de los estados.

UP: el botón habilitado sin interacción.

OVER: el botón habilitado y recibe la interacción del ratón encima (MouseEvent.OVER)

DOWN: el botón habilitado y recibe la interacción del cursor al presionar el botón izquierdo del razón estando encima del botón (MouseEvent.PRESS)

DISABLED: estado del botón deshabilitado, no reaccione frente a interacciones

Disparan el evento MouseEvent.CLICK si reciben un click del ratón.

**Componente Arrastrable**

Los componentes arrastrables serán componentes que pueden ser desplazados por la escena de forma interactiva por el usuario.

Podemos tener dos clases de componentes arrastrables:

- Botones que se desplazarán junto al cursor mientras estén en el estado "down" (con el botón izquierdo del ratón apretado).

- Botones que se desplazarán junto al cursor tras haber recibido un click del ratón, y dejarán de ser arrastrables al volver a recibir otro click.

Junto con la implementación de sus clases y la creación de sus gráficos completan los objetivos hasta mediados de noviembre.

## 4.3. DISEÑO DEL JUEGO: PETRINETGAME

Al juego que pretendemos crear con este proyecto se le ha dado el nombre de PetriNetGame, en el que tendremos como trasfondo reparar o modificar mecanismos. El comportamiento de cada uno de estos mecanismos vendrá determinado por una Red de Petri, se especificará el comportamiento deseado y el jugador deberá hacer uso de las herramientas disponibles para modificar la Red de Petri original con el fin de que cumpla los requisitos especificados.

Estas máquinas funcionarán de forma determinista. Para evitar el indeterminismo en el juego, hemos decidido usar Redes de Petri priorizadas para definir las máquinas, con la condición de que no puede haber dos transiciones con la misma prioridad.

### 4.3.1. NIVELES

El juego se dividirá en diversos niveles, en cada cuál habrá que completar un puzzle para superarlo. En cada nivel irá progresando la dificultad, y se irán añadiendo nuevas herramientas a disposición de los jugadores según vayan avanzando en su conocimiento de las Redes de Petri.

Los niveles puedes estar desbloqueados o bloqueados. Los niveles desbloqueados pueden ser jugados siempre que se desee. Los niveles desbloqueados no pueden ser jugados hasta que sean desbloqueados.

Al empezar el juego el primer nivel estará desbloqueado y los demás niveles bloqueados. Al completar un nivel, el siguiente nivel se desbloqueará.

Para completar el juego se deberán haber completado todos los niveles.

Se puede continuar jugando al juego para aumentar la puntuación en niveles ya superados.

### 4.1.2 PUZZLES

Cada puzzle será un mecanismo representado por elementos físicos externos enlazadas a elementos de una Red de Petri, y las especificaciones que debe cumplir el funcionamiento correcto de la máquina.

El jugador deberá colocar y modificar los elementos necesarios de la Red de Petri para que cumpla las especificaciones dadas.

Para realizar esta comprobación se realizarán pruebas con los elementos externos del puzzle y se comprobará si en cada caso se cumplen las especificaciones oportunas.

### 4.1.3 PUNTUACIÓN

En cada nivel se valorará usar las piezas mínimas para resolverlo.

Se premiará a los jugadores que jueguen bien otorgándoles hasta 3 estrellas tras completar cada nivel:

- 1ª estrella: Puzzle resuelto.

- 2ª estrella: Puzzle resuelto sin muchas piezas.

- 3ª estrella: Puzzle resuelto con pocas piezas.

Al principio se pensó en dar mejores puntuaciones por resolver los puzzles más rápidamente, pero al añadir la rejugabilidad de los puzzles, una vez es conocida la solución del puzzle es muy sencillo recordar los pasos necesarios para completar el puzzle, así que se quitó esta opción y se le dio más importancia a la capacidad de pensar cómo resolver los puzzles y cómo hacerlo con una Red de Petri más sencilla (con menos piezas) para conseguir mejor puntuación, a la vez que se le da todo el tiempo que requiera el jugador para pensar con detenimiento cada puzzle.

## 4.4. DISEÑO DE LOS NIVELES

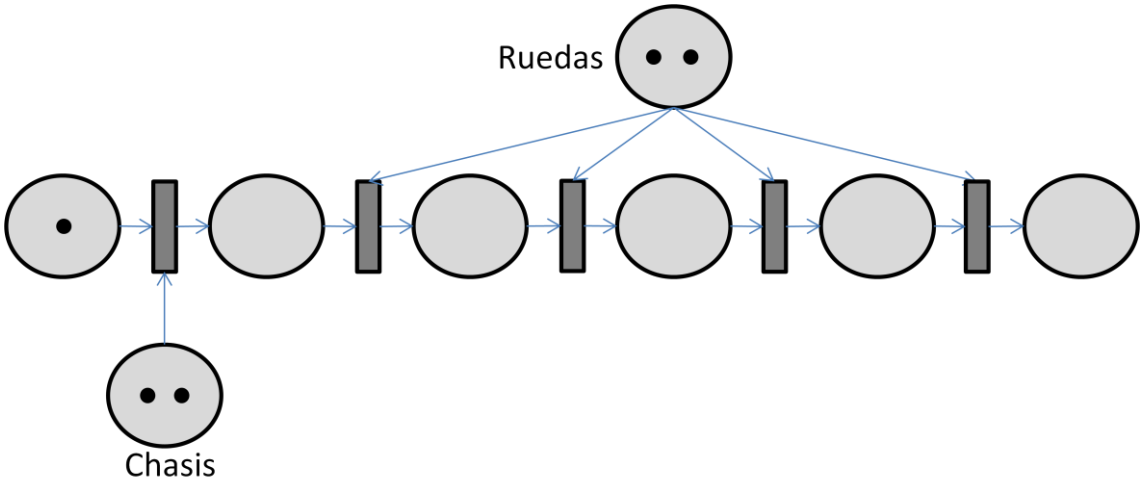
Este punto empezó a finales de noviembre con los primeros niveles, pero ha sido ampliado y modificado desde su creación, añadiendo y quitando puzzles según se tenía una percepción mejor de la experiencia del usuario con los puzzles.

Para simplificar el aprendizaje e introducir los conceptos de forma más gradual, se añadió la adquisición progresiva de las herramientas con las que interactuar con las Redes de Petri para facilitar el aprendizaje añadiendo los conceptos mínimos en cada puzzle, incorporando una breve explicación de los conceptos y herramientas introducidos en cada nivel para facilitar el aprendizaje.

A continuación se detallan los diseños de los distintos niveles que finalmente se quedaron en la configuración final.

### 4.4.1. MONTAJE DE COCHE

Máquina inicial:




**Breve introducción a las Redes de Petri (lugares, transiciones, arcos, marcas).**

**Herramientas**: poner y quitar marcas

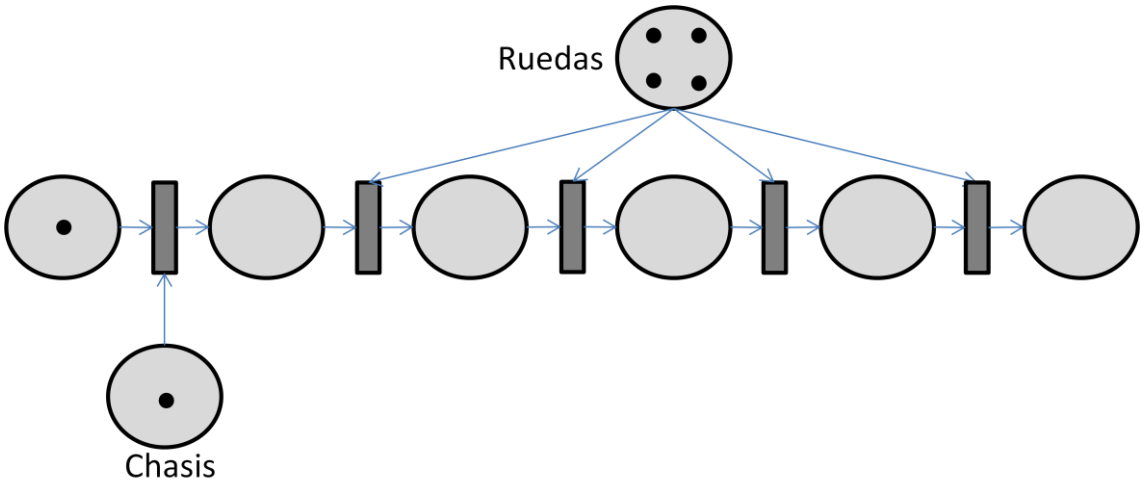
**Especificaciones**:

No puede haber un proceso de montaje de coche a medias.

Se debe de montar un coche entero, con un chasis y 4 ruedas.

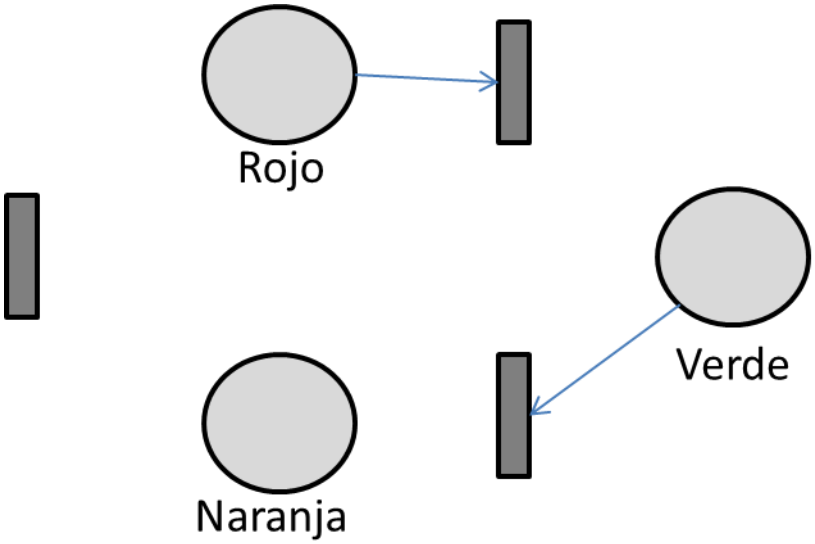
No deben sobrar piezas cuando se acabe de montar el coche.

Solución:



### 4.4.2. SEMÁFORO

Máquina inicial:




**Herramientas**: poner y quitar marcas, arcos, goma de borrar

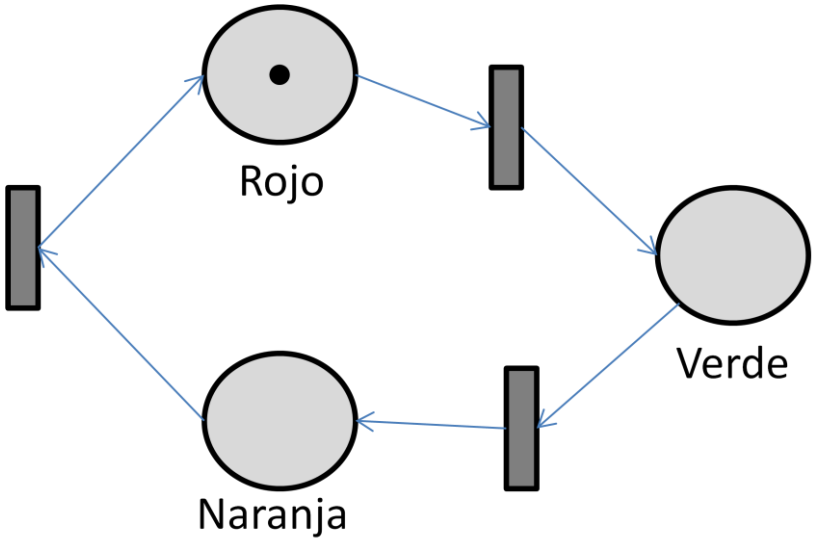
**Especificaciones**:

El semáforo empieza en rojo.

El semáforo debe tener una luz encendida, ni más ni menos.

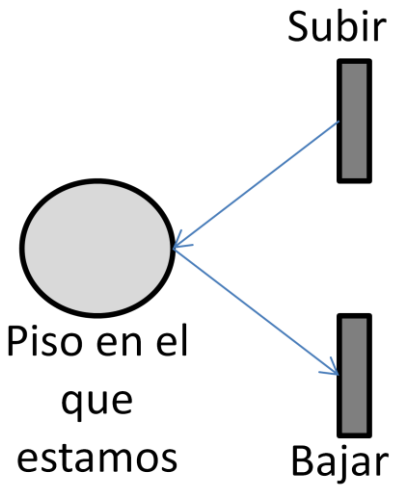
El semáforo debe ir de rojo a verde, de verde a naranja y de naranja a rojo, y ninguna otra combinación.

Solución:



### 4.4.3. ASCENSOR

Máquina inicial:



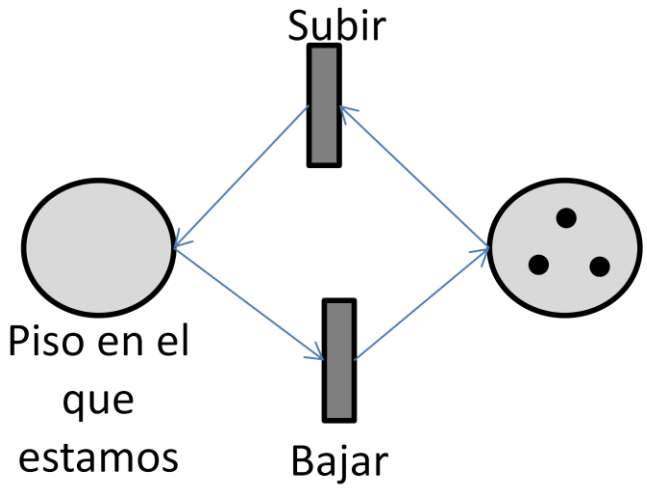

**Herramientas**: poner y quitar marcas, arcos, goma de borrar, lugares

**Especificaciones**:

El ascensor debe empezar en la planta baja.

El ascensor sólo debe poder subir y bajar entre el bajo(piso 0) y el tercer piso (piso 3).

Solución:




El nuevo lugar representa los pisos que podemos subir

### 4.4.4. CINTA TRANSPORTADORA CON PROCESO DE FABRICACIÓN

Máquina inicial:


**Breve explicación de las prioridades en las transiciones.**

**Herramientas**: poner y quitar marcas, arcos, goma de borrar, lugares, transiciones

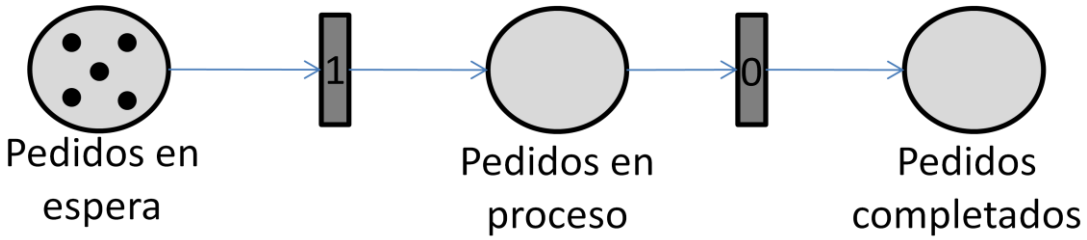
**Especificaciones**:

Debemos empezar con 5 pedidos en espera, sin productos en proceso ni fabricados.

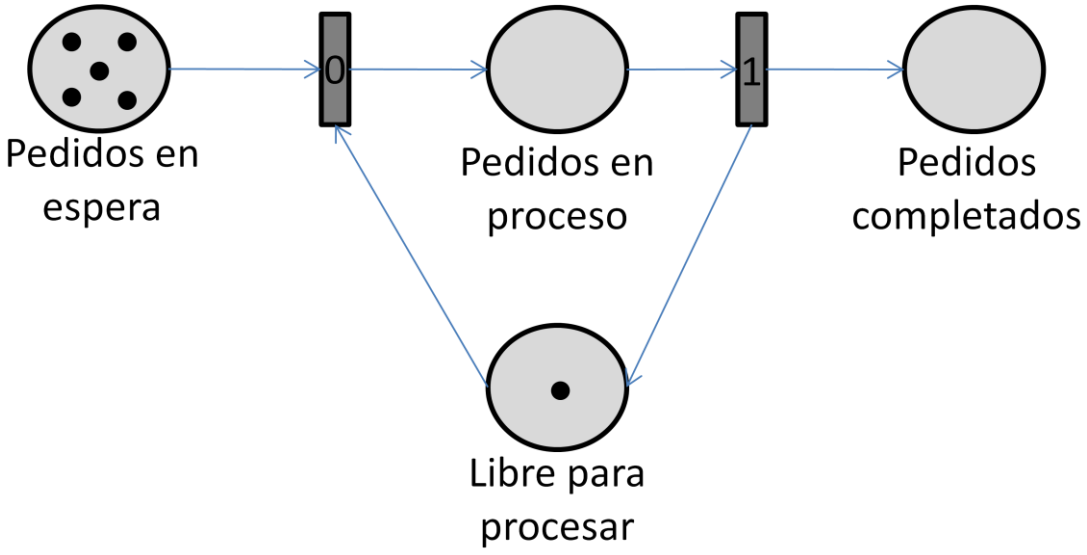
Los pedidos en espera deben pasar por el lugar de proceso, y acabar en pedidos completados.

No puede haber más de 1 pedido en proceso al mismo tiempo.

Solución:




Solución alternativa, con las transiciones intercambiadas:



### 4.4.5. MÁQUINA DE REFRESCOS

Máquina inicial:

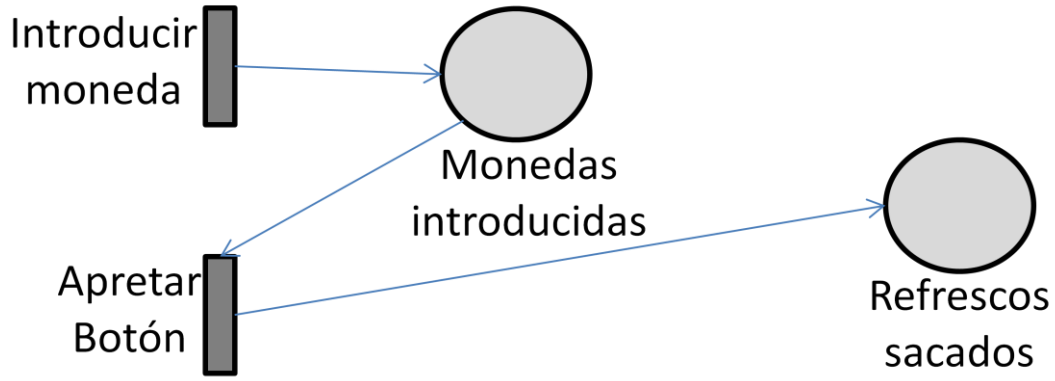

**Herramientas**: poner y quitar marcas, arcos, goma de borrar, lugares, transiciones

**Especificaciones**:

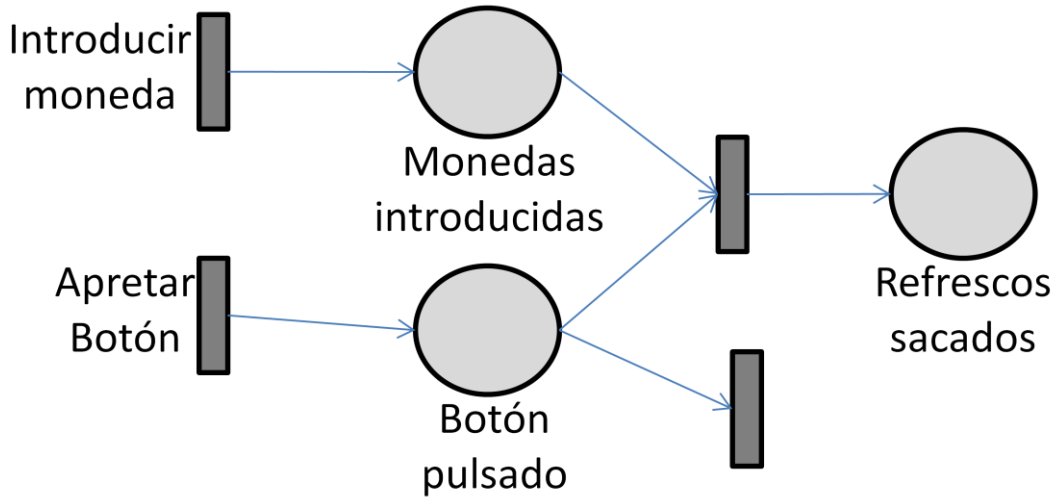
La máquina sólo debe sacar refrescos si se pulsa el botón habiendo monedas introducidas en la máquina, consumiendo una de estas monedas en el proceso.

No deben salir refrescos cuando se inserten monedas.

Solución:

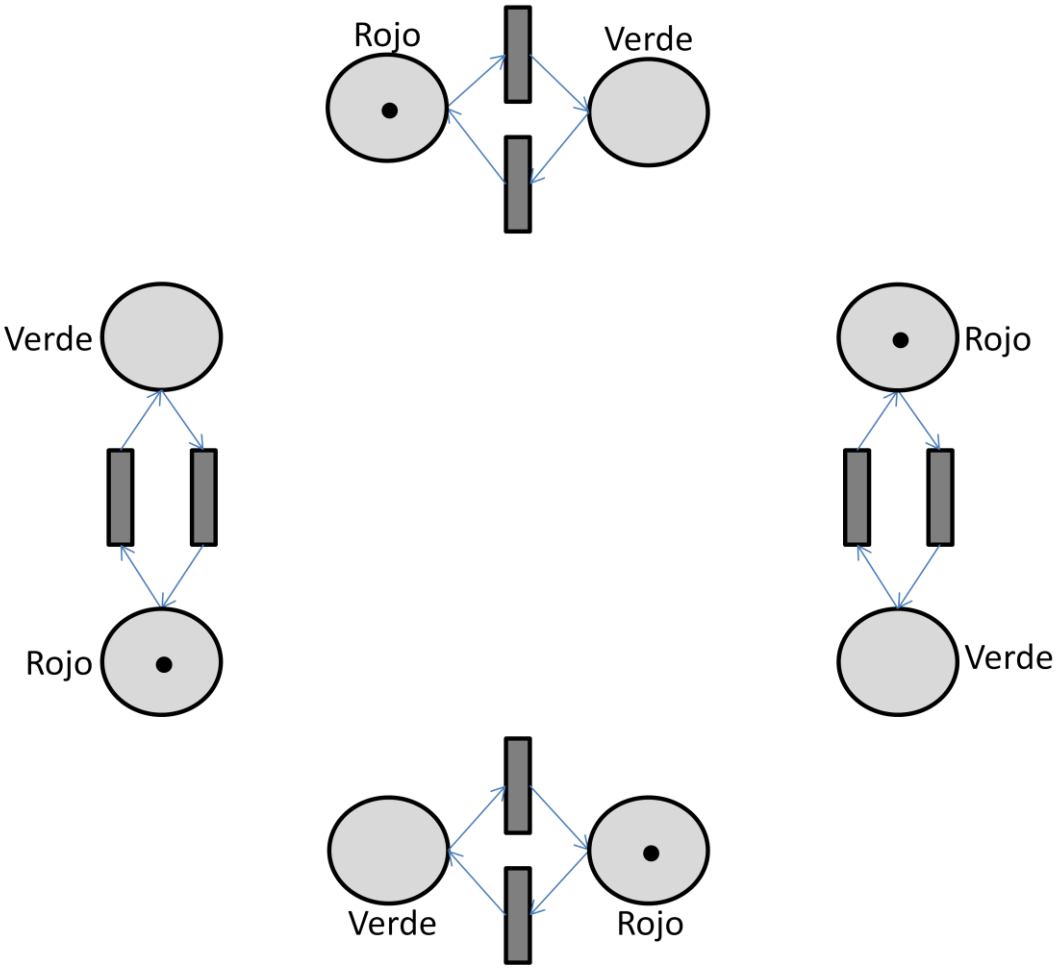



Solución sin deshabilitar las pulsaciones de botón:




### 4.4.6. CRUCE DE SEMÁFOROS

Máquina inicial:




**Herramientas**: poner y quitar marcas, arcos, goma de borrar, lugares, transiciones

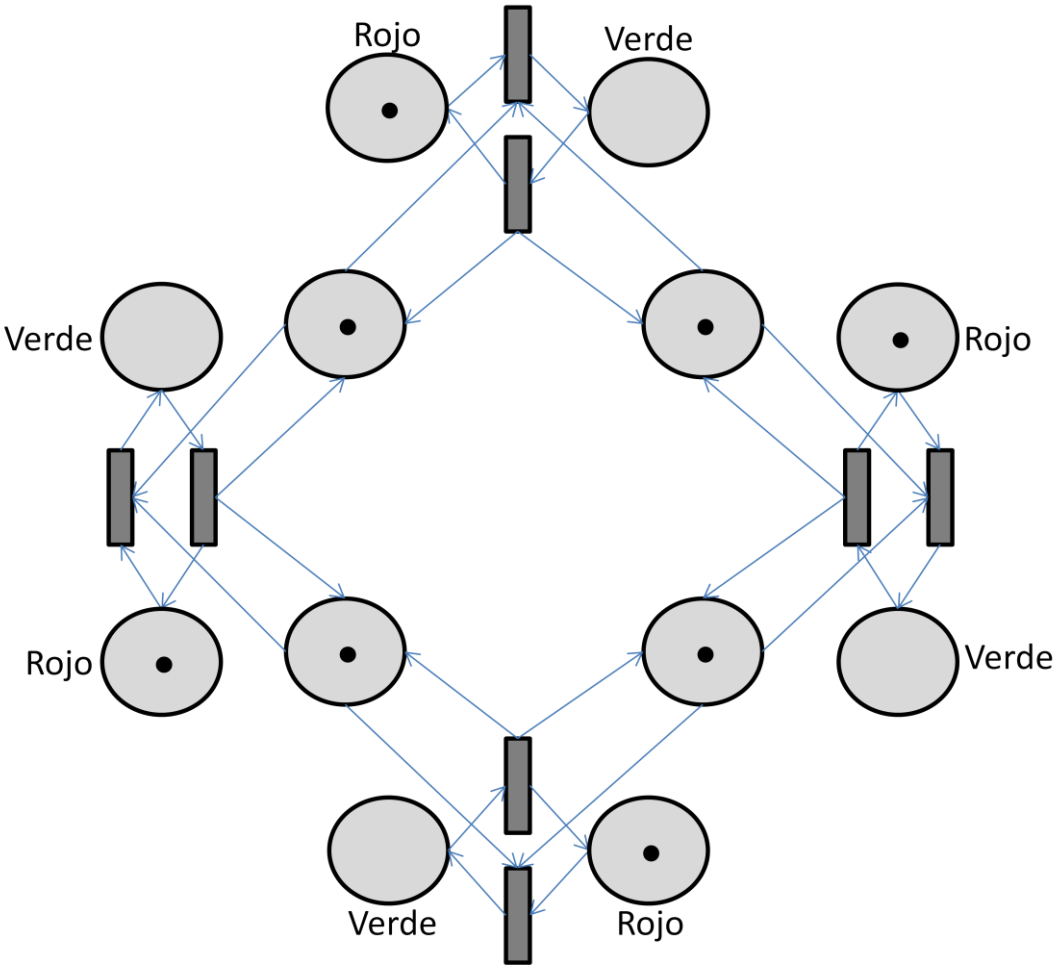
**Especificaciones**:

Todos los semáforos empiezan en rojo.

