INGENIERÍA INFORMÁTICA

Juegos en Java

Realizado por:

Alejandro Raposo López

Dirigido por:

José Ramón Portillo Fernández

Departamento

Matemática Aplicada I

Sevilla, DICIEMBRE, 2012

Agradecimientos

No puedo comenzar de otra manera que agradeciendo a mis padres su inestimable apoyo, no sólo económico, sino emocional durante todos estos años. Sin ellos, no habría llegado hasta donde hoy me encuentro.

A mis amigos y compañeros, por su apoyo, sugerencias y tantas horas de dudas resueltas.

Por supuesto, al tutor de proyecto José Ramón Portillo Fernández, por su guía y camino que me han llevado a buen fin en la realización de este proyecto.

Y a todos aquellos anónimos de internet, que tantas dudas me han resueltos cuando no sabía dónde acudir.

En definitiva, gracias a todas las personas que han colaborado para hacer posible que estas aplicaciones puedan haberse terminado y serán quizás, en un futuro, referencia para nuevos proyectos.

Índice

1. Definición de objetivos

El proyecto consistirá en la implementación de dos conocidos juegos: Picross y Pumba, aplicando los conocimientos y destrezas que cada una requieren.

Picross, es un juego de lógica cuyo objetivo es rellenar un tablero de juego de acuerdo a unas indicaciones de los márgenes.

El juego dispondrá de tres niveles de dificultad: fácil, normal y difícil. Acorde al tamaño del tablero.

Se implementará además un temporizador para recoger el tiempo empleado en la resolución del problema. Con ello, se pretende estimular al jugador para lograr resolver los juegos en cada vez menos tiempo.

En segundo lugar, Pumba, es la implementación de un popular juego de cartas consistente en tratar de quedarse sin ellas antes de que lo hagan tus rivales. Algunas cartas no aportarán ningún valor añadido para el juego, sin embargo, otras aplicarán un efecto a la partida.

Computacionalmente supone mayor desafío, ya que, al estar rivalizando contra oponentes, se deberán aplicar estrategias e inteligencia artificial.

Por ejemplo, la carta 1, permite hacer robar al jugador de su elección una carta, en cierto momento de la partida es conveniente usarla para evitar que un jugador a punto de ganar alcance su objetivo. Pero dicho jugador puede devolverle la jugada aumentando el efecto, entrando así en un duelo por la permanencia.

2. Análisis temporal y de costes de desarrollo

La elaboración del presente proyecto requiere unos tiempos de desarrollo repartidos entre distintas tareas. Si bien, el tiempo se ve incrementado debido a las situaciones de duda y el coste temporal de solucionarlas.

Las distintas categorías son las siguientes:

- *Investigación*: Tiempo de aprendizaje de conocimientos y resolución de dudas.
- *Implementación*: Tiempo dedicado al desarrollo del software.
- *Pruebas*: Tiempo de revisión de objetivos y detección de fallos o defectos en el producto final.
- *Documentación*: Tiempo dedicado a la elaboración del presente documento.

Las horas son aproximaciones de los tiempos acumulados día tras día a lo largo del desarrollo.

• Picross

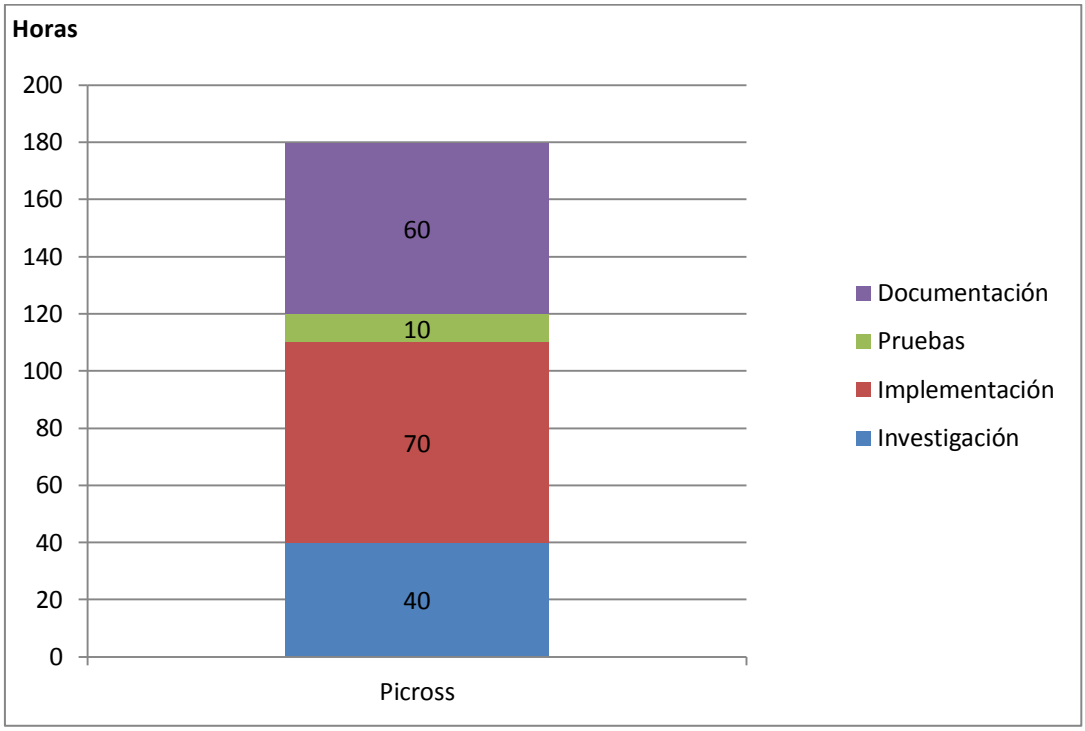


Ilustración 1. Picross

Como se aprecia en la ilustración 1, el tiempo total empleado para el desarrollo de Picross es de *180 horas*, repartidas en las distintas categorías.

Los costes de investigación, para esta aplicación incluyen: aprendizaje de la API gráfica de Java, librerías Swing y manejo de eventos.

Implementación refiere al número de horas de programación.

Los costes de pruebas engloban el tiempo sometido a la aplicación para verificar el correcto funcionamiento, visualización e implantación en distintos equipos.

La fase de documentación corresponde con la elaboración de las secciones pertinentes del presente documento.

