INSTITUTO DE EDUCACIÓN SECUNDARIA
I.E.S. CONSELLERIA - FAMILIA PROFESIONAL DE INFORMATICA Y COMUNICACIONES

Juego de la “Brisca” en HTML5, PHP y MySQL

PROYECTO DE DESARROLLO DE APLICACIONES WEB

NÚMERO DAW2013-08

Autor: Andrés Leone Gámez

Tutor individual: Mercedes Pellicer Gómez

ÍNDICE

IMÁGENES

1. Introducción

1.1. Objetivos

Este proyecto pretende conseguir la realización de un sitio web que permita al usuario jugar al juego de cartas la brisca tanto contra la máquina como contra otros usuarios haciendo uso de las tecnologías PHP, MySQL y HTML5.

Para ello, se realizará todo lo necesario para que los usuarios puedan registrarse e iniciar sesión en la web, podrán elegir jugar contra la máquina u otros usuarios así como la cantidad de rivales que varían de uno a tres. En caso de tratarse de una partida contra otros usuarios, estos contarán con un chat con el que podrán comunicarse entre sí.

Se almacenará el historial de los chats, el historial de las partidas contra otros usuarios, las puntuaciones máximas, cantidad de partidas ganadas y cantidad de partidas perdidas contra la máquina y contra otros usuarios.

Todo esto se realizará en una web capaz de representar en el navegador el juego de cartas y permitir que el usuario interactúe con el juego.

1.1.1. Resultados esperados

Se espera que la web permita que múltiples usuarios puedan tener una cuenta en la web que les permita jugar simultáneamente entre ellos, se guarden las estadísticas de las partidas que disputen, puedan comunicarse durante el transcurso de una partida y en caso de jugar contra la máquina que la partida que tenga lugar sea similar a la disputada contra un humano. Además, se espera que se pueda jugar contra la máquina sin necesidad de registrarse en el portal.

La web contendrá todo lo necesario para que, desde la interfaz mostrada en el navegador, el usuario pueda realizar todas las tareas descritas en los objetivos y de la forma más intuitiva posible.

1.1.2. Origen de la idea

La idea proviene del deseo del autor del proyecto por aprender a realizar un juego multijugador para navegador sin usar complementos para el navegador como pueden ser flash o java por la cantidad de ventajas que supone, entre ellas la habilidad de que el juego sea multiplataforma, siendo necesario únicamente un navegador compatible con HTML5, CSS3 y JS.

1.1.3. Justificación del proyecto

El proyecto consiste en la creación de una web funcional e intuitiva con las tecnologías PHP, MySQL, HTML5, CSS3 y JS aprendidas durante el transcurso del CFGS. Esto supondrá la ampliación de los conocimientos prácticos y teóricos aprendidos durante

el curso sobre dichas tecnologías además de la ampliación de una práctica de JAVA consistente en un chat multiusuario realizada durante el curso y la migración de la lógica de dicha práctica a otro lenguaje de programación. Por último, se realizará una Inteligencia artificial, siendo la primera vez que realizo una.

1.2. Requisitos mínimos

Es necesario un servidor compartido como mínimo con php y base de datos mysql, acceso a los archivos del sistema y la extensión PDO instalada con el driver de SQLite y soporte para el fichero htaccess para control del cache. En caso de usar un ordenador, este necesitaría mínimo 64 mb de ram, 500 MB de disco duro y conexión a internet.

1.3. Método de trabajo y organización del proyecto

Primero se realizará una web sin usuarios ni bases de datos en la que se pueda jugar únicamente contra la máquina usando JS para el control del juego. Esto se conseguirá mediante la programación de la lógica del juego y de la inteligencia artificial rival del usuario.

Una vez funcione correctamente se harán las modificaciones necesarias para incluir el modo de juego contra otros usuarios. Estas modificaciones serán realizar la parte de la web que manejará a los usuarios, trasladas la lógica del juego del cliente al servidor y realizar un chat para los usuarios que sea aprovecharlo para sincronizar la partida.

Es necesario que el modo de juego contra la máquina funcione sin ningún fallo antes de poder pasar a la siguiente fase ya que se convertirá el código JS necesario para la lógica del juego a PHP. Para conseguir esto se dividirá la lógica en funciones lo suficientemente simples como para poder comprobar que funcionan correctamente con facilidad.

2. Brisca

2.1. Descripción

La brisca es un juego de naipes que se juega con la baraja española. El juego requiere de 2 a 4 jugadores y puede jugarse por parejas dos a dos. Gana el jugador que consiga acumular la mayor cantidad de puntos. Las cartas con puntos son el 1, 3, 12 11 y 10 de los cuatro palos de la baraja, valiendo 11, 10, 4, 3 y 2 puntos cada una respectivamente, sumando un total de 120 puntos. Por tanto, una puntuación superior de 60 supone una victoria segura. De las 48 cartas de la baraja, el 2, 4, 6, 7, 8 y 9 no valen puntos pero si tienen valor en el juego a la hora de decidir quién se lleva las cartas en cada mano de la partida. El juego contempla la posibilidad de empate al ser los 120 puntos divisibles entre las distintas cantidades de oponentes, considerándose este resultado ni victoria ni derrota.

Se considerará que se conocen las normas de este juego en el resto de la memoria [1].

2.2. Variantes

Un problema importante de este juego es la falta de una normativa fija y por ello la existencia de distintas variantes. Algunas de ellas son grandes diferencias de juego al aplicar restricciones o permitir jugadas que cambian por completo estrategias del juego.

Las variantes elegidas para esta web son:

- No hay obligación de asistir (subir la carta más alta) a diferencia del juego de naipes llamado tute.
- No se puede intercambiar la carta de muestra por una carta de menor valor que se tenga en la mano.
- Por cada partida realizada la carta de muestra es elegida de forma aleatoria en lugar de ir rotando los palos de forma ordenada resultando en una cantidad aleatoria de sucesiones de cartas de muestra de un mismo palo para una sucesión de partidas consecutivas.
- En caso de poseer las cartas 12, 3 y 1 del palo de la carta de muestra en la primera mano de la partida el juego continúa en lugar de considerarse una victoria automática (Mano Negra o Conquista).

3. Juego contra máquina

3.1. Conocimientos previos

Antes de empezar debemos tener en cuenta qué necesitaremos a lo largo del proyecto. Lo que necesitaremos será las imágenes de las cartas, las posiciones que podrán ocupar las cartas, poder desplazar las cartas entre las posiciones, un marcador de puntos por cada jugador, un mensaje de victoria o derrota, poder elegir número de oponentes y un mensaje de victoria o derrota. Además se podrá especificar la dificultad de la inteligencia artificial.

Se necesita html para maquetar la zona de juego en el navegador. Por ello usaremos un elemento div por cada posición posible para las cartas junto con una id identificativa de la posición que representan para poder, mediante js, desplazar las cartas a una posición.

El posicionamiento de los divs de posición para las cartas se realizará usando css. Las posiciones serán posicionadas de forma absoluta y de forma relativa a la pantalla mediante porcentajes. Esto en un futuro permitiría fácilmente, mediante css media queries resituar las posiciones dependiendo del tamaño de la ventana del navegador y mejorando así la interfaz dependiendo de la pantalla. A su vez, la lógica del juego quedará aislada del posicionamiento de las cartas siendo necesario únicamente conocer los divs de posición, la clase del div de posición que define la posición del div y el ancho y el alto de la carta para poder centrarla en la posición requerida. Las animaciones de las cartas moviéndose en la pantalla se realizarán aprovechando la habilidad *transition* de CSS3, lo cual permitirá no hacer las animaciones usando javascript cargando este trabajo al navegador.

La mayor parte del trabajo en este punto se encontrará en el javascript. La lógica del juego y la inteligencia artificial serán escritas en javascript. La lógica del juego deberá ser escrita de forma que permita reiniciar la partida una vez terminada sin necesidad de actualizar la partida, tendrá que poder ser extendida con facilidad cuando lleguemos al punto de jugar contra otros jugadores (en lugar de decidir las jugadas, validarlas y decidir ganador desde javascript, se realizará desde el servidor y se enviará la decisión a javascript, por lo que estas instrucciones se aislarán a funciones lo suficientemente independientes para funcionar bajo las órdenes del modo de juego contra otros jugadores).

La mejor solución será la realización de un objeto para así separar variables y funciones a un scope, aislando el código y evitando que pueda sea modificado sin querer por culpa de variables globales.

3.2. Entorno de juego

Los elementos en el entorno de juego serán de 2 a 4 marcadores de puntuación, 22 posiciones a donde podrán moverse las cartas, un fondo el cual será un tapete verde similar a una mesa de juego, 48 imágenes de las cartas de la baraja española, un div con

espacio para informar de la victoria, derrota o empate como resultado de la partida y que permita reiniciar la partida o dejar de jugar y un menú que permita la selección de cantidad de rivales y dificultad antes de comenzar la partida.

3.2.1. Base

La base del javascript serán un conjunto de funciones capaces de hacer los cálculos más básicos a partir de cartas.

La nomenclatura de las cartas será la inicial del palo (Oro, Bastos, Espadas, Copas) en mayúsculas seguido del número de carta y se almacenará en una cadena de texto. Por ejemplo, la carta 7 de oros se escribiría *O7*.

El próximo paso es guardar en variables la información básica de las cartas. Estas variables almacenarán: Las 48 cartas posibles con palo y número, distintos números de las cartas, distintos palos de las cartas, relación entre número de carta y puntuación.

A continuación necesitamos funciones que operen con las cartas. Estas funciones se encargarán de: Obtener el número de una carta, Obtener el palo de una carta, calcular los puntos que vale una carta, calcular los puntos que valen un grupo de cartas, comprobar si de un grupo de cartas alguna es de un palo en concreto y por último, ordenar un grupo de cartas por valor (tanto numérica como de puntuación) dado el palo de la muestra y el palo que manda en la mesa (pudiendo ser similares o pudiendo faltar el palo que manda en la mesa).

Este conjunto de funciones requieren de las variables anteriormente mencionadas para poder hacer dichos cálculos. Estos cálculos permitirán decidir quién es el ganador de una mano del juego, calcular los puntos ganados por los jugadores y decidir el ganador. Además serán necesarias para que la inteligencia artificial pueda realizar comparaciones entre cartas.

Una vez hecho, se realizaron múltiples pruebas para comprobar que los valores retornados por las funciones eran los correctos. Gracias a lo simple de cada función fue sencillo comprobar cada una de ellas.

Nota: Los arrays y objetos en javascript almacenados en variables no se duplican al igualar dos variables. Algunas funciones requieren modificar algunos de los arrays anteriores por lo que es necesario llamar a la función *slice* del array pasando el parámetro *0* para obtener así una copia del array y evitar modificar el original, con la consecuencia de obtener errores en futuros cálculos que usen dicho array.

Nota: La función *parseInt* convierte una cadena de texto con números a un número. Esta función en caso de recibir un número que comience por 0 interpreta que se trata de un número en base 8. La única solución para trabajar en base 10.

3.2.2. Objeto IABrisca

Ahora que tenemos la base para realizar los cálculos, para evitar colisiones con variables futuras se aislará todo este código a un objeto llamado IABrisca. Este objeto será

el núcleo del código y será realizado de forma que pueda ser implementado en el proyecto mediante callbacks, prescindiendo así de modificar el código para implementar cambios que son independientes de la lógica del juego.

El objeto realizado es una función que para ser usada requiere de hacer un *new* a IABrisca en una variable.

Aprovechando que tenemos un objeto con todo lo necesario para hacer los cálculos para el juego, IABrisca será un contenedor de más funciones y objetos. Estos objetos se encargarán de controlar un jugador real, controlar un jugador manejado por inteligencia artificial, controlar un jugador manejado mediante órdenes del modo contra otros jugadores, controlar el estado de la mesa de juego y funciones vacías para los callback que se definirán después de crear el objeto.

3.3. Inteligencia Artificial (IA)

Una inteligencia artificial se puede definir como una entidad capaz de razonar por sí misma. En este caso se hará una inteligencia artificial muy básica cuyas salidas y entradas de información están muy limitadas y se conocen.