Los semáforos pueden estar en rojo o en verde, ni pueden tener ambas luces apagadas ni ambas encendidas.

No pueden haber dos semáforos contiguos en verde. (¡Provocaría un accidente!)

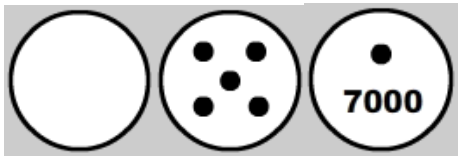
Solución:



## 4.5. LOS COMPONENTES BÁSICOS

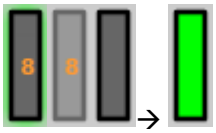
Durante el mes de diciembre se crearon los componentes básicos, tanto sus gráficos, como su lógica. Todos ellos extenderían el concepto de componente, con la capacidad de ser arrastrables. Se publicó una demo básica en la página web.

Los lugares se dibujaron como un círculo con la capacidad de albergar marcas, representadas gráficamente con puntos desde 0 a 5 marcas, y con un número para valores mayores.



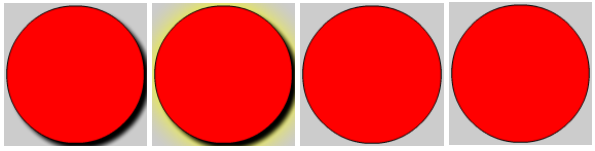

Las transiciones se crearon como un rectángulo, que tiene representaciones visuales distintas según su prioridad, representada por un número que representa el orden de disparo ante conflictos; un halo verde alrededor cuando la transición está habilitada (tiene concesión en el marcado), que desaparece al igual que se vuelve medio transparente cuando está deshabilitada(deja de tener concesión).

Para mostrar el proceso de disparo de una transición se creó una animación que cambia el color de la transición progresivamente al mismo verde que se utiliza para mostrar que está habilitada, y tras la animación pasa al estado habilitado o deshabilitado según proceda.



Los arcos se definieron como la unión de una segmento y una punta de flecha en su extremo final, y se crearon con la posibilidad de configurar fácilmente su posición de origen y final, grosor de línea y tamaño de la punta de flecha del final, lo que fue de gran utilidad para afinar las proporciones más adecuadas para su fácil visualización y facilidad de interactuación posterior.

Para los botones se utilizó la clase SimpleButton incorporada en Flash, creando un botón de prueba con cada uno de sus estados (up, over, down) y su zona de acción, igual a su estado de down (ver 4.2-Botón, para la definición de los estados del botón).



## 4.6. REPLANIFICACIÓN Y BARRA DE HERRAMIENTAS

En enero se replanificó el proyecto, así como sus objetivos, debido a que tras estudiar varias de las aplicaciones de edición de Redes de Petri (RdP) se definió el concepto de "barra de herramientas", así en lugar de tener componentes sueltos y arrastrarlos y unirlos con las piezas del puzzle, se optó por un sistema de herramientas, mucho más intuitivo y preciso sistema de edición de RdP.

Además, junto con la posibilidad de conectar diversos componentes con arcos y moverlos libremente, se implementó el gestor de arcos que redibuja los arcos entre los componentes unidos, siempre utilizando puntos cardinales de los objetos y utilizando la distancia más corta entre ambos, para minimizar el tamaño de los arcos.

Durante los meses de enero y febrero se implementó este sistema hasta definir la barra de herramientas final, en la que se ajustaron las mínimas herramientas para poder completar todos los puzzles, dándoles la mayor funcionalidad posible, así como iconos muy representativos y autoexplicativos para hacerla lo más intuitiva y usable posible.

Con esta barra de herramientas se permite que al hacer click en una de ellas (pues cada una de ella es un botón en sí), pasamos a estar utilizando es herramienta, realizando las acciones oportunas en lo que posteriormente se llamaría la "mesa de trabajo" (workbench).

Para saber en cada momento qué herramienta se estaba usando, se incluyó un pequeño decorador con la apariencia de la herramienta en el propio cursor, así como un arco ficticio que une el primer componente seleccionado por la herramienta de creación de arcos con el cursor. Para las herramientas que crean lugares y transiciones su decorador es un lugar o una transición respectivamente, que marca el lugar dónde se colocará el componente en caso de hacer click en la mesa de trabajo en ese momento.

A la hora de mover los distintos componentes creados por el escenario se le dio además la funcionalidad de "traer al frente" el componente que estaba siendo arrastrado, de tal forma que al arrastrar un componente nada puede molestar al usuario mientras lo mueve dejándolo fuera de su vista, a la vez que permite reordenar la profundidad de los elementos creados por el usuario.

Con esta barra de herramientas se dio la posibilidad de:

 Mover los componentes / Dejar de usar la herramienta actual

 Crear un nuevo lugar

 Crear una nueva transición

 Comenzar un arco desde un componente. Crear un arco entre el primer componente seleccionado y el segundo (siempre que sea posible)

 Borrar lugares, transiciones y arcos.

 Añadir una marca a un lugar

 Borrar una marca de un lugar

 Iniciar la prueba del puzzle para comprobar si está completado, o en su defecto, recibir el aviso del fallo por el que no está completo el puzzle.

Inicialmente se creó el botón de PLAY, junto con botón de STOP simplemente para iniciar o detener la ejecución de la Red de Petri creada por el usuario, pero se decidió cambiar esta funcionalidad para directamente testear los puzzles, cuya comprobación incluye la ejecución y detención de la Red de Petri del usuario, lo que nos lleva a nuestro siguiente punto.

## 4.7. CREACIÓN DE NIVEL, PUZZLE Y COMPROBACIÓN DE PUZZLE

El mes de marzo se propuso crear la primera demo con un puzzle completo, para ello el primer paso fue crear un básico sistema de navegación, permitiendo acceder a los distintos puzzles, o salir de estos para volver a la pantalla de selección de nivel.

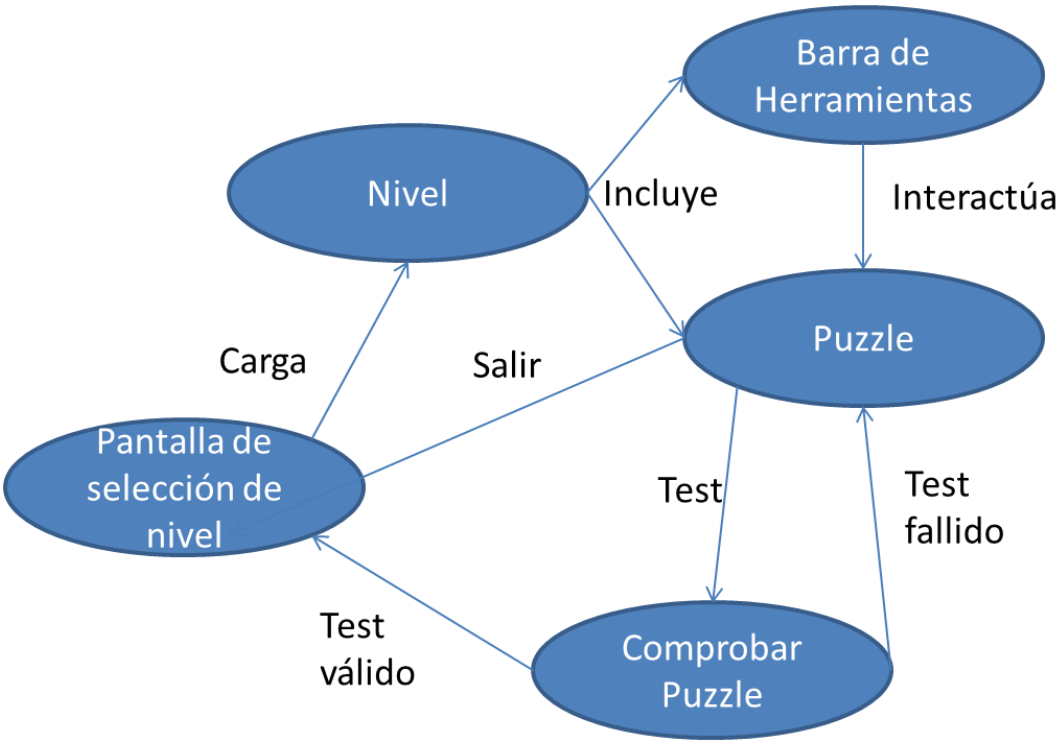



Ilustración 7: Sistema de puzzles

El segundo paso fue crear las instrucciones que muestran las especificaciones que debe cumplir el mecanismo para dar el puzzle por bueno. Se muestran nada más entrar a un puzzle, y pueden ser consultadas posteriormente así como abandonar el puzzle y volver a la pantalla de selección de nivel.

El verdadero reto del proyecto fue crear el proceso de creación y evaluación de los puzzles diseñados. Cada uno de los puzzles creados tiene un estado inicial previsto, así como unas reglas definidas para darlo por completado.

Para crear la experiencia deseada se requería la combinación de una red fija, que consistiría de las piezas enlazadas a componentes externos de máquina, en los que podríamos medir y comprobar las especificaciones dadas, con la Red de Petri (RdP) del usuario.

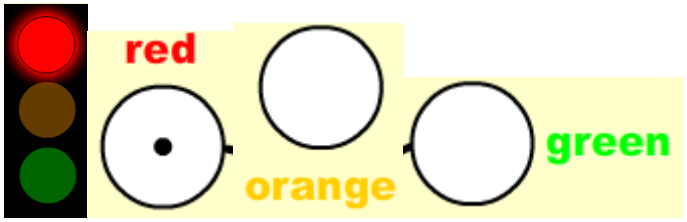
Para ello se implementaron los lugares, transiciones y arcos especiales, tales que no podían ser borrados ni arrastrados por la zona de edición de la RdP, pero que sin embargo eran interactuables con la RdP del usuario, pudiendo establecer arcos entre los componentes especiales del puzzle y los del usuario, así como cambiar el número de marcas de los lugares.

Estos componentes especiales, además de sus cualidades especiales, serían los componentes que uniríamos a las representaciones gráficas de algunos elementos destacados de los puzzles, de tal manera que cualquier acción o cambio sobre ellos se

viera reflejado visualmente en su extensión gráfica enlazada. Las transiciones especiales estarían condicionadas a los eventos externos de sus eventos enlazados.

Por ejemplo: al introducir una moneda en una ranura para monedas, sería la condición necesaria para que se intentara disparar su transición enlazada (si está habilitada).


O las luces de este semáforo estarán encendidas o no dependiendo de la presencia de marcas o no en los lugares correspondientes.




La comprobación de la completitud del puzzle podía ser automática e instantánea, pero se optó por realizar test paso a paso de tal manera que el usuario puede ver la evolución tanto del sistema de la RdP combinada como de los elementos externos asociados, y ver en qué momento se incumple alguna de las restricciones especificadas si se da el caso de que el test falle en medio de la simulación.

Para evitar que los usuarios puedan interactuar con la RdP mientras se testea, se deshabilita la interfaz, así como se oscurece levemente para dar la sensación de que hay algo entre la aplicación y el usuario, que no permite interactuar en ese momento.

En cada uno de los pasos de simulación que se efectúan para validar los puzzles, se hacen las comprobaciones pertinentes para validar las restricciones del puzzle, así como se permite la ejecución de la RdP del usuario tras cada paso, comprobando que no rompe las restricciones durante su ejecución y limitando el número de disparos posibles con la intención de evitar la ejecución infinita de una RdP con ciclos, o sin condición de parada.

## 4.8. PUZZLES Y SISTEMA DE AVANCE

Durante el mes de abril se había planificado la implantación de la herramienta en un entorno real, pero este paso se realizó en diciembre, y se creó un procedimiento para poder actualizar las versiones del juego y publicarlo en el entorno web fácilmente.

Así pues, durante los meses de abril y mayo se crearon el resto de puzzles, así como el sistema de puntuación y avance de estos mismos, incluyendo un sistema automático que guarda los datos de las puntuaciones y desbloqueos en el sistema de cookies de Flash (SharedObjects), con lo que podemos dar una mínima persistencia al avance de los jugadores aunque recarguen la página o cierren el navegador y nos visiten en un futuro.

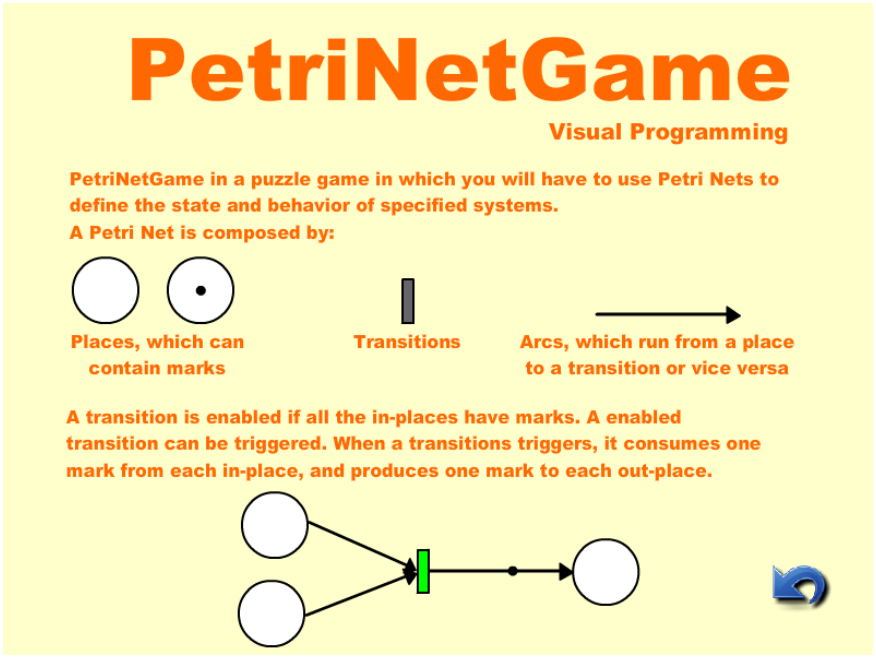
Durante este periodo se procedió a publicar una demo con varios puzzles para sus testeo por otro usuarios, valorando sus opiniones y mejorando la usabilidad y la apariencia de la herramienta.

Como aplicación destinada a ser utilizada por usuarios no expertos, se prestó especial atención a los comentarios y críticas de los beta-testers durante su primera experiencia de juego.

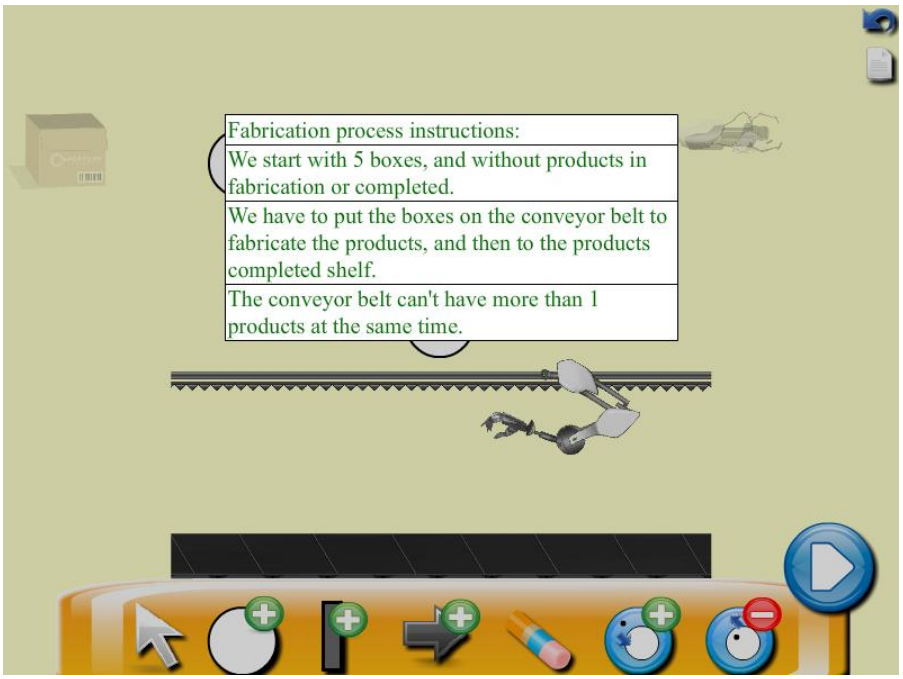
El comentario más común era la ausencia de una explicación básica de las Redes de Petri para empezar con el primer puzzle, para ello en junio se incluyó un nuevo apartado en la pantalla principal en la que se explica brevemente el funcionamiento de las Redes de Petri.

## 5. LA APLICACIÓN: PETRINETGAME

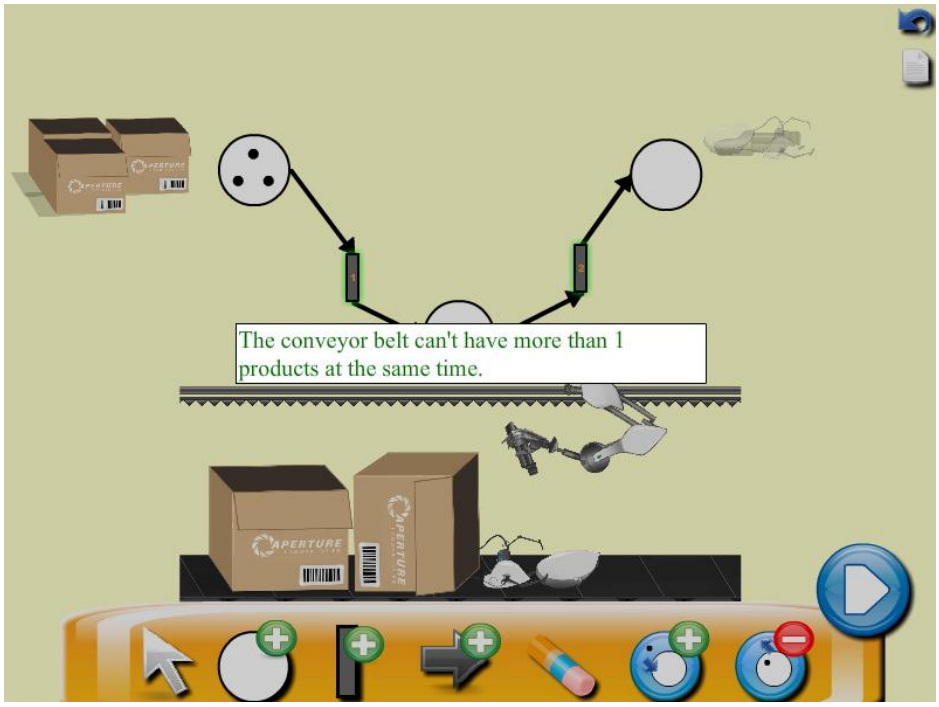

Esta es la pantalla inicial de la aplicación: PetriNetGame. En ella se visualiza rápidamente el estado de los niveles, si los tenemos completados y con qué puntuación, además de poder entrar en cualquiera de ellos que no esté bloqueado o consultar las instrucciones.



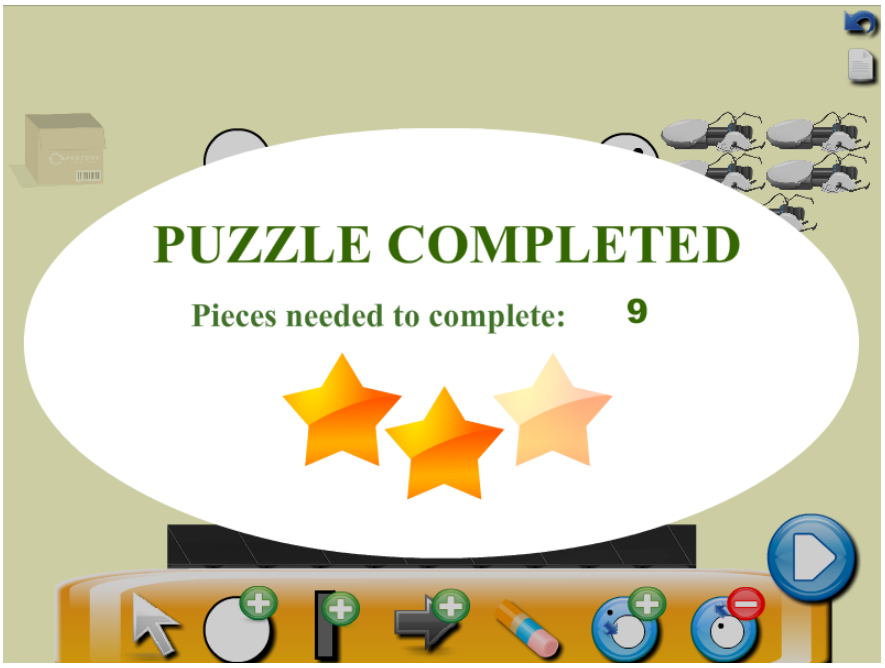
Nada más entrar a cualquier nivel, se nos muestran las instrucciones divididas por especificaciones, que son necesarias para completarlo. Entonces procedemos a interactuar con la Red de Petri del puzzle, usando la barra inferior de herramientas, para solucionarlo.




Estas mismas especificaciones son los mensajes de aviso cuando el puzzle falla por incumplir esa misma especificación.



Tras completar la comprobación del puzzle con éxito podremos observar el panel de "puzzle completado", así como las piezas utilizadas y la puntuación obtenida en el puzzle. Y una vez cerrado este panel, volveremos al panel inicial de selección de nivel dónde veremos el nuevo estado de los niveles si hemos mejorado.



# 6. TRABAJOS FUTUROS

La capacidad expresiva y el formalismo de las Redes de Petri (RdP) junto con la gran flexibilidad y adaptabilidad de la herramienta abren muchas posibilidades de expansión y desarrollo futuro.

Una de las expansiones más inmediatas sería continuar diseñando más puzzles, más niveles y más herramientas (ej: arcos inhibidores), añadiéndolos al final del actual flujo de niveles, mezclándolos o añadiendo otro flujo de niveles (conocidos como mundos en gamificación).

Tomando el camino de expandir la herramienta con más contenido de niveles y puzzles, incluso se podría intentar definir de forma formal los puzzles y así permitir la creación de puzzles mediante archivos de configuración, o mejor aún con una herramienta de edición de puzzles en la que podamos definir las reglas y relaciones de cada uno de los elementos con las partes de la RdP del puzzle, definir las restricciones necesarias para completarlo y los test de simulación a comprobar.

Otra posible expansión, es que se intente automatizar el proceso de comprobación de los puzzles, definiendo formalmente las restricciones de los sistemas de tal manera que el propio juego se encargue de diseñar los tests y comprobaciones necesarias para poder afirmar que se cumplen las restricciones del puzzle.

Con esta mejora se podría pensar en la verificación automática de los puzzles en sí, para comprobar si pueden ser completados o son imposibles de resolver.

Aunque se ha intentado crear una interfaz lo más usable e intuitiva posible, se han ido mejorando tanto los gráficos como la estética de la aplicación, y se han agregado muchos gráficos de personas externas al proyecto, pero se podría mejorar la aplicación actual incluyendo gráficos y animaciones más elaboradas, tanto para las piezas de los puzzles, los menús, botones, paneles y fondos.

Otra de las mejoras más artísticas del proyecto sería la incorporación de música y sonidos en la aplicación, para mejorar la inmersión y entretenimiento del juego.

Otras mejoras más externas a la aplicación en sí sería la creación de un ranking online, de la gente que ha completado los puzzles, o adaptar la aplicación para utilizar las redes sociales, poder compartir tus resultados y competir con tus amigos. Con la mejora del editor de puzzles se podría incluso mandar esos puzzles a tus amigos para ver si son capaces de completarlos.

Una de las mejoras o añadidos más solicitados por los testers de la aplicación es la de “volver atrás” o reiniciar el puzzle, para volver a empezar el puzzle (igual que si sales y

entras al nivel) o volver al estado de la RdP antes de iniciar los test de comprobación de los puzzles.

Tomando otra vertiente distinta del objetivo más abstracto del proyecto, se podrían diseñar otra clase de herramientas o videojuegos que acerquen conceptos técnicos al público general, aprovechando su representación visual, mostrando su utilidad en entornos reales y facilitando su aprendizaje, como por ejemplo: los diagramas de estado, los distintos tipos de grafos de UML…

Siguiendo con esta idea de crear herramientas que faciliten el aprendizaje de conceptos y herramientas técnicas al público general se podría crear una página web a modo de repositorio de estas herramientas para poder elegir cuál de ellas queremos aprender, divididas por tipos o por su utilidad práctica en diversos campos del mundo real.

Por otro lado, enfocando la expansión por el desarrollo automático dirigido por modelos, se podría diseñar una mejora que analizara las Redes de Petri creadas por el usuario y se generara código en uno o varios lenguajes implementando el diseño de la Red de Petri.

# 7. CONCLUSIONES

Hay muchas conclusiones que podemos obtener tanto del desarrollo como del resultado del proyecto.

Quizás lo más importante que hemos aprendido de este proyecto es la enorme dificultad que supone simplificar conceptos complejos y abstractos para hacerlos simples y divertidos de aprender, y cómo solucionar la mayoría de todas estas dificultades.