- ## Pumba

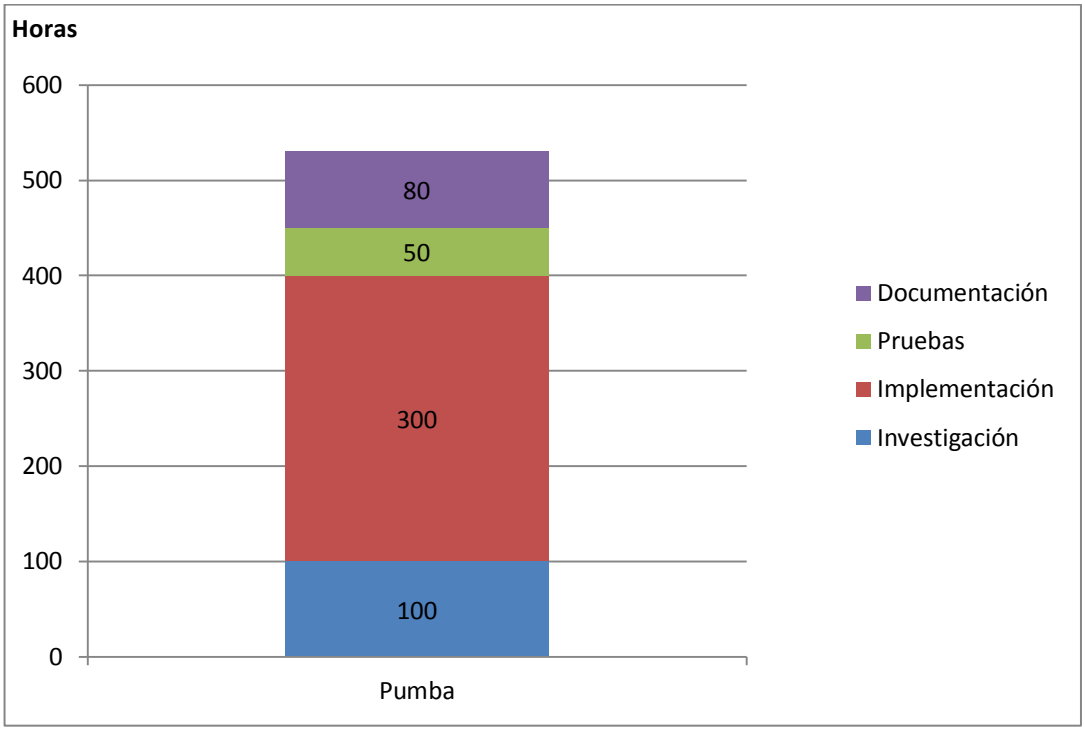


Ilustración 2

Como se aprecia en la ilustración 2, el tiempo total empleado para el desarrollo de Pumba es de *530 horas*, repartidas en las distintas categorías.

Los costes de investigación, para esta aplicación incluyen: aprendizaje extenso de la API gráfica de Java, librerías Swing, manejo de eventos, patrón de diseño modelo-vista-controlador (MVC) e inteligencia artificial.

Implementación refiere al número de horas de programación.

Los costes de pruebas engloban el tiempo sometido a la aplicación para verificar el correcto funcionamiento, visualización e implantación en distintos equipos.

La fase de documentación corresponde con la elaboración de las secciones pertinentes a dicha aplicación del presente documento.

A continuación se muestra en la ilustración 3 una visión conjunta de los tiempos de desarrollo de ambas aplicaciones:

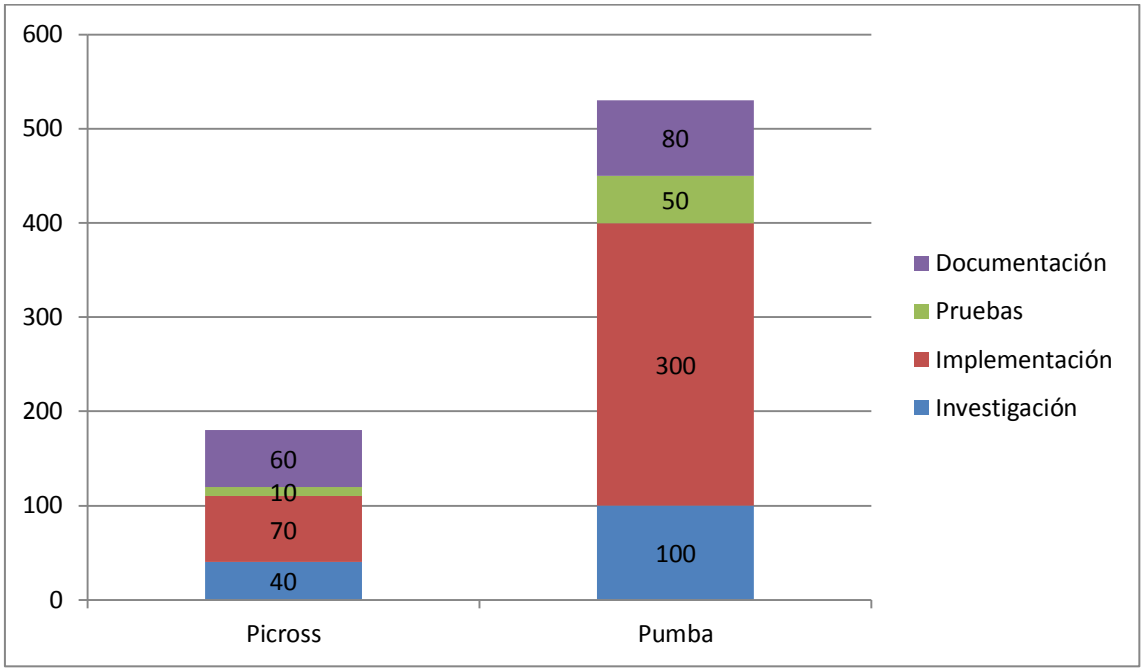


Ilustración 3

Como se aprecia comparativamente, la aplicación Pumba supone mayores costes en una proporción aproximada 3:1.

Finalmente puede apreciarse en la ilustración 4 los tiempos totales de desarrollo del proyecto:

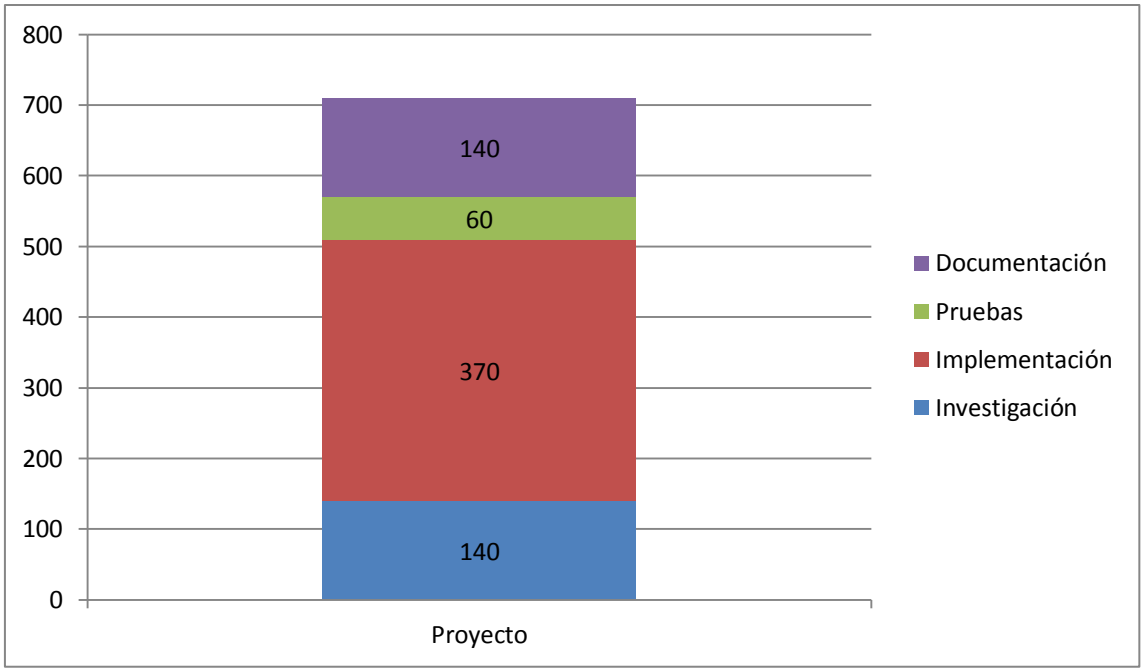


Ilustración 4

Los tiempos totales dedicados al proyecto ascienden a **710 horas**, desglosadas en las categorías de la siguiente manera:

- Documentación: 140 horas.
- Pruebas: 60 horas.
- Implementación: 370 horas.
- Investigación: 140 horas.

3. Análisis de antecedentes y aportación realizada

- ### Picross

En 1987, Non Ishida, un editor de gráficos japoneses, ganó un concurso en Tokio por las imágenes utilizando el diseño de rejilla de luces de rascacielos que se activa o se desactiva. Coincidentemente, un profesional japonés de puzles llamado Tetsuya Nishio inventó el este mismo rompecabezas denominado originalmente Nonogram.

Nonogram, también conocido como Hanjie, Paint by Numbers, Griddlers, etc. son rompecabezas de imágenes de lógica en el que las celdas en una red debe estar en blanco o de color de acuerdo con los números a la izquierda y arriba de la red para revelar una imagen oculta. En este tipo puzzle, los números miden la cantidad de celdas continuas coloredas que hay en cualquier fila o columna. Por ejemplo, una pista de "4 8 3" significaría que hay grupos de cuatro, ocho y tres plazas llenas, en ese orden, con al menos un cuadrado en blanco entre los grupos sucesivos.

Estos rompecabezas son con frecuencia blancos y negros, pero también puede ser de color. Si está coloreado, las pistas de números son también de color para indicar el color de los cuadrados. Dos números de diferentes colores pueden tener un espacio entre ellas. Por ejemplo, un negro de cuatro seguido de un rojo dos podría significar cuatro cajas negras, algunos espacios vacíos, y dos cajas de color rojo, o podría simplemente significar cuatro cajas negras seguido inmediatamente por dos rojas.

Nonograms no tienen límites teóricos sobre el tamaño, y no se limitan a los diseños cuadrados.

Para la primera aplicación, Picross, pueden encontrarse implementaciones en la red, pero todas ellas sujetas a una determinada tecnología, sistema operativo o plataforma.

La intención de éste proyecto es, por la naturaleza de su sencillez y gracias al lenguaje de programación empleado, hacer el juego apto para usar en distintos sistemas operativos tanto de arquitecturas PC como MAC, así como en dispositivos móviles.

- ## Pumba

Entre los juegos populares de naipes en España se encuentra Pumba, también conocido como chúpate dos, hijoputa o más lejanamente, uno.

Las reglas son diversas según el lugar donde se juegue, sin llegar a un consenso en el reglamento.

Por nuestra parte, Pumba, no dispone de ninguna implementación en la red de tan popular juego de cartas, por lo que el aporte es totalmente novedoso.

Al igual que ocurre en el Picross, se han empleado estructuras muy sencillas que permitan ejecutar el juego en distintos entornos, incluso en sistemas de bajo rendimiento, pudiendo ser el juego accesible al máximo número de jugadores.

4. Análisis de requisitos, diseño e implementación

4.1 Análisis de requisitos

• Picross

- *Requisitos funcionales*: El sistema deberá generar de manera aleatoria tableros de juego a resolver.
  El sistema deberá implementa un contador de tiempo.

- *Requisitos de información*: El sistema deberá almacenar el estado del tablero solución, el del tablero a resolver así como el estado de cada una de las celdas que los componen.

- *Restricciones*: El sistema deberá limitar el número de tableros a resolver a uno por cada aplicación, aunque pueden ejecutarse más de una aplicación al mismo tiempo.

- *Requisitos de interfaz*: El sistema no necesita establecer comunicación con otros subsistemas.

- *Requisitos no funcionales*: El sistema deberá funcionar en ordenadores personales independientemente del sistema operativo o el entorno gráfico siempre que sean compatibles con la máquina virtual Java.

• Pumba

- *Requisitos funcionales*: El sistema deberá generar partidas distintas para cada ejecución, esto es reparto aleatorio de cartas para los jugadores.
  El sistema deberá proporcionar el tiempo suficiente para que el jugador tome decisiones sin apremiar al mismo.
  El sistema deberá implementar inteligencia artificial para los jugadores controlados por ordenador.

- *Requisitos de información*: El sistema deberá almacenar información sobre los siguientes conceptos relevantes: carta (palo y número), baraja, jugador (nombre, controlador humano/máquina, reparto de cartas) y juego (turno actual, cartas a robar en el turno, sentido de la partida, mazo de robar, mazo de descarte, última jugada, número de jugadores y clasificación de los jugadores ganadores).

- *Restricciones*: El sistema deberá garantizar el orden de los turnos de acuerdo a las reglas del juego.

- *Requisitos de interfaz*: El sistema no necesita establecer comunicación con otros subsistemas.

- *Requisitos no funcionales*: El sistema deberá funcionar en ordenadores personales independientemente del sistema operativo o el entorno gráfico siempre que sean compatibles con la máquina virtual Java.

4.2 Diseño

Para la aplicación Picross, aunque resulte relativamente sencillo las fases de diseño e implementación se pondrán de manifiesto el uso de antipatrones de diseño y la falta de un ciclo de vida de software.

Un antipatrón es un patrón de diseño que invariablemente conduce a una mala solución para un problema.