3.3.1. Objetivo

El objetivo es el de desarrollar una inteligencia artificial con el objetivo final de vencer a otros jugadores en el juego de la Brisca. Es importante que esta IA sea capaz de ganar con los menos fallos posibles en la lógica usada para tratar de vencer ya que más adelante se añadirá azar para así poner niveles de dificultad. Por tanto, la mayor dificultad será la IA sin azar. Aunque en este caso la decisión ha sido que el nivel más difícil sea la IA sin azar, esta no es necesariamente la mejor decisión ya que, al tratarse de un conjunto de órdenes preescritas estáticas, para esquivar el problema que supone la repetición de un patrón de forma continuada provocando que sea posible percatarse de los patrones que sigue la IA y poder así contrarrestarlos, es ideal incluir una ligera cantidad de azar en las decisiones, provocando pequeños fallos o jugadas inesperadas que se salen del patrón. La razón por la cual no ha sido realizado de esta forma es para poder competir con el patrón de la IA.

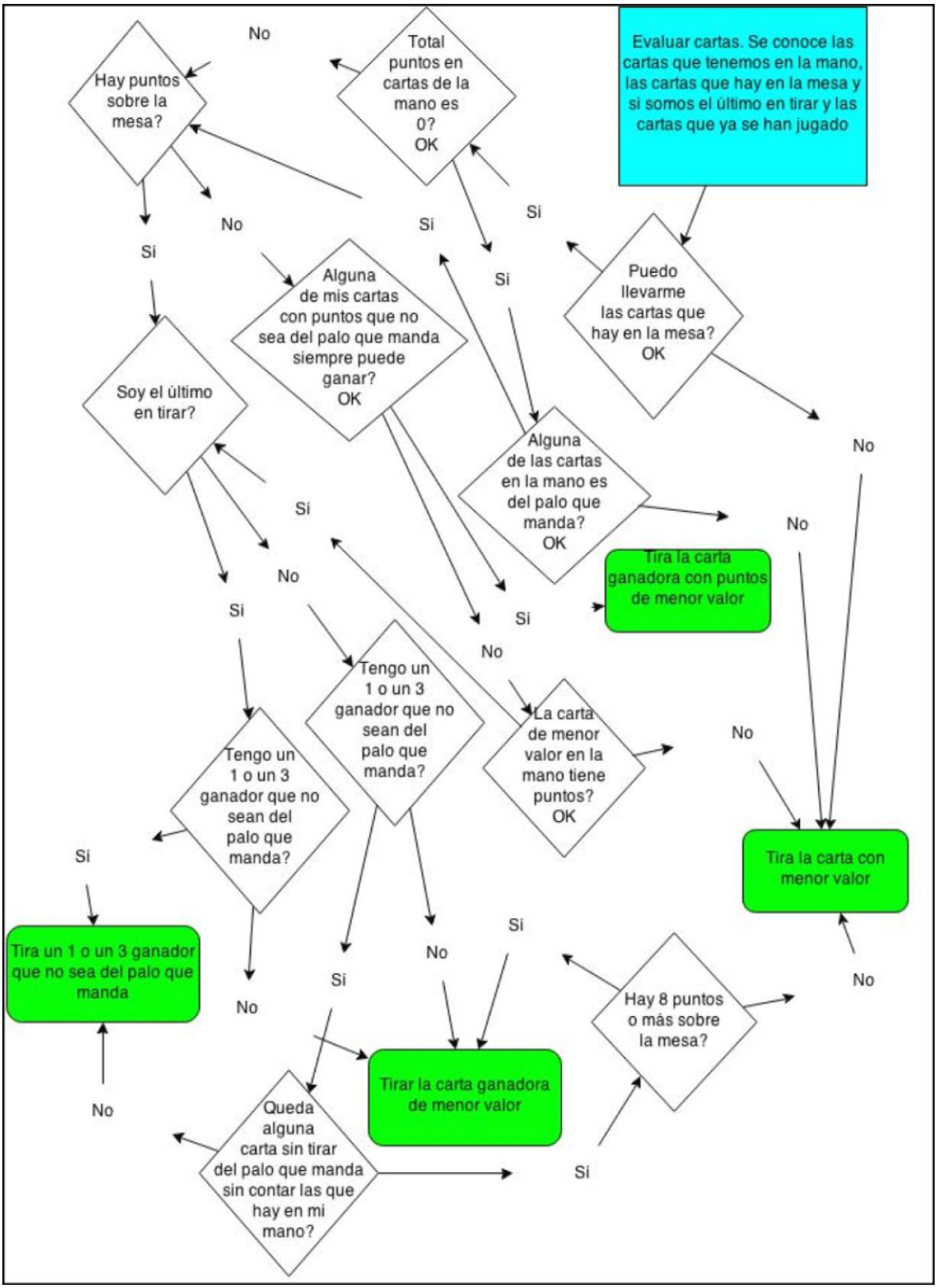
3.3.2. Planteamiento

El planteamiento inicial de esta IA se ha realizado mediante la observación de varias partidas del juego junto respuestas que deberían de darse en ciertas situaciones. Ya que algunas jugadas pueden variar dependiendo de qué cartas han sido anteriormente es necesario almacenar en la memoria de la IA las cartas que se han jugado. La IA debe tener acceso a las mismas entradas de información que un jugador tendría. Estas son las cartas que el jugador tiene en la mano, las cartas que hay sobre la mesa (incluyendo la carta de muestra), las cartas jugadas anteriormente, el orden de tiro y la posición que la IA ocupa en dicho orden. Para esta IA, tanto el orden de tiro como la posición no se han tenido en cuenta, en su lugar se ha tenido en cuenta únicamente si el jugador es el último en tirar.

Una IA sería capaz de recordar partidas anteriores y aprender de ellas, realizar distintas estrategias a lo largo de una partida para probar al resto de jugadores o incluso percatarse de que un rival está siguiendo una estrategia conocida y decidir usar una estrategia capaz de contrarrestar la del rival. Esto abarcaría una cantidad de tiempo muy grande tanto en aprendizaje como en pruebas, por lo que ha sido necesario buscar un punto en el que la IA sea suficientemente funcional como para suponer un reto y tardar una cantidad de tiempo razonable en realizarla.

Lo primero para realizar la IA es diseñar un diagrama de flujo. No todas las IA requieren de un diagrama de flujo para ser diseñadas pero es el método más sencillo de plasmar las comprobaciones que hará la IA hasta dar con una respuesta (lanzar una carta) y a la vez comprobar fácilmente siguiendo el diagrama cómo actuará una vez se pase el diagrama a código.

El primer diagrama resultó no ser el definitivo. Fue necesaria la realización de 4 diagramas hasta dar con uno que consiguiera que la IA no realizara jugadas sin sentido.



1. Diagrama de flujo de la IA. El cuadrado azul representa el punto de partida y los cuadrados azules los de salida.

3.3.3. Codificación de la IA

Una vez está listo el diagrama de flujo se puede proceder a convertirlo en código. Ello supone traducir las preguntas que se formulan en el mismo a condiciones programadas. Para ello, se realizó una función por cada una de las condiciones del diagrama y una función por cada uno de los finales, un total de 14 funciones (16 originalmente pero agrupadas se agruparon funciones). La realización de las preguntas se hizo usando el código de la base del objeto IABrisca.

3.3.4. Búsqueda de fallos

Esta fue la parte más importante. En un principio se realizaron pruebas mediante la llamada a la función con los parámetros de entrada (cartas en la mesa, cartas en la mano, palo de la muestra, palo que manda en la mesa, ¿último en tirar?, todas las cartas jugadas). Este fue un proceso lento de comprobar el funcionamiento de la IA, por lo que fue necesario implementar la IA en el objeto IABrisca y hacer que jugara contra sí misma en partidas con diferentes cantidades de rivales para poder ver los fallos. Esto provocó que los puntos 3.3.5. (Inclusión de la IA en el objeto IABrisca) y 3.4 (Apariencia) fueran realizados al mismo tiempo que este punto lo suficiente como para poder ver sin necesidad de la consola de javascript del navegador como jugaba la IA.

Tras múltiples partidas aparecieron muchas jugadas inadecuadas. Usando la consola para mostrar los parámetros de entrada de todas las jugadas y la carta que la IA elegía tirar junto con el juego visualizándose en la pantalla se hizo mucho más sencillo encontrar los fallos, los cuales resultaron ser más de lógica que fallos de código.

Los fallos de programación fueron los fallos más sencillos de identificar y arreglar. Para encontrar y solucionar estos problemas se usó la consola del navegador y puntos de interrupción en el código para ir comprobando los valores que tenían las variables. Ya que gracias a la lógica del diagrama conocemos la respuesta que debe de darse ante cierta entrada, localizar los fallos de programación fue sencillo.

Los fallos de lógica de la IA fueron los fallos complicados de arreglar. Supusieron rehacer el diagrama de flujo 3 veces. El primer diagrama contaba con 6 condiciones y 3 finales distintos, el segundo diagrama con 9 condiciones y 3 finales, el tercer diagrama con 10 condiciones y 3 finales, el cuarto y último con 11 condiciones y 4 finales.

Dado que una vez hecho el diagrama y vuelto a pasar a código continuaba avanzando con la estética, para el cuarto diagrama ya se podía jugar contra la máquina. Por ello decidí subir a un servidor de internet todo lo necesario para que funcionara y dejé que amigos y desconocidos probaran su funcionamiento. Por suerte hubo varias personas que lo probaron y no encontraron nada raro en las jugadas de la IA. Puesto que yo tampoco conseguía encontrar más fallos de lógica decidí dejar de modificar la IA ya que era capaz de jugar una partida y continué con los otros puntos.

3.3.5. Inclusión de la IA en el objeto IABrisca

Una vez terminada la IA se incluyó dentro del objeto IABrisca como una función con funciones dentro. La función principal recibe los parámetros necesarios para poder

calcular la jugada mientras que las funciones internas son las condiciones codificadas del diagrama de flujo Esta función se introdujo dentro del objeto base de IABrisca ya que era común a todos los jugadores IA.

3.3.6. Jugadores

Dentro del objeto IABrisca se realizan 3 objetos con la misma interfaz (mismas funciones y variables como mínimo) encargados de representar a un jugador: jugador IA, jugador humano y jugador callback (jugador que se usará en el modo contra otros jugadores. Este objeto se realizó cuando se llegó al punto 4.4.2. (Planteamiento), en la parte de dentro de una sala). Las variables comunes a los tres objetos son la id del jugador, array de cartas en la mano, array de cartas jugadas en la partida por todos los jugadores y array de cartas ganadas. Las funciones son el inicializador del jugador, parámetros de entrada la id del jugador; función que pide al jugador que lance una carta, parámetro de entrada una función callback que se debe llamar con la carta que se quiere tirar; función para robar una carta, parámetro de entrada la carta a ser robada; función para ganar la mano, parámetro de entrada las cartas ganadas de la mesa; función para recibir el listado de cartas jugadas en la actual mano, parámetro de entrada las cartas jugadas.

En un principio la función para pedir al jugador que lance una carta tenía un return con la carta que se debía tirar. Ya que un jugador humano no lanza la carta en el momento en el que se le permite lanzarla a diferencia de la IA, se cambió a una petición de lanzamiento la cual incluye una función que se llama, en el caso de la IA tras un timeout simulando que la IA está pensando, tras un onclick sobre una carta en la pantalla en el caso del humano y tras llamar a la función por orden del servidor haciendo que el jugador lance lo que el servidor indique para el modo contra otros jugadores.

Por último, para añadir distintos niveles de dificultad a la IA se modificará ligeramente el objeto IA. Cuando se instancie un nuevo jugador IA y se inicie se podrá indicar el porcentaje de probabilidades de que la IA se equivoque, lo que significa que el lanzamiento que realizará será calculado aleatoriamente en lugar de por el patrón de la IA en caso de que el azar indicado al inicializar el jugador sea mayor que 0 y el cálculo de azar sea inferior al porcentaje indicado de azar. En caso de indicar un 100% de azar al inicializar el jugador este tirará al azar todas las cartas. Las dificultades elegidas son difícil, con 0% de azar, medio, con 20% de azar y fácil, con 50% de azar. En ocasiones el azar puede realizar mejores jugadas que el patrón de la IA y teniendo en cuenta que gran parte de la dificultad reside en el azar de las cartas repartidas a cada jugador, la posibilidad de una partida en la que el porcentaje de aleatoriedad de lanzamiento del jugador IA sea irrelevante existe.