Es importante ser consciente de que cuando nos enfrentamos a un proyecto muy distinto a lo que hemos hecho hasta ahora, siempre van a haber cambios tanto en el diseño como en la planificación, y hay que ser capaz de rediseñar las partes del proyecto necesarias y poder asumir la nueva replanificación para poder seguir llevando a cabo el proyecto.

Otro punto muy importante ha sido la acotación y planificación del proyecto para poder llevarlo a cabo, ya que los proyectos de este tipo sufren a menudo las consecuencias de no acotar planes muy ambiciosos y acaban siendo abandonados sin ni siquiera una beta o una demo funcional. Siempre es posible añadir más cosas a un proyecto de este tipo, pero hay que ser capaz de centrarse en lo más importante del proyecto y dejar para iteraciones posteriores los añadidos más superfluos o los casos de uso que no pertenezcan a la parte central del proyecto.

Como se esperaba con la motivación del proyecto, ha sido posible acercar las Redes de Petri (RdP) a un número relevante de personas que han sido capaces de completar los puzzles de la aplicación mientras se divertían y se han dado cuenta poco a poco de las cualidades y el poder expresivo de estas. Sin embargo, el interés en la capacidad de expresar formalmente el comportamiento de sistemas asíncronos ha sido bajo, ya que la mayoría de la gente no necesita de tal grado de expresividad y formalismo para llevar a cabo sus proyectos o diseños.

## 8. BIBLIOGRAFÍA

[1] M. Ajmone Marsan, Gianfranco Balbo, Gianni Conte, Susanna Donatelli, Giuliana Franceschinis - "Modelling With Generalised Stochastic Petri Nets" - capítulo 4

[2] Pieter J. Mostermany, Martin Ottery, Hilding Elmqvistz – "Modeling Petri Nets As Local Constraint Equations For Hybrid Systems Using Modelicatm" - Petri Net Semantics

[3] Santiago C. Pérez - "Modelación, Simulación De Funcionamiento Y Evaluación De Prestaciones De Protocolos De Red Con Redes De Petri. Desarrollo De Herramientas Académicas De Enseñanza", 2006

[4] Wil van der Aalst, Vincent Almering, Hermen Wijbenga – "Interactive Tutorials on Petri Nets"
http://www.informatik.uni-hamburg.de/TGI/PetriNets/introductions/aalst/

[5] Sklenar Jaroslav – "Example Petri Nets"
http://staff.um.edu.mt/jskl1/pnexamps.html

[6] Tadao Murata – "Petri Nets: Properties, Analysis and Applications"
http://embedded.eecs.berkeley.edu/Research/hsc/class.F03/ee249/discussionpapers/PetriNets.pdf

[7] James L. Peterson – "Petri Nets"
http://www.rose-hulman.edu/Users/faculty/young/CS-Classes/csse373/Spring2009/Resources/peterson77.pdf

[8] Carl Adam Petri , Wolfgang Reisig - "Petri net.Scholarpedia"
http://www.scholarpedia.org/article/Petri_net

[9] G. Rozenburg, J. Engelfriet – "Elementary Net Systems"

[10] Ariel Sabiguero - "Nomenclatura y definiciones básicas de Redes de Petri"

[11] Carlos Aguirre - "Redes de Petri Estocásticas"

[12] Peter J. Haas – "Stochastic Petri Nets For Modelling And Simulation"

# 9. ANEXO: CÓDIGO FUENTE

A continuación se adjunta y detalla todo el código desarrollado para la implementación tanto del loader de la aplicación como del juego en sí. El orden de las clases de más específica e independiente a más general facilita la comprensión de su lectura sin tener que saltar entre clases para entender su funcionamiento.

Se ha elegido una aproximación por eventos, de tal forma que los elementos interactivos más externos tengan las dependencias mínimas y sean reutilizables en cualquier otro entorno.

## 9.1. CÓDIGO FUENTE DEL LOADER

```
package util
{
    /**
     * Operaciones útiles para Urls
     * @author Alex
     */
    public class UrlUtil
    {

        /**
         * Devuelve la url de la carpeta que contiene el fichero de la
url especificada
         * @param   fileUrl la url del fichero
         * @return  la ruta de la carpeta
         */
        public static function
getFolderUrlFromFileUrl(fileUrl:String):String
        {
            //Separa la ruta del fichero en subcadenas separadas por
"/"
            var domain:Array = fileUrl.split("/");

            var folderUrl:String = "";
            //Tomamos todas las subcadenas excepto la última
            for(var i:int=0;i<domain.length-1;i++){
                folderUrl += domain[i] + "/";
            }

            return folderUrl;
        }

        /**
         * Devuelve una etiqueta para evitar la cache si se añade a
una url
         */
        public static function getNoCacheTag():String
        {
            return "?nocache=" + Math.floor(Math.random() * 10000 +
1);
        }

    }
```

```
}
```

```
package loading
{
    import flash.display.DisplayObject;
    import flash.display.Loader;
    import flash.display.MovieClip;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.KeyboardEvent;
    import flash.events.ProgressEvent;
    import flash.net.URLRequest;
    import flash.system.Security;
    import flash.text.TextField;
    import util.UrlUtil;

    /**
     * Loader simple de una aplicación. Provee la url base.
     * Se deben sobreescribir "animationUrl" y "applicationUrl" con
las rutas específicas
     * Provee métodos que sobreescribir para gestionar los progresos
de carga (onAnimationLoadProgress, onApplicationLoadProgress,
onApplicationAssetsLoadProgress)
     * 1. Carga la animación
     * 2. Carga la aplicación mientras avanza la animación
     * 3. Cuando la aplicación está cargada, muestra la carga de los
assets de la aplicación
     * 4. Cuando la aplicación está lista, cambiamos la animación de
carga por la aplicación
     * @author Alex
     */
    public class AbstractLoader extends Sprite
    {
        /** Url base*/
        protected var baseUrl:String;

        /** Loader con el que cargar la animación y la aplicación */
        protected var loader:Loader;

        /** Animación de carga */
        protected var animation:MovieClip;

        /** Aplicación a cargar */
        protected var application:DisplayObject;

        /**TextField de Debug*/
        protected var debugTextField:TextField;

        public function AbstractLoader():void
        {
            addDebugTextField();

            Security.allowDomain('*');

            if (stage) init();
            else addEventListener(Event.ADDED_TO_STAGE, init);
        }

        /**
```

```
         * Añade un log de debug
         */
        private function addDebugTextField():void
        {
            debugTextField = new TextField();
            debugTextField.text = "Debug TextField";
            debugTextField.width = 700;
            debugTextField.visible = false;
            addChild(debugTextField);
        }

        /**
         * Inicializa el loader, y con él la carga de su animación de
carga
         * @param   e   evento de añadido a escena
         */
        private function init(e:Event = null):void
        {
            removeEventListener(Event.ADDED_TO_STAGE, init);

            stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);

            //obtenemos la ruta dónde está la aplicación
            baseUrl =
UrlUtil.getFolderUrlFromFileUrl(root.loaderInfo.url);
            //baseUrl =
"http://files.petrinetgame.webnode.es/200000028-1fb2721357/";
            trace("baseUrl = " + baseUrl);

            loadLoaderAnimation();
        }

        /**
         * Al recibir un evento de teclado de "tecla_presionada"
         * @param   e   evento de teclado
         */
        private function onKeyDown(e:KeyboardEvent):void
        {
            //Al presionar la tecla D
            if (e.keyCode == 68/*D*/ || e.keyCode == 100/*d*/) {
                //visualizamos o no el texto de debug, alternando el
estado anterior de visibilidad
                debugTextField.visible = !debugTextField.visible;
            }
        }

        /**
         * Carga la animación del loader
         */
        private function loadLoaderAnimation():void
        {
            loader = new Loader();

loader.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
onAnimationLoadProgress);
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE,
onAnimationLoadComplete);

loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR,
onError);
```

```
            var urlRequest:URLRequest = new URLRequest(animationUrl);
            loader.load( urlRequest);

            debugTextField.text = animationUrl;
        }

        /**
         * Si hay algún error en la carga de la animación de carga
         * @param   e   evento de error
         */
        private function onError(e:IOErrorEvent):void
        {
            debugTextField.text = "ERROR " + e.text;
            trace("ERROR " + e.text);
        }

        /**
         * Cuando progresa la carga de la animación de carga
         * @param   e   evento de progreso
         */
        protected function
onAnimationLoadProgress(e:ProgressEvent):void
        {
            debugTextField.text = "onAnimationLoadProgress " +
e.bytesLoaded;

            //TO OVERRIDE
        }

        /**
         * Cuando se completa la carga de la animación de carga
         * @param   e evento de carga completa
         */
        protected function onAnimationLoadComplete(e:Event):void
        {
            debugTextField.text = "onAnimationLoadComplete";


loader.contentLoaderInfo.removeEventListener(ProgressEvent.PROGRESS,
onAnimationLoadProgress);

loader.contentLoaderInfo.removeEventListener(Event.COMPLETE,
onAnimationLoadComplete);

loader.contentLoaderInfo.removeEventListener(IOErrorEvent.IO_ERROR,
onError);

            //recogemos la animación de carga
            animation = loader.content as MovieClip;
            //y la añadimos a escena
            addChild(animation);

            loadApplication();
        }

        /**
         * Carga la aplicación
         */
        private function loadApplication():void
```

```
        {

loader.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
onApplicationLoadProgress);
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE,
onApplicationLoadComplete);

loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR,
onError);
            loader.load( new URLRequest(applicationUrl));
        }

        /**
         * Cuando progresa la carga de la aplicación
         * @param   e evento de progreso
         */
        protected function
onApplicationLoadProgress(e:ProgressEvent):void
        {
            //TO OVERRIDE
        }

        /**
         * Cuando se completa la carga de la aplicación
         * @param   e evento de carga completa
         */
        private function onApplicationLoadComplete(e:Event):void
        {

loader.contentLoaderInfo.removeEventListener(ProgressEvent.PROGRESS,
onApplicationLoadProgress);

loader.contentLoaderInfo.removeEventListener(Event.COMPLETE,
onApplicationLoadComplete);

loader.contentLoaderInfo.removeEventListener(IOErrorEvent.IO_ERROR,
onError);

            //recogemos la aplicación
            application = loader.content;

            application.addEventListener(ProgressEvent.PROGRESS,
onApplicationAssetsLoadProgress);
            application.addEventListener(Event.COMPLETE,
onApplicationAssetsLoadComplete);

            //inicializamos la aplicación
            Object(application).init();
        }

        /**
         * Cuando progresa la carga de los assets de la aplicación
         * @param   e evento de progreso
         */
        protected function
onApplicationAssetsLoadProgress(e:ProgressEvent):void
        {
            //TO OVERRIDE
        }
```

```
        /**
         * Cuando termina la carga de los assets de la aplicación
         * @param   e evento de carga completa
         */
        private function onApplicationAssetsLoadComplete(e:Event):void
        {
            application.removeEventListener(ProgressEvent.PROGRESS,
onApplicationAssetsLoadProgress);
            application.removeEventListener(Event.COMPLETE,
onApplicationAssetsLoadComplete);

            changeToApplication();
        }

        /**
         * Quita la animación y pone en su lugar la aplicación
         */
        private function changeToApplication():void
        {
            removeChild(animation);
            addChild(application);
        }


        /**
         * Ruta de la animación
         */
        protected function get animationUrl():String
        {
            //TO OVERRIDE WITH THE SPECIFIC URL
            return "";
        }

        /**
         * Ruta de la aplicación
         */
        protected function get applicationUrl():String
        {
            //TO OVERRIDE WITH THE SPECIFIC URL
            return "";
        }
    }

}
```

```
package loading
{
    import flash.events.Event;
    import flash.events.ProgressEvent;
    import util.UrlUtil;
    /**
     * Loader simple de una aplicación con animación de progreso
     * @author Alex
     */
    public class SimpleLoader extends AbstractLoader
    {

        public function SimpleLoader()
        {
            super();
```

```
        }

        /**
         * Cuando se completa la carga de la animación de carga
         * @param   e evento de carga completa
         */
        override protected function
onAnimationLoadComplete(e:Event):void
        {
            super.onAnimationLoadComplete(e);

            //inicializamos la animación
            if (animation && animation.loader) {
                if (animation.loader.bar_mc) {
                    //escalamos la mitad de la barra con el % de carga
                    animation.loader.bar_mc.scaleX = 0;
                }
                if (animation.loader.loader_txt) {
                    //mostramos el % de carga
                    animation.loader.loader_txt.text = 0 + "%";
                }
            }
        }

        /**
         * Cuando progresa la carga de la aplicación
         * @param   e evento de progreso
         */
        override protected function
onApplicationLoadProgress(e:ProgressEvent):void
        {
            var proportion:Number = getProportionFromProgressEvent(e);

            if (animation && animation.loader) {
                if (animation.loader.bar_mc) {
                    //escalamos la mitad de la barra con el % de carga
                    animation.loader.bar_mc.scaleX = proportion /2;
                }
                if (animation.loader.loader_txt) {
                    //mostramos el % de carga
                    animation.loader.loader_txt.text =
Math.floor(proportion * 100 /2) + "%";
                }
            }
        }

        /**
         * Cuando progresa la carga de los assets de la aplicación
         * @param   e evento de progreso
         */
        override protected function
onApplicationAssetsLoadProgress(e:ProgressEvent):void
        {
            var proportion:Number = getProportionFromProgressEvent(e);

            if (animation && animation.loader) {
                if (animation.loader.bar_mc) {
                    //escalamos la barra con el % de carga
                    animation.loader.bar_mc.scaleX = 0.5 + proportion
/2;
```

```
                }
                if (animation.loader.loader_txt) {
                    //mostramos el % de carga
                    animation.loader.loader_txt.text = 50 +
Math.floor(proportion * 100 /2) + "%";
                }
            }
        }

        /**
         * Devuelve la proporción entre los bytes cargados y los
totales de un evento de progreso de carga
         * @param   e   evento de progreso
         * @return  la proporción entre los bytes cargados y los
totales
         */
        private function
getProportionFromProgressEvent(e:ProgressEvent):Number
        {
            //recogemos los bytes totales a descargar
            var total:Number = e.bytesTotal;
            //recogemos los bytes descargados
            var loaded:Number = e.bytesLoaded;

            var proportion:Number;

            //si aún no tenemos los datos del tamaño total de bytes de
la carga
            if (total == 0) {
                //consideramos la proporción como 0
                proportion = 0;
            } else {
                proportion = loaded / total;
            }

            return Math.min(proportion, 1);
        }

        /**
         * Ruta de la animación
         */
        override protected function get animationUrl():String
        {
            //hardcodemos la ruta de los assets en Webnode porque
gestiona los archivos subidos en servidores distintos, sin unificar
rutas
            if (baseUrl.indexOf("webnode") != -1) {
                return
"http://files.petrinetgame.webnode.es/200000020-
92bfc93b95/loader_gfx.swf" + UrlUtil.getNoCacheTag();
            } else {
                return baseUrl + "gfx/loader_gfx.swf";
            }
        }

        /**
         * Ruta de la aplicación
         */
        override protected function get applicationUrl():String
        {
```

```
            //hardcodemos la ruta de los assets en Webnode porque
gestiona los archivos subidos en servidores distintos, sin unificar
rutas
            if (baseUrl.indexOf("webnode") != -1) {
                return
"http://files.petrinetgame.webnode.es/200000030-
0458005ae2/PetriNetsGame.swf" + UrlUtil.getNoCacheTag();
            } else {
                return baseUrl + "PetriNetsGame.swf";
            }
        }

    }

}
```

## 9.2. CÓDIGO FUENTE DE LA APLICACIÓN

La librería de assets (AssetsLibrary) es la clase que carga y suministra todos los assets de la aplicación, como se definió en el punto 4.2. Con los Sprites que suministra la librería de assets se crean los componentes (Component), que son la clase base para todos los elementos gráficos con lógica, excepto para los botones.

```
package assets
{
    import flash.display.Loader;
    import flash.display.LoaderInfo;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.EventDispatcher;
    import flash.events.IOErrorEvent;
    import flash.events.ProgressEvent;
    import flash.net.URLRequest;
    /**
     * Singleton encargada de gestionar la librería de assets
     * Se le pueden añadir librerías externas para que las cargue
     * Se le pueden pedir Sprites de esas librerías por nombre
     * @author Alex
     */
    public class AssetsLibrary extends EventDispatcher
    {
        /** TAMAÑO PREVISTO DE UN ASSET, se utiliza como previsión de
tamaño mientras no se tienen datos reales */
        private static const ASSET_BYTES_SIZE:Number = 200;

        /** Instancia singleton */
        private static var _instance:AssetsLibrary;

        /** Cerradura de singleton */
        private static var canCreateInstance:Boolean = false;

        /** Urls de las librerías que se desean cargar */
        private var pendingExternalLibrariesUrls:Vector.<String>;

        /** Loaders de las librerías cargándose */
        private var loaders:Vector.<Loader>;
```

```
        /** Cargas completadas desde el comienzo de la última carga */
        private var completedLoads:int;

        /** Librerías externas cargadas */
        private var libraries:Vector.<LoaderInfo>;

        /**
         * Constructor singleton
         */
        public function AssetsLibrary()
        {
            if (!canCreateInstance) {
                throw new Error("AssetsLibrary es Singleton, no debe
ser instanciada");
            }
            pendingExternalLibrariesUrls = new Vector.<String>();
            loaders = new Vector.<Loader>();
            libraries = new Vector.<LoaderInfo>();
        }

        /**
         * @see EventDispatcher
         */
        public static function addEventListener(type:String,
listener:Function, useCapture:Boolean=false, priority:int=0,
useWeakReference:Boolean=false):void
        {
            instance.addEventListener(type, listener, useCapture,
priority, useWeakReference);
        }

        /**
         * @see EventDispatcher
         */
        public static function removeEventListener(type:String,
listener:Function, useCapture:Boolean=false):void
        {
            instance.removeEventListener(type, listener, useCapture);
        }

        /**
         * Creación de instancia singleton
         */
        private static function get instance():AssetsLibrary
        {
            if (_instance == null) {
                canCreateInstance = true;
                _instance = new AssetsLibrary();
                canCreateInstance = false;
            }
            return _instance;
        }

        /**
         * Obtiene un Sprite de las librerías, buscando por nombre
         * @param   className   el nombre de la clase exportada
         * @return  una instancia del gráfico de la librería de la
clase solicitada
         */
        public static function getSprite(className:String):Sprite
```

```
        {
            for (var i:int = 0; i < instance.libraries.length; i++)
            {
                if
(instance.libraries[i].applicationDomain.hasDefinition(className)) {
                    var def:Class =
instance.libraries[i].applicationDomain.getDefinition(className) as
Class;
                    return new def();
                }
            }
            throw new Error("Can'f find " + className + " in the
AssetsLibrary");
        }

        /**
         * Añade una librería externa para ser cargada
         * @param   url la url de la librería externa
         */
        public static function addExternalSwf(url:String):void
        {
            instance.pendingExternalLibrariesUrls.push(url);
        }

        /**
         * Da la orden de cargar las librerías pendientes de carga
         */
        public static function loadPendingLibraries():void
        {
            instance.completedLoads = 0;

            for (var i:int = 0; i <
instance.pendingExternalLibrariesUrls.length; i++)
            {
                //añadimos a cargar las librerías pendientes

instance.addLibraryLoad(instance.pendingExternalLibrariesUrls[i]);
            }
            //reiniciamos la lista de librerías
            instance.pendingExternalLibrariesUrls = new
Vector.<String>;
        }

        /**
         * Carga una librería externa
         * @param   url url de la librería externa
         */
        private function addLibraryLoad(url:String):void
        {
            //creamos el loader de la librería y escuchamos sus
eventos
            var loader:Loader = new Loader();

loader.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
onLibraryLoadProgress);
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE,
onLibraryLoadComplete);

loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR,
onError);
```

```
            //lo añadimos a la lista de loaders
            loaders.push(loader);

            //comenzamos la carga de la librería externa
            loader.load( new URLRequest(url));
        }

        /**
         * Al avanzar la carga de una librería
         * @param   e evento de progreso
         */
        private function onLibraryLoadProgress(e:ProgressEvent):void
        {
            var progressEvent:ProgressEvent = new
ProgressEvent(ProgressEvent.PROGRESS);
            progressEvent.bytesLoaded = getAllLoadsBytesLoaded();
            progressEvent.bytesTotal = getAllLoadsBytesTotal();

            dispatchEvent(progressEvent);
        }

        /**
         * Devuelve todos los bytes cargados de todas las cargas
         * @return  el total de bytes cargados
         */
        private function getAllLoadsBytesLoaded():Number
        {
            var allLoadsBytesLoaded:Number = 0;
            for (var i:int = 0; i < loaders.length; i++)
            {
                allLoadsBytesLoaded +=
loaders[i].contentLoaderInfo.bytesLoaded;
            }
            return allLoadsBytesLoaded;
        }

        /**
         * Devuelve el total de bytes de todas las cargas
         * @return  el tamaño total de las cargas
         */
        private function getAllLoadsBytesTotal():Number
        {
            var allLoadsBytesTotal:Number = 0;
            for (var i:int = 0; i < loaders.length; i++)
            {
                var bytesTotal:Number =
loaders[i].contentLoaderInfo.bytesTotal;
                //si el tamaño total es 0 (porque aún no ha comenzado
la carga)
                if (bytesTotal == 0) {
                    //suponemos un tamaño standar
                    bytesTotal = ASSET_BYTES_SIZE;
                }

                allLoadsBytesTotal += bytesTotal;
            }
            return allLoadsBytesTotal;
        }
```

```
        /**
         * Al completarse la carga de una librería
         * @param   e   evento de carga completa
         */
        private function onLibraryLoadComplete(e:Event):void
        {
            //quitamos los listeners de la carga completada
            var dispatcher:EventDispatcher = e.target as
EventDispatcher;
            dispatcher.removeEventListener(ProgressEvent.PROGRESS,
onLibraryLoadProgress);
            dispatcher.removeEventListener(Event.COMPLETE,
onLibraryLoadComplete);
            dispatcher.removeEventListener(IOErrorEvent.IO_ERROR,
onError);

            //añadimos la librería a las librerías cargadas
            libraries.push(e.target as LoaderInfo);

            //anotamos que se ha completado una carga
            completedLoads++;

            //si hemos completado todas las cargas
            if (completedLoads == loaders.length) {
                onAllLoadsCompleted();
            }
        }

        /**
         * Cuando todas las cargas se han completado
         */
        private function onAllLoadsCompleted():void
        {
            dispatchEvent(new Event(Event.COMPLETE));
        }

        /**
         * Si hay algún error en la carga de una librería
         * @param   e   evento de error
         */
        private function onError(e:IOErrorEvent):void
        {
            trace("IOERROR " + e.text);
        }

    }

}
```

```
package core
{
    import assets.AssetsLibrary;
    import flash.display.DisplayObjectContainer;
    import flash.display.MovieClip;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.EventDispatcher;
    import flash.events.MouseEvent;
    import flash.ui.Mouse;
    import flash.ui.MouseCursor;
```

```
    /**
     * Clase que envuelve un Sprite
     * Puede definirse como arrastrable para permitir arrastrar el
componente usando el ratón
     * @author Alex
     */
    public class Component extends EventDispatcher
    {
        /**
         * Indica que se ha movido
         */
        public static const MOVED:String = "MOVED";

        /** El Sprite que envuelve el componente */
        private var _sprite:Sprite;

        public function Component(sprite:Sprite=null,
className:String=null)
        {
            if (sprite) {
                _sprite = sprite;
            } else if (className) {
                _sprite = AssetsLibrary.getSprite(className);
            } else {
                _sprite = new Sprite();
            }

            _sprite.addEventListener(MouseEvent.CLICK, onSpriteClick);

            addChildren();
        }

        /**
         * Cuando se ha hecho click sobre el Sprite del componente
         * @param   e evento de ratón
         */
        private function onSpriteClick(e:MouseEvent):void
        {
            dispatchEvent(e);
        }

        protected function addChildren():void
        {
            //TO OVERRIDE
        }

        public function get parent():DisplayObjectContainer
        {
            return _sprite.parent;
        }

        public function set parent(value:DisplayObjectContainer):void
        {
            if (value) {
                value.addChild(_sprite);
            } else {
                if(_sprite.parent){
                    _sprite.parent.removeChild(_sprite);
                }
```

```
        }
    }

    public function get x():Number
    {
        return _sprite.x;
    }

    public function set x(value:Number):void
    {
        _sprite.x = value;
        dispatchEvent(new Event(MOVED));
    }

    public function get y():Number
    {
        return _sprite.y;
    }

    public function set y(value:Number):void
    {
        _sprite.y = value;
        dispatchEvent(new Event(MOVED));
    }

    public function get width():Number
    {
        return _sprite.width;
    }

    public function set width(value:Number):void
    {
        _sprite.width = value;
    }

    public function get height():Number
    {
        return _sprite.height;
    }

    public function set height(value:Number):void
    {
        _sprite.height = value;
    }

    public function get sprite():Sprite
    {
        return _sprite;
    }

    public function get mc():MovieClip
    {
        if (_sprite && _sprite is MovieClip) {
            return _sprite as MovieClip;
        } else {
            return null;
        }
    }

    /**
```

```
         * Le da o le quita la capacidad de ser arrastrado con el
ratón
         */
        public function set draggable(value:Boolean):void
        {
            if (value) {
                if (!sprite.hasEventListener(MouseEvent.MOUSE_DOWN)) {
                    sprite.addEventListener(MouseEvent.MOUSE_DOWN,
onMouseDown);
                }
                if (!sprite.hasEventListener(MouseEvent.MOUSE_UP)) {
                    sprite.addEventListener(MouseEvent.MOUSE_UP,
onMouseUp);
                }
                if (!sprite.hasEventListener(MouseEvent.MOUSE_OVER)) {
                    sprite.addEventListener(MouseEvent.MOUSE_OVER,
onMouseOver);
                }
                if (!sprite.hasEventListener(MouseEvent.MOUSE_OUT)) {
                    sprite.addEventListener(MouseEvent.MOUSE_OUT,
onMouseOut);
                }
            } else {
                sprite.removeEventListener(MouseEvent.MOUSE_DOWN,
onMouseDown);
                sprite.removeEventListener(MouseEvent.MOUSE_UP,
onMouseUp);
                sprite.removeEventListener(MouseEvent.MOUSE_OVER,
onMouseOver);
                sprite.removeEventListener(MouseEvent.MOUSE_OUT,
onMouseOut);
            }
        }

        /**
         * Cuando el ratón deja de estar encima del Sprite del
componente
         * @param   e evento de ratón
         */
        private function onMouseOut(e:MouseEvent):void
        {
            //usamos el cursor automático
            Mouse.cursor = MouseCursor.AUTO;
        }

        /**
         * Cuando el ratón pasa a estar encima del Sprite del
componente
         * @param   e   evento del ratón
         */
        private function onMouseOver(e:MouseEvent):void
        {
            //usamos el cursor de mano
            Mouse.cursor = MouseCursor.HAND;
        }

        /**
         * Cuando se aprieta el botón del ratón estando sobre el
Sprite del componente
         * @param   e   evento del ratón
```

```
         */
        private function onMouseDown(e:MouseEvent):void
        {
            //atendemos el movimiento del ratón
            sprite.addEventListener(MouseEvent.MOUSE_MOVE,
onMouseMove);
            //lo ponemos encima de todo sus hermanos
            sprite.parent.addChild(sprite);
            //y permitimos que se arrastre
            sprite.startDrag();
        }

        /**
         * Al moverse el ratón sobre el Sprite del componente
         * @param   e evento de ratón
         */
        private function onMouseMove(e:MouseEvent):void
        {
            dispatchEvent(new Event(MOVED));
        }

        /**
         * Cuando se deja de apretar el botón del ratón estando sobre
el Sprite del componente
         * @param   e evento del ratón
         */
        private function onMouseUp(e:MouseEvent):void
        {
            //detenemos el arrastre
            sprite.stopDrag();
            sprite.removeEventListener(MouseEvent.MOUSE_MOVE,
onMouseMove);
        }

    }
```

Estas clases describen los componentes básicos de las Redes de Petri: lugares, transiciones y arcos. Nótese que es indiferente añadir a una transición t un lugar p de entrada, como añadir al lugar p una transición t de salida.