Al documentarse los antipatrones, además de los patrones de diseño, se dan argumentos a los diseñadores de sistemas para no escoger malos caminos, partiendo de documentación disponible en lugar de simplemente la intuición.

El término se origina inspirado en el libro Design Patterns, escrito por un grupo de autores conocido como Gang of Four, y que aglutina un conjunto de buenas soluciones de programación. Los autores bautizaron dichas soluciones con el nombre de "patrones de diseño" por analogía con el mismo término, usado en arquitectura. El libro Anti-Patterns (de William Brown, Raphael Malveau, Skip McCormick y Tom Mowbray, junto con la más reciente incorporación de Scott Thomas) describe los antipatrones como la contrapartida natural al estudio de los patrones de diseño. El estudio formal de errores que se repiten permite reconocer y reconducir los elementos involucrados hacia una mejor solución. Los antipatrones no se mencionan en el libro original de Design Patterns, puesto que éste es anterior.

Los antipatrones se consideran una parte importante de una buena práctica de programación. Es decir, un buen programador procurará evitar los antipatrones siempre que sea posible, lo que requiere su reconocimiento e identificación tan pronto como sea posible, dentro del ciclo de vida del software.

El concepto de antipatrón se puede aplicar a la ingeniería en general, e incluso a cualquier tarea realizada por el hombre. Aunque no se escucha con frecuencia fuera del campo ingenieril, la noción está ampliamente extendida.

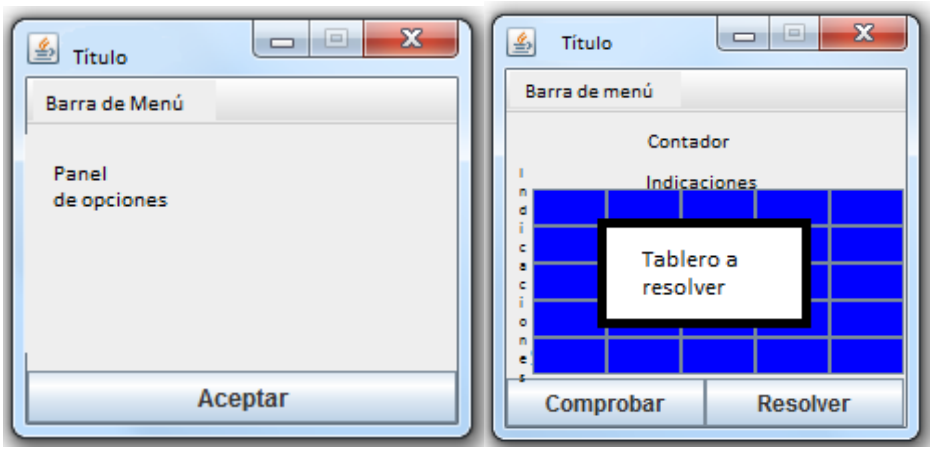
Una vez comprobadas en la aplicación Picross las dificultades que presentan, se obtará para la aplicación Pumba por el uso de patrones de diseño y un determinado ciclo de vida.

- ## Picross

El diseño de interfaz de la aplicación se basa en el concepto minimalista, simple y e intuitivo que permita al jugador inexperto iniciar una partida de manera rápida, evitando así información y mensajes superfluos.

Internamente, siguiendo los objetivos de hacer una aplicación ligera, para los equipos más modestos, se ha optado por el empleo de componentes ligeros propios de las librerías Swing de Java. Principalmente JPanel, JButton y JMenu.

Inicialmente la interfaz presentará la barra de título, una barra de menú con las opciones pertinentes del juego, un panel de opciones con la dificultad de la partida y el botón de comenzar.

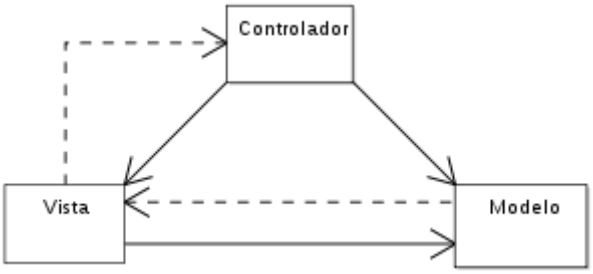


- ## Pumba

El diseño general se basa en el patrón **MVC**:

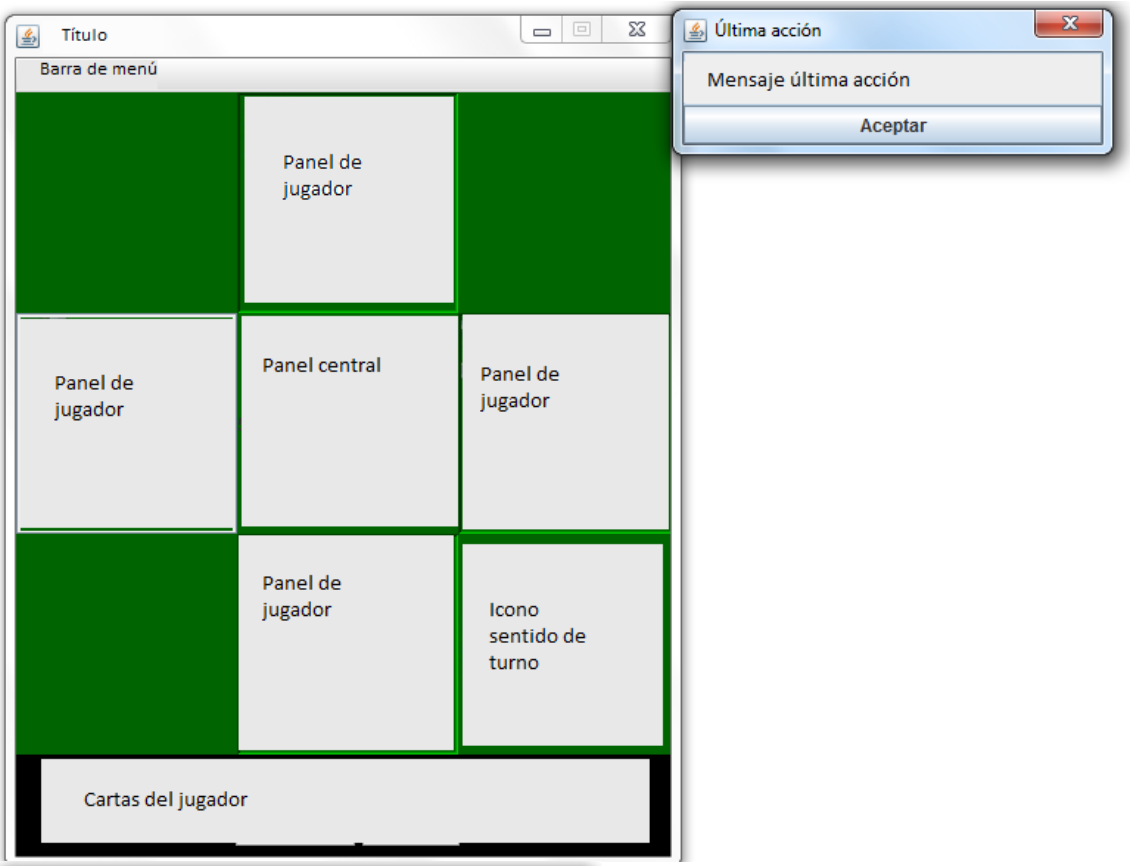
Modelo Vista Controlador (MVC) es un patrón o modelo de abstracción de desarrollo de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos.