3.3.7. Mesa

El objeto mesa del objeto IABrisca se encarga de organizar la partida llevando el control del juego. Cuenta con variables para conocer las cartas que hay en el mazo, la carta de muestra, las cartas de los jugadores que hay en la mesa, a qué jugador pertenecen y el orden de lanzamiento, quién ganó la última mano. A su vez, cuenta con

diferentes funciones: para inicializar la mesa instanciando el mazo de cartas y eligiendo al azar la carta de muestra; para insertar a los jugadores en la mesa que reparte las cartas a los jugadores y decide quien comienza aleatoriamente; para repartir cartas a los jugadores llamando a la función de los jugadores que recibe la petición de robar; para conocer si quedan manos por jugar; para controlar cada jugada y decidir cuál es el próximo movimiento, pudiendo ser que lance el siguiente jugador, que se elija un ganador de la mano, repartan cartas en caso de quedar y se intente seguir con la siguiente mano; para resetear la mano y evitar conflictos con el comienzo de una nueva mano; Para intentar seguir con la siguiente mano, repartiendo cartas en caso de quedar si es que todavía no está terminada la partida o elegir el ganador en caso de que la partida haya terminado; para pedir que un jugador lance una carta; callback para que un jugador lance a la mesa; para que un jugador robe una carta; para que un jugador gane la mano.

Por último se añadieron varias funciones y objetos más a IABrisca para hacer posible ciertos cálculos necesarios para manejar los jugadores, para poder activar o desactivar console.log mediante el uso de un objeto cuya única función era llamar a console.log en caso de que estuviera activado y una función necesaria para poder buscar un elemento en un array mediante la función indexOf.

3.4. Apariencia

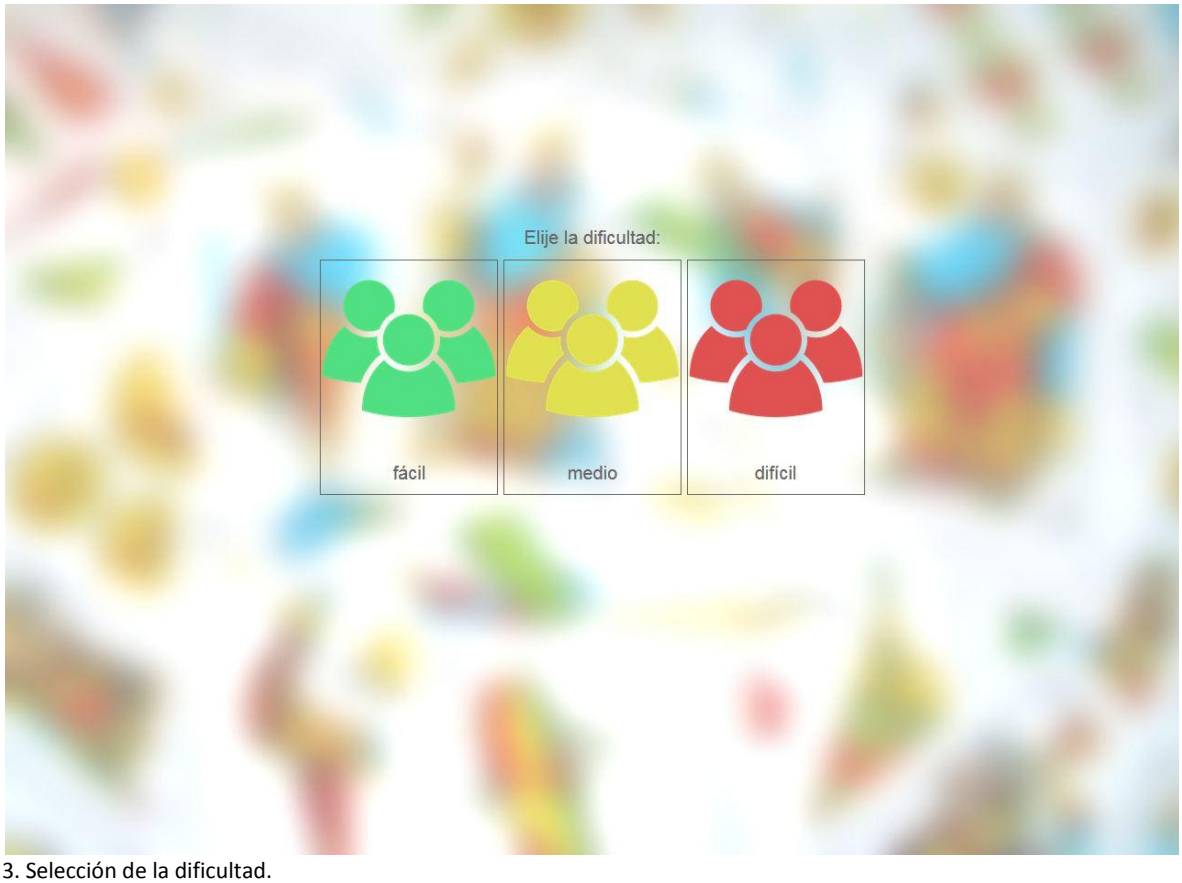
Para dibujar la mesa en el navegador se usó un div que ocupaba toda la pantalla con el fondo de una imagen que simulaba un tapete de mesa de juego de naipes, se puso los anteriormente mencionados divs para posiciones (posicionados mediante una clase css) y se pusieron dos capas ocultas que ocupaban toda la pantalla para los menús de selección de cantidad de jugadores y la dificultad.

Las cartas se dibujan usando la etiqueta img para poder redimensionarlas. Mediante el alto y ancho de la ventana se calcula el tamaño de las cartas. No se puede indicar dicho tamaño en una clase a causa de la relación de aspecto entre el ancho y el alto ya que se pretende que el tamaño de la carta varíe dependiendo del tamaño de la ventana del navegador, pudiendo así dibujar las cartas con el máximo tamaño posible.

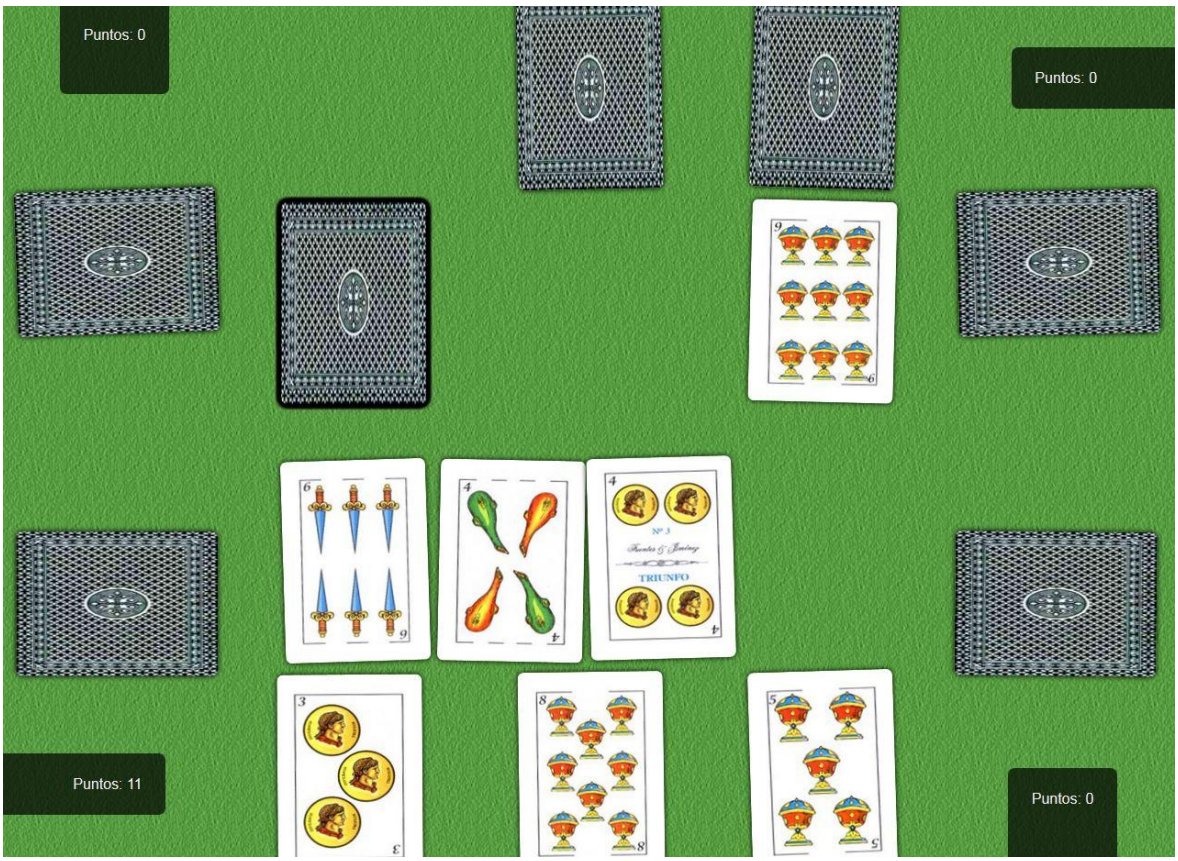
Como extra para la apariencia las cartas se posicionan ligeramente descentradas y rotadas mediante azar de la posición donde deben situarse dando así la sensación de, al ser lanzadas gracias a la animación, caer de forma algo errática intentando simular un lanzamiento más parecido al de una persona que el de una máquina, pero sin dejar de colocar las cartas en la pantalla lo mejor posible para que se pueda jugar.

3.4.1. Capturas

2. Selección del número oponentes.



3. Selección de la dificultad.



4. Juego contra la máquina.

3.4.2. Animaciones

Una parte importante son las animaciones. Estas permiten al usuario ver con facilidad los desplazamientos que realizan las cartas y la mayor parte del tiempo en el juego son las animaciones las que están teniendo lugar en la ventana. Las animaciones pueden realizarse de dos formas distintas: Usando css o usando javascript. Decidí usar css ya que supuse que, siendo el navegador el encargado de hacerla iría mejor. Tras múltiples pruebas y búsqueda de información quedó patente que esto no es verdad y que en realidad depende del navegador, siendo firefox capaz de hacer fluidas las animaciones mientras que chrome las hacía entrecortadas. Esto se debe a que los motores de renderizado de estos dos navegadores trabajan de forma distinta por lo que sería necesario hacer cambios en las animaciones dependiendo de qué navegador sea el que renderiza.

Teniendo en cuenta que las animaciones funcionaban y para no alargar el proceso en más puntos (dudas similares a esta han ocurrido a lo largo del proyecto pero por falta de tiempo no todas han podido resolverse) ha sido necesario continuar y en caso de sobrar tiempo arreglarlo. Al final no hubo tiempo para arreglarlo por lo que las animaciones funcionan todavía entrecortadas en el navegador chrome.

La función encargada de desplazar las cartas por la ventana recibe el nombre de la carta que debe desplazar y la posición que debe ocupar. Esta función encuentra el elemento html que representa la carta mediante id el cual es "*carta_*" seguido de el nombre de la carta. Una vez se tiene el objeto se le aplica la clase de la posición que debe

ocupar y por último, para centrar la carta, se le indica margen superior e izquierdo negativo con un ligero azar para recrear la posición errática junto con una rotación usando transform de css3 también aleatoria dentro de un rango de valores máximos y minimos. En caso de que la carta esté dirigida para los jugadores de los laterales de la ventana las cartas se inclinan 90 grados. La transición entre posiciones la realiza el navegador siguiendo la orden transition de css. En caso de que la carta se dirija a un jugador distinto al jugador humano se cambia la imagen de la carta y se pone la de una carta invertida para evitar que el usuario conozca la carta que tienen los rivales.

3.5. Victoria y derrota

El último punto para hacer que el modo contra la máquina sea funciona es el final de la partida. El juego puede terminar de tres formas distintas, con el jugador ganando, perdiendo o empatando con la máquina. Estos finales son los mismos en caso de jugar contra otros jugadores por lo que el mensaje que al terminar una partida se mostrará un mensaje con el estado de la victoria, los puntos ganados y las opciones de reiniciar partida o volver al menú principal (dejar de jugar).