Como se puede ver, la lógica de los arcos la gestiona el "ArcManager", mientras que la clase "ArrowedArc" sólo es el componente visual de la flecha que va de un punto a otro.

Por último se incluyen las clases de lugares y transiciones especiales, que corresponden a las partes indestructibles de los puzzles.

```
package game.components
{
    import core.Component;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import flash.events.TimerEvent;
    import flash.text.TextField;
    import flash.utils.Timer;
```

```
/**
 * Lugar que puede:
     * contener marcas
     * tener transiciones de entrada
     * tener transiciones de salida
 *
 * @author Alex
 */
public class Place extends Component
{
    public static const MARKS_CHANGED:String = "MARKS_CHANGED";
    public static const MARK_ADDED:String = "MARK_ADDED";
    public static const MARK_REMOVED:String = "MARK_REMOVED";

    /**
     * Número de marcas contenidas
     */
    private var _numMarks:int;

    /**
     * Transiciones de entrada
     */
    private var inTransitions:Vector.<Transition>;

    /**
     * Transiciones de salida
     */
    private var outTransitions:Vector.<Transition>;

    public function Place(sprite:Sprite=null)
    {
        _numMarks = 0;
        inTransitions = new Vector.<Transition>;
        outTransitions = new Vector.<Transition>;

        super(sprite, "place");
    }

    override protected function addChildren():void
    {
        if (mc) {
            mc.gotoAndStop("0m");
        }
    }

    /**
     * Añade una marca
     */
    public function addMark():void
    {
        numMarks++;
        dispatchEvent(new Event(MARK_ADDED));
    }

    /**
     * Quita una marca, si la hay
     */
    public function removeMark():void
    {
        if(numMarks > 0){
```

```
                numMarks--;
                dispatchEvent(new Event(MARK_REMOVED));
            }
        }

        /**
         * Añade una transición de entrada, si no está ya incluída
         * @param   transition la transición a añadir
         */
        public function addInTransition(transition:Transition):void
        {
            var index:int = inTransitions.indexOf(transition);
            if (index == -1) {
                inTransitions.push(transition);
                transition.addOutPlace(this);
            }
        }

        /**
         * Quita una transición de entrada, si está
         * @param   transition  la transición a quitar
         */
        public function removeInTransition(transition:Transition):void
        {
            var index:int = inTransitions.indexOf(transition);
            if (index != -1) {
                inTransitions.splice(index, 1);
                transition.removeOutPlace(this);
            }
        }

        /**
         * Añade una transición de salida, si no está ya incluída
         * @param   transition la transición a añadir
         */
        public function addOutTransition(transition:Transition):void
        {
            var index:int = outTransitions.indexOf(transition);
            if (index == -1) {
                outTransitions.push(transition);
                transition.addInPlace(this);
            }
        }

        /**
         * Quita una transición de salida, si está
         * @param   transition  la transición a quitar
         */
        public function
removeOutTransition(transition:Transition):void
        {
            var index:int = outTransitions.indexOf(transition);
            if (index != -1) {
                outTransitions.splice(index, 1);
                transition.removeInPlace(this);
            }
        }

        public function get numMarks():int
        {
```

```
            return _numMarks;
        }

        public function set numMarks(value:int):void
        {
            _numMarks = value;

            dispatchEvent(new Event(MARKS_CHANGED));

            if (_numMarks <= 5) {
                mc.gotoAndStop(_numMarks + "m");
            } else {
                mc.gotoAndStop("nm");
                var numMarksTxt:TextField = mc["num_marks_txt"];
                if (numMarksTxt) {
                    numMarksTxt.text = _numMarks.toString();
                }
            }
        }

    }

}
```

```
package game.components
{
    import core.Component;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import flash.events.TimerEvent;
    import flash.text.TextField;
    import flash.utils.Timer;
    /**
     * Transición que puede tener:
         * lugares de entrada
         * lugares de salida
     *  Puede estar habilitada o no
     *  Tiene prioridad, se utiliza a la hora de evaluarse para
disparse antes que otras, con las que comparte lugares de entrada
     * @author Alex
     */
    public class Transition extends Component
    {
        /**
         * Tiempo que tras haberse disparado la transición se
comprueba su estado para saber si está activada o desactivada
         */
        private const TRIGGER_ANIMATION_TIME:int = 400;

        /**
         * Lugares de entrada
         */
        private var inPlaces:Vector.<Place>;

        /**
         * Lugares de salida
         */
        private var outPlaces:Vector.<Place>;

        /**
```

```
         * Si está habilitada
         */
        private var _enabled:Boolean = true;

        /**
         * Si está siendo disparada
         */
        private var _triggered:Boolean;

        /**
         * Prioridad de la transición. A menor número, mayor prioridad
         */
        private var _priority:int;

        /**
         * Etiqueta de la prioridad
         */
        protected var _priorityTextField:TextField;

        /**
         * Timer de espera durante la animación de disparo para
recalcular el estado de la transición
         */
        protected var triggerTimer:Timer;

        public function Transition(sprite:Sprite=null)
        {
            inPlaces = new Vector.<Place>;
            outPlaces = new Vector.<Place>;

            super(sprite, "transition");
        }

        override protected function addChildren():void
        {
            if (mc) {
                mc.gotoAndStop("enabled");
            }

            if (sprite && sprite["priority_txt"] &&
sprite["priority_txt"] is TextField) {
                _priorityTextField =
TextField(sprite["priority_txt"]);
            }

            priority = 0;
        }

        /**
         * Añade un lugar de entrada, si no está ya incluído
         * @param   place   el lugar a añadir
         */
        public function addInPlace(place:Place):void
        {
            var index:int = inPlaces.indexOf(place);
            if (index == -1) {
                inPlaces.push(place);
                place.addEventListener(Place.MARKS_CHANGED,
checkEnabled);
                checkEnabled();
```

```
                place.addOutTransition(this);
            }
        }

        /**
         * Quita un lugar de entrada, si lo hay
         * @param   place   el lugar a quitar
         */
        public function removeInPlace(place:Place):void
        {
            var index:int = inPlaces.indexOf(place);
            if (index != -1) {
                inPlaces.splice(index, 1);
                place.removeEventListener(Place.MARKS_CHANGED,
checkEnabled);
                checkEnabled();
                place.removeOutTransition(this);
            }
        }

        /**
         * Comprueba si la transición está habilitada y actualiza su
estado
         * @param   e   evento de cambio de marcas de un lugar de
entrada
         */
        private function checkEnabled(e:Event=null):void
        {
            for (var i:int = 0; i < inPlaces.length; i++)
            {
                if (inPlaces[i].numMarks <= 0) {
                    enabled = false;
                    return;
                }
            }

            enabled = true;
        }

        /**
         * Añade un lugar de salida, si no está ya incluído
         * @param   place   el lugar a añadir
         */
        public function addOutPlace(place:Place):void
        {
            var index:int = outPlaces.indexOf(place);
            if (index == -1) {
                outPlaces.push(place);
                place.addInTransition(this);
            }
        }

        /**
         * Quita un lugar de salida, si lo hay
         * @param   place   el lugar a quitar
         */
        public function removeOutPlace(place:Place):void
        {
            var index:int = outPlaces.indexOf(place);
            if (index != -1) {
```

```
                outPlaces.splice(index, 1);
                place.removeInTransition(this);
            }
        }

        /**
         * Comprueba si está habilitada la transición. En caso
afirmativo se dispara
         */
        public function checkTrigger():void
        {
            if (enabled) {
                trigger();
            }
        }

        /**
         * Dispara la transición.
         * Quita una marca de todos sus lugares de entrada y pone una
marca en todos sus lugares de salida
         */
        public function trigger():void
        {
            triggered = true;

            if (triggerTimer) {
                triggerTimer.stop();
            }

            triggerTimer = new Timer(TRIGGER_ANIMATION_TIME, 1);
            triggerTimer.addEventListener(TimerEvent.TIMER,
onTimerEnds);
            triggerTimer.start();

            //quitamos una marca en cada uno de los lugares de entrada
            for (var i:int = 0; i < inPlaces.length; i++)
            {
                inPlaces[i].removeMark();
            }

            //añadimos una marca en cada uno de los lugares de salida
            for (var j:int = 0; j < outPlaces.length; j++)
            {
                outPlaces[j].addMark();
            }
        }

        /**
         * Cuando termina el timer de espera de la animación de
disparo
         * @param   e   evento de timer
         */
        protected function onTimerEnds(e:TimerEvent):void
        {
            //dejamos de mostrar que la transición ha sido disparada
            triggered = false;
        }

        public function get enabled():Boolean
        {
```

```
            return _enabled;
        }

        public function set enabled(value:Boolean):void
        {
            _enabled = value;

            if (!triggered) {

                if (enabled) {
                    if (mc) {
                        mc.gotoAndStop("enabled");
                    }
                } else {
                    if (mc) {
                        mc.gotoAndStop("disabled");
                    }
                }
            }
        }

        public function get triggered():Boolean
        {
            return _triggered;
        }

        public function set triggered(value:Boolean):void
        {
            _triggered = value;

            if (triggered) {
                if (mc) {
                    mc.gotoAndStop("triggered");
                }
            } else {
                checkEnabled();
            }
        }

        public function get priority():int
        {
            return _priority;
        }

        public function set priority(value:int):void
        {
            _priority = value;

            if (_priorityTextField) {
                //mostramos la prioridad + 1, para empezar en 1 en
lugar de 0
                //así la prioridad indica el orden de disparo
                _priorityTextField.text = (_priority +1).toString();
            }
        }

    }

}
```

```
package game.components
{
    import core.Component;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import flash.geom.Point;
    /**
     * Arco dirigido
     * @author Alex
     */
    public class ArrowedArc extends Component
    {
        public static const ARC_CLICKED:String = "ARC_CLICKED";

        /**
         * Línea del arco
         */
        private var arc:Component;

        /**
         * Punta de flecha
         */
        private var arrowEnd:Component;

        /**
         * Punto de comienzo
         */
        private var _startPoint:Point;

        /**
         * Punto final al que señala el arco
         */
        private var _endPoint:Point;

        /**
         * El ancho de la línea
         */
        private var lineThickness:Number = 4; //2;

        /**
         * Escala de la punta
         */
        private var arrowEndScale:Number = 1.5;

        /**
         * Si es borrable por el usuario
         */
        private var _isUserErasable:Boolean = true;


        public function ArrowedArc()
        {
            super();
        }

        public function get isUserErasable():Boolean
        {
            return _isUserErasable;
        }
```

```
        public function set isUserErasable(value:Boolean):void
        {
            _isUserErasable = value;
        }

        override protected function addChildren():void
        {
            //efectos de "over" para las líneas
            /*sprite.addEventListener(MouseEvent.MOUSE_OVER,
function():void {
                lineThickness = 6;
                arrowEndScale = 2;
                drawArrow();
            });
            sprite.addEventListener(MouseEvent.MOUSE_OUT,
function():void {
                lineThickness = 2;
                arrowEndScale = 1.5;
                drawArrow();
            });
            */
            sprite.addEventListener(MouseEvent.CLICK, function():void
{
                dispatchEvent(new Event(ARC_CLICKED));
            });

            arc = new Component();
            arrowEnd = new Component(null, "arrow_end");
        }

        /**
         * Comprueba si tiene punto de inicio de fin
         * En caso afirmativo dibuja el arco
         */
        private function checkPoints():void
        {
            if (_startPoint && endPoint) {
                drawArrow();
            }
        }

        /**
         * Dibujamos la flecha del punto de salida al de llegada
         */
        private function drawArrow():void
        {
            //calculamos los catetos, como las diferencias en los ejes
X e Y
            var catA:Number = endPoint.y - startPoint.y;
            var catB:Number = endPoint.x - startPoint.x;

            //calculamos el ángulo que forma la recta que une los
puntos
            var radianAngle:Number = Math.atan2(catA, catB);
            var angle:Number = 180 * radianAngle / Math.PI;
            //trace("angle -> " + angle);

            //limpiamos los gráficos del arco
            arc.sprite.graphics.clear();
            //creamos el arco entre los dos puntos
```

```
            //arc.sprite.graphics.lineStyle(5, 0xFF0000, 0.5);
            arc.sprite.graphics.lineStyle(lineThickness, 0, 1);
            arc.sprite.graphics.moveTo(startPoint.x, startPoint.y);
            arc.sprite.graphics.lineTo(endPoint.x, endPoint.y);

            //ponemos la punta de la flecha con el ángulo calculado
            arrowEnd.sprite.rotation = angle;
            //con su escala
            arrowEnd.sprite.scaleX = arrowEndScale;
            arrowEnd.sprite.scaleY = arrowEndScale;

            //en el punto final
            arrowEnd.x = endPoint.x;
            arrowEnd.y = endPoint.y;

            //añadimos las partes de la flecha a escena
            arc.parent = sprite;
            arrowEnd.parent = sprite;
        }

        public function set arrowEndVisible(value:Boolean):void
        {
            arrowEnd.sprite.visible = value;
        }

        public function get startPoint():Point
        {
            return _startPoint;
        }

        public function set startPoint(value:Point):void
        {
            _startPoint = value;

            checkPoints();
        }

        public function get endPoint():Point
        {
            return _endPoint;
        }

        public function set endPoint(value:Point):void
        {
            _endPoint = value;

            checkPoints();
        }

    }

}
```

```
package game.managers
{
    import core.Component;
    import flash.events.Event;
    import flash.geom.Point;
    import game.components.ArrowedArc;
    import game.components.Place;
    import game.components.Transition;
```

```
    /**
     * Crea y gestiona los arcos entre lugares y transiciones
     * Atiende al movimiento de los componentes con arcos redibujando
adecuadamente los arcos involucrados
     * @author Alex
     */
    public class ArcManager
    {
        //Estas 3 listas estarán conectadas,
        //cada componente del primero estará conectada con el
componente del tercero en la misma posición, mediante el arco de la
misma posición del segundo
        /** Lista de componentes de los que salen arcos */
        private var startComponents:Vector.<Component>;
        /** Lista de arcos */
        private var arcs:Vector.<ArrowedArc>;
        /** Lista de componentes a los que llegan los arcos */
        private var endComponents:Vector.<Component>;

        public function ArcManager()
        {
            startComponents = new Vector.<Component>;
            arcs = new Vector.<ArrowedArc>;
            endComponents = new Vector.<Component>;
        }

        /**
         * Añade un arco entre ambos componentes, siempre y cuando uno
sea un lugar y el otro una transición, y no exista ya ese arco
         * @param   startComponent  componentes del que sale el arco
         * @param   endComponent    componente al que llega el arco
         */
        public function addArc(startComponent:Component,
endComponent:Component):ArrowedArc
        {
            var arc:ArrowedArc;

            if(!existsArc(startComponent, endComponent)){
                if (startComponent is Place && endComponent is
Transition) {

Place(startComponent).addOutTransition(Transition(endComponent));
                    arc = addArcBetweenComponents(startComponent,
endComponent);
                } else if (startComponent is Transition &&
endComponent is Place) {

Transition(startComponent).addOutPlace(Place(endComponent));
                    arc = addArcBetweenComponents(startComponent,
endComponent);
                }
            }

            return arc;
        }

        /**
         * Quita un arco entre ambos componentes, siempre y cuando
exista ya ese arco
         * @param   startComponent  componentes del que sale el arco
```

```
         * @param   endComponent    componente al que llega el arco
         */
        public function removeArc(startComponent:Component,
endComponent:Component):void
        {
            for (var i:int = 0; i < startComponents.length; i++)
            {
                if (startComponents[i] == startComponent &&
endComponents[i] == endComponent) {
                    //quitamos el arco de la escena
                    arcs[i].parent = null;

                    //quitamos la unión entre componentes
                    if (startComponent is Place) {

Place(startComponent).removeOutTransition(Transition(endComponent));
                    } else if (startComponent is Transition) {

Transition(startComponent).removeOutPlace(Place(endComponent));
                    }

                    //quitamos las referencias al arco de la lista
                    startComponents.splice(i, 1);
                    arcs.splice(i, 1);
                    endComponents.splice(i, 1);
                }
            }

            //si ya no quedan arcos en alguno de los dos componentes
            //dejamos de atender su movimiento para redibujar arcos
            if (startComponents.indexOf(startComponent) == -1 &&
endComponents.indexOf(startComponent) == -1) {
                startComponent.removeEventListener(Component.MOVED,
redrawArcs);
            }
            if (startComponents.indexOf(endComponent) == -1 &&
endComponents.indexOf(endComponent) == -1) {
                endComponent.removeEventListener(Component.MOVED,
redrawArcs);
            }
        }

        /**
         * Quita un arco entre dos componentes, siempre y cuando
exista ese arco
         * @param   arc el arco a eliminar
         */
        public function removeArcDirectly(arc:ArrowedArc):void
        {
            //buscamos el arco y sus extremos, y lo borramos
            for (var i:int = 0; i < arcs.length; i++)
            {
                if (arcs[i] == arc) {
                    removeArc(startComponents[i], endComponents[i]);
                }
            }
        }

        /**
         * Quita todos los arcos del componente
```

```
         * @param   component   el componente al que quitar todos los
arcos
         */
        public function removeAllArcsFrom(component:Component):void
        {
            //recorremos las listas alrevés para evitar saltarnos una
posición al ir eliminando posiciones
            for (var i:int = startComponents.length-1; i >= 0; i--)
            {
                if (startComponents[i] == component) {
                    removeArc(component, endComponents[i]);
                }
            }

            for (var j:int = endComponents.length-1; j >= 0; j--)
            {
                if (endComponents[j] == component) {
                    removeArc(startComponents[j], component);
                }
            }
        }

        /**
         * Comprueba si ya existe ese mismo arco
         * @param   startComponent  el primer componente
         * @param   endComponent    el segundo componente
         * @return  si ya existe el mismo arco
         */
        private function existsArc(startComponent:Component,
endComponent:Component):Boolean
        {
            for (var i:int = 0; i < startComponents.length; i++)
            {
                if (startComponents[i] == startComponent &&
endComponents[i] == endComponent) {
                    return true;
                }
            }

            return false;
        }

        /**
         * Añade un arco gráfico entre ambos componentes, y atiende
por si se mueven para redibujarlo
         * @param   startComponent  componente del que sale el arco
         * @param   endComponent    componente al que llega el arco
         */
        private function
addArcBetweenComponents(startComponent:Component,
endComponent:Component):ArrowedArc
        {
            //creamos el arco entre ambos componentes
            var arc:ArrowedArc = new ArrowedArc();

            //en la misma escena que el componente del que sale el
arco,
            //debajo del todo menos del fondo y el puzzle(decorado),
            //para poder interactuar con todos los demás componentes
            startComponent.parent.addChildAt(arc.sprite, 2);
```

```
            //atendemos por si se mueven los componentes para
redibujar sus arcos
            if (!startComponent.hasEventListener(Component.MOVED)) {
                startComponent.addEventListener(Component.MOVED,
redrawArcs);
            }
            if (!endComponent.hasEventListener(Component.MOVED)) {
                endComponent.addEventListener(Component.MOVED,
redrawArcs);
            }

            startComponents.push(startComponent);
            arcs.push(arc);
            endComponents.push(endComponent);

            drawArc(arc);

            return arc;
        }

        /**
         * Redibuja todos los arcos del componente, tanto los de
entrada como los de salida
         * @param   e
         */
        private function redrawArcs(e:Event):void
        {
            if (e && e.target is Component) {
                var component:Component = e.target as Component;

                for (var i:int = 0; i < startComponents.length; i++)
                {
                    if (startComponents[i] == component) {
                        drawArc(arcs[i]);
                    }
                    if (endComponents[i] == component) {
                        drawArc(arcs[i]);
                    }
                }
            }
        }

        /**
         * Dibuja un arco entre dos componentes. Si ya existía se
redibuja
         * @param   arc el arco a dibujar
         */
        private function drawArc(arc:ArrowedArc):void
        {
            //buscamos la posición del arco
            var position:int;
            for (var i:int = 0; i < arcs.length; i++)
            {
                if (arcs[i] == arc) {
                    position = i;
                }
            }
            //y con ella, a los componentes involucrados
            var startComponent:Component = startComponents[position];
```

```
            var endComponent:Component = endComponents[position];

            //dibujamos el arco de forma adecuada, con la distancia
más corta entre componentes
            drawShortestArc(startComponent, arc, endComponent);
        }

        /**
         * Dibuja el arco, con la distancia más corta entre Sprites
         * @param   startComponent  componente de salida
         * @param   arc arco
         * @param   endComponent    componente de entrada
         */
        private function drawShortestArc(startComponent:Component,
arc:ArrowedArc, endComponent:Component):void
        {
            //Escogemos los puntos susceptibles de ser puntos de los
extremos del arco
            var startPoints:Vector.<Point> = new Vector.<Point>;
            startPoints.push( up(startComponent),
left(startComponent), right(startComponent), down(startComponent) );
            var endPoints:Vector.<Point> = new Vector.<Point>;
            endPoints.push( up(endComponent), left(endComponent),
right(endComponent), down(endComponent) );

            var startPoint:Point;
            var endPoint:Point;

            var shortestDistance:Number = Number.MAX_VALUE;
            var auxDistance:Number;

            //nos quedamos los puntos cuya distancia sea mínima
            for (var i:int = 0; i < startPoints.length; i++)
            {
                for (var j:int = 0; j < endPoints.length; j++)
                {
                    auxDistance = Point.distance(startPoints[i],
endPoints[j]);
                    if (auxDistance < shortestDistance) {
                        shortestDistance = auxDistance;
                        startPoint = startPoints[i];
                        endPoint = endPoints[j];
                    }
                }
            }

            //trazamos el arco desde los puntos más próximos
            arc.startPoint = startPoint;
            arc.endPoint = endPoint;

        }

        /**
         * Devuelve el punto central superior del Componente
         * @param   component el componente
         * @return  el punto central superior
         */
        private function up(component:Component):Point
        {
            return new Point(component.x + component.width / 2,
```

```
component.y);
        }

        /**
         * Devuelve el punto central izquierdo del Componente
         * @param   component el componente
         * @return  el punto central izquierdo
         */
        private function left(component:Component):Point
        {
            return new Point(component.x, component.y +
component.height/2);
        }

        /**
         * Devuelve el punto central derecho del Componente
         * @param   component el componente
         * @return  el punto central derecho
         */
        private function right(component:Component):Point
        {
            return new Point(component.x + component.width,
component.y + component.height/2);
        }

        /**
         * Devuelve el punto central inferior del Componente
         * @param   component el componente
         * @return  el punto central inferior
         */
        private function down(component:Component):Point
        {
            return new Point(component.x + component.width/2,
component.y + component.height);
        }

    }

}
```

```
package game.components
{
    import flash.display.Sprite;
    /**
     * Lugar especial de los puzzles
     * @author Alex
     */
    public class SpecialPlace extends Place
    {

        public function SpecialPlace(sprite:Sprite=null)
        {
            super(sprite);
        }

        override public function set draggable(value:Boolean):void
        {
            //no es arrastrable
        }

    }
```

```
}
```

```
package game.components
{
    import flash.display.Sprite;
    import flash.events.TimerEvent;
    import flash.utils.Timer;
    /**
     * Transición especial de los puzzles
     * @author Alex
     */
    public class SpecialTransition extends Transition
    {
        public function SpecialTransition(sprite:Sprite=null)
        {
            super(sprite);
        }

        override protected function addChildren():void
        {
            super.addChildren();
            //no tienen prioridad, así que quitamos la etiqueta de la
prioridad
            if (_priorityTextField) {
                _priorityTextField.visible = false;
            }
        }

        override public function set draggable(value:Boolean):void
        {
            //no es arrastrable
        }

    }

}
```