- **Modelo**: Esta es la representación específica de la información con la cual el sistema opera. En resumen, el modelo se limita a lo relativo de la *vista* y su *controlador* facilitando las presentaciones visuales complejas. El sistema también puede operar con más datos no relativos a la presentación, haciendo uso integrado de otras lógicas de negocio y de datos afines con el sistema modelado.
- **Vista**: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.
- **Controlador**: Este responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y, probablemente, a la vista.



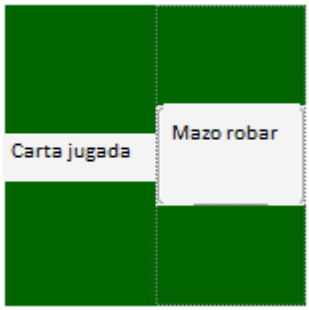
En la aplicación que nos ocupa, el componente *vista* está diseñado como una única ventana conteniendo al principio un panel donde el usuario introduce su nombre para la partida. Posteriormente la vista quedará dividida en zonas bien diferenciadas para cada uno de los jugadores que participan en la partida. Cada una de las zonas muestran el tipo de control que usan (humano o máquina), el nombre del jugador y el número de cartas que posee. Finalmente se muestra una ventana donde aparece el orden de ganadores de la partida.

La vista proporciona, además, una barra de menú con diferentes acciones: iniciar una partida, salir del juego, mostrar las reglas del juego y una ayuda acerca de la aplicación.


Título
Barra de menú
Panel de jugador
Panel de jugador
Panel central
Panel de jugador
Panel de jugador
Icono sentido de turno
Cartas del jugador
Última acción
Mensaje última acción
Aceptar


Título
Barra de menú
Área para el nombre
Logo del juego
Comenzar

Panel Jugador 1



Panel Central 1

El componente *modelo* recoge las reglas del juego: sucesión de turnos, efectos de las cartas y en su caso la inteligencia artificial de la máquina.

La inteligencia artificial consta de las decisiones cuando obligan a un jugador a robar cartas, pudiendo devolver el efecto acumulado y redirigir el efecto en caso de ser posible (carta uno). Otra de las funciones de la inteligencia artificial es dictar el orden de despojo de las cartas cuando un jugador controlado por la máquina echa una carta de rey (carta número 13).

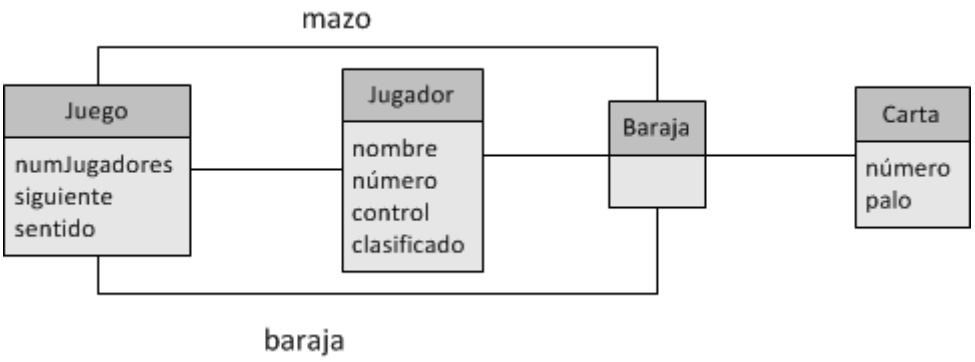


Diagrama de Clases 1

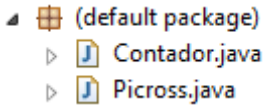
El *controlador* recoge los eventos generados por la vista, concretamente los botones de aceptar turno, introducir el nombre, robar carta, cerrar ventanas (instrucciones, acerca de la aplicación y ventana principal), accionar menús (juego, ayuda), elementos de menú (juego nuevo, salir del juego, mostrar instrucciones y mostrar acerca de la aplicación).

4.3 Implementación

No es intención de este documento aportar la totalidad del código fuente empleado en la implementación de las aplicaciones, sin embargo se comentarán las clases implicadas a modo de resumen e incluirán fragmentos de código que resulten interesantes por su naturaleza.

- **Picross**

La estructura de clases se muestra en la siguiente ilustración:



- Contador.java: Implementa el contador de tiempo de juego.
- Picross.java: Clase principal. Integra la lógica, la vista y el manejador de eventos.

Implementación del temporizador: anexo 1.

- ## Pumba

El esquema general para la parte de modelo de la aplicación corresponde a un algoritmo voraz, con la siguiente estructura:

```
inicializa()
mientras no fin()

        tratar()
fin-mientras
```

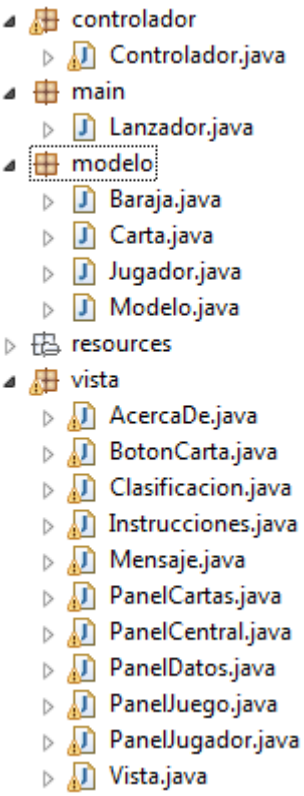
El cuerpo de cada una de las funciones es el siguiente:

- *inicializa():* Establece los valores iniciales para las variables.

- *fin()*: Condición de parada que se satisface cuando sólo un jugador posee cartas en la mano.

- *tratar()*: Establece un turno. Independientemente del control del jugador, debe echar una carta, si es posible, robar e incluso correr el turno.