Para calcular quien es el ganador lo haremos desde IABrisca. Al terminal la última ronda llamará a una función callback con el parámetro de los puntos que ha ganado cada jugador. Esta función lanzará el cartel de victoria, derrota o empate dependiendo del parámetro de entrada.

Para poder reiniciar partida es necesario que IABrisca sea independiente. Esto permite que se destruya el script y todo lo generado por él en pantalla. Por último, se vuelve a preguntar la cantidad de contrincantes, dificultad y se crea de nuevo una instancia de IABrisca, se configuran los parámetros necesarios del objeto como los callbacks y valores iniciales y se inicia la partida.

3.6. Todo junto

Para terminar el modo contra la máquina falta unir todas las piezas mencionadas hasta ahora. El html contiene 3 capas: Selección de cantidad de contrincantes, selección de dificultad de contrincantes y visualización del juego. Se iniciará en la capa de selección de cantidad de contrincantes siendo la única visible de las 3. Una vez el usuario seleccione una de las opciones se cerrará la capa y se abrirá la de selección de dificultad. Esta cambia dependiendo de la cantidad de contrincantes elegidos por lo que se escribe mediante javascript desde la llamada del botón clicado en la capa anterior para indicar el número de oponentes. Una vez clicada la dificultad se lanzará una función que instanciará el objeto IABrisca, configurará los callback con funciones que se encargarán de mover las cartas, cambiar la puntuación en la ventana, pedir que el jugador lance carta y que se muestre el mensaje del final. A continuación se instancian las cartas en la mesa sobre el mazo y después se pide al objeto IABrisca que inicie la mesa. Se crea la cantidad de jugadores solicitada, tanto humano como máquina o máquinas, se introducen en la mesa y se inicia la partida, lo cual provoca que IABrisca reparta las cartas a los jugadores y de paso al primer lanzamiento, tomando el control del resto de juego hasta que llame al callback de fin de la partida.

4. Juego contra otros jugadores

4.1. Conocimientos previos

Antes de empezar debemos tener en cuenta ciertos elementos. El juego se desarrollará en salas donde entrarán usuarios para jugar. Los usuarios necesitarán registrarse y mantener una sesión iniciada en la página para poder acceder al modo de juego contra otros usuarios, poder cerrar sesión, hacer login, crear y entrar en salas de juego, hablar entre ellos mediante un chat. Las jugadas pueden y deben enviarse a través de ese mismo chat. El servidor debe aplicar la misma lógica que el javascript IABrisca en las jugadas, debe iniciar y terminar una partida, determinar ganador, guardar en la base de datos las estadísticas del jugador tras la partida y controlar la salida y entrada de jugadores a una sala.

Se necesita php para realizar un control de usuarios (registro, inicio de sesión y fin de sesión), portar la lógica de IABrisca de javascript a php, realizar un chat con filtrado por salas, usar la lógica del juego para administrar la partida, diseñar un modo de comunicar el usuario con el servidor para transmitir y recibir órdenes junto con el chat de los usuarios y hacer estadísticas de las partidas.

Se usará MySQL para almacenar los datos del usuario y sus estadísticas, además de para guardar el estado del usuario con respecto a las salas (en ninguna sala o en una sala, conociendo cual). La base de datos no se pensó por completo al inicio del proyecto por lo que fue necesario remodelarla en varias ocasiones. El historial de los chats junto al historial de jugadas realizadas será guardado, pero no en la base de datos MySQL ya que el diseño del historial requiere de diversas tablas. Guardar varias tablas por partida en una base de datos resultaría en una bajada de rendimiento conforme pasa el tiempo además de una pérdida del orden en la base de datos por la multitud de tablas acumuladas.

Se usará SQLite para almacenar el historial de mensajes, órdenes y jugadas de las salas y la configuración de las mismas. Se decidió elegir esta tecnología ya que permite generar bases de datos en archivos individuales y trabajar con ellos mediante consultas SQL, siendo muy similar en lo más básico de SQL.

La información que queremos guardar es posible fragmentarla por salas. Esto significa que una vez una sala sea cerrada, ya que se ha fragmentado dicha información a un archivo dejará de cargarse en memoria permitiendo reducir el uso de ram y en caso de querer retirar historial del servidor para almacenarlo en un lugar distinto o borrarlo, bastará con mover los archivos correspondientes (uno por base de datos SQLite) sin mayor preocupación.

En javascript necesitaremos duplicar y modificar la parte que ya tenemos para jugar en el modo contra la máquina y adaptarla al modo contra otros jugadores. Para ello, necesitamos realizar un chat mediante llamadas asíncronas al servidor (AJAX) y enviar y recibir órdenes, jugadas, mensajes, avisos y todo lo necesario para que una partida online pueda llevarse a cabo.

Usaremos prácticamente el mismo HTML del modo contra con la única modificación de incluir espacio para un chat y realizarlo.

4.2. Registro y login de usuarios

Necesitamos mantener un registro de usuarios para poder administrar el historial de las partidas que realizan. Sería posible realizar el modo online sin registro de usuarios mediante el uso de cookies o sesiones de php.

4.2.1. Descripción

El registro y login de usuarios consiste en guardar en una base de datos un nombre de usuario y una contraseña por cada usuario para poder identificarlos de forma individual y permanente. A su vez sirve para que el usuario pueda verificar su identidad evitando ser suplantado.

Se usará el registro de usuarios para poder almacenar estadísticas del usuario en el modo contra otros jugadores y en el modo contra la máquina. Más adelante se tratará la fiabilidad de dichas estadísticas, sobre todo en el modo contra la máquina.

4.2.2. Objetivo

El objetivo es la realización de un formulario de registro y un formulario de login en html, uno o varios archivos php en el servidor encargados de realizar consultas a la base de datos con el objetivo de registrar, iniciar sesión y cerrar sesión de un usuario y mantenimiento de una sesión mediante una cookie y reconocimiento de la misma desde el servidor.

4.2.3. Diseño de la tabla de usuarios y login en MySQL

La tabla de usuarios contará con los siguientes campos: ID, Nombre, Apellido, Nick, Password, Email, Fecha de registro, Fecha de último acceso, ip de último acceso, validación de email de registro, sala actual, número de victorias contra otros jugadores, número de derrotas contra otros jugadores, puntuación máxima conseguida contra otros jugadores, número de victorias contra la máquina, número de derrotas contra la máquina, puntuación máxima conseguida contra otros jugadores. ID se usará como un número que representa al usuario de forma inequívoca con facilidad. Nick y password serán usados para validar el usuario y nick será el identificador con el que otros usuarios identificarán al usuario. Email se usará para validar el usuario en el futuro al igual que para poder informar de cambios importantes en la web (existen herramientas en internet capaces de generar correos electrónicos válidos y funcionales de forma temporal que pueden ser usados para validar el usuario por lo que la finalidad de este campo es informativo y de seguridad para el usuario únicamente en caso de que el usuario indique su correo real en el campo). Victorias, derrotas y puntuaciones máximas para los modos contra la máquina y otros jugadores servirán para almacenar las estadísticas del usuario. Sala servirá para indicar si el usuario se encuentra en una sala y en cual, valiendo -1 en caso de no estar en ninguna sala o bien valiendo el valor de la ID de la sala en la tabla de salas (se realizará a continuación). Nombre, apellido, Fecha de registro, fecha de último inicio de sesión y IP del último inicio de sesión serán irrelevantes en este proyecto.

La tabla login se encargará de controlar las sesiones de los usuarios. Para ello almacenará ID de un usuario, contenido de la cookie que lo identificará como usuario con sesión iniciada y tiempo máximo que puede estar la sesión de la cookie abierta.

4.2.4. Registro

Para registrar un usuario se usará un formulario html con los campos Nombre, Apellidos, Correo, Repetición de correo, Nick, Contraseña y Repetición de contraseña. El formulario se enviará a un php que validará los campos en busca de caracteres inválidos, comprobará que todos los campos estén completos y que aquellos que deben de coincidir por ser repeticiones, coincidan. En caso de que todos estas condiciones se cumplan se generará una cadena de textos y número de forma aleatoria y se introducirá junto al nuevo usuario en la base de datos guardando la cadena generada de forma aleatoria en el campo de validación de usuario de la base de datos. Después, se enviará un correo al usuario con la url de un php del sitio con dos variables GET en la url, una que contendrá la misma cadena generada anteriormente de forma aleatoria y otra que contendrá la ID del usuario en la base de datos. Al abrir el usuario el email y clicar la url de su interior cargará el php y en caso de que ambas variables existan y sean limpiadas de caracteres problemáticos para la base de datos se buscará en la base de datos una coincidencia en las columnas ID y user validado dadas dichas variables. De existir una coincidencia se vaciará el contenido del campo de user validado.

4.2.5. Login

Para iniciar sesión un usuario usará un formulario html con los campos nick y contraseña. Este formulario se enviará a un php que validará los campos para evitar ataques a la base de datos y después buscará una coincidencia. De haberla se deduce que el usuario que está intentando iniciar sesión se corresponde con el usuario coincidente de la base de datos por lo que se le envía al usuario una cookie con una cadena generada de forma aleatoria y se guarda dicha cadena en la base de datos junto con una fecha de expiración y la ID del usuario en la tabla de login.

Todos los archivo php del servidor que requieran de un usuario con sesión iniciada preguntarán por esta cookie mediante una función que unificará las preguntas de comprobar si la cookie es válida y si hay alguna coincidencia en la base de datos. De haberla y ser la fecha de la base de datos mayor que la fecha actual, se recuperarán los datos del usuario en cuestión para que el servidor pueda usarlos.

4.2.6. Logout

Para cerrar sesión se eliminará cualquier cookie de sesión, se validará la cookie actual y de ser válida se borrará toda cookie guardada en la base de datos que coincida con la cookie del usuario.

4.2.7. Seguridad

La seguridad es un aspecto muy importante del manejo del registro de usuarios. Se ha intentado garantizar un mínimo de seguridad teniendo en cuenta que no es posible una seguridad completa y fiable al igual que se requiere de muchos conocimientos y experiencia en el campo para poder garantizar la robustez de la seguridad de una aplicación frente a ataques.

Para tratar de garantizar la seguridad de la aplicación se ha usado (todas las medidas están situadas en el lado del servidor, resultando inútil situarlas en el lado del cliente):

- La función MD5 para guardar un hash de las contraseñas de los usuarios, modificadas ligeramente para evitar que un diccionario de contraseñas MD5 rompa la contraseña. Esto únicamente retrasa el tiempo necesario para encontrar la contraseña detrás de un MD5 ya que se puede averiguar la cadena de texto que ha generado un hash mediante la fuerza bruta.
- Una función que retira de una cadena de texto todos los caracteres con algún sentido para SQL evitando que se puedan realizar instrucciones no deseadas a la base de datos.

4.3. Comunicación entre usuarios. Chat

4.3.1. Descripción

Un chat permite la comunicación entre los participantes del mismo mediante, en este caso, mensajes de texto. Estos mensajes dependiendo de la aplicación de chat serán enviados de forma directa al usuario o usuarios de destino o en su lugar se enviará a un servidor encargado de reenviar el mensaje entre los usuarios. Mediante javascript es posible crear redes p2p (peer to peer) que serían capaces de enviar los mensajes entre usuarios sin necesidad de un intermediario mediante el uso de WebRTC, compatible con Firefox, Chrome y Opera en sus más recientes versiones. En este caso el chat que realizaremos servirá para enviar y recibir mensajes de los usuarios además de enviar órdenes al servidor indicando jugadas realizadas por el usuario y recibir órdenes del servidor para indicar movimientos de cartas, victoria de un jugador para cada mano, la finalización de la partida, entrada y salida de usuarios en la sala, bloqueo de la sala en caso de que estando iniciada alguien abandone la partida y otras, pudiendo ampliarse.

4.3.2. Objetivo

El objetivo del chat es sincronizar la partida entre los usuarios que disputan una partida. Cada sala tendrá un chat independiente y se guardará el historial del chat teniendo en cuenta que contendrá mensajes de usuarios y ordenes que permitan que la partida tenga lugar. Por tanto, el chat desde el servidor tendrá que poder aplicar la lógica del juego de la Brisca en caso de que no se trate de mensajes entre usuarios.

4.3.3. Planteamiento

El chat consistente en mensajes para usuarios será visible en la ventana del navegador en un div dedicado al mismo. Dicho div contendrá un formulario que enviará mediante ajax mensajes a otros usuarios. El mismo ajax servirá para enviar jugadas.