A continuación se adjuntan las clases de los componentes visuales que forman parte de la interfaz de usuario, tanto de botones y barras de herramientas como de popups emergentes.

```
package game.components
{
    import core.Component;
    import flash.events.MouseEvent;
    /**
     * Mensaje de fallo de un puzzle
     * @author Alex
     */
    public class FailMessage extends Component
    {

        public function FailMessage(levelNum:int, failNum:int)
        {
            super(null, "instructions_" + levelNum + "_" + failNum);
        }

        override protected function addChildren():void
        {
            sprite.addEventListener(MouseEvent.CLICK, onClick);
        }

        private function onClick(e:MouseEvent):void
        {
            sprite.removeEventListener(MouseEvent.CLICK, onClick);
            parent = null;
        }

    }

}
```

```
package game.components
{
    import core.Component;
    import flash.events.MouseEvent;
    /**
     * Instrucciones de un nivel
     * Al hacerles click desaparecen
     * @author Alex
     */
    public class Instructions extends Component
    {

        public function Instructions(levelNum:int)
        {
            super(null, "instructions_" + levelNum);
        }

        override protected function addChildren():void
        {
            addEventListener(MouseEvent.CLICK, onClick);
        }

        private function onClick(e:MouseEvent):void
        {
            removeEventListener(MouseEvent.CLICK, onClick);
```

```
            parent = null;
        }

    }

}
```

```
package game.events
{
    import flash.events.Event;
    /**
     * Eventos del panel de información de nivel
     * @author Alex
     */
    public class LevelInfoEvent extends Event
    {
        /**
         * Se desea iniciar el nivel
         */
        public static const LEVEL_INIT:String = "LEVEL_INIT";

        /**
         * El número identificador del nivel
         */
        private var _levelNum:int;

        public function LevelInfoEvent(type:String, levelNum:int)
        {
            _levelNum = levelNum;
            super(type);
        }

        public function get levelNum():int
        {
            return _levelNum;
        }

        public function set levelNum(value:int):void
        {
            _levelNum = value;
        }

    }

}
```

```
package game.components
{
    import core.Component;
    import flash.display.MovieClip;
    import flash.display.SimpleButton;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.text.TextField;
    import game.events.LevelInfoEvent;
    /**
     * Muestra la información del nivel (si está superado o no, y en
caso de estarlo, con cuántas estrellas)
     * Además da la posibilidad de hacerle click para iniciar ese
nivel
     * @author Alex
     */
```

```
    public class LevelInfo extends Component
    {
        /**
         * Número del nivel
         */
        private var _levelNum:int;

        /**
         * Si está bloqueado
         */
        private var _locked:Boolean;

        /**
         * Si está completado o no
         */
        private var _completed:Boolean;

        /**
         * Número de estrellas del nivel superado
         */
        private var _numStars:int;

        /**
         * Botón para iniciar el nivel
         */
        private var levelButton:SimpleButton;


        public function LevelInfo(sprite:Sprite)
        {
            super(sprite);
        }

        override protected function addChildren():void
        {
            if (sprite && sprite["level_btn"] && sprite["level_btn"]
is SimpleButton) {
                levelButton = SimpleButton(sprite["level_btn"]);
                levelButton.addEventListener(MouseEvent.CLICK,
onLevelBtnClick);
            }

            if (sprite && sprite["levelnum_txt"] &&
sprite["levelnum_txt"] is TextField) {
                TextField(sprite["levelnum_txt"]).mouseEnabled =
false;
            }
        }

        /**
         * Al hacer click en el botón del nivel
         * @param   e   evento de ratón
         */
        private function onLevelBtnClick(e:MouseEvent):void
        {
            if(!locked){
                dispatchEvent(new
LevelInfoEvent(LevelInfoEvent.LEVEL_INIT, levelNum));
            }
        }
```

```
        public function get levelNum():int
        {
            return _levelNum;
        }

        public function set levelNum(value:int):void
        {
            _levelNum = value;

            if (sprite && sprite["levelnum_txt"] &&
sprite["levelnum_txt"] is TextField) {
                TextField(sprite["levelnum_txt"]).text =
_levelNum.toString();
            }
        }

        public function get completed():Boolean
        {
            return _completed;
        }

        public function set completed(value:Boolean):void
        {
            _completed = value;

            if (sprite && sprite["solved_tick"] &&
sprite["solved_tick"] is MovieClip) {
                MovieClip(sprite["solved_tick"]).visible = value;
            }
        }

        public function get numStars():int
        {
            return _numStars;
        }

        public function set numStars(value:int):void
        {
            _numStars = value;
            if (sprite && sprite["level_star_1"] &&
sprite["level_star_1"] is MovieClip && sprite["level_star_2"] &&
sprite["level_star_2"] is MovieClip && sprite["level_star_3"] &&
sprite["level_star_3"] is MovieClip) {
                //mostramos las estrellas conseguidas, y dejamos medio
transparentes las que falten por conseguir
                MovieClip(sprite["level_star_1"]).alpha = 0.3;
                MovieClip(sprite["level_star_2"]).alpha = 0.3;
                MovieClip(sprite["level_star_3"]).alpha = 0.3;

                if (_numStars > 0) {
                    MovieClip(sprite["level_star_1"]).alpha = 1;
                }

                if (_numStars > 1) {
                    MovieClip(sprite["level_star_2"]).alpha = 1;
                }

                if (_numStars > 2) {
                    MovieClip(sprite["level_star_3"]).alpha = 1;
```

```
                }
            }
        }

        public function get locked():Boolean
        {
            return _locked;
        }

        public function set locked(value:Boolean):void
        {
            _locked = value;

            if (_locked) {
                levelButton.enabled = false;
                levelButton.alpha = 0.3;
            }
        }

    }

}
```

```
package game.events
{
    import flash.events.Event;
    /**
     * Eventos de la barra de herramientas
     * @author Alex
     */
    public class ToolsBarEvent extends Event
    {
        /**
         * Se pide utilizar la herramienta CURSOR
         */
        public static const CURSOR:String = "CURSOR";

        /**
         * Se pide utilizar la herramienta ADD_PLACE
         */
        public static const ADD_PLACE:String = "ADD_PLACE";

        /**
         * Se pide utilizar la herramienta ADD_TRANSITION
         */
        public static const ADD_TRANSITION:String = "ADD_TRANSITION";

        /**
         * Se pide utilizar la herramienta ADD_ARC
         */
        public static const ADD_ARC:String = "ADD_ARC";

        /**
         * Se pide utilizar la herramienta ERASER
         */
        public static const ERASER:String = "ERASER";

        /**
         * Se pide utilizar la herramienta ADD_MARK
         */
        public static const ADD_MARK:String = "ADD_MARK";
```

```
        /**
         * Se pide utilizar la herramienta REMOVE_MARK
         */
        public static const REMOVE_MARK:String = "REMOVE_MARK";

        /**
         * Se pide ejecutar la prueba del puzzle actual
         */
        public static const PLAY:String = "PLAY";

        /**
         * Se pide detener la ejecución actual
         */
        public static const STOP:String = "STOP";

        public function ToolsBarEvent(type:String)
        {
            super(type);
        }
    }

}
```

```
package game.components
{
    import core.Component;
    import flash.display.SimpleButton;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import game.events.ToolsBarEvent;
    /**
     * Barra de herramientas para editar una Red de Petri
     * @author Alex
     */
    public class ToolsBar extends Component
    {

        /**
         * Botón para volver al cursor normal
         */
        private var _cursorBtn:SimpleButton;

        /**
         * Botón para añadir lugares
         */
        private var _addPlaceBtn:SimpleButton;

        /**
         * Botón para poner transiciones
         */
        private var _addTransitionBtn:SimpleButton;

        /**
         * Botón para poner arcos
         */
        private var _addArcBtn:SimpleButton;

        /**
         * Botón para borrar componentes (goma de borrar)
         */
```

```
        private var _eraserBtn:SimpleButton;

        /**
         * Botón para añadir marcas en los lugares
         */
        private var _addMarkBtn:SimpleButton;

        /**
         * Botón para quitar marcas en los lugares
         */
        private var _removeMarkBtn:SimpleButton;

        /**
         * Botón de play. Para disparar las transiciones
         */
        private var _playBtn:SimpleButton;

        public function ToolsBar(sprite:Sprite=null,
className:String=null)
        {
            super(sprite, className);
        }

        override protected function addChildren():void
        {
            //recogemos los botones de la barra de herramientas

            _cursorBtn = sprite["cursor_btn"];
            if (_cursorBtn) {
                _cursorBtn.addEventListener(MouseEvent.CLICK,
onCursorBtnClick);
            }

            _addPlaceBtn =  sprite["add_place_btn"];
            if (_addPlaceBtn) {
                _addPlaceBtn.addEventListener(MouseEvent.CLICK,
onAddPlaceBtnClick);
            }

            _addTransitionBtn = sprite["add_transition_btn"];
            if (_addTransitionBtn) {
                _addTransitionBtn.addEventListener(MouseEvent.CLICK,
onAddTransitionBtnClick);
            }

            _addArcBtn = sprite["add_arc_btn"];
            if (_addArcBtn) {
                _addArcBtn.addEventListener(MouseEvent.CLICK,
onAddArcBtnClick);
            }

            _eraserBtn = sprite["eraser_btn"];
            if (_eraserBtn) {
                _eraserBtn.addEventListener(MouseEvent.CLICK,
onEraserBtnClick);
            }

            _addMarkBtn = sprite["add_mark_btn"];
            if (_addMarkBtn) {
                _addMarkBtn.addEventListener(MouseEvent.CLICK,
```

```
onAddMarkBtnClick);
            }

            _removeMarkBtn = sprite["remove_mark_btn"];
            if (_removeMarkBtn) {
                _removeMarkBtn.addEventListener(MouseEvent.CLICK,
onRemoveMarkBtnClick);
            }

            _playBtn = sprite["play_btn"];
            if (_playBtn) {
                _playBtn.addEventListener(MouseEvent.CLICK,
onPlayBtnClick);
            }
        }

        /**
         * Al hacer click en el botón de PLAY
         * @param   e evento de ratón
         */
        private function onPlayBtnClick(e:MouseEvent):void
        {
            dispatchEvent(new ToolsBarEvent(ToolsBarEvent.PLAY));
        }

        /**
         * Al hacer click en el botón de CURSOR
         * @param   e evento de ratón
         */
        private function onCursorBtnClick(e:MouseEvent):void
        {
            dispatchEvent(new ToolsBarEvent(ToolsBarEvent.CURSOR));
        }

        /**
         * Al hacer click en el botón de ADD_PLACE
         * @param   e evento de ratón
         */
        private function onAddPlaceBtnClick(e:MouseEvent):void
        {
            dispatchEvent(new ToolsBarEvent(ToolsBarEvent.ADD_PLACE));
        }

        /**
         * Al hacer click en el botón de ADD_TRANSITION
         * @param   e evento de ratón
         */
        private function onAddTransitionBtnClick(e:MouseEvent):void
        {
            dispatchEvent(new
ToolsBarEvent(ToolsBarEvent.ADD_TRANSITION));
        }

        /**
         * Al hacer click en el botón de ADD_ARC
         * @param   e evento de ratón
         */
        private function onAddArcBtnClick(e:MouseEvent):void
        {
            dispatchEvent(new ToolsBarEvent(ToolsBarEvent.ADD_ARC));
```

```
        }

        /**
         * Al hacer click en el botón de ERASER
         * @param   e evento de ratón
         */
        private function onEraserBtnClick(e:MouseEvent):void
        {
            dispatchEvent(new ToolsBarEvent(ToolsBarEvent.ERASER));
        }

        /**
         * Al hacer click en el botón de ADD_MARK
         * @param   e evento de ratón
         */
        private function onAddMarkBtnClick(e:MouseEvent):void
        {
            dispatchEvent(new ToolsBarEvent(ToolsBarEvent.ADD_MARK));
        }

        /**
         * Al hacer click en el botón de REMOVE_MARK
         * @param   e evento de ratón
         */
        private function onRemoveMarkBtnClick(e:MouseEvent):void
        {
            dispatchEvent(new
ToolsBarEvent(ToolsBarEvent.REMOVE_MARK));
        }

        public function get addPlaceBtn():SimpleButton
        {
            return _addPlaceBtn;
        }

        public function get addTransitionBtn():SimpleButton
        {
            return _addTransitionBtn;
        }

        public function get addArcBtn():SimpleButton
        {
            return _addArcBtn;
        }

        public function get eraserBtn():SimpleButton
        {
            return _eraserBtn;
        }

        public function get cursorBtn():SimpleButton
        {
            return _cursorBtn;
        }

    }

}
```