Las clases empleadas en la implementación están reflejadas en la siguiente figura:



Existen 3 paquetes principales, correspondiendo a los 3 elementos del patrón MVC, son: modelo, vista y controlador; además de los paquetes "resources" y "main".

Sin entrar en detalles, comentaremos la función de cada una de las clases de sus correspondientes paquetes:

- Main

- *Lanzador.java*: Crea los objetos principales: modelo, vista y controlador, establece las relaciones entre ellas y lanza la aplicación.

- Resources

- Alberga las imágenes empleadas en la aplicación.

- Modelo

- *Baraja.java*: Modela una baraja como una colección de cartas y permite barajar.
- Carta.java: Modela la información sobre un naipe.
- *Jugador.java*: Recoge la información de un jugador, tales como el nombre y el reparto de cartas que posee.
- *Modelo.java*: Implementa la lógica de la aplicación y las funciones necesarias para la inteligencia artificiar de los jugadores controlados por la máquina.

- Vista

- *AcercaDe.java*: Muestra una ayuda acera del juego y su autor.

- *BotonCarta.java*: Botón que almacena una carta del reparto del jugador humano.
- *Clasificación.java*: Presenta en pantalla en orden descendente los ganadores del juego.
- *Instrucciones.java*: Permite visualizar las reglas del juego.
- *PanelCartas.java*: Alberga las cartas del jugador humano.
- *PanelCentral*.java: Muestra la última carta jugada y el mazo donde los jugadores robarán cartas.
- *PanelDatos.java*: Alberga el logotipo de la aplicación y recoge el nombre que usará el jugador.
- *PanelJuego.java*: Presenta las zonas de los jugadores, el panel central, el sentido del turno y el panel de cartas del jugador humano.
- *PanelJugador.java*: Muestra icono representativo del control del jugador (ordenador para control de máquina y monigote para control humano), el nombre del jugador y el número de cartas en la mano.
- *Vista.java*: Engloba la aplicación general.

    - Controlador
- *Controlador.java*: Manejador de eventos.

A continuación se incluirá los fragmentos de código que resultan interesantes:

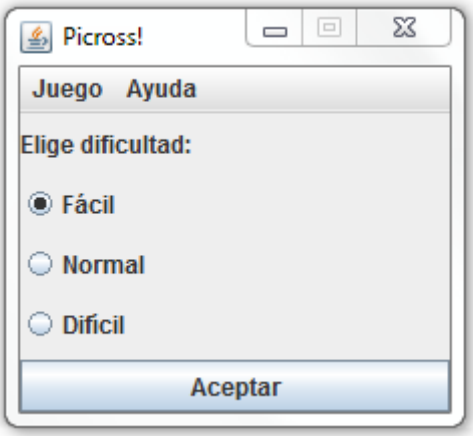
Parte de la inteligencia artificial: anexo 2

Inteligencia artificial aplicada a la carta 13: anexo 3.

5 Manual de usuario

• Picross

Tras ejecutar el juego, presenta la siguiente interfaz:



Por defecto viene seleccionada la dificultad fácil.
La dificultad elegida condiciona el tamaño del tablero a resolver, disponiendo de 3 dificultades: fácil, normal y difícil, con tableros de 5x5, 10x10 y 15x15 casillas respectivamente.

Una vez elegida la dificultad, pulsamos el botón “Aceptar“ y comienza el juego.

Como se aprecia en la parte superior, en color rojo, tenemos el contador de tiempo, que seguirá incrementándose mientras el tablero no esté resuelto.
En los márgenes de cada fila y columna aparecen las indicaciones de cuántos *grupos de casillas negras* se encuentran en dicha ficha o columna.
Por ejemplo, en la primera fila aparece la indicación “111” indicando que en dicha fila hay tres grupos de una casilla negra cada una.

Como es lógico dos grupos deben estar separados, al menos, por una casilla blanca, de lo contrario formarían un único grupo.
En la primera columna se muestra la indicación “21”, esto quiere decir que, de arriba a abajo, primero encontraremos un grupo de dos casillas negras y luego un grupo de una casilla negra.
Toda casilla no negra, es blanca.
Sumando el total de casillas de esa fila, llegamos a la siguiente conclusión: 2+1 casillas son negras, restan 2 que deben ser blancas. Entre cada grupo debe haber, al menos, una casilla blanca. Por consiguiente, resta dónde colocar una casilla blanca que complete la columna.

Marcar una casilla se hace con un simple clic de ratón, y quedará coloreada en negro, un siguiente clic colocará dicha casilla en blanco.
Tenemos además un nuevo color, gris, que pueden usarse para hacer “supuestos”. Ante indecisiones pensamos: supongamos que ésta casilla es negra, entonces otras tendrán tal color…
Tanto si llegamos a resolver la situación de duda como si llegamos a una contradicción, el supuesto inicial pasa a ser una seguridad que nos permite colorear la/s casilla/s con su color correspondiente.

Una vez completado el tablero, con sus casillas negras y blancas, se pulsa sobre el botón comprobar y podemos obtener uno de dos resultados:

- Es correcto (Figura 1): el tiempo se detiene y el botón comprobar pasa a colorearse en verde. ¡Enhorabuena! La próxima vez intenta resolverlo en menos tiempo o en una dificultad mayor.
- Es incorrecto (Figura 2): el tiempo no se detiene, sigue contando, y el botón comprobar pasa a colorearse en rojo. No se desanime, busque dónde ha podido cometer el error, soluciónelo y pulse en comprobar tantas veces como necesite.

Figura 1

Figura 2

Si por el contrario desiste en el intento pero desea conocer el resultado del tablero, puede pulsar en el botón resolver y obtendrá la solución.
Eso sí, el contador de tiempo se detendrá a un valor falseado, de tal forma que no podrá presumir de haber resuelto el ejercicio. Prueba de ello se muestra en la figura 3.

Figura 3

- **Pumba**

Una vez lanzada la aplicación, nos presenta la interfaz de la figura 4.

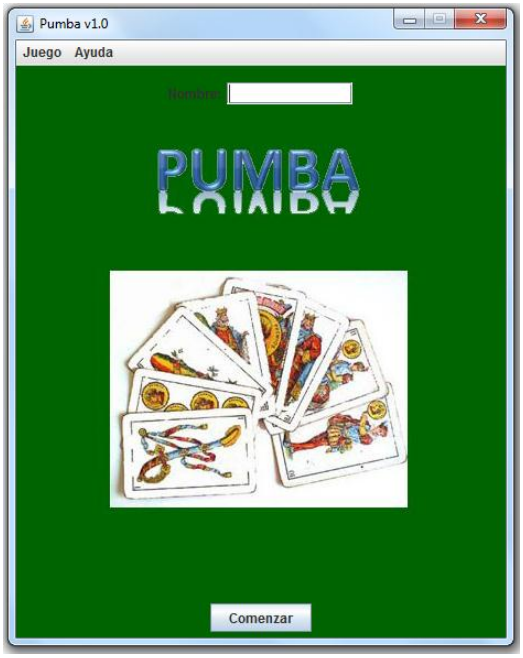


Figura 4

La interfaz muestra en la parte superior una barra de menú con opciones para reiniciar un juego, salir de la aplicación, mostrar las reglas del juego y una ayuda acerca del mismo.