Un segundo ajax cargará un php del servidor encargado de comprobar si hay mensajes nuevos desde la última vez que se comprobaron. Cada vez que el javascript reciba mensajes guardará la ID del último mensaje y preguntará al php indicando mediante una variable por POST si hay nuevos mensajes en el chat. El php entrará en un bucle con fin que comprobará si en el SQLite correspondiente a la sala hay nuevos mensajes. En caso de no haber esperará una determinada cantidad de tiempo y repetirá el bucle en caso de quedar repeticiones restantes o bien retornará un JSON con los mensajes a partir de la ID indicada por el javascript y saldrá del bucle. Cada vez que se termine la conexión con el php javascript abrirá una nueva conexión ajax al php pasado una cantidad determinada de tiempo. Estas esperas sirven para no saturar el servidor.

No todos los mensajes del juego van dedicado a todos, por ello se usará privacidad en los mensajes indicando quien debe recibirlos o quien no debe recibirlos para así conseguir que órdenes como la de repartir cartas a los usuarios mantenga privada la carta robada para el usuario que la recibe y evitar así la realización de trampas.

Nota: El modo contra la máquina es vulnerable a trampas como demuestra el javascript escrito por un usuario que probó el juego y que se puede ver en la página web http://pastebin.com/9nZhCLr8

El historial, como se explicó anteriormente, se guarda en bases de datos en archivos mediante SQLite. La base de datos SQLite tendrá la siguiente estructura: Tabla mensajes con los campos *n* (ID), *usuario*, *mensaje* y *privacidad*; Tabla usuarios con los campos *ID*, *primero*, *lanza*, *hueco_sala* y *pareja*; Tabla cartas con los campos *carta*, *posición*, *propietario*, *palo_manda_siempre* y *palo_manda_mesa*; Tabla estados con los campos *clave* y *valor*.

La tabla mensajes almacena el usuario que envía un mensaje, el mensaje y para quien va dedicado mediante privacidad.

La tabla usuarios almacena la ID de cada usuario, en *primero* mediante un 1 o un 0 marca quien es el usuario que ha comenzado la mano, en *lanza* guarda si ya ha lanzado carta, *hueco_sala* guarda la posición en la mesa del jugador para calcular el siguiente jugador y *pareja* tiene un número que identifica a la pareja, teniendo dos jugadores el mismo número en caso de estar jugando la partida en parejas.

La tabla cartas almacena cada carta, la posición que ocupa en la mesa y en caso de ser lanzada, *propietario* adquiere el valor de la ID del usuario que ha lanzado la carta. En caso de que algún usuario gane la mano, *propietario* indicará la ID del usuario ganador de la mano. *Palo_manda_siempre* y *palo_manda_mesa* tendrán como valor 1 o 0 indicando el palo de la muestra y el palo de la primera carta tirada en la mesa respectícamente.

La tabla estados se usará para indicar si la sala se trata de una partida en parejas o no, si ha sido iniciada y si está detenida. La sala se marcará iniciada y comenzará la partida tan pronto como en la sala se encuentre el número de usuarios que la sala requiere. En

caso de que estando la sala iniciada un usuario abandone la misma se marcará detenida y la partida se dentendrá.

4.3.4. Desarrollo del cliente

Una función realizará una llamada ajax a un php con parámetros post y se usará para enviar mensajes. Estos parámetros dependerán de desde dónde se llame, teniendo los parámetros sala y msg (mensaje) en caso de que el usuario envíe un mensaje por el chat y teniendo los parámetros sala y jugada en caso de lanzar una carta.

Una función realizará una llamada ajax de forma regular para comprobar si hay nuevos mensajes llamando al mismo php que la función anterior con los parámetros sala y ult (ID del último mensaje). Esta función enviará la salida a una función que mediante ifs y switch llamará a funciones de IABrisca, moverá cartas y llamará a la función que muestra aviso de victoria o derrota.

Parte del código usado para el modo contra la máquina es aislado a un archivo con funciones para ser usadas en el modo contra otros jugadores. Se hacer pequeñas modificaciones de ciertas partes de funciones para que sean compatibles con ambos modos de juego.

4.3.5. Desarrollo del servidor

El servidor está compuesto por un php que consulta a la base de datos y al archivo con la base de datos SQLite. Es necesario disponer de la lógica de la Brisca en el servidor por lo que se portará el código necesario del objeto IABrisca de javascript a php.

4.3.6. Problemas con SQLite

El problema que surgió con SQLite fue que por defecto en caso de que se realice a la misma base de datos SQLite dos conexiones simultaneas, la que llegue más tarde fallará. Es posible solventar esto gracias a la especificación del valor de una variable llamada busybox, la cual provoca la realización de consultas de forma continua al archivo para comprobar si es posible acceder a el durante el periodo de tiempo que se indique en la variable. De esta forma únicamente fallará en caso de que el tiempo de espera finalice sin poder acceder al archivo. Para evitar que falle es necesario cerrar la conexión si ya no es necesario realizar más consultas

Otro problema que surgió más tarde fue la carencia de las funciones nativas de php para manejar archivos SQLite3 en el servidor gratuito donde decidí hacer pruebas. Esto provocó la migración de estas funciones al objeto PDO, el cual tiene un driver capaz de manejas bases de datos SQLite. Las variaciones entre el objeto PDO y las funciones nativas para SQLite son la forma de abrir el archivo y especificar la variable busybox, el tipo de variable que retornan los resultados, siendo similar pero no igual y que el objeto PDO cierra la conexión tras desaparecer la variable que contiene el objeto (mediante una llamada a unset con el objeto como parámetro) mientras que era necesario cerrar la conexión en el objeto SQLite mediante una función con ese objetivo.

4.4. Jugar online

4.4.1. Objetivo

El objetivo es que el usuario pueda jugar una partida contra otros jugadores a la vez que puede chatear con ellos. Además debe conocer el estado de la partida de forma visual de todos los eventos relevantes para la partida.

Los usuarios podrán crear partidas, cerrarlas y conectarse a ya existentes conociendo la cantidad de jugadores en la sala, la cantidad de jugadores requeridos para que la partida se inicie y si se jugará por parejas.

El servidor del juego será autoritativo, lo que significa que el usuario indicará la petición de realización de un movimiento y el servidor decidirá la realización de dicho movimiento, comunicando después al usuario el resultado de su decisión, siendo entonces cuando el usuario realiza el movimiento.

4.4.2. Planteamiento

Se creará un formulario en html para crear una sala llamando a un php. Los campos que tendrá son el nombre de la sala, la cantidad de jugadores y en caso de ser 4 la cantidad de jugadores, si se jugará por parejas.

Se creará un html que listará las salas abiertas, permitirá filtrar por cantidad de jugadores y por parejas y permitirá conectarse a una sala.

El usuario accederá a una sala en caso de no estar previamente en una distinta. De estarlo, se solicitará que el usuario que retorne a la sala en la que se encontraba o que abandode la sala.

La creación de salas se realizará mediante la generación de un archivo SQLite para la sala creada y la creación de una fila en la base de datos con las características de la sala. Ya que no tiene sentido abrir todos los archivos SQLite para listarlos, en la selección de salas los datos básicos de la sala se recogen y guardan en la base de datos para así poder realizar con facilidad un listado de salas abiertas.

Una vez todos los jugadores de una sala la abandonen se borrará de la base de datos, dejando el archivo SQLite sin manipular.

Dentro de una sala se dispondrá de de un botón para abandonar la sala. Mientras que no estén todos los usuarios necesarios para comenzar la partida podrán usar el chat para hablar entre ellos. Una vez se encuentren todos los usuarios necesarios la partida se iniciará de forma automática introduciendo en el chat las órdenes necesarias para que el javascript instancia lo básico de la partida en la ventana. Después se repartirán las cartas y dará paso al primer jugador para lanzar. En caso de que un jugador abandone la sala se cambiará el estado de la sala a detenida y se mantendrá el chat pero se bloquearán las futuras jugadas.

4.4.3. Control del juego desde el chat

Para controlar el juego desde el chat se inserta desde el servidor mensajes de diversos tipos tanto cuando un jugador realiza un movimiento como cuando habla. Los mensajes se agregan en las siguientes situaciones: Cuando un usuario entra en la partida, cuando un usuario abandona la partida, cuando un usuario realiza una jugada y cuando un usuario escribe un mensaje. Para realizar esto se reutiliza el campo del mensaje introduciendo un objeto json con el tipo de mensaje y su valor. En el caso de la última situación no únicamente se inserta un objeto del tipo mensaje con el contenido de lo enviado por el usuario. En el resto de situaciones se realizan comprobaciones previas.

Gracias a que los envíos de información se hacen por un canal distinto al recibimiento de los datos es posible realizar pausas en el php que recibe los datos mientras va actualizando el SQLite, permitiendo de esa forma cosas como repartir cartas de una en una y no todas de golpe. Otras soluciones habrían sido posibles, como órdenes más concretas con marca de tiempo que indique cuando deben de realizarse o el desfase de tiempo en segundos entre orden y orden.

4.4.4. Lógica en el servidor

Es necesario disponer de la lógica que desarrollamos en javascript en el servidor para poder validar las jugadas y realizar los cálculos necesarios para poder controlar la partida en el modo entre varios jugadores.

Portaremos el objeto IABrisca de jabascript al servidor. Para portar el código de a php se creó una clase con el mismo nombre y se copio y pegó el código del objeto javascript dentro del php. Únicamente resultó ser necesario el sub objeto Base de IABrisca, por lo que se realizó una copia completa de este objeto, tanto variables como funciones. Se tradujeron las funciones, probándolas una por una para confirmar que el funcionamiento era el mismo, y se hizo también una copia de las variables, algo más sencillo. Esta clase de php, sus funciones y variables se definieron como estáticas para facilitar su uso. De esta forma ya se pudo realizar cálculos para decidir el ganador de cada mano, el ganador de una partida

Para validar una jugada se hace uso de la base de datos SQLite y del objeto IABrisca, todo en el lado del servidor puesto que no se puede ni debe confiar en el cliente en este tipo de tareas.

Se usa la base de datos SQLite para comprobar si el usuario puede tirar ya que es su turno de tirar. De serlo, se comprueba si intenta tirar una carta que tiene en la mano. De ser así, se realiza el lanzamiento mediante el envío de un mensaje por el chat con la orden que te todos los jugadores hagan que el jugador en cuestión lance la carta y se modifica la base de datos SQLite para indicar que el jugador ya no tiene la carta lanzada y que la carta ahora está en la mesa, indicando que la propiedad es del jugador que ha lanzado la carta. En caso de el usuario que ha lanzado no sea el último en tirar se da paso al siguiente jugador para que lance cambiando en la base de datos SQLite la variable lanza del jugador actual de próximo a lanzar a lanzado y la del próximo jugador, calculado mediante la posición que ocupa en la mesa, de no lanzado a próximo a lanzar. Después se envía el mensaje al chat de ordenar que lance el usuario que debe lanzar a continuación.

De lo contrario, se decidirá el ganador de la mano mediante la clase IABrisca, se modificará la clase SQLite para indicar que el usuario ganador es ahora propietario de las cartas, que ya no se encuentran en la mesa y se enviará el mensaje de que el usuario ganador recibe las cartas. Después se repartirá una carta a cada jugador en caso de quedar cartas en el mazo, teniendo en cuenta la muestra para repartirla cuando se acabe el mazo de cartas, mediante un mensaje en el chat por cada carta repartida y la modificación de la base de datos SQLite para indicar las nuevas posiciones de las cartas repartidas.

Se determinará el ganador en la última jugada de la partida, cuando no queden cartas por repartir y el usuario que lance se quede sin cartas y sea el último en tirar. En este momento se realizará lo necesario para indicar qué usuario gana la mano tal y como se realizaba en el punto anterior pero después se calculará qué jugador es el ganador mediante la suma de los puntos de todas las cartas ganadas. Una vez esté claro el ganador se enviará por chat el veredicto para que el javascript lo interprete y muestra a cada usuario si es ganador, perdedor o ha empatado.

Una vez terminado el modo contra otros jugadores se incluyó el modo de juego por parejas. Esto supuso un problema ya que no se tuvo en cuenta en el diseño.