Las siguientes clases definen los puzzles definidos en el punto 4.4, así como la clase base de todos ellos.

```
package game.puzzles
{
    import core.Component;
    import flash.events.Event;
    import flash.events.TimerEvent;
    import flash.utils.Timer;
    import game.components.FailMessage;
    import game.managers.WorkbenchManager;
    /**
     * Clase abstracta con las cosas comunes de los puzzles
     * @author Alex
     */
    public class Puzzle extends Component
    {
        /**
         * Pide ejecutar al Red de Petri del usuario hasta que se
detenga
         */
        public static const PLAY_UNTIL_STOP:String =
"PLAY_UNTIL_STOP";

        /**
         * Informa de que el test del puzzle ha sido un éxito
         */
        public static const SUCCESS:String = "SUCCESS";

        /**
         * Informa de que el test del puzzle ha sido un fracaso
         */
        public static const FAIL:String = "FAIL";

        /**
         * Tiempo (en milisegundos) de retraso en pedir que se ejecute
la Red de Petri del usuario
         */
        private const PLAY_ULTIL_STOP_DELAY_TIME:int = 500;

        //el número de puzzle
        private var _puzzleNum:int;

        //el paso del test en el que estamos
        protected var testStep:int = 0;

        public function Puzzle(puzzleNum:int):void
        {
            _puzzleNum = puzzleNum;
            super(null, "puzzle_" + puzzleNum);
        }

        /**
         * Prueba el puzzle
         */
        public function testLevel():void
        {
            //primera etapa de test
            testStep = 0;
```

```
            continueTest();
        }

        /**
         * Prepara el puzzle en el Workbench a través del manager
         * @param   workbenchManager el manager del workbench donde se
quiere preparar el puzzle
         */
        public function
setPuzzleOnWorkbench(workbenchManager:WorkbenchManager):void
        {
            //TO OVERRIDE
        }

        /**
         * Realiza las diversas comprobaciones y acciones que
determinan si el puzzle está resuelto o no
         * Entre caso y caso, se pone la Red de Petri en marcha hasta
que para, y se sigue con el siguiente paso
         */
        public function continueTest():void
        {
            //TO OVERRIDE
        }

        /**
         * Se pide inciar la Red de Petri del usuario, pero con un
retraso
         */
        protected function delayedPlayUntilStop():void
        {
            var timer:Timer = new Timer(500, 1);
            timer.addEventListener(TimerEvent.TIMER, playUntilStop,
false, 0, true);
            timer.start();
        }

        /**
         * Se solicita que se ejecute la Red de Petri del usuario
         * @param   e evento de timer
         */
        protected function playUntilStop(e:Event=null):void
        {
            dispatchEvent(new Event(PLAY_UNTIL_STOP));
        }

        /**
         * El test del puzzle ha determinado que ha fallado
         * @param   failType    el tipo de fallo
         */
        protected function fail(failType:int):void
        {
            trace("fail " + testStep);

            var failMessage:FailMessage = new FailMessage(_puzzleNum,
failType);
            failMessage.parent = parent.parent;
            //workbench -> level

            //ponemos el mensaje de error como hijo del level,
```

```
centrado
            failMessage.x = (parent.width - failMessage.width) / 2;
            failMessage.y = (parent.height - failMessage.height) / 2;

            //con modal debajo del todo
            var modal:Component = new Component(null, "modal_bg");
            failMessage.sprite.addChildAt(modal.sprite, 0);
            modal.x = -failMessage.x;
            modal.y = -failMessage.y;

            dispatchEvent(new Event(FAIL));
        }

        /**
         * El test del puzzle ha determinado que ha sido completado
con éxito
         */
        protected function success():void
        {
            trace("success");
            dispatchEvent(new Event(SUCCESS));
        }

    }

}
```

```
package game.puzzles
{
    import flash.display.MovieClip;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import game.components.Place;
    import game.components.SpecialPlace;
    import game.components.SpecialTransition;
    import game.managers.WorkbenchManager;
    /**
     * Puzzle 1
     * Montaje de coche
     * @author Alex
     */
    public class Puzzle1 extends Puzzle
    {
        //animación de montaje de coche
        private var carAssembly:MovieClip;

        //transiciones
        private var state1ToState2Transition:SpecialTransition;
        private var state2ToState3Transition:SpecialTransition;
        private var state3ToState4Transition:SpecialTransition;
        private var state4ToState5Transition:SpecialTransition;
        private var state5ToState6Transition:SpecialTransition;

        //lugares de estado del montaje
        private var state1Place:SpecialPlace;
        private var state2Place:SpecialPlace;
        private var state3Place:SpecialPlace;
        private var state4Place:SpecialPlace;
        private var state5Place:SpecialPlace;
        private var state6Place:SpecialPlace;
```

```
        //lugares de componentes
        private var chassisPlace:SpecialPlace;
        private var wheelsPlace:SpecialPlace;

        public function Puzzle1()
        {
            super(1);
        }

        override protected function addChildren():void
        {
            //transiciones
            state1ToState2Transition = new
SpecialTransition(sprite["state1_to_state2_transition"]);
            state2ToState3Transition = new
SpecialTransition(sprite["state2_to_state3_transition"]);
            state3ToState4Transition = new
SpecialTransition(sprite["state3_to_state4_transition"]);
            state4ToState5Transition = new
SpecialTransition(sprite["state4_to_state5_transition"]);
            state5ToState6Transition = new
SpecialTransition(sprite["state5_to_state6_transition"]);

            //lugares
            state1Place = new SpecialPlace(sprite["state1_place"]);
            state2Place = new SpecialPlace(sprite["state2_place"]);
            state3Place = new SpecialPlace(sprite["state3_place"]);
            state4Place = new SpecialPlace(sprite["state4_place"]);
            state5Place = new SpecialPlace(sprite["state5_place"]);
            state6Place = new SpecialPlace(sprite["state6_place"]);

            chassisPlace = new SpecialPlace(sprite["chassis_place"]);
            wheelsPlace = new SpecialPlace(sprite["wheels_place"]);

            //montaje coche
            carAssembly = MovieClip(sprite["car_assembly"]);
            carAssembly.gotoAndStop("no_car");

            //enlazamos montaje de coche con los lugares de estado
            state1Place.addEventListener(Place.MARKS_CHANGED,
onStateMarksChanged);
            state2Place.addEventListener(Place.MARKS_CHANGED,
onStateMarksChanged);
            state3Place.addEventListener(Place.MARKS_CHANGED,
onStateMarksChanged);
            state4Place.addEventListener(Place.MARKS_CHANGED,
onStateMarksChanged);
            state5Place.addEventListener(Place.MARKS_CHANGED,
onStateMarksChanged);
            state6Place.addEventListener(Place.MARKS_CHANGED,
onStateMarksChanged);
        }

        private function onStateMarksChanged(e:Event):void
        {
            if (state6Place.numMarks != 0) {
                carAssembly.gotoAndStop("4wheels");
                return;
            }
```

```
            if (state5Place.numMarks != 0) {
                carAssembly.gotoAndStop("3wheels");
                return;
            }

            if (state4Place.numMarks != 0) {
                carAssembly.gotoAndStop("2wheels");
                return;
            }

            if (state3Place.numMarks != 0) {
                carAssembly.gotoAndStop("1wheels");
                return;
            }

            if (state2Place.numMarks != 0) {
                carAssembly.gotoAndStop("0wheels");
                return;
            }

            carAssembly.gotoAndStop("no_car");

        }

        override public function
setPuzzleOnWorkbench(workbenchManager:WorkbenchManager):void
        {
            //añadimos las marcas iniciales
            state1Place.addMark();
            chassisPlace.addMark();
            chassisPlace.addMark();
            wheelsPlace.addMark();
            wheelsPlace.addMark();

            //añadimos las transiciones

workbenchManager.addSpecialTransition(state1ToState2Transition);

workbenchManager.addSpecialTransition(state2ToState3Transition);

workbenchManager.addSpecialTransition(state3ToState4Transition);

workbenchManager.addSpecialTransition(state4ToState5Transition);

workbenchManager.addSpecialTransition(state5ToState6Transition);

            //añadimos los lugares
            workbenchManager.addSpecialPlace(state1Place);
            workbenchManager.addSpecialPlace(state2Place);
            workbenchManager.addSpecialPlace(state3Place);
            workbenchManager.addSpecialPlace(state4Place);
            workbenchManager.addSpecialPlace(state5Place);
            workbenchManager.addSpecialPlace(state6Place);

            workbenchManager.addSpecialPlace(chassisPlace);
            workbenchManager.addSpecialPlace(wheelsPlace);

            //añadimos los arcos especiales
            //entre estados de fabricación del coche
            workbenchManager.addSpecialArc(state1Place,
```

```
state1ToState2Transition);
            workbenchManager.addSpecialArc(state1ToState2Transition,
state2Place);
            workbenchManager.addSpecialArc(state2Place,
state2ToState3Transition);
            workbenchManager.addSpecialArc(state2ToState3Transition,
state3Place);
            workbenchManager.addSpecialArc(state3Place,
state3ToState4Transition);
            workbenchManager.addSpecialArc(state3ToState4Transition,
state4Place);
            workbenchManager.addSpecialArc(state4Place,
state4ToState5Transition);
            workbenchManager.addSpecialArc(state4ToState5Transition,
state5Place);
            workbenchManager.addSpecialArc(state5Place,
state5ToState6Transition);
            workbenchManager.addSpecialArc(state5ToState6Transition,
state6Place);
            //chasis
            workbenchManager.addSpecialArc(chassisPlace,
state1ToState2Transition);
            //ruedas
            workbenchManager.addSpecialArc(wheelsPlace,
state2ToState3Transition);
            workbenchManager.addSpecialArc(wheelsPlace,
state3ToState4Transition);
            workbenchManager.addSpecialArc(wheelsPlace,
state4ToState5Transition);
            workbenchManager.addSpecialArc(wheelsPlace,
state5ToState6Transition);
        }

        /**
         * Comprueba si el nivel está completado, realizando diversas
pruebas
         */
        override public function testLevel():void
        {
            //primera etapa de test
            testStep = 0;
            continueTest();
        }

        /**
         * Realiza las diversas comprobaciones y acciones que
determinan si el puzzle está resuelto o no
         * Entre caso y caso, se pone la Red de Petri en marcha hasta
que para, y se sigue con el siguiente paso
         */
        override public function continueTest():void
        {

            switch (testStep)
            {
                case 0:
                    //comprobamos que no hay ningún coche montándose o
montado
                    if (state2Place.numMarks != 0  ||
state3Place.numMarks != 0 || state4Place.numMarks != 0 ||
```

```
state5Place.numMarks != 0 || state6Place.numMarks != 0) {
                        fail(1);
                        return;
                    }

                    //intentamos montar el chasis
                    state1ToState2Transition.checkTrigger();
                    break;

                case 1:
                    //comprobamos que el chasis está montado
                    if (state2Place.numMarks != 1) {
                        fail(2);
                        return;
                    }

                    //intentamos montar la primera rueda
                    state2ToState3Transition.checkTrigger();
                    break;

                case 2:
                    //comprobamos que la primera rueda está montada
                    if (state3Place.numMarks != 1) {
                        fail(2);
                        return;
                    }

                    //intentamos montar la segunda rueda
                    state3ToState4Transition.checkTrigger();
                    break;

                case 3:
                    //comprobamos que la segunda rueda está montada
                    if (state4Place.numMarks != 1) {
                        fail(2);
                        return;
                    }

                    //intentamos montar la tercera rueda
                    state4ToState5Transition.checkTrigger();
                    break;

                case 4:
                    //comprobamos que la tercera rueda está montada
                    if (state5Place.numMarks != 1) {
                        fail(2);
                        return;
                    }

                    //intentamos montar la cuarta rueda
                    state5ToState6Transition.checkTrigger();
                    break;

                case 5:
                    //comprobamos que la cuarta rueda está montada
                    if (state6Place.numMarks != 1) {
                        fail(2);
                        return;
                    }
```

```
                    //comprobamos que no nos sobran piezas
                    if (chassisPlace.numMarks != 0 ||
wheelsPlace.numMarks != 0) {
                        fail(3);
                        return;
                    }

                    //si llegamos aquí, puzzle completado
                    success();
                    return;
                    break;

                default:
            }

            testStep++;

            delayedPlayUntilStop();
        }
    }

}
```

```
package game.puzzles
{
    import flash.display.MovieClip;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import game.components.Place;
    import game.components.SpecialPlace;
    import game.components.SpecialTransition;
    import game.managers.WorkbenchManager;
    /**
     * Puzzle 2
     * Semáforo
     * @author Alex
     */
    public class Puzzle2 extends Puzzle
    {
        private var redLight:MovieClip;
        private var orangeLight:MovieClip;
        private var greenLight:MovieClip;

        private var redToGreenTransition:SpecialTransition;
        private var greenToOrangeTransition:SpecialTransition;
        private var orangeToRedTransition:SpecialTransition;

        private var redPlace:SpecialPlace;
        private var greenPlace:SpecialPlace;
        private var orangePlace:SpecialPlace;

        public function Puzzle2()
        {
            super(2);
        }

        override protected function addChildren():void
        {
            //transiciones
            redToGreenTransition = new
SpecialTransition(sprite["red_to_green_transition"]);
```

```
            greenToOrangeTransition = new
SpecialTransition(sprite["green_to_orange_transition"]);
            orangeToRedTransition = new
SpecialTransition(sprite["orange_to_red_transition"]);

            //lugares
            redPlace = new SpecialPlace(sprite["red_place"]);
            greenPlace = new SpecialPlace(sprite["green_place"]);
            orangePlace = new SpecialPlace(sprite["orange_place"]);

            //luces
            redLight = MovieClip(sprite["red_light"]);
            redLight.gotoAndStop("off");

            orangeLight = MovieClip(sprite["orange_light"]);
            orangeLight.gotoAndStop("off");

            greenLight = MovieClip(sprite["green_light"]);
            greenLight.gotoAndStop("off");

            //enlazamos luces con lugares
            redPlace.addEventListener(Place.MARKS_CHANGED,
function():void {
                if (redPlace.numMarks == 0) {
                    redLight.gotoAndStop("off");
                }else {
                    redLight.gotoAndStop("on");
                }
            });

            orangePlace.addEventListener(Place.MARKS_CHANGED,
function():void {
                if (orangePlace.numMarks == 0) {
                    orangeLight.gotoAndStop("off");
                }else {
                    orangeLight.gotoAndStop("on");
                }
            });

            greenPlace.addEventListener(Place.MARKS_CHANGED,
function():void {
                if (greenPlace.numMarks == 0) {
                    greenLight.gotoAndStop("off");
                }else {
                    greenLight.gotoAndStop("on");
                }
            });
        }

        override public function
setPuzzleOnWorkbench(workbenchManager:WorkbenchManager):void
        {
            //añadimos las transiciones

workbenchManager.addSpecialTransition(redToGreenTransition);

workbenchManager.addSpecialTransition(greenToOrangeTransition);

workbenchManager.addSpecialTransition(orangeToRedTransition);
```

```
            //añadimos los lugares
            workbenchManager.addSpecialPlace(redPlace);
            workbenchManager.addSpecialPlace(greenPlace);
            workbenchManager.addSpecialPlace(orangePlace);

            //añadimos los arcos especiales
            workbenchManager.addSpecialArc(redPlace,
redToGreenTransition);
            workbenchManager.addSpecialArc(greenPlace,
greenToOrangeTransition);
        }

        /**
         * Comprueba si el nivel está completado, realizando diversas
pruebas
         */
        override public function testLevel():void
        {
            //primera etapa de test
            testStep = 0;
            continueTest();
        }

        /**
         * Realiza las diversas comprobaciones y acciones que
determinan si el puzzle está resuelto o no
         * Entre caso y caso, se pone la Red de Petri en marcha hasta
que para, y se sigue con el siguiente paso
         */
        override public function continueTest():void
        {

            switch (testStep)
            {
                case 0:
                    //comprobamos que el semáforo está en rojo
                    if (!isOn(redLight)) {
                        fail(1);
                        return;
                    }

                    //comprobamos que hay una y sólo una luz encendida
en el semáforo
                    if (numLightsOn != 1) {
                        fail(2);
                        return;
                    };

                    //intentamos pasar de rojo a verde
                    redToGreenTransition.checkTrigger();
                    break;

                case 1:
                    //comprobamos que el semáforo está en verde
                    if (!isOn(greenLight)) {
                        fail(3);
                        return;
                    }

                    //comprobamos que hay una y sólo una luz encendida
```

```
en el semáforo
                    if (numLightsOn != 1) {
                        fail(2);
                        return;
                    };

                    //intentamos pasar de verde a naranja
                    greenToOrangeTransition.checkTrigger();
                    break;

                case 2:
                    //comprobamos que el semáforo está en naranja
                    if (!isOn(orangeLight)) {
                        fail(3);
                        return;
                    }

                    //comprobamos que hay una y sólo una luz encendida
en el semáforo
                    if (numLightsOn != 1) {
                        fail(2);
                        return;
                    };

                    //intentamos pasar de naranja a rojo
                    orangeToRedTransition.checkTrigger();
                    break;

                case 3:
                    //comprobamos que el semáforo está en rojo
                    if (!isOn(redLight)) {
                        fail(3);
                        return;
                    }

                    //comprobamos que hay una y sólo una luz encendida
en el semáforo
                    if (numLightsOn != 1) {
                        fail(2);
                        return;
                    };

                    //si llegamos hasta aquí, puzzle completado
                    success();
                    return;
                    break;

                default:
            }

            testStep++;

            delayedPlayUntilStop();
        }

        /**
         * Comprueba si una luz está encedida
         * @param   light la luz a comprobar
         * @return  si la luz está encendida
         */
```

```
        private function isOn(light:MovieClip):Boolean
        {
            if (light.currentFrameLabel == "on") {
                return true;
            } else {
                return false;
            }
        }

        /**
         * Devuelve el número de luces encendidas del semáforo
         * @return el número de luces encendidas del semáforo
         */
        private function get numLightsOn():int
        {
            var numLightsOn:int = 0;

            if (redLight.currentFrameLabel == "on") {
                numLightsOn++;
            }

            if (orangeLight.currentFrameLabel == "on") {
                numLightsOn++;
            }

            if (greenLight.currentFrameLabel == "on") {
                numLightsOn++;
            }

            return numLightsOn;
        }

    }

}
```

```
package game.puzzles
{
    import flash.display.MovieClip;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import game.components.Place;
    import game.components.SpecialPlace;
    import game.components.SpecialTransition;
    import game.managers.WorkbenchManager;
    /**
     * Puzzle 3
     * Ascensor
     * @author Alex
     */
    public class Puzzle3 extends Puzzle
    {
        private var elevator:MovieClip;
        private var upBtn:MovieClip;
        private var downBtn:MovieClip;

        private var upTransition:SpecialTransition;
        private var downTransition:SpecialTransition;

        private var currentFloorPlace:SpecialPlace;
```

```
        public function Puzzle3()
        {
            super(3);
        }

        override protected function addChildren():void
        {
            //transiciones
            upTransition = new
SpecialTransition(sprite["up_transition"]);
            downTransition = new
SpecialTransition(sprite["down_transition"]);

            //lugares
            currentFloorPlace = new
SpecialPlace(sprite["current_floor_place"]);

            //botones
            upBtn = MovieClip(sprite["up_btn_animation"]);
            upBtn.gotoAndStop(1);

            //test
            upBtn.addEventListener(MouseEvent.CLICK,
function(e:Event=null):void {
                pressUpBtn();
            });

            downBtn = MovieClip(sprite["down_btn_animation"]);
            downBtn.gotoAndStop(1);

            //test
            downBtn.addEventListener(MouseEvent.CLICK,
function(e:Event=null):void {
                pressDownBtn();
            });

            //ascensor
            elevator = MovieClip(sprite["elevator"]);
            elevator.gotoAndStop("floor0");

            currentFloorPlace.addEventListener(Place.MARKS_CHANGED,
function():void {
                var floor:int = Math.min(currentFloorPlace.numMarks,
3);
                elevator.gotoAndStop("floor"+floor);
            });
        }

        override public function
setPuzzleOnWorkbench(workbenchManager:WorkbenchManager):void
        {
            //añadimos las transiciones
            workbenchManager.addSpecialTransition(upTransition);
            workbenchManager.addSpecialTransition(downTransition);

            //añadimos los lugares
            workbenchManager.addSpecialPlace(currentFloorPlace);

            //añadimos los arcos especiales
            workbenchManager.addSpecialArc(upTransition,
```

```
currentFloorPlace);
            workbenchManager.addSpecialArc(currentFloorPlace,
downTransition);
        }

        /**
         * Comprueba si el nivel está completado, realizando diversas
pruebas
         */
        override public function testLevel():void
        {
            //primera etapa de test
            testStep = 0;
            continueTest();
        }

        /**
         * Realiza las diversas comprobaciones y acciones que
determinan si el puzzle está resuelto o no
         * Entre caso y caso, se pone la Red de Petri en marcha hasta
que para, y se sigue con el siguiente paso
         */
        override public function continueTest():void
        {

            switch (testStep)
            {
                case 0:
                    //comprobamos que el ascensor está en el bajo
                    if (currentFloorPlace.numMarks != 0) {
                        fail(1);
                        return;
                    }

                    //comprobamos que el ascensor puede subir al
primer piso
                    if (!upTransition.enabled) {
                        fail(2);
                        return;
                    }

                    //comprobamos que el ascensor no puede bajar
                    if (downTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //intentamos bajar
                    pressDownBtn();
                    break;

                case 1:
                    //comprobamos que el ascensor sigue en el bajo
                    if (currentFloorPlace.numMarks != 0) {
                        fail(1);
                        return;
                    }

                    //comprobamos que el ascensor puede subir al
primer piso
```

```
                    if (!upTransition.enabled) {
                        fail(2);
                        return;
                    }

                    //comprobamos que el ascensor no puede bajar
                    if (downTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //intentamos subir al primer piso
                    pressUpBtn();
                    break;

                case 2:
                    //comprobamos que el ascensor está en el primer
piso
                    if (currentFloorPlace.numMarks != 1) {
                        fail(2);
                        return;
                    }

                    //comprobamos que podemos subir al segundo piso
                    if (!upTransition.enabled) {
                        fail(2);
                        return;
                    }

                    //comprobamos que podemos bajar al bajo
                    if (!downTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //intentamos subir al segundo piso
                    pressUpBtn();
                    break;

                case 3:
                    //comprobamos que el ascensor está en el segundo
piso
                    if (currentFloorPlace.numMarks != 2) {
                        fail(2);
                        return;
                    }

                    //comprobamos que podemos subir al tercer piso
                    if (!upTransition.enabled) {
                        fail(2);
                        return;
                    }

                    //comprobamos que podemos bajar al primer piso
                    if (!downTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //intentamos subir al tercer piso
```

```
                    pressUpBtn();
                    break;

                case 4:
                    //comprobamos que el ascensor está en el tercer
piso
                    if (currentFloorPlace.numMarks != 3) {
                        fail(2);
                        return;
                    }

                    //comprobamos que no podemos subir
                    if (upTransition.enabled) {
                        fail(2);
                        return;
                    }

                    //comprobamos que podemos bajar al segundo piso
                    if (!downTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //intentamos subir
                    pressUpBtn();
                    break;

                case 5:
                    //comprobamos que el ascensor sigue en el tercer
piso
                    if (currentFloorPlace.numMarks != 3) {
                        fail(2);
                        return;
                    }

                    //comprobamos que no podemos subir
                    if (upTransition.enabled) {
                        fail(2);
                        return;
                    }

                    //comprobamos que podemos bajar al segundo piso
                    if (!downTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //intentamos bajar al segundo piso
                    pressDownBtn();
                    break;

                case 6:
                    //comprobamos que el ascensor está en el segundo
piso
                    if (currentFloorPlace.numMarks != 2) {
                        fail(2);
                        return;
                    }

                    //comprobamos que podemos subir al tercer piso
```

```
                    if (!upTransition.enabled) {
                        fail(2);
                        return;
                    }

                    //comprobamos que podemos bajar al primer piso
                    if (!downTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //intentamos bajar al primer piso
                    pressDownBtn();
                    break;

                case 7:
                    //comprobamos que el ascensor está en el primer
piso
                    if (currentFloorPlace.numMarks != 1) {
                        fail(2);
                        return;
                    }

                    //comprobamos que podemos subir al segundo piso
                    if (!upTransition.enabled) {
                        fail(2);
                        return;
                    }

                    //comprobamos que podemos bajar al bajo
                    if (!downTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //intentamos bajar al bajo
                    pressDownBtn();
                    break;

                case 8:
                    //comprobamos que el ascensor está en el bajo
                    if (currentFloorPlace.numMarks != 0) {
                        fail(1);
                        return;
                    }

                    //comprobamos que el ascensor puede subir al
primer piso
                    if (!upTransition.enabled) {
                        fail(2);
                        return;
                    }

                    //comprobamos que el ascensor no puede bajar
                    if (downTransition.enabled) {
                        fail(3);
                        return;
                    }
                    //si llegamos hasta aquí, puzzle completado
                    success();
```

```
                    return;
                    break;

                default:
            }

            testStep++;

            delayedPlayUntilStop();
        }

        /**
         * Simula que se presiona el botón del ascensor de subir
         */
        private function pressUpBtn():void
        {
            upBtn.gotoAndPlay(1);
            upTransition.checkTrigger();
        }

        /**
         * Simula que se presiona el botón del ascensor de bajar
         */
        private function pressDownBtn():void
        {
            downBtn.gotoAndPlay(1);
            downTransition.checkTrigger();
        }

    }

}
```

```
package game.puzzles
{
    import flash.display.MovieClip;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import game.components.Place;
    import game.components.SpecialPlace;
    import game.components.SpecialTransition;
    import game.managers.WorkbenchManager;
    /**
     * Puzzle 4
     * Cinta de montaje
     * @author Alex
     */
    public class Puzzle4 extends Puzzle
    {
        //cajas
        private var boxes:MovieClip;
        //cinta de montaje
        private var conveyorBelt:MovieClip;
        //productos montados
        private var products:MovieClip;

        private var boxesPlace:SpecialPlace;
        private var conveyorBeltPlace:SpecialPlace;
        private var productsPlace:SpecialPlace;
```

```
        public function Puzzle4()
        {
            super(4);
        }

        override protected function addChildren():void
        {
            //lugares
            boxesPlace = new SpecialPlace(sprite["boxes_place"]);
            conveyorBeltPlace = new
SpecialPlace(sprite["conveyor_belt_place"]);
            productsPlace = new
SpecialPlace(sprite["products_place"]);

            //animaciones
            boxes = MovieClip(sprite["boxes"]);
            boxes.gotoAndStop("0boxes");

            boxesPlace.addEventListener(Place.MARKS_CHANGED,
function():void {
                var numBoxes:int = Math.min(boxesPlace.numMarks, 5);
                boxes.gotoAndStop(numBoxes + "boxes");
            });

            conveyorBelt = MovieClip(sprite["conveyor_belt"]);
            conveyorBelt.gotoAndStop("empty");

            conveyorBeltPlace.addEventListener(Place.MARKS_CHANGED,
function():void {
                switch (conveyorBeltPlace.numMarks)
                {
                    case 0:
                        conveyorBelt.gotoAndStop("empty");
                        break;
                    case 1:
                        conveyorBelt.gotoAndStop("building");
                        break;
                    default:
                        conveyorBelt.gotoAndStop("crash");
                        break;
                }
            });

            products = MovieClip(sprite["products"]);
            products.gotoAndStop("0products");

            productsPlace.addEventListener(Place.MARKS_CHANGED,
function():void {
                var numProducts:int = Math.min(productsPlace.numMarks,
5);
                products.gotoAndStop(numProducts + "products");
            });

        }

        override public function
setPuzzleOnWorkbench(workbenchManager:WorkbenchManager):void
        {
            //añadimos los lugares
            workbenchManager.addSpecialPlace(boxesPlace);
```

```
            workbenchManager.addSpecialPlace(conveyorBeltPlace);
            workbenchManager.addSpecialPlace(productsPlace);

            workbenchManager.transitionsTriggerLimit = 20;
        }

        /**
         * Comprueba si el nivel está completado, realizando diversas
pruebas
         */
        override public function testLevel():void
        {
            //nos aseguramos de que no replicamos listeners
            conveyorBeltPlace.removeEventListener(Place.MARK_ADDED,
onConveyorBeltMarkAdded);

            //primera etapa de test
            testStep = 0;
            toProductionUnits = 0;
            continueTest();
        }

        private var toProductionUnits:int = 0;

        /**
         * Realiza las diversas comprobaciones y acciones que
determinan si el puzzle está resuelto o no
         * Entre caso y caso, se pone la Red de Petri en marcha hasta
que para, y se sigue con el siguiente paso
         */
        override public function continueTest():void
        {

            switch (testStep)
            {
                case 0:
                    //comprobamos que hay 5 cajas, ningún producto
montándose y ninguno montado
                    if (boxesPlace.numMarks != 5 ||
conveyorBeltPlace.numMarks != 0 || productsPlace.numMarks != 0) {
                        fail(1);
                        return;
                    }


conveyorBeltPlace.addEventListener(Place.MARK_ADDED,
onConveyorBeltMarkAdded);
                    break;

                case 1:
                    //comprobamos que ya no quedan cajas, ningún
producto montándose, que se han montado 5 productos y tenemos los 5
montados
                    if (boxesPlace.numMarks != 0 ||
conveyorBeltPlace.numMarks != 0 || toProductionUnits != 5 ||
productsPlace.numMarks != 5 ) {
                        fail(2);
                        return;
                    }
```

```
                    //si llegamos hasta aquí, puzzle completado
                    success();
                    return;
                    break;

                default:
            }

            testStep++;

            delayedPlayUntilStop();
        }

        private function onConveyorBeltMarkAdded(e:Event):void
        {
            if (conveyorBeltPlace.numMarks > 1) {
                fail(3);
            }
            //añadimos que se ha producido un producto más
            toProductionUnits++;
        }



    }

}
```

```
package game.puzzles
{
    import flash.display.MovieClip;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import flash.events.TimerEvent;
    import flash.utils.Timer;
    import game.components.Place;
    import game.components.SpecialPlace;
    import game.components.SpecialTransition;
    import game.managers.WorkbenchManager;
    /**
     * Puzzle 5
     * Máquina de refrescos
     * @author Alex
     */
    public class Puzzle5 extends Puzzle
    {
        private var coinSlot:MovieClip;
        private var coinSlotTransition:SpecialTransition;

        private var refreshmentBtn:MovieClip;
        private var refreshmentBtnTransition:SpecialTransition;

        private var refreshmentSlot:MovieClip;
        private var refreshmentSlotPlace:SpecialPlace;

        public function Puzzle5()
        {
            super(5);
        }

        override protected function addChildren():void
```

```
        {
            coinSlot = MovieClip(sprite["coin_slot"]);
            coinSlot.gotoAndStop(1);
            coinSlotTransition = new
SpecialTransition(sprite["coin_slot_transition"]);

            //test
            coinSlot.addEventListener(MouseEvent.CLICK,
function(e:Event=null):void {
                insertCoin();
            });

            refreshmentBtn = MovieClip(sprite["refreshment_btn"]);
            refreshmentBtn.gotoAndStop(1);
            refreshmentBtnTransition = new
SpecialTransition(sprite["refreshment_btn_transition"]);

            //test
            refreshmentBtn.addEventListener(MouseEvent.CLICK,
function(e:Event=null):void {
                pushRefreshmentBtn();
            });

            refreshmentSlot = MovieClip(sprite["refreshment_slot"]);
            refreshmentSlot.gotoAndStop(1);
            refreshmentSlotPlace = new
SpecialPlace(sprite["refreshment_slot_place"]);

            refreshmentSlotPlace.addEventListener(Place.MARK_ADDED,
function():void {
                refreshmentSlot.gotoAndPlay(1);
                numRefreshments++;
            });

            //test
            //refreshmentSlot.addEventListener(MouseEvent.CLICK,
function(e:Event=null):void {
                //refreshmentSlot.gotoAndPlay(1);
                //success();
            //});
        }

        override public function
setPuzzleOnWorkbench(workbenchManager:WorkbenchManager):void
        {
            //colocamos la transición especial que representa que han
insertado una moneda al lado de la ranura
            workbenchManager.addSpecialTransition(coinSlotTransition);

            //colocamos la transición especial que representa que han
pulsado el botón de refresco

workbenchManager.addSpecialTransition(refreshmentBtnTransition);

            //colocamos el lugar especial que representa los refrescos
que han salido de la máquina
            workbenchManager.addSpecialPlace(refreshmentSlotPlace);
        }
```

```
        //contamos el número de refrescos que sale de la máquina
        private var numRefreshments:int = 0;
        /**
         * Comprueba si el nivel está completado, realizando diversas
pruebas
         */
        override public function testLevel():void
        {
            //reseteamos la cuenta del número de refrescos que sale de
la máquina
            numRefreshments = 0;

            //primera etapa de test
            testStep = 0;
            continueTest();
        }

        /**
         * Realiza las diversas comprobaciones y acciones que
determinan si el puzzle está resuelto o no
         * Entre caso y caso, se pone la Red de Petri en marcha hasta
que para, y se sigue con el siguiente paso
         */
        override public function continueTest():void
        {
            switch (testStep)
            {
                case 0:
                    pushRefreshmentBtn();
                    break;

                case 1:
                    //comprobamos que como le hemos dado al botón de
comprar refresco sin haber insertado ninguna moneda, no salen
refrescos
                    if (numRefreshments != 0) {
                        fail(1);
                        return;
                    }
                    insertCoin();
                    break;

                case 2:
                    //comprobamos que tras haber insertado una moneda,
tampoco salen refrescos
                    if (numRefreshments != 0) {
                        fail(2);
                        return;
                    }
                    pushRefreshmentBtn();
                    break;

                case 3:
                    //comprobamos que al haber una moneda dentro de la
máquina, le hemos dado al botón de comprar refresco y sale un refresco
                    if (numRefreshments != 1) {
                        fail(3);
                        return;
                    }
                    pushRefreshmentBtn();
```

```
                    break;

                case 4:
                    //comprobamos que como ya no quedan monedas, no
sale refresco
                    if (numRefreshments != 1) {
                        fail(1);
                        return;
                    }
                    insertCoin();
                    break;

                case 5:
                    //comprobamos que tras haber insertado una moneda,
tampoco salen refrescos
                    if (numRefreshments != 1) {
                        fail(2);
                        return;
                    }
                    insertCoin();
                    break;

                case 6:
                    //comprobamos que tras haber insertado una moneda,
tampoco salen refrescos
                    if (numRefreshments != 1) {
                        fail(2);
                        return;
                    }
                    insertCoin();
                    break;

                case 7:
                    //comprobamos que tras haber insertado una moneda,
tampoco salen refrescos
                    if (numRefreshments != 1) {
                        fail(2);
                        return;
                    }
                    pushRefreshmentBtn();
                    break;

                case 8:
                    //comprobamos que al haber 3 monedas dentro de la
máquina, le hemos dado al botón de comprar refresco y sale un refresco
                    if (numRefreshments != 2) {
                        fail(3);
                        return;
                    }
                    pushRefreshmentBtn();
                    break;

                case 9:
                    //comprobamos que al haber 2 monedas dentro de la
máquina, le hemos dado al botón de comprar refresco y sale un refresco
                    if (numRefreshments != 3) {
                        fail(3);
                        return;
                    }
                    //si llegamos aqui, puzzle resuelto
```

```
                    success();
                    return;
                    break;

                default:
            }

            testStep++;

            delayedPlayUntilStop();
        }

        /**
         * Inserta una moneda
         */
        private function insertCoin():void
        {
            coinSlot.gotoAndPlay(1);
            coinSlotTransition.checkTrigger();
        }

        /**
         * Pulsa el botón de refresco
         */
        private function pushRefreshmentBtn():void
        {
            refreshmentBtn.gotoAndPlay(1);
            refreshmentBtnTransition.checkTrigger();
        }

    }

}
```

```
package game.puzzles
{
    import flash.display.MovieClip;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import flash.events.TimerEvent;
    import flash.utils.Timer;
    import game.components.Place;
    import game.components.SpecialPlace;
    import game.components.SpecialTransition;
    import game.managers.WorkbenchManager;
    /**
     * Puzzle 6
     * Máquina de refrescos
     * @author Alex
     */
    public class Puzzle6 extends Puzzle
    {
        //semáforo norte
        private var northGreenLight:MovieClip;
        private var northRedLight:MovieClip;
        private var northGreenPlace:SpecialPlace;
        private var northRedPlace:SpecialPlace;
        private var northGreenToRedTransition:SpecialTransition;
        private var northRedToGreenTransition:SpecialTransition;
```

```
        //semáforo sur
        private var southGreenLight:MovieClip;
        private var southRedLight:MovieClip;
        private var southGreenPlace:SpecialPlace;
        private var southRedPlace:SpecialPlace;
        private var southGreenToRedTransition:SpecialTransition;
        private var southRedToGreenTransition:SpecialTransition;

        //semáforo oeste
        private var westGreenLight:MovieClip;
        private var westRedLight:MovieClip;
        private var westGreenPlace:SpecialPlace;
        private var westRedPlace:SpecialPlace;
        private var westGreenToRedTransition:SpecialTransition;
        private var westRedToGreenTransition:SpecialTransition;

        //semáforo este
        private var eastGreenLight:MovieClip;
        private var eastRedLight:MovieClip;
        private var eastGreenPlace:SpecialPlace;
        private var eastRedPlace:SpecialPlace;
        private var eastGreenToRedTransition:SpecialTransition;
        private var eastRedToGreenTransition:SpecialTransition;

        public function Puzzle6()
        {
            super(6);
        }

        override protected function addChildren():void
        {
            //semáforo norte
            northGreenLight = MovieClip(sprite["north_green_light"]);
            northRedLight = MovieClip(sprite["north_red_light"]);
            northGreenPlace = new
SpecialPlace(sprite["north_green_place"]);
            northRedPlace = new
SpecialPlace(sprite["north_red_place"]);
            northGreenToRedTransition = new
SpecialTransition(sprite["north_green_to_red_transition"]);
            northRedToGreenTransition = new
SpecialTransition(sprite["north_red_to_green_transition"]);

            //semáforo sur
            southGreenLight = MovieClip(sprite["south_green_light"]);
            southRedLight = MovieClip(sprite["south_red_light"]);
            southGreenPlace = new
SpecialPlace(sprite["south_green_place"]);
            southRedPlace = new
SpecialPlace(sprite["south_red_place"]);
            southGreenToRedTransition = new
SpecialTransition(sprite["south_green_to_red_transition"]);
            southRedToGreenTransition = new
SpecialTransition(sprite["south_red_to_green_transition"]);

            //semáforo oeste
            westGreenLight = MovieClip(sprite["west_green_light"]);
            westRedLight = MovieClip(sprite["west_red_light"]);
            westGreenPlace = new
SpecialPlace(sprite["west_green_place"]);
```

```
            westRedPlace = new SpecialPlace(sprite["west_red_place"]);
            westGreenToRedTransition = new
SpecialTransition(sprite["west_green_to_red_transition"]);
            westRedToGreenTransition = new
SpecialTransition(sprite["west_red_to_green_transition"]);

            //semáforo este
            eastGreenLight = MovieClip(sprite["east_green_light"]);
            eastRedLight = MovieClip(sprite["east_red_light"]);
            eastGreenPlace = new
SpecialPlace(sprite["east_green_place"]);
            eastRedPlace = new SpecialPlace(sprite["east_red_place"]);
            eastGreenToRedTransition = new
SpecialTransition(sprite["east_green_to_red_transition"]);
            eastRedToGreenTransition = new
SpecialTransition(sprite["east_red_to_green_transition"]);

            //enlazamos luces con lugares
            joinPlaceToLight(northGreenPlace, northGreenLight);
            joinPlaceToLight(northRedPlace, northRedLight);
            joinPlaceToLight(southGreenPlace, southGreenLight);
            joinPlaceToLight(southRedPlace, southRedLight);
            joinPlaceToLight(westGreenPlace, westGreenLight);
            joinPlaceToLight(westRedPlace, westRedLight);
            joinPlaceToLight(eastGreenPlace, eastGreenLight);
            joinPlaceToLight(eastRedPlace, eastRedLight);

            //inicializamos los semáforos en rojo
            northGreenLight.gotoAndStop("off");
            southGreenLight.gotoAndStop("off");
            westGreenLight.gotoAndStop("off");
            eastGreenLight.gotoAndStop("off");

            northRedPlace.addMark();
            southRedPlace.addMark();
            westRedPlace.addMark();
            eastRedPlace.addMark();
        }

        /**
         * Enlaza el comportamiento de una luz conforme a las marcas
del lugar pasado como referencia
         * @param   place   el lugar que definirá el comportamiento de
la luz
         * @param   light   la luz cuyo comportamiento estará definido
por el lugar
         */
        private function joinPlaceToLight(place:SpecialPlace,
light:MovieClip):void
        {
            place.addEventListener(Place.MARKS_CHANGED,
function():void {
                if (place.numMarks == 0) {
                    light.gotoAndStop("off");
                }else {
                    light.gotoAndStop("on");
                }
            });
        }
```

```
        override public function
setPuzzleOnWorkbench(workbenchManager:WorkbenchManager):void
        {
            //colocamos los lugares
            workbenchManager.addSpecialPlace(northGreenPlace);
            workbenchManager.addSpecialPlace(northRedPlace);

            workbenchManager.addSpecialPlace(southGreenPlace);
            workbenchManager.addSpecialPlace(southRedPlace);

            workbenchManager.addSpecialPlace(westGreenPlace);
            workbenchManager.addSpecialPlace(westRedPlace);

            workbenchManager.addSpecialPlace(eastGreenPlace);
            workbenchManager.addSpecialPlace(eastRedPlace);

            //colocamos las transiciones

workbenchManager.addSpecialTransition(northGreenToRedTransition);

workbenchManager.addSpecialTransition(northRedToGreenTransition);


workbenchManager.addSpecialTransition(southGreenToRedTransition);

workbenchManager.addSpecialTransition(southRedToGreenTransition);


workbenchManager.addSpecialTransition(westGreenToRedTransition);

workbenchManager.addSpecialTransition(westRedToGreenTransition);


workbenchManager.addSpecialTransition(eastGreenToRedTransition);

workbenchManager.addSpecialTransition(eastRedToGreenTransition);

            //colocamos los arcos
            workbenchManager.addSpecialArc(northGreenPlace,
northGreenToRedTransition);
            workbenchManager.addSpecialArc(northGreenToRedTransition,
northRedPlace);
            workbenchManager.addSpecialArc(northRedPlace,
northRedToGreenTransition);
            workbenchManager.addSpecialArc(northRedToGreenTransition,
northGreenPlace);

            workbenchManager.addSpecialArc(southGreenPlace,
southGreenToRedTransition);
            workbenchManager.addSpecialArc(southGreenToRedTransition,
southRedPlace);
            workbenchManager.addSpecialArc(southRedPlace,
southRedToGreenTransition);
            workbenchManager.addSpecialArc(southRedToGreenTransition,
southGreenPlace);

            workbenchManager.addSpecialArc(westGreenPlace,
westGreenToRedTransition);
            workbenchManager.addSpecialArc(westGreenToRedTransition,
westRedPlace);
```

```
            workbenchManager.addSpecialArc(westRedPlace,
westRedToGreenTransition);
            workbenchManager.addSpecialArc(westRedToGreenTransition,
westGreenPlace);

            workbenchManager.addSpecialArc(eastGreenPlace,
eastGreenToRedTransition);
            workbenchManager.addSpecialArc(eastGreenToRedTransition,
eastRedPlace);
            workbenchManager.addSpecialArc(eastRedPlace,
eastRedToGreenTransition);
            workbenchManager.addSpecialArc(eastRedToGreenTransition,
eastGreenPlace);
        }

        /**
         * Comprueba si el nivel está completado, realizando diversas
pruebas
         */
        override public function testLevel():void
        {
            //primera etapa de test
            testStep = 0;
            continueTest();
        }

        /**
         * Realiza las diversas comprobaciones y acciones que
determinan si el puzzle está resuelto o no
         * Entre caso y caso, se pone la Red de Petri en marcha hasta
que para, y se sigue con el siguiente paso
         */
        override public function continueTest():void
        {
            switch (testStep)
            {
                case 0:
                    //comprobamos que todos los semáforos están en
rojo
                    if (!isOn(northRedLight) || !isOn(southRedLight)
|| !isOn(westRedLight) || !isOn(eastRedLight)) {
                        fail(1);
                        return;
                    }

                    //comprobamos que ningún semáforo tiene más o
menos de 1 luz encendida
                    if (!numSemaphoresLightsCheck()) {
                        fail(2);
                        return;
                    }

                    //comprobamos que podemos poner en verde cualquier
semáforo
                    if (!northRedToGreenTransition.enabled ||
!southRedToGreenTransition.enabled ||
!westRedToGreenTransition.enabled ||
!eastRedToGreenTransition.enabled) {
                        fail(3);
                        return;
```

```
                    }

                    //comprobamos que no podemos poner en rojo ningún
semáforo
                    if (northGreenToRedTransition.enabled ||
southGreenToRedTransition.enabled || westGreenToRedTransition.enabled
|| eastGreenToRedTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //ponemos en verde el semáforo norte
                    northRedToGreenTransition.checkTrigger();

                    break;

                case 1:
                    //comprobamos que ningún semáforo tiene más o
menos de 1 luz encendida
                    if (!numSemaphoresLightsCheck()) {
                        fail(2);
                        return;
                    }

                    //comprobamos que todos están en rojo, menos el
norte que está en verde
                    if (!isOn(northGreenLight) || !isOn(southRedLight)
|| !isOn(westRedLight) || !isOn(eastRedLight)) {
                        fail(4);
                        return;
                    }

                    //comprobamos que sólo puede ponerse en verde el
semáforo sur
                    if (northRedToGreenTransition.enabled ||
!southRedToGreenTransition.enabled || westRedToGreenTransition.enabled
|| eastRedToGreenTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //comprobamos que sólo puede ponerse en rojo el
semáforo norte
                    if (!northGreenToRedTransition.enabled ||
southGreenToRedTransition.enabled || westGreenToRedTransition.enabled
|| eastGreenToRedTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //ponemos en verde el semáforo sur
                    southRedToGreenTransition.checkTrigger();

                    break;

                case 2:
                    //comprobamos que ningún semáforo tiene más o
menos de 1 luz encendida
                    if (!numSemaphoresLightsCheck()) {
                        fail(2);
```

```
                        return;
                    }

                    //comprobamos que norte y sur están en verde, y el
resto en rojo
                    if (!isOn(northGreenLight) ||
!isOn(southGreenLight) || !isOn(westRedLight) || !isOn(eastRedLight))
{
                        fail(4);
                        return;
                    }

                    //comprobamos que ningún semáforo puede ponerse en
verde
                    if (northRedToGreenTransition.enabled ||
southRedToGreenTransition.enabled || westRedToGreenTransition.enabled
|| eastRedToGreenTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //comprobamos que sólo pueden ponerse en rojo los
semáforos norte y sur
                    if (!northGreenToRedTransition.enabled ||
!southGreenToRedTransition.enabled || westGreenToRedTransition.enabled
|| eastGreenToRedTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //ponemos en rojo el semáforo norte
                    northGreenToRedTransition.checkTrigger();

                    break;

                case 3:
                    //comprobamos que ningún semáforo tiene más o
menos de 1 luz encendida
                    if (!numSemaphoresLightsCheck()) {
                        fail(2);
                        return;
                    }

                    //comprobamos que todos están en rojo, menos el
sur que está en verde
                    if (!isOn(northRedLight) || !isOn(southGreenLight)
|| !isOn(westRedLight) || !isOn(eastRedLight)) {
                        fail(4);
                        return;
                    }

                    //comprobamos que sólo puede ponerse en verde el
semáforo norte
                    if (!northRedToGreenTransition.enabled ||
southRedToGreenTransition.enabled || westRedToGreenTransition.enabled
|| eastRedToGreenTransition.enabled) {
                        fail(3);
                        return;
                    }
```

```
                    //comprobamos que sólo puede ponerse en rojo el
semáforo sur
                    if (northGreenToRedTransition.enabled ||
!southGreenToRedTransition.enabled || westGreenToRedTransition.enabled
|| eastGreenToRedTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //ponemos en rojo el semáforo sur
                    southGreenToRedTransition.checkTrigger();

                    break;

                case 4:
                    //comprobamos que ningún semáforo tiene más o
menos de 1 luz encendida
                    if (!numSemaphoresLightsCheck()) {
                        fail(2);
                        return;
                    }

                    //comprobamos que todos los semáforos están en
rojo
                    if (!isOn(northRedLight) || !isOn(southRedLight)
|| !isOn(westRedLight) || !isOn(eastRedLight)) {
                        fail(4);
                        return;
                    }

                    //comprobamos que podemos poner en verde cualquier
semáforo
                    if (!northRedToGreenTransition.enabled ||
!southRedToGreenTransition.enabled ||
!westRedToGreenTransition.enabled ||
!eastRedToGreenTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //comprobamos que no podemos poner en rojo ningún
semáforo
                    if (northGreenToRedTransition.enabled ||
southGreenToRedTransition.enabled || westGreenToRedTransition.enabled
|| eastGreenToRedTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //ponemos en verde el semáforo oeste
                    westRedToGreenTransition.checkTrigger();
                    break;

                case 5:
                    //comprobamos que ningún semáforo tiene más o
menos de 1 luz encendida
                    if (!numSemaphoresLightsCheck()) {
                        fail(2);
                        return;
                    }
```

```
                    //comprobamos que todos están en rojo, menos el
oeste que está en verde
                    if (!isOn(northRedLight) || !isOn(southRedLight)
|| !isOn(westGreenLight) || !isOn(eastRedLight)) {
                        fail(4);
                        return;
                    }

                    //comprobamos que sólo puede ponerse en verde el
semáforo este
                    if (northRedToGreenTransition.enabled ||
southRedToGreenTransition.enabled || westRedToGreenTransition.enabled
|| !eastRedToGreenTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //comprobamos que sólo puede ponerse en rojo el
semáforo oeste
                    if (northGreenToRedTransition.enabled ||
southGreenToRedTransition.enabled || !westGreenToRedTransition.enabled
|| eastGreenToRedTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //ponemos en verde el semáforo este
                    eastRedToGreenTransition.checkTrigger();
                    break;

                case 6:
                    //comprobamos que ningún semáforo tiene más o
menos de 1 luz encendida
                    if (!numSemaphoresLightsCheck()) {
                        fail(2);
                        return;
                    }

                    //comprobamos que este y oeste están en verde, y
el resto en rojo
                    if (!isOn(northRedLight) || !isOn(southRedLight)
|| !isOn(westGreenLight) || !isOn(eastGreenLight)) {
                        fail(4);
                        return;
                    }

                    //comprobamos que ningún semáforo puede ponerse en
verde
                    if (northRedToGreenTransition.enabled ||
southRedToGreenTransition.enabled || westRedToGreenTransition.enabled
|| eastRedToGreenTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //comprobamos que sólo pueden ponerse en rojo los
semáforos este y oeste
                    if (northGreenToRedTransition.enabled ||
southGreenToRedTransition.enabled || !westGreenToRedTransition.enabled
```

```
|| !eastGreenToRedTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //ponemos en rojo el semáforo oeste
                    westGreenToRedTransition.checkTrigger();

                    break;

                case 7:
                    //comprobamos que ningún semáforo tiene más o
menos de 1 luz encendida
                    if (!numSemaphoresLightsCheck()) {
                        fail(2);
                        return;
                    }

                    //comprobamos que todos están en rojo, menos el
este que está en verde
                    if (!isOn(northRedLight) || !isOn(southRedLight)
|| !isOn(westRedLight) || !isOn(eastGreenLight)) {
                        fail(4);
                        return;
                    }

                    //comprobamos que sólo puede ponerse en verde el
semáforo oeste
                    if (northRedToGreenTransition.enabled ||
southRedToGreenTransition.enabled || !westRedToGreenTransition.enabled
|| eastRedToGreenTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //comprobamos que sólo puede ponerse en rojo el
semáforo este
                    if (northGreenToRedTransition.enabled ||
southGreenToRedTransition.enabled || westGreenToRedTransition.enabled
|| !eastGreenToRedTransition.enabled) {
                        fail(3);
                        return;
                    }

                    //ponemos en rojo el semáforo este
                    eastGreenToRedTransition.checkTrigger();

                    break;

                case 8:
                    //comprobamos que ningún semáforo tiene más o
menos de 1 luz encendida
                    if (!numSemaphoresLightsCheck()) {
                        fail(2);
                        return;
                    }

                    //comprobamos que todos los semáforos están en
rojo
                    if (!isOn(northRedLight) || !isOn(southRedLight)
```

```
|| !isOn(westRedLight) || !isOn(eastRedLight)) {
                            fail(4);
                            return;
                        }

                        //comprobamos que podemos poner en verde cualquier
semáforo
                        if (!northRedToGreenTransition.enabled ||
!southRedToGreenTransition.enabled ||
!westRedToGreenTransition.enabled ||
!eastRedToGreenTransition.enabled) {
                            fail(3);
                            return;
                        }

                        //comprobamos que no podemos poner en rojo ningún
semáforo
                        if (northGreenToRedTransition.enabled ||
southGreenToRedTransition.enabled || westGreenToRedTransition.enabled
|| eastGreenToRedTransition.enabled) {
                            fail(3);
                            return;
                        }

                        //si llegamos aqui, puzzle resuelto
                        success();
                        return;
                        break;

                    default:
                }

                testStep++;

                delayedPlayUntilStop();
            }

            /**
             * Comprueba que todos los semáforos no tienen las dos luces
encendidas o apagadas
             * @return si todo está en regla
             */
            private function numSemaphoresLightsCheck():Boolean
            {
                if ((isOn(northGreenLight) && isOn(northRedLight)) ||
(!isOn(northGreenLight) && !isOn(northRedLight))) {
                    return false;
                }

                if ((isOn(southGreenLight) && isOn(southRedLight)) ||
(!isOn(southGreenLight) && !isOn(southRedLight))) {
                    return false;
                }

                if ((isOn(westGreenLight) && isOn(westRedLight)) ||
(!isOn(westGreenLight) && !isOn(westRedLight))) {
                    return false;
                }

                if ((isOn(eastGreenLight) && isOn(eastRedLight)) ||
```

```
(!isOn(eastGreenLight) && !isOn(eastRedLight))) {
                return false;
            }

            return true;
        }

        /**
         * Comprueba si una luz está encedida
         * @param   light la luz a comprobar
         * @return  si la luz está encendida
         */
        private function isOn(light:MovieClip):Boolean
        {
            if (light.currentFrameLabel == "on") {
                return true;
            } else {
                return false;
            }
        }

    }

}
```