En la zona central encontramos un campo de texto para introducir el nombre del jugador y un botón, comenzar, para empezar la partida.

Como se ilustra en la figura 5, el escenario de juego está dividido en una cuadrícula de 3x3 casillas en el centro y un panel en la zona inferior.

Cada casilla de jugador contiene un icono de un ordenador o un monigote para simbolizar el control, máquina o humano respectivamente. Además un número para indicar la cantidad de cartas que posee en ese momento.

El jugador que posee el turno tiene resaltado su recuadro en color blanco, frente al resto de jugadores que no presenta resalto.

En la esquina inferior derecha se muestra unas flechas ilustrando el sentido del turno actual.

En la parte inferior de la interfaz se muestra las cartas que el jugador humano tiene en ese momento. Como es el caso, los botones están desactivados ya que no es el turno de ese jugador.

Adicionalmente, aparece un mensaje auxiliar con la última acción realizada. En la figura vemos como el jugador “PC1” en su turno ha usado la carta “5 de oro”.

Este mensaje permanece hasta que el jugador pulse el botón.

De esta manera, el jugador humano dispone de cuánto tiempo necesite para reflexionar las jugadas de los demás jugadores y planificar la suya propia.

Como se ve en la figura 6, cuando es el turno del jugador humano, su casilla se resalta y activan las cartas que pueden ser jugadas en ese turno de acuerdo a la última jugada, que puede verse en el centro.

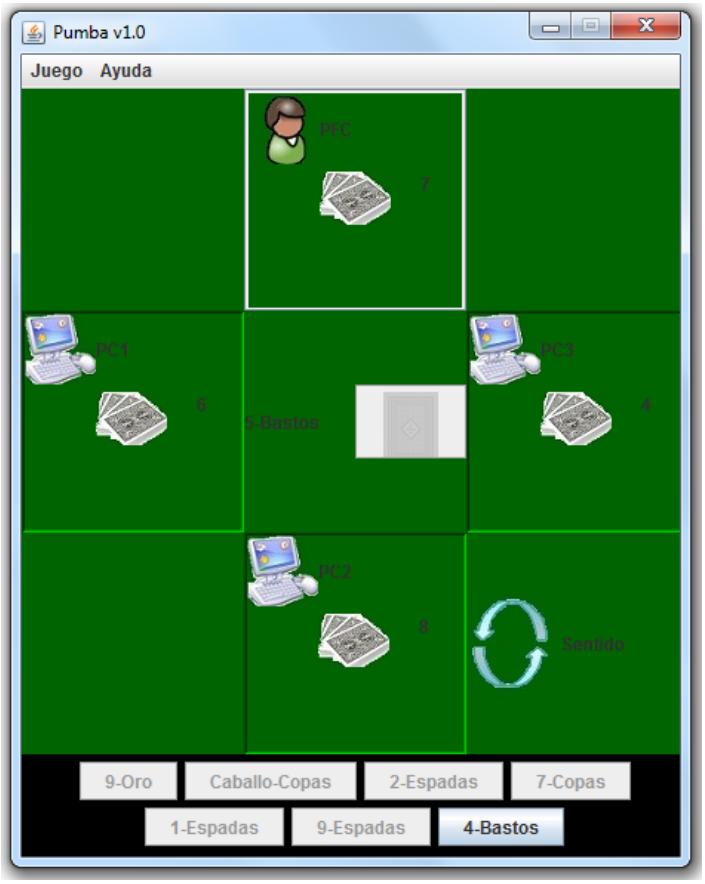


Figura 6

Una vez que todos los jugadores se han clasificado, la aplicación muestra el panel de ganadores, como se aprecia en la figura 7.

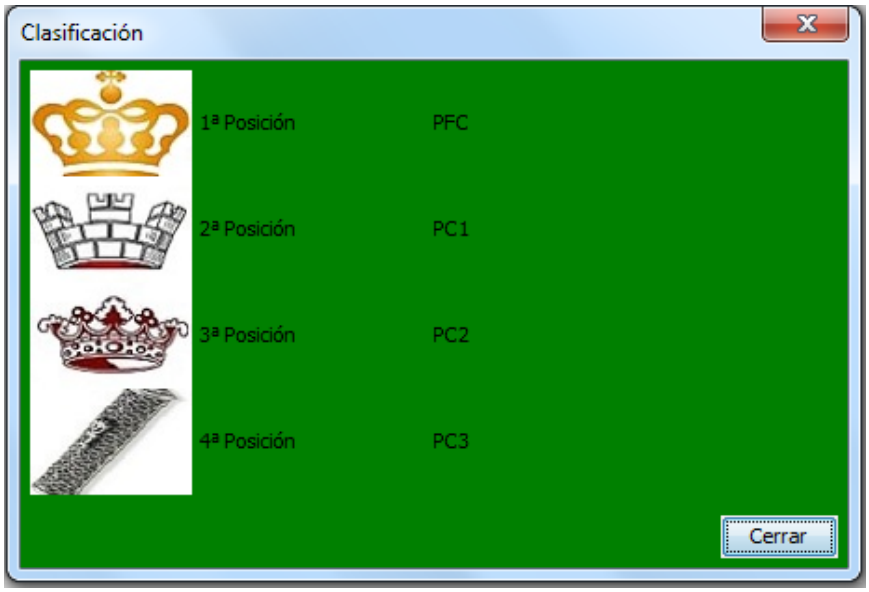


Figura 7

6 Pruebas

Ambas aplicaciones han sido sometidas a pruebas para garantizar su funcionamiento y estabilidad en distintos sistemas operativos y máquinas con características diferentes.

Los equipos probados son los siguientes:

- PC con sistema operativo Windows XP.
- PC con sistema operativo Windows 7.
- PC con sistema operativo Ubuntu (Linux).
- MAC.

Los resultados son satisfactorios en todos los casos.

7 Comparación con otras alternativas

- ## Picross

Esta aplicación puede encontrarse en diferentes formatos y precios, ya sea papel como en la versión impresa de la revista de pasatiempos Logic[1], en aplicación web[2] o disponible para consolas[3].

La versión impresa conlleva un coste económico. No está informatizada, por lo que su corrección automática no es posible. No aporta temporización automática que motive al jugador. Además el número de tableros a resolver está limitado por las páginas que la revista dedique al juego.