Para conseguir el juego por parejas se realizaron los siguientes añadidos:

En el javascript:

- se guardaban las ID de los jugadores y el grupo de la pareja al que pertenecían.
- Cada vez que un jugador ganaba la mesa, su pareja también recibía los puntos
- Cuando la partida terminaba y se indicaba el jugador ganador, en caso de ser la pareja del usuario se mostraba también el mensaje de victoria.

En el php:

- Se hizo un final distinto del juego en la lógica en el que en lugar de jugadores se manejaba la id de los grupos.
- La id de cada grupo hacía referencia al sumatorio de los puntos de sus integrantes.

4.4.5. Seguridad

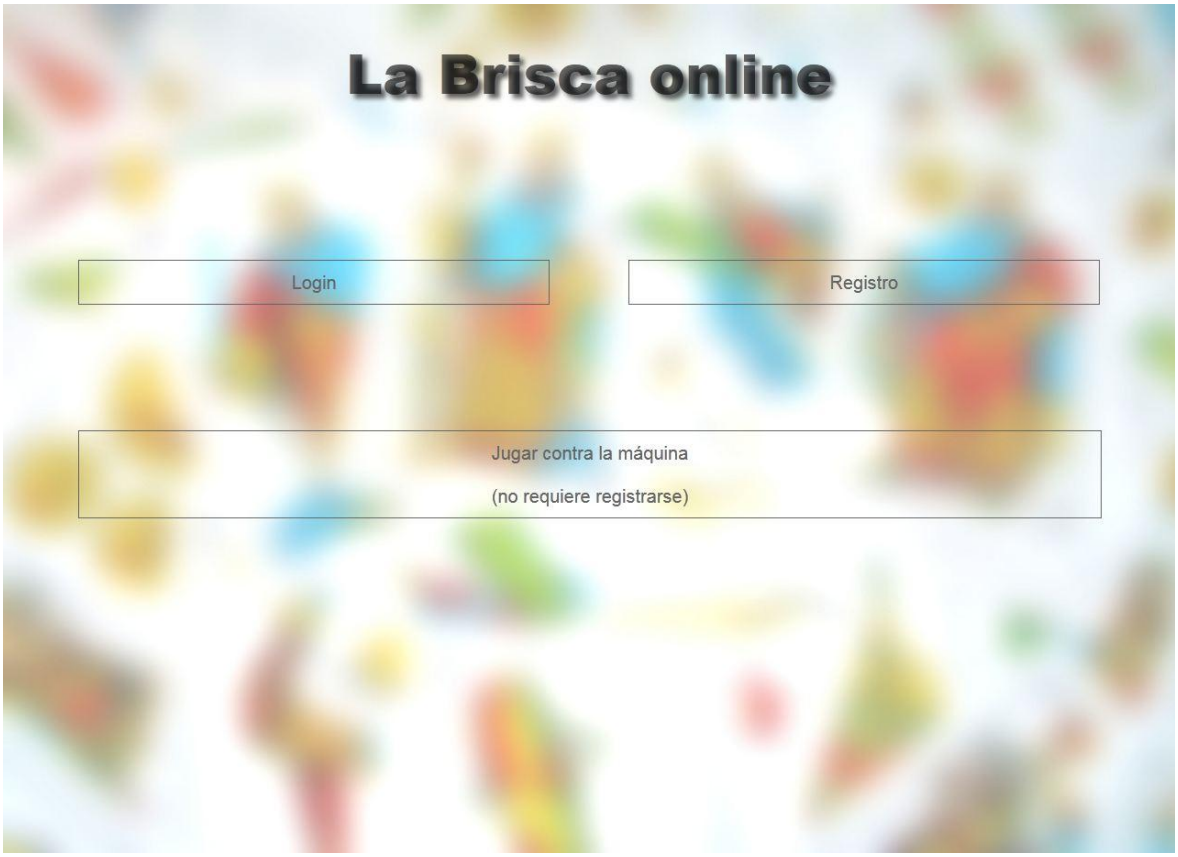
La seguridad nuevamente es un punto importante ya que en el transcurso de una partida se espera que los usuarios ni hagan ni puedan hacer trampas. Esta seguridad se ha conseguido mediante la replicación de la partida en el servidor, convirtiendo a los usuarios en observadores de la partida y únicamente cuando el servidor solicita a un usuario que intervenga en la partida el usuario puede intervenir. Además, la intervención del usuario está restringida a los únicos movimientos que puede realizar.

Esto significa que el usuario únicamente muestra por pantalla la partida tal y como está en el servidor y ya que los movimientos que el usuario puede hacer están restringidos, la posibilidad de realizar trampas es prácticamente nula. Nunca se puede garantizar la nulidad de la posibilidad de hacer trampas ya que no solo depende de la

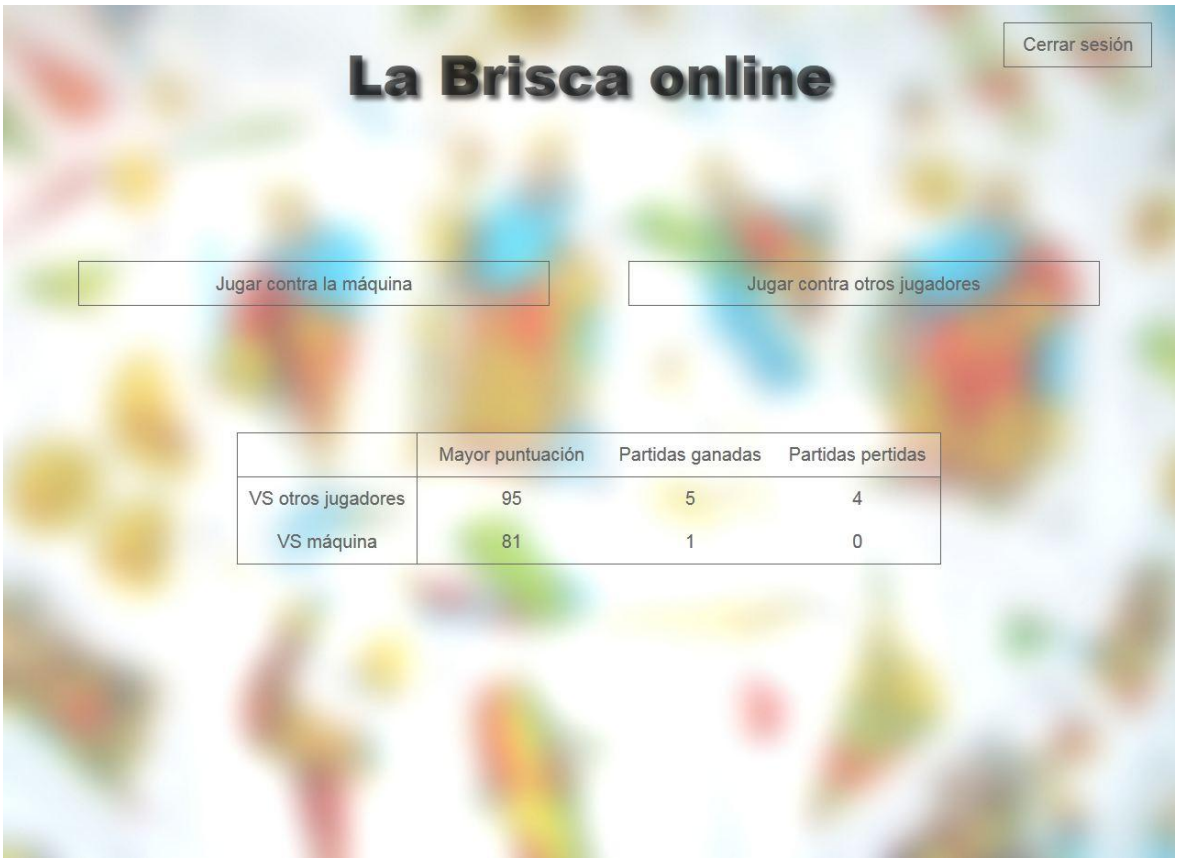
lógica usada para detenerlas, además depende de que esté bien llevada a cabo dicha lógica y que el lenguaje usado no posea vulnerabilidades explotables para realizar trampas. Muchos puntos que no se pueden asegurar con certeza.

PHP por defecto detiene un proceso en ejecución perteneciente a un usuario en caso de que el usuario corte la conexión con la web. Esto es bueno para ahorrar recursos pero es un problema en caso de que el php deba realizar cambios internos. En el caso de que el php que se está ejecutando deba escribir ya sea en la base de datos MySQL o la base de datos SQLite, se cancela este comportamiendo obligando a que el php permaneza funcionando hasta que el límite de tiempo de ejecución se supere o termine la ejecución del script. Para ello se ha hecho uso de la función ignore_user_abort con el parámetro true para indicar que el abandono del usuario no debe detener la ejecución del script php. Esta función se puede llamar con el parámetro false en cualquier momento para reanudar el funcionamiento por defecto.