El punto central de desarrollo, además de los puzzles anteriores que se adaptaron a este sistema, fue el editor de Redes de Petri o gestor de mesa de trabajo (WorkBenchManager), en los que es de vital importancia la gestión de los diferentes estados posibles de la edición y comprobación de puzzles.

```
package game.managers
{
    import core.Component;
    import flash.events.Event;
    import flash.net.SharedObject;
    import game.components.Level;
    import game.components.SuccessPanel;
    import game.puzzles.Puzzle;
    import game.puzzles.Puzzle1;
    import game.puzzles.Puzzle2;
    import game.puzzles.Puzzle3;
    import game.puzzles.Puzzle4;
    import game.puzzles.Puzzle5;
    import game.puzzles.Puzzle6;
    /**
     * Se encarga de gestionar los distintos niveles del juego
     * @author Alex
     */
    public class LevelManager
    {
        /**
         * El nivel actual
         */
        private var currentLevel:Level;

        /**
         * El puzzle actual
         */
        private var currentPuzzle:Puzzle;

        /**
         * Panel modal que deshabilita la interfaz
         */
        private var modal:Component;

        /**
         * Guarda el número de estrellas conseguidas en la última
comprobación exitosa de puzzle
         */
        private var stars:int;

        public function LevelManager()
        {

        }

        /**
         * Obtiene el nivel especificado
         * @param   levelNum    el número del nivel deseado
         * @return  el nivel deseado
         */
        public function getLevel(levelNum:int):Level
        {
```

```
            //creamos el nivel
            currentLevel = new Level(levelNum);

            //añadimos el puzzle del nivel
            addPuzzle(levelNum, currentLevel.workBench);

            //el nivel recién creado muestra las instrucciones
            currentLevel.showInstructions();

            return currentLevel;
        }

        /**
         * Añade el puzzle del nivel
         * @param   levelNum el número del nivel deseado
         * @param   workBench   la mesa de trabajo dónde colocar las
piezas del puzzle
         */
        private function addPuzzle(levelNum:int,
workBench:Component):void
        {
            //creamos el puzzle correspondiente a cada nivel
            switch (levelNum)
            {
                case 1:
                    currentPuzzle = new Puzzle1();
                    break;
                case 2:
                    currentPuzzle = new Puzzle2();
                    break;
                case 3:
                    currentPuzzle = new Puzzle3();
                    break;

                case 4:
                    currentPuzzle = new Puzzle4();
                    break;

                case 5:
                    currentPuzzle = new Puzzle5();
                    break;

                case 6:
                    currentPuzzle = new Puzzle6();
                    break;

                default:
            }

            //añadimos el puzzle a la mesa de trabajo
            currentPuzzle.parent = workBench.sprite;

            currentPuzzle.addEventListener(Puzzle.PLAY_UNTIL_STOP,
onPlayUntilStop);
            currentPuzzle.addEventListener(Puzzle.SUCCESS, onSuccess);
            currentPuzzle.addEventListener(Puzzle.FAIL, onFail);

            if (currentLevel.workbenchManager &&
!currentLevel.workbenchManager.hasEventListener(WorkbenchManager.INIT_
TEST)) {
```

```
currentLevel.workbenchManager.addEventListener(WorkbenchManager.INIT_T
EST, onInitTest);
            };


currentPuzzle.setPuzzleOnWorkbench(currentLevel.workbenchManager);
        }

        /**
         * Cuando se recibe la petición del nivel de iniciar la
comprobación de un puzzle
         * @param   e   evento de inicio de test de un puzzle
         */
        private function onInitTest(e:Event):void
        {
            //deshabilitar interfaz mientras se ejecuta el test
            modal = new Component(null, "modal_bg");
            modal.parent = currentLevel.parent;

            currentPuzzle.testLevel();
        }

        /**
         * Cuando la comprobación de un puzzle falla
         * @param   e   evento de fallo en puzzle
         */
        private function onFail(e:Event):void
        {
            trace("onFail");
            //dejamos de atender al evento de que se ha parado la Red
de Petri
            if (currentLevel.workbenchManager &&
currentLevel.workbenchManager.hasEventListener(WorkbenchManager.STOPPE
D)) {

currentLevel.workbenchManager.removeEventListener(WorkbenchManager.STO
PPED, onPetriNetStopped);
            };

            //detener la Red de Petri si está en marcha
            currentLevel.workbenchManager.stop();

            //habilitar interfaz, pues ha finalizado el test
            if (modal) {
                modal.parent = null;
            }
        }

        /**
         * Cuando la comprobación de un puzzle falla
         * @param   e   evento de fallo en puzzle
         */
        private function onSuccess(e:Event=null):void
        {
            //habilitar interfaz tras finalizar el test
            if (modal) {
                modal.parent = null;
            }
```

```
            //Ventana de Puzzle completado
            var piecesUsed:int =
currentLevel.workBench.sprite.numChildren - getBasePiecesByLevel();
            stars = getStarsByLevel(piecesUsed);

            //panel de victoria
            var successPanel:SuccessPanel = new
SuccessPanel(piecesUsed, stars);
            successPanel.parent = currentLevel.parent;

            //centrado
            successPanel.x = (successPanel.parent.width -
successPanel.width) / 2;
            successPanel.y = (successPanel.parent.height -
successPanel.height) / 2;

            //con modal detrás
            var modalBG:Component = new Component(null, "modal_bg");
            successPanel.sprite.addChildAt(modalBG.sprite, 0);

            modalBG.x = -successPanel.x;
            modalBG.y = -successPanel.y;

            //guardamos la puntuación obtenida, si supera la anterior
            var mySharedObject:SharedObject =
SharedObject.getLocal("petrinetgame");
            if (mySharedObject.data["puzzle" + currentLevel.levelNum]
!= undefined && mySharedObject.data["puzzle" + currentLevel.levelNum]
< stars) {
                mySharedObject.data["puzzle"+currentLevel.levelNum] =
stars;
                mySharedObject.flush();
            }

            //cuando quitemos el panel de victoria

successPanel.sprite.addEventListener(Event.REMOVED_FROM_STAGE,
onSuccessPanelRemovedFromStage);
        }

        /**
         * Devuelve las estrellas ganadas en cada nivel según el
número de piezas usadas
         * @param   piecesUsed  piezas usadas en el nivel
         * @return  las estrellas ganadas
         */
        private function getStarsByLevel(piecesUsed:int):int
        {
            stars = 1;
            switch (currentLevel.levelNum)
            {
                case 1:
                    stars = 3;
                    break;

                case 2:
                    if (piecesUsed <= 4) {
                        stars = 3;
                    } else if (piecesUsed <= 5) {
                        stars = 2
```

```
                }
                break;

            case 3:
                if (piecesUsed <= 3) {
                    stars = 3;
                } else if (piecesUsed <= 4) {
                    stars = 2;
                }
                break;

            case 4:
                if (piecesUsed <= 6) {
                    stars = 3;
                } else if (piecesUsed <= 9) {
                    stars = 2;
                }
                break;

            case 5:
                if (piecesUsed <= 4) {
                    stars = 3;
                } else if (piecesUsed <= 10) {
                    stars = 2;
                }
                break;

            case 6:
                if (piecesUsed <= 20) {
                    stars = 3;
                } else if (piecesUsed <= 24) {
                    stars = 1;
                }
            default:
        }

        return stars;
    }

    /**
     * Devuelve las piezas con las que empieza el puzzle
     * @return  el número de piezas con las que empieza el puzzle
     */
    private function getBasePiecesByLevel():int
    {
        switch (currentLevel.levelNum)
        {
            case 1:
                return 30;
                break;

            case 2:
                return 10;
                break;

            case 3:
                return 7;
                break;

            case 4:
```

```
                    return 5;
                    break;

                case 5:
                    return 5;
                    break;

                case 6:
                    return 34;
                    break;

                default:
            }

            return 0;
        }

        /**
         * Cuando la ventana de victoria se cierra
         * @param   e   evento de sacado de escena
         */
        private function onSuccessPanelRemovedFromStage(e:Event):void
        {
            (e.target).removeEventListener(Event.REMOVED_FROM_STAGE,
onSuccessPanelRemovedFromStage);

            //salimos del nivel actual
            currentLevel.exit();
        }

        /**
         * Cuando el puzzle solicita que se ejecute la Red de Petri
del usuario hasta pararse
         * @param   e   evento de puzzle
         */
        private function onPlayUntilStop(e:Event):void
        {
            trace("onPlayUntilStop");
            if (currentLevel.workbenchManager &&
!currentLevel.workbenchManager.hasEventListener(WorkbenchManager.STOPP
ED)) {

currentLevel.workbenchManager.addEventListener(WorkbenchManager.STOPPE
D, onPetriNetStopped);
            };

            currentLevel.workbenchManager.play();
        }

        /**
         * Cuando la Red de Petri del usuario se detiene
         * @param   e   evento de la mesa de trabajo
         */
        private function onPetriNetStopped(e:Event):void
        {
            trace("onPetriNetStopped");
            currentPuzzle.continueTest();
        }

    }
```

```
}
```

```
package game.managers
{
    import core.Component;
    import flash.display.Stage;
    import flash.events.Event;
    import flash.events.EventDispatcher;
    import flash.events.MouseEvent;
    import flash.events.TimerEvent;
    import flash.geom.Point;
    import flash.utils.Timer;
    import game.components.ArrowedArc;
    import game.components.Place;
    import game.components.SpecialPlace;
    import game.components.SpecialTransition;
    import game.components.ToolsBar;
    import game.components.Transition;
    import game.events.ToolsBarEvent;

    /**
     * Gestiona la edición de una Red de Petri dentro de un Workbench,
con una barra de herramientas
     * @author Alex
     */
    public class WorkbenchManager extends EventDispatcher
    {
        /**
         * se informa de que la Red de Petri del usuario se ha parado
         */
        static public const STOPPED:String = "STOPPED";

        /**
         * se pide iniciar el test
         */
        static public const INIT_TEST:String = "INIT_TEST";

        /**
         * Margen del decorador del cursor respecto del cursor
         */
        private const MOUSE_DECORATOR_MARGIN:int = 5;

        /**
         * Tiempo (en milisegundos) tras la última transición
disparada para volver a comprobar la siguiente
         */
        private const CHECK_NEXT_TRIGGER_TIME:int = 1000;


        /**
         * Número límite de transiciones que se pueden disparar sin
detener la Red de Petri del usuario
         */
        public var transitionsTriggerLimit:int = 4;

        /**
         * Timer de espera para la comprobación/ejecución de otra
transición mientras la Red de Petri del usuario está en marcha
         */
        private var playTimer:Timer;
```

```
        /**
         * Timer de espera para indicar que la Red de Petri del
usuario se ha detenido
         */
        private var stopTimer:Timer;

        /**
         * Escena de la mesa de trabajo
         */
        private var _stage:Stage;

        /**
         * Fondo en el que colocar los componentes de la Red de Petri
         */
        private var _workbench:Component;

        /**
         * Barra de herramientas
         */
        private var _toolsBar:ToolsBar;

        /**
         * Gestor de arcos
         */
        private var _arcManager:ArcManager;

        /**
         * Vector de transiciones. Ordenado por prioridad
         */
        private var _transitions:Vector.<Transition>;

        /**
         * Estado actual del editor
         */
        private var _currentState:String;

        /**
         * El último lugar clicado
         */
        private var clickedPlace:Place;

        /**
         * La última transición clicada
         */
        private var clickedTransition:Transition;

        /**
         * Decorador del cursor (para saber qué herramienta tenemos
seleccionada)
         */
        private var cursorDecorator:Component;

        /**
         * Arco falso que une el componente inicial del arco con el
cursor
         */
        private var fakeArc:ArrowedArc;

        /**
```

```
         * Contador de las transiciones que se han disparado desde la
última vez que se inició la ejecución de la Red de Petri del usuario
         * Se utiliza para limitar el número de transiciones que
pueden dispararse en la Red de Petri del usuario
         */
        private var transitionsTriggered:int;

        //ESTADOS POSIBLES
        /**
         * Cuando se está ejecutando la Red de Petri del usuario
         */
        private const PLAY:String = "PLAY";

        /**
         * Cuando se detiene la Red de Petri del usuario
         */
        private const STOP:String = "STOP";

        /**
         * La herramienta es el cursor básico
         */
        private const CURSOR:String = "CURSOR";

        /**
         * La herramienta es añadir lugares
         */
        private const PLACE:String = "PLACE";

        /**
         * La herramienta es añadir transiciones
         */
        private const TRANSITION:String = "TRANSITION";

        /**
         * La herramienta es añadir arcos
         */
        private const ARC:String = "ARC";

        /**
         * La herramienta es añadir arcos y ya está el lugar inicial
escogido
         */
        private const PLACE_OUT_ARC:String = "PLACE_OUT_ARC";

        /**
         * La herramienta es añadir arcos y ya está la transición
inicial escogida
         */
        private const TRANSITION_OUT_ARC:String =
"TRANSITION_OUT_ARC";

        /**
         * La herramienta es borrar
         */
        private const ERASER:String = "ERASER";

        /**
         * La herramienta es añadir marcas
         */
        private const ADD_MARK:String = "ADD_MARK";
```

```
        /**
         * La herramienta es quitar marcas
         */
        private const REMOVE_MARK:String = "REMOVE_MARK";


        /**
         * Constructor
         * @param   workbench mesa de trabajo
         * @param   toolsBar    barra de herramientas
         */
        public function WorkbenchManager(workbench:Component,
toolsBar:ToolsBar)
        {
            _arcManager = new ArcManager();

            _transitions = new Vector.<Transition>;

            _workbench = workbench;

            //gestionamos los eventos de la mesa de trabajo y de la
escena
            if (_workbench) {
                _workbench.addEventListener(MouseEvent.CLICK,
onWorkbenchClick);


_workbench.sprite.addEventListener(Event.ADDED_TO_STAGE,
function():void {
                    if (_workbench.sprite && _workbench.sprite.stage)
{
                        _stage = _workbench.sprite.stage;
                        _stage.addEventListener(MouseEvent.MOUSE_MOVE,
onMouseMoveOnStage);
                    }
                });


_workbench.sprite.addEventListener(Event.REMOVED_FROM_STAGE,
function():void {
                    currentState = CURSOR;
                });
            }

            _toolsBar = toolsBar;

            addToolsBarListeners();

            currentState = CURSOR;
        }

        /**
         * Añade los listeners de la barra de herramientas
         */
        private function addToolsBarListeners():void
        {
            if (_toolsBar) {
                _toolsBar.addEventListener(ToolsBarEvent.CURSOR,
onCursor);
```

```
                _toolsBar.addEventListener(ToolsBarEvent.ADD_PLACE,
onAddPlace);

_toolsBar.addEventListener(ToolsBarEvent.ADD_TRANSITION,
onAddTransition);
                _toolsBar.addEventListener(ToolsBarEvent.ADD_ARC,
onAddArc);
                _toolsBar.addEventListener(ToolsBarEvent.ERASER,
onEraser);
                _toolsBar.addEventListener(ToolsBarEvent.ADD_MARK,
onAddMark);
                _toolsBar.addEventListener(ToolsBarEvent.REMOVE_MARK,
onRemoveMark);
                _toolsBar.addEventListener(ToolsBarEvent.PLAY,
onPlay);
                _toolsBar.addEventListener(ToolsBarEvent.STOP,
onStop);
            }
        }

        /**
         * Quita los listeners de la barra de herramientas
         */
        private function removeToolsBarListeners():void
        {
            //quitamos todos los listeners excepto del play y stop
            if (_toolsBar) {
                _toolsBar.removeEventListener(ToolsBarEvent.CURSOR,
onCursor);
                _toolsBar.removeEventListener(ToolsBarEvent.ADD_PLACE,
onAddPlace);

_toolsBar.removeEventListener(ToolsBarEvent.ADD_TRANSITION,
onAddTransition);
                _toolsBar.removeEventListener(ToolsBarEvent.ADD_ARC,
onAddArc);
                _toolsBar.removeEventListener(ToolsBarEvent.ERASER,
onEraser);
                _toolsBar.removeEventListener(ToolsBarEvent.ADD_MARK,
onAddMark);

_toolsBar.removeEventListener(ToolsBarEvent.REMOVE_MARK,
onRemoveMark);
            }
        }

        /**
         * Cuando la barra de herramientas pide STOP
         * @param   e evento de la barra de herramientas
         */
        private function onStop(e:ToolsBarEvent=null):void
        {
            currentState = STOP;
        }

        /**
         * Cuando la barra de herramientas pide PLAY (iniciar test)
         * @param   e evento de la barra de herramientas
         */
        public function onPlay(e:ToolsBarEvent=null):void
```

```
        {
            dispatchEvent(new Event(INIT_TEST));

            //currentState = PLAY;
        }

        /**
         * Cuando la barra de herramientas pide la herramienta CURSOR
         * @param   e evento de la barra de herramientas
         */
        private function onCursor(e:ToolsBarEvent=null):void
        {
            currentState = CURSOR;
        }

        /**
         * Cuando la barra de herramientas pide la herramienta PLACE
         * @param   e evento de la barra de herramientas
         */
        private function onAddPlace(e:ToolsBarEvent=null):void
        {
            currentState = PLACE;
        }

        /**
         * Cuando la barra de herramientas pide la herramienta
TRANSITION
         * @param   e evento de la barra de herramientas
         */
        private function onAddTransition(e:ToolsBarEvent=null):void
        {
            currentState = TRANSITION;
        }

        /**
         * Cuando la barra de herramientas pide la herramienta ARC
         * @param   e evento de la barra de herramientas
         */
        private function onAddArc(e:ToolsBarEvent=null):void
        {
            currentState = ARC;
        }

        /**
         * Cuando la barra de herramientas pide la herramienta ERASER
         * @param   e evento de la barra de herramientas
         */
        private function onEraser(e:ToolsBarEvent=null):void
        {
            currentState = ERASER;
        }

        /**
         * Cuando la barra de herramientas pide la herramienta
ADD_MARK
         * @param   e evento de la barra de herramientas
         */
        private function onAddMark(e:ToolsBarEvent=null):void
        {
            currentState = ADD_MARK;
```

```
        }

        /**
         * Cuando la barra de herramientas pide la herramienta
REMOVE_MARK
         * @param   e evento de la barra de herramientas
         */
        private function onRemoveMark(e:ToolsBarEvent=null):void
        {
            currentState = REMOVE_MARK;
        }

        /**
         * Al recibir un click en la mesa de trabajo
         * @param   e evento de ratón
         */
        private function onWorkbenchClick(e:MouseEvent=null):void
        {
            //trace("_workbench.sprite.numChildren -> " +
_workbench.sprite.numChildren);
            //obtenemos el punto donde se ha hecho click relativo a la
mesa de trabajo
            var localPoint:Point = _workbench.sprite.globalToLocal(new
Point(_stage.mouseX, _stage.mouseY));

            switch (_currentState)
            {
                case PLACE:
                    createNewPlace(localPoint);
                    break;

                case TRANSITION:
                    var newTransition:Transition =
createNewTransition(localPoint);
                    addTransition(newTransition);
                    break;

                default:
                    //trace(_currentState + " state not implemented");
            }
        }

        /**
         * Crea una nueva transición en el punto escogido
         * @param   localPoint  el punto dónde colocar la transición
         * @return  la transición creada
         */
        private function
createNewTransition(localPoint:Point):Transition
        {
            var newTransition:Transition = new Transition();
            newTransition.addEventListener(MouseEvent.CLICK,
onTransitionClick);
            newTransition.draggable = true;
            newTransition.parent = _workbench.sprite;
            newTransition.x = localPoint.x + MOUSE_DECORATOR_MARGIN;
            newTransition.y = localPoint.y + MOUSE_DECORATOR_MARGIN;

            return newTransition;
        }
```

```
        /**
         * Añade una transición especial
         * @param   specialTransition la transición especial
         */
        public function
addSpecialTransition(specialTransition:SpecialTransition):void
        {
            specialTransition.addEventListener(MouseEvent.CLICK,
onSpecialTransitionClick);
            specialTransition.parent = _workbench.sprite;
        }

        /**
         * Cuando le hacen click a una transición especial
         * @param   e   evento de ratón
         */
        private function onSpecialTransitionClick(e:MouseEvent):void
        {
            trace("onSpecialTransitionClick");

            switch (currentState)
            {
                case ARC:
                    clickedTransition = Transition(e.target);
                    //iniciaremos el arco como arco de salida de la
transición
                    currentState = TRANSITION_OUT_ARC;
                    fakeArc = _arcManager.addArc(clickedTransition,
cursorDecorator);
                    fakeArc.sprite.alpha = 0.5;
                    break;
                case PLACE_OUT_ARC:
                    clickedTransition = Transition(e.target);
                    //acabaremos el arco como arco de entrada de la
transición
                    var arc:ArrowedArc =
_arcManager.addArc(clickedPlace, clickedTransition);
                    if(arc){
                        arc.addEventListener(MouseEvent.CLICK,
onArcClick);
                    }
                    //y volveremos al estado de iniciar otro arco
                    currentState = ARC;
                    break;
                default:
            }
        }

        /**
         * Crea un nuevo lugar en el punto escogido
         * @param   localPoint  el punto dónde colocar la transición
         */
        private function createNewPlace(localPoint:Point):void
        {
            var newPlace:Place = new Place();
            newPlace.addEventListener(MouseEvent.CLICK, onPlaceClick);
            newPlace.draggable = true;
            newPlace.parent = _workbench.sprite;
            newPlace.x = localPoint.x + MOUSE_DECORATOR_MARGIN;
```

```
            newPlace.y = localPoint.y + MOUSE_DECORATOR_MARGIN;
        }

        /**
         * Añade un lugar especial
         * @param   specialPlace    el lugar especial
         */
        public function
addSpecialPlace(specialPlace:SpecialPlace):void
        {
            specialPlace.addEventListener(MouseEvent.CLICK,
onSpecialPlaceClick);
            specialPlace.parent = _workbench.sprite;
        }

        /**
         * Cuando le hacen click a un lugar especial
         * @param   e   evento de ratón
         */
        private function onSpecialPlaceClick(e:MouseEvent):void
        {
            trace("onSpecialPlaceClick");

            switch (currentState)
            {
                case ARC:
                    clickedPlace = Place(e.target);
                    //iniciaremos el arco como arco de salida del
lugar
                    currentState = PLACE_OUT_ARC;
                    fakeArc = _arcManager.addArc(clickedPlace,
cursorDecorator);
                    fakeArc.sprite.alpha = 0.5;
                    break;
                case TRANSITION_OUT_ARC:
                    clickedPlace = Place(e.target);
                    //acabaremos el arco como arco de entrada del
lugar
                    var arc:ArrowedArc =
_arcManager.addArc(clickedTransition, clickedPlace);
                    if(arc){
                        arc.addEventListener(MouseEvent.CLICK,
onArcClick);
                    }
                    //y volveremos al estado de iniciar otro arco
                    currentState = ARC;
                    break;
                case ADD_MARK:
                    clickedPlace = Place(e.target);
                    //añadimos 1 marca al lugar clicado
                    clickedPlace.addMark();
                    break;
                case REMOVE_MARK:
                    clickedPlace = Place(e.target);
                    //quitamos 1 marca al lugar clicado
                    clickedPlace.removeMark();
                    break;
                default:
            }
        }
```

```
        /**
         * Añade un arco especial
         * @param   startComponent  componentes del que sale el arco
         * @param   endComponent    componente al que llega el arco
         */
        public function addSpecialArc(startComponent:Component,
endComponent:Component):void
        {
            var arc:ArrowedArc = _arcManager.addArc(startComponent,
endComponent);
            arc.isUserErasable = false;
        }

        /**
         * Añade una transición a la lista de transiciones que se
dispararán automáticamente con la Red de Petri en marcha
         * @param   newTransition   la nueva transición a insertar
         */
        private function addTransition(newTransition:Transition):void
        {
            _transitions.push(newTransition);

            //actualizamos su prioridad
            newTransition.priority = _transitions.length - 1;
        }

        /**
         * Quita una transición de la lista de transiciones que se
dispararán automáticamente con la Red de Petri en marcha
         * Se actualizan las prioridades del resto de transiciones
         * @param   transition la transición a eliminar
         */
        private function removeTransition(transition:Transition):void
        {
            for (var i:int = 0; i < _transitions.length; i++)
            {
                _transitions[i].priority = i;
                if (_transitions[i] == transition) {
                    _transitions.splice(i, 1);
                    i--;
                }
            }
        }

        /**
         * Al hacer click en un lugar
         * @param   e   evento de ratón
         */
        private function onPlaceClick(e:MouseEvent):void
        {
            trace("onPlaceClick");

            switch (currentState)
            {
                case ARC:
                    clickedPlace = Place(e.target);
                    //iniciaremos el arco como arco de salida del
lugar
                    currentState = PLACE_OUT_ARC;
```

```
                    fakeArc = _arcManager.addArc(clickedPlace,
cursorDecorator);
                    fakeArc.sprite.alpha = 0.5;
                    break;
                case TRANSITION_OUT_ARC:
                    clickedPlace = Place(e.target);
                    //acabaremos el arco como arco de entrada del
lugar
                    var arc:ArrowedArc =
_arcManager.addArc(clickedTransition, clickedPlace);
                    if(arc){
                        arc.addEventListener(MouseEvent.CLICK,
onArcClick);
                    }
                    //y volveremos al estado de iniciar otro arco
                    currentState = ARC;
                    break;
                case ERASER:
                    clickedPlace = Place(e.target);
                    clickedPlace.removeEventListener(MouseEvent.CLICK,
onPlaceClick);
                    _arcManager.removeAllArcsFrom(clickedPlace);
                    clickedPlace.parent = null;
                    break;
                case ADD_MARK:
                    clickedPlace = Place(e.target);
                    //añadimos 1 marca al lugar clicado
                    clickedPlace.addMark();
                    break;
                case REMOVE_MARK:
                    clickedPlace = Place(e.target);
                    //quitamos 1 marca al lugar clicado
                    clickedPlace.removeMark();
                    break;
                default:
            }
        }

        /**
         * Al hacer click en una transición
         * @param   e   evento de ratón
         */
        private function onTransitionClick(e:MouseEvent):void
        {
            trace("onTransitionClick");

            switch (currentState)
            {
                case ARC:
                    clickedTransition = Transition(e.target);
                    //iniciaremos el arco como arco de salida de la
transición
                    currentState = TRANSITION_OUT_ARC;
                    fakeArc = _arcManager.addArc(clickedTransition,
cursorDecorator);
                    fakeArc.sprite.alpha = 0.5;
                    break;
                case PLACE_OUT_ARC:
                    clickedTransition = Transition(e.target);
                    //acabaremos el arco como arco de entrada de la
```

```
transición
                    var arc:ArrowedArc =
_arcManager.addArc(clickedPlace, clickedTransition);
                    if(arc){
                        arc.addEventListener(MouseEvent.CLICK,
onArcClick);
                    }
                    //y volveremos al estado de iniciar otro arco
                    currentState = ARC;
                    break;
                case ERASER:
                    clickedTransition = Transition(e.target);

clickedTransition.removeEventListener(MouseEvent.CLICK,
onTransitionClick);
                    _arcManager.removeAllArcsFrom(clickedTransition);
                    clickedTransition.parent = null;
                    removeTransition(clickedTransition);
                    break;
                default:
            }
        }

        /**
         * Al hacer click en un arco
         * @param   e   evento de ratón
         */
        private function onArcClick(e:MouseEvent):void
        {
            switch (currentState)
            {
                case ERASER:
                    var clickedArc:ArrowedArc = ArrowedArc(e.target);
                    if(clickedArc.isUserErasable){
                        _arcManager.removeArcDirectly(clickedArc);
                        clickedArc.parent = null;
                    }
                break;
                default:
            }
        }

        /**
         * Cuando se mueva el ratón por la escena
         * @param   e   evento de ratón
         */
        private function onMouseMoveOnStage(e:MouseEvent):void
        {
            //si tenemos decorador del cursor, lo colocamos dónde el
cursor con un margen
            if (cursorDecorator && _stage) {
                cursorDecorator.x = _stage.mouseX +
MOUSE_DECORATOR_MARGIN;
                cursorDecorator.y = _stage.mouseY +
MOUSE_DECORATOR_MARGIN;
            }

            //si tenemos un arco falso, movemos la punta del arco
falso a dónde el cursor, con un margen
            if (fakeArc && cursorDecorator) {
```

```
                fakeArc.endPoint = new Point(_stage.mouseX +
MOUSE_DECORATOR_MARGIN, _stage.mouseY + MOUSE_DECORATOR_MARGIN);
            }
        }

        public function get currentState():String
        {
            return _currentState;
        }

        public function set currentState(value:String):void
        {
            if (value != _currentState) {
                //quitamos el decorador actual del cursor, si lo hay
                if (cursorDecorator) {
                    cursorDecorator.parent = null;
                    cursorDecorator = null;
                }

                //si teníamos el arco falso siguiendo al cursor, lo
quitamos
                if (_arcManager && fakeArc) {
                    _arcManager.removeArcDirectly(fakeArc);
                }

                _currentState = value;
                //trace("_currentState = " + _currentState);

                switch (_currentState)
                {
                    case PLACE:
                        cursorDecorator = new Place();
                        break;

                    case TRANSITION:
                        cursorDecorator = new Component(null,
"transition_background");
                        break;

                    case ARC:
                        cursorDecorator = new Component(null,
"black_arrow");
                        cursorDecorator.sprite.scaleX = 0.5;
                        cursorDecorator.sprite.scaleY = 0.5;
                        break;

                    case PLACE_OUT_ARC:
                        cursorDecorator = new Transition();
                        cursorDecorator.sprite.scaleX = 0;
                        cursorDecorator.sprite.scaleY = 0;
                        break;

                    case TRANSITION_OUT_ARC:
                        cursorDecorator = new Place();
                        cursorDecorator.sprite.scaleX = 0;
                        cursorDecorator.sprite.scaleY = 0;
                        break;

                    case ERASER:
                        cursorDecorator = new Component(null,
```

```
"eraser");
                        cursorDecorator.sprite.scaleX = 0.5;
                        cursorDecorator.sprite.scaleY = 0.5;
                        break;

                    case ADD_MARK:
                        cursorDecorator = new Component(null,
"add_mark_decorator");
                        cursorDecorator.sprite.scaleX = 0.5;
                        cursorDecorator.sprite.scaleY = 0.5;
                        break;

                    case REMOVE_MARK:
                        cursorDecorator = new Component(null,
"remove_mark_decorator");
                        cursorDecorator.sprite.scaleX = 0.5;
                        cursorDecorator.sprite.scaleY = 0.5;
                        break;

                    case PLAY:
                        //nada
                        break;

                    case STOP:
                        stop();
                        break;

                    default:
                        //nothing here
                }

                //si queremos un decorador en el cursor, lo colocamos
en la posición del cursor con un margen, medio transparente
                if (cursorDecorator) {
                    cursorDecorator.parent = _stage;
                    cursorDecorator.x = _stage.mouseX +
MOUSE_DECORATOR_MARGIN;
                    cursorDecorator.y = _stage.mouseY +
MOUSE_DECORATOR_MARGIN;
                    cursorDecorator.sprite.alpha = 0.5;
                }

            }
        }

        /**
         * Se van disparando las transiciones en orden de prioridad,
hasta que no se puede disparar ninguna
         */
        public function play():void
        {
            trace("play");
            currentState = PLAY;
            transitionsTriggered = 0;
            checkNextTransition();
        }

        /**
         * Comprueba la siguiente transición que debe dispararse, y la
dispara si se cumplen los requisitos
```

```
         */
        private function checkNextTransition():void
        {
            //trace("checkNextTransition");
            //si no hemos llegado al límite de disparos
            if (transitionsTriggered < transitionsTriggerLimit) {

                var transition:Transition;
                //recorremos todas las transiciones (del usuario)
                for (var i:int = 0; i < _transitions.length; i++)
                {
                    transition = _transitions[i];
                    //si está habilitada
                    if (transition.enabled) {
                        transitionsTriggered++;
                        //preparamos el timer de espera para la
próxima comprobación
                        playTimer = new Timer(CHECK_NEXT_TRIGGER_TIME,
1);
                        playTimer.addEventListener(TimerEvent.TIMER,
onPlayTimerTick);
                        playTimer.start();

                        transition.checkTrigger();
                        return;
                    }
                }
            }

            //los disparon se paran
            currentState = STOP;
        }

        /**
         * Cuando el timer de espera entre comprobaciones de disparo
de transiciones se dispara
         * @param   e   evento de timer
         */
        private function onPlayTimerTick(e:TimerEvent):void
        {
            playTimer.removeEventListener(TimerEvent.TIMER,
onPlayTimerTick);
            checkNextTransition();
        }

        /**
         * Detiene el disparo de más transiciones
         */
        public function stop():void
        {
            trace("stop");
            if (playTimer) {
                playTimer.removeEventListener(TimerEvent.TIMER,
onPlayTimerTick);
                playTimer.stop();
            }

            stopTimer = new Timer(CHECK_NEXT_TRIGGER_TIME, 1);
            stopTimer.addEventListener(TimerEvent.TIMER, delayedStop);
            stopTimer.start();
```

```
        }

        /**
         * Cuando el timer de espera para informar de que la Red de
Petri del usuario se ha detenido
         * @param   e   evento de timer
         */
        private function delayedStop(e:TimerEvent):void
        {
            stopTimer.removeEventListener(TimerEvent.TIMER,
delayedStop);
            dispatchEvent(new Event(STOPPED));
            currentState = CURSOR;
        }

    }

}
```