La aplicación web, presenta una interfaz e instrucciones en inglés, por lo que los nuevos usuarios que no dominen la lengua no podrán aprender a jugarlo. Por ser una aplicación propietaria e integrada en la web no puede ser descargada ni estar disponible en todo momento. Como en la versión impresa, el número de tableros es limitado, esta vez por la programación.

La versión de consola presenta los siguientes inconvenientes: es imprescindible disponer de la consola, que su precio junto al del juego puede ser prohibitivo. Igual que la aplicación web, el número de tableros a resolver es limitado, inherente a su programación. Tampoco dispone de contador de tiempo.

La aplicación, que en este proyecto se presenta una serie de ventajas: gratuito, reducido tamaño, interfaz sencilla y deductiva, multiplataforma y elementos estimulantes.

- ## Pumba

Esta aplicación, aunque famosa en su versión de naipes, no se encuentra implementada. El aporte que aquí se presenta es totalmente novedoso.

Debido al limitado tiempo y al hecho de realizar el proyecto individualmente, el juego implementa las características principales, pero podría ser ampliado con nuevas características: diseño de las cartas, sonidos, posibilidad de juego multijugador, chat, cambiar el número de jugadores, etc. Se dejará como ampliación para futuros alumnos que desearan retomar el proyecto.

8 Conclusiones

La intención de este proyecto es aplicar los conocimientos adquiridos a lo largo de los estudios de ingeniería así como demostrar las aptitudes para desarrollar una obra de dicha naturaleza y demostrar capacidades de adaptación y aprendizaje de nuevos conceptos.

Algunos de los conocimientos adquiridos que se han aplicado al presente proyecto varían desde la programación básica al uso de patrones de diseño con inteligencia artificial.

En la aplicación Picross se pone de manifiesto la problemática de los antipatrones cuando un proyecto adquiere ciertas dimensiones.

Es por eso que en la aplicación Pumba se ha requerido el uso de patrones y un proceso de ingeniería de software.

Si bien, la falta de experiencia propia y el limitado tiempo hacen patente que los productos sean mejorables y ampliables. Se deja para proyectos futuros la continuación de las aplicaciones que aquí se han presentado, por ejemplo la inclusión de sonido y suporte multijugador.

9 Bibliografía

[1] Juegos de lógica, Logic. Fecha consulta: 5 noviembre 2012.
http://www.demente.com/revistas/13-Logic.html

[2] Aplicación web Picross. Fecha consulta: 5 noviembre 2012.

http://www.minijuegos.com/Picross-Quest/7141

[3] Picross para Nintendo DS. Fecha consulta: 5 noviembre

http://www.game.es/Product/Default.aspx?SKU=048598

[4]API Java Swing. Fecha consulta: 14 Octubre 2012.

http://www.javahispano.org/portada/2011/7/5/java-basico-con-ejemplos.html

[5] Patrón MVC. Fecha consulta: 14 Octubre 2012

http://jc-mouse.blogspot.com.es/2011/12/patron-mvc-en-java-con-netbeans.html

[6] Modelo Vista Controlador. Fecha consulta: 5 noviembre 2012

http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

[7] Tutorial layouts. Fecha consulta: 14 Octubre 2012

http://www.javahispano.org/portada/2011/8/1/tutorial-de-layouts.html

10. Anexos

- ## Anexo 1: Contador.

```
public void contar(){
	tActual = System.currentTimeMillis();

	long tiempo = tActual - tReferencia;

	tiempo /= 1000;

	seg = (int)tiempo%60;
	min = (int)(tiempo-seg)/60;
}
```

- ## Anexo 2: parte de la inteligencia artificial.

```
private int jugadorMenosCartas() {
	int max = Integer.MAX_VALUE;
	int jug = 0;

	for (int i = 0; i < jugadores.size(); i++) {
		if (i != sig) {
			if (getJugador(i).size() < max) {
				jug = i;
				max = getJugador(i).size();
			}
		}
	}
	return jug;
}
```

- ## Anexo 3: *Inteligencia artificial aplicada a la carta 13.*

```
private void estrategiaRey(Carta c) {
   List<Carta> laux = jugTurno.getReparto();
   List<Carta> cands2 = new LinkedList<Carta>();
   Carta caux;
   for (int i = 0; i < jugTurno.size(); i++) {
	caux = (Carta) laux.get(i);
	if (caux.getPalo() == c.getPalo()) {
		if (caux.getNumero() > 2 && caux.getNumero() < 11
		    && caux.getNumero() != 7)
			cands1.add(caux);
		else
			cands2.add(caux);
		}
	}

	if (cands1.size() + cands2.size() == 0) {
```

```
                        try {
                                Thread.sleep(1000);
                        } catch (InterruptedException e) {
                                System.out.println("Expcepción Sleep.\nSaliendo");
                                System.exit(1);
                        }
                        Carta rob = null;
                        jugTurno.roba(baraja, mazo);
                        rob = jugTurno.getReparto().get(jugTurno.size() - 1);
                        if (rob.getPalo() == c.getPalo() || rob.getNumero() == 13)
                        {
                                mazo.add(rob);
                                jugTurno.remove(rob);
                                if (rob.getNumero() == 13)
                                        estrategiaRey(rob);
                                else {
}
                        } else {

                                try {
                                        Thread.sleep(1000);
                                } catch (InterruptedException e) {
                                        System.out.println("Expcepción
                                                Sleep.\nSaliendo");
                                        System.exit(1);
                                }
                                jugTurno.roba(baraja, mazo);
                                rob = (Carta)
                                jugTurno.getReparto().get(jugTurno.size() - 1);
                                if (rob.getPalo() == c.getPalo() ||
                                    rob.getNumero() == 13) {
                                        mazo.add(rob);
                                        jugTurno.remove(rob);
                                        if (rob.getNumero() == 13)
                                                estrategiaRey(rob);
                                        else {
                                                efecto(rob.getNumero());
                                        }
                                } else {
                                        setSig();

                                }
                        }
                } else {
                        Carta cart = null;
                        while (cands1.size() != 0) {
                                cart = (Carta) cands1.remove(0);
                                mazo.add(cart);
                                jugTurno.remove(cart);
                                try {
                                        Thread.sleep(1000);
                                } catch (InterruptedException e) {
                                        System.out.println("Expcepción
                                                Sleep.\nSaliendo");
                                        System.exit(1);
                                }
                        }
                        if (cands2.size() != 0) {
                                cart = (Carta) cands2.remove(0);
```

```
                        mazo.add(cart);
                        jugTurno.remove(cart);
                        efecto(cart.getNumero());

                } else
                        setSig();
        }
}
```