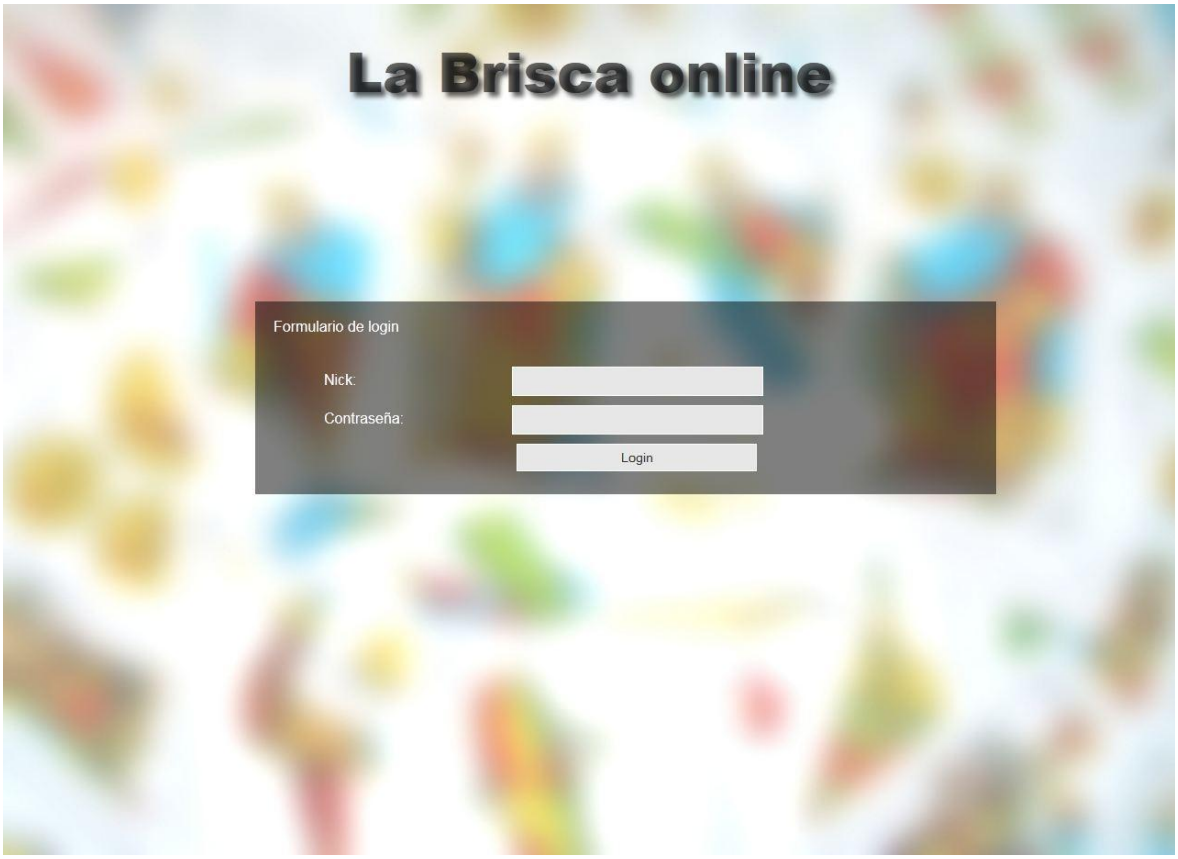
4.4.6. Apariencia



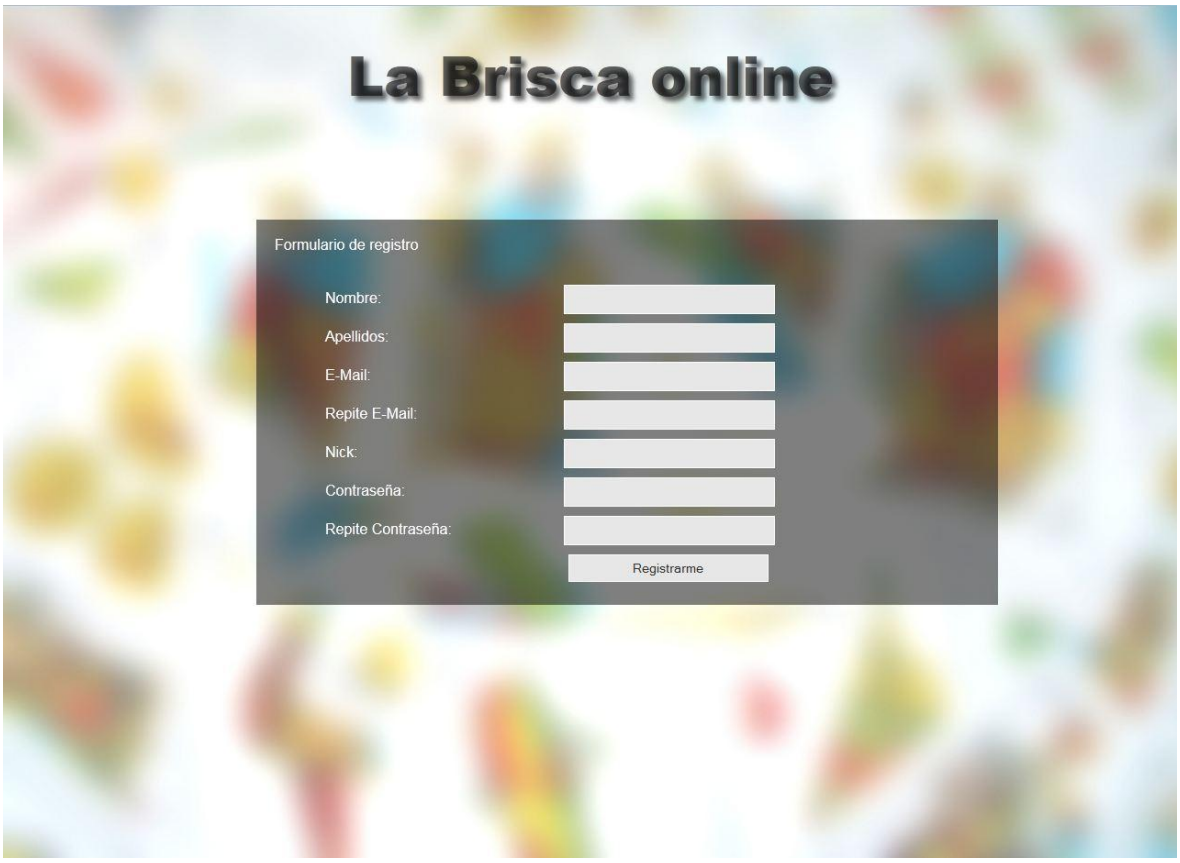
5. Index de la web.



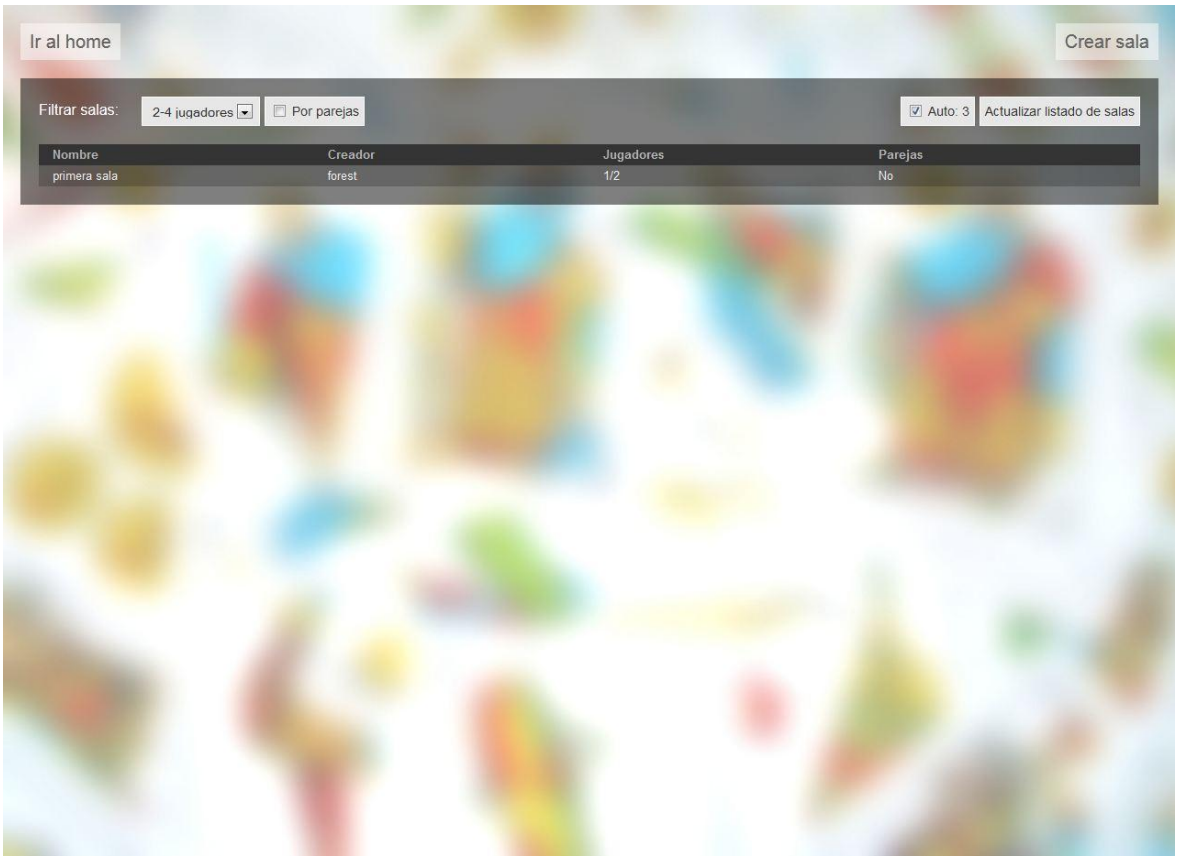
6. Index con la sesión iniciada.



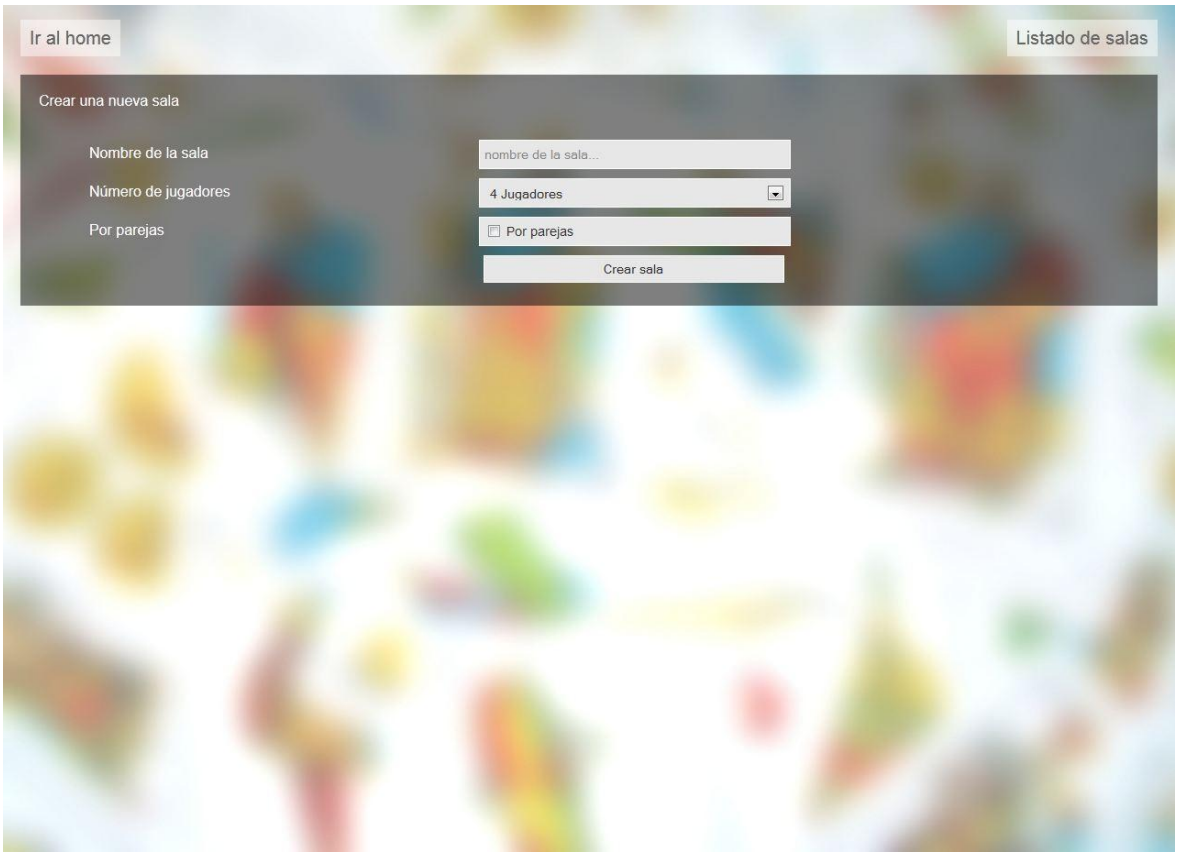
7. Formulario de login.



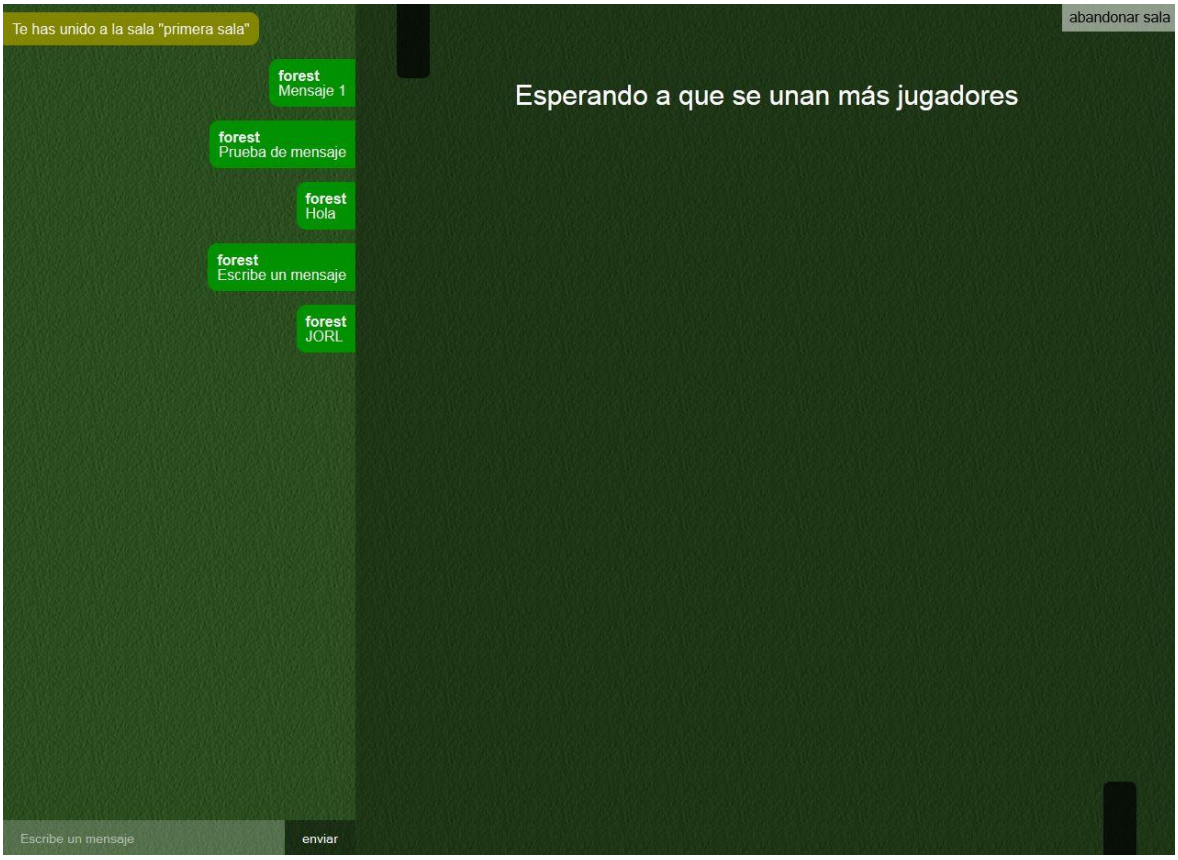
8. Formulario de registro.