Por último, la clase principal de la aplicación, que se encarga cargar los assets necesarios para la aplicasción, inicializa las clases principales y gestiona la escena principal mostrando el nivel correspondiente o la pantalla inicial de selección de nivel.

```
package game
{
    import assets.AssetsLibrary;
    import core.Component;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.ProgressEvent;
    import flash.system.Security;
    import game.components.Level;
    import game.components.LevelSelectPanel;
    import game.events.LevelInfoEvent;
    import game.managers.LevelManager;
    import util.UrlUtil;

    /**
     * Aplicación de PetriNetsGame
     * @author Alex
     */
    public class Main extends Sprite
    {
        //La url desde la que se está ejecutando la aplicación
        private var baseUrl:String;

        //Gestor de niveles
        private var levelManager:LevelManager;

        /**
         * El panel que está visualizándose en este momento
         */
        private var currentPanel:Component;

        public function Main():void
        {
            levelManager = new LevelManager();

            Security.allowDomain('*');

            //si no nos está cargando nadie
            if (!loaderInfo.loader) {
                //nos inicializamos solos
                init();
            }

            if (stage) onAddedToStage();
            else addEventListener(Event.ADDED_TO_STAGE,
onAddedToStage);
        }

        /**
         * Al ser añadido a escena
         * @param   e   evento de añadido a escena
         */
        private function onAddedToStage(e:Event=null):void
        {
            removeEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
```

```
            //gestión de stage ...
        }

        /**
         * Inicializa la aplicación
         */
        public function init():void
        {
            //obtenemos la ruta dónde está la aplicación como ruta
base
            baseUrl =
UrlUtil.getFolderUrlFromFileUrl(root.loaderInfo.url);
            trace("baseUrl = " + baseUrl);

            loadAssets();
        }

        /**
         * Carga los assets necesarios para la aplicación
         */
        private function loadAssets():void
        {
            AssetsLibrary.addExternalSwf(mainGraphicsUrl);

            AssetsLibrary.addEventListener(ProgressEvent.PROGRESS,
onAssetsLoadProgress);
            AssetsLibrary.addEventListener(Event.COMPLETE,
onAssetsLoadComplete);

            AssetsLibrary.loadPendingLibraries();
        }

        /**
         * Cuando progresa la carga de los assets de la aplicación
         * @param   e   evento de progreso
         */
        private function onAssetsLoadProgress(e:ProgressEvent):void
        {
            //replicamos el evento hacia arriba, para que el loader
sepa cómo va la carga de assets de la aplicación
            dispatchEvent(e);
        }

        /**
         * Cuando se completa la carga de assets
         * @param   e   evento de carga completada
         */
        private function onAssetsLoadComplete(e:Event):void
        {
            //replicamos el evento hacia arriba, para que el loader
sepa que la aplicación ya tiene sus assets necesarios cargados
            dispatchEvent(e);

            showLevelSelectPanel();
        }

        /**
         * Muestra el panel de selección de nivel
         * @param   e   evento de navegación de salir de un nivel
         */
```

```
        private function showLevelSelectPanel(e:Event = null):void
        {
            //si hay algún panel mostrándose lo quitamos
            if (currentPanel) {
                currentPanel.parent = null;
            }

            //creamos y mostramos el panel de selección de nivel
            currentPanel = new LevelSelectPanel();
            currentPanel.addEventListener(LevelInfoEvent.LEVEL_INIT,
onLevelInit);
            currentPanel.parent = this;
        }

        /**
         * Cuando recibimos un evento de navegación de entrar a un
nivel
         * @param   e   evento de navegación de ir a un nivel
         */
        private function onLevelInit(e:LevelInfoEvent):void
        {
            trace ("init level -> " + e.levelNum);
            //si hay algún panel mostrándose lo quitamos
            if (currentPanel) {
                currentPanel.parent = null;
            }

            //le pedimos el nivel deseado al gestor de niveles, y
mostramos el nivel
            currentPanel = levelManager.getLevel(e.levelNum);
            currentPanel.addEventListener(Level.EXIT,
showLevelSelectPanel);
            currentPanel.parent = this;
        }

        /**
         * Ruta de los gráficos principales
         */
        private function get mainGraphicsUrl():String
        {
            //hardcodemos la ruta de los assets en Webnode porque
gestiona los archivos subidos en servidores distintos, sin unificar
rutas
            if (baseUrl.indexOf("webnode") != -1) {
                return
"http://files.petrinetgame.webnode.es/200000031-
b06ffb261e/main_gfx.swf" + UrlUtil.getNoCacheTag();
            } else {
                return baseUrl + "gfx/main_gfx.swf";
            }
        }
    }

}
```