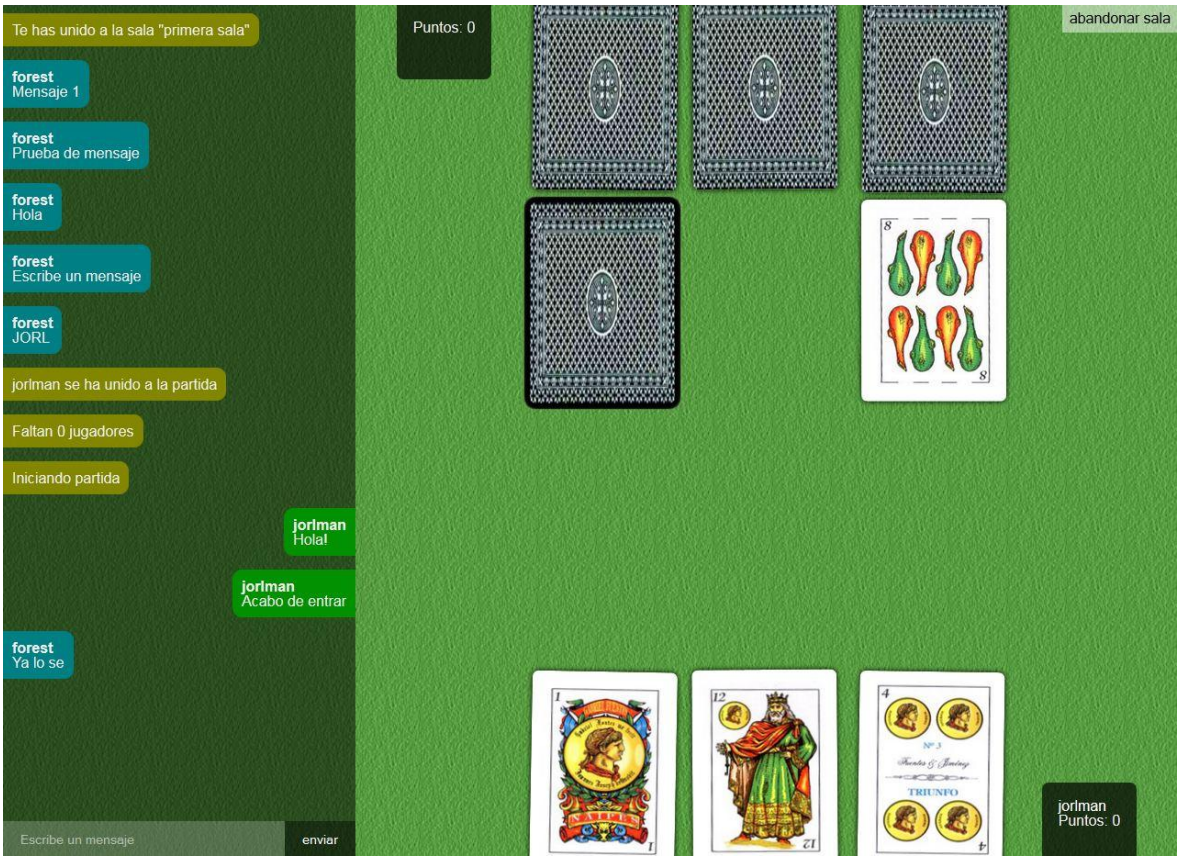
9. Sala de espera con selección de salas.



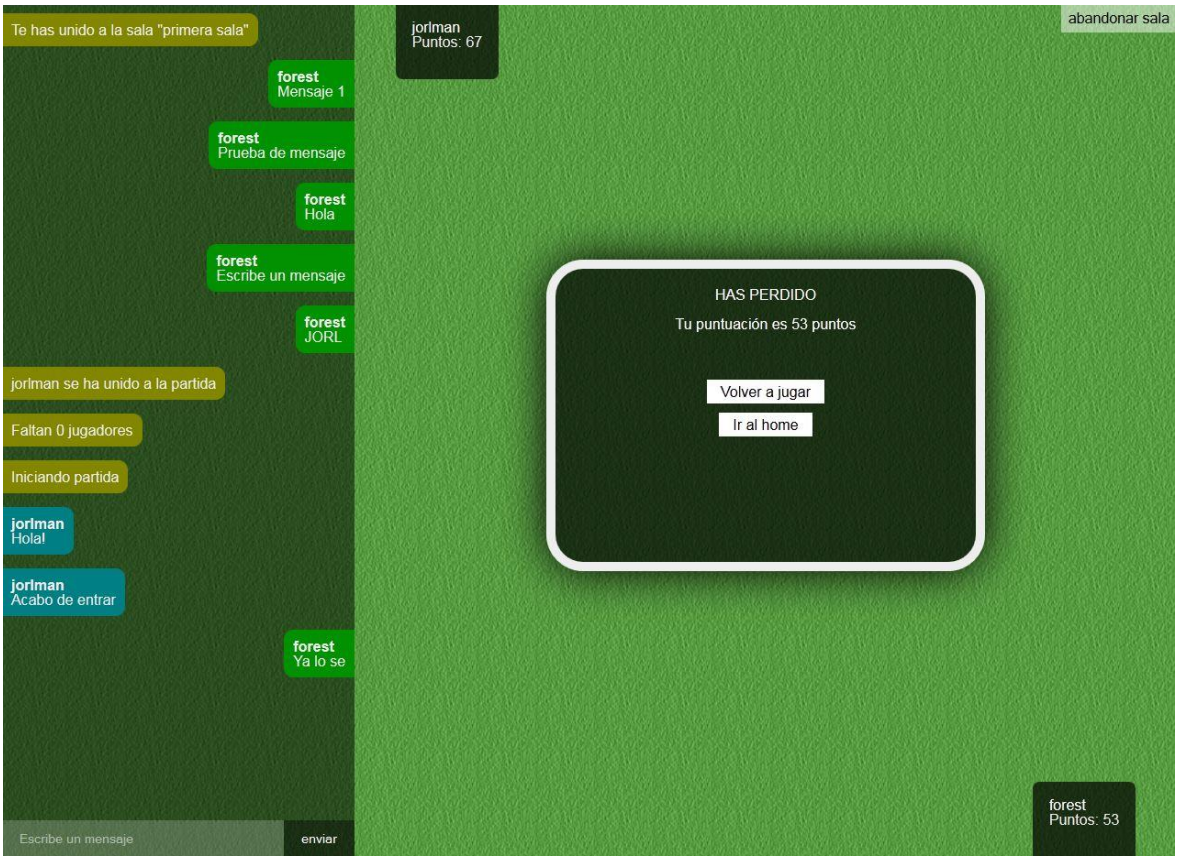
10. Formulario de creación de salas.



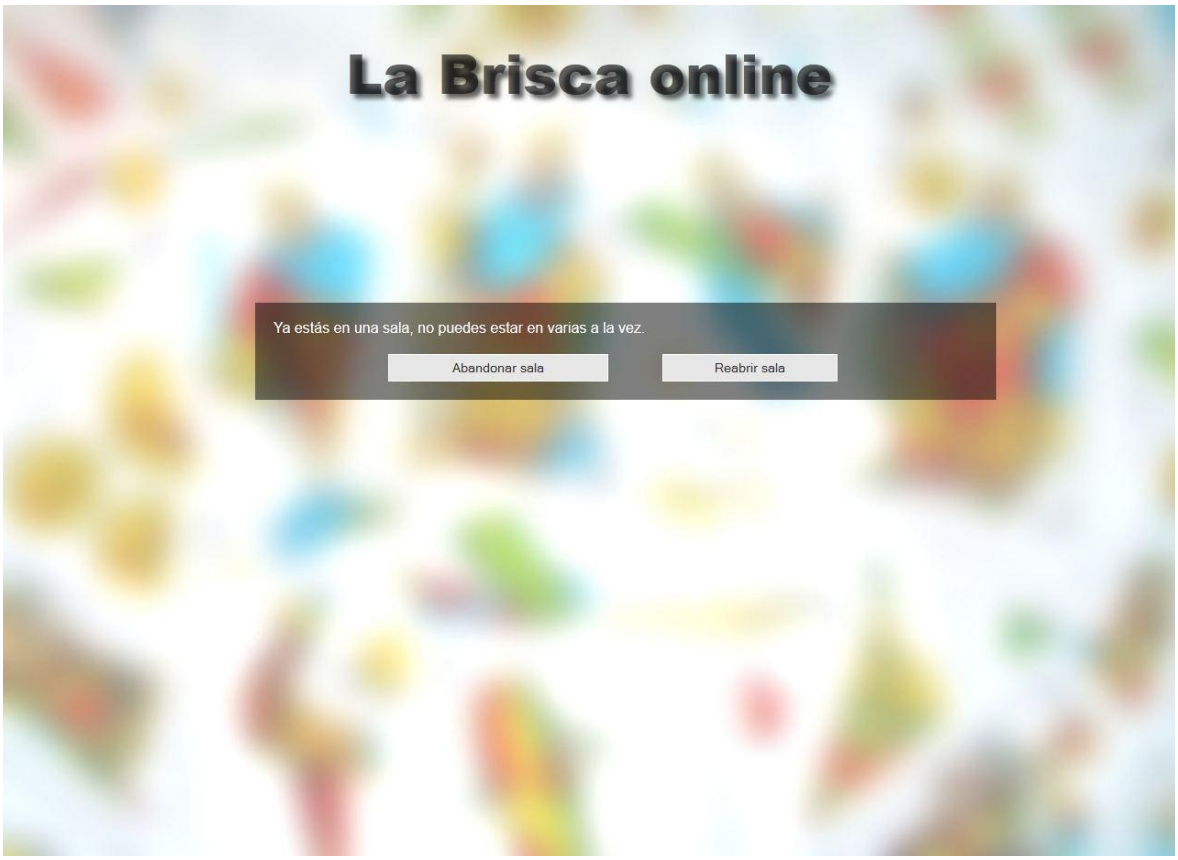
11. Dentro de una sala, esperando más usuarios.



12. Dentro de una sala, jugando con otros usuarios.



13. Dentro de una sala, juego terminado.



14. Intentando entrar en una sala cuando ya se está en una.

4.5. Estadísticas

Las estadísticas guardadas en este proyecto sobre los usuarios en el uso del modo contra la máquina y el modo contra otros jugadores se limita a almacenar la puntuación máxima obtenida, la cantidad de victorias y la cantidad de derrotas en el transcurso de todas las partidas que el jugador realice dependiendo de el modo de juego.

Las estadísticas se mostrarán en el index de la web en caso de que el usuario tenga la sesión iniciada.

4.5.1. Juego contra otros jugadores

Las estadísticas en este modo de juego se calculan desde el servidor una vez una partida ha terminado al realizar un usuario la última jugada. Para ello como última instrucción después de enviar los mensajes a los usuarios indicando el fin de la partida se ejecuta en la base de datos por cada usuario en la partida un incremento en 1 de las partidas ganadas o perdidas dependiendo de si se ha ganado o perdido, o bien no haciendo nada en caso de empate y guardando el mayor número entre la puntuación conseguida y la anterior puntuación conseguida.

4.5.2. Juego contra máquina

Las estadísticas en este modo de juego se calculan del mismo modo que en el punto anterior pero recibiendo los datos de si se ha ganado, perdido o empatado y la puntuación desde el cliente por javascript mediante una llamada ajax al php que realizará los cálculos necesarios.

4.5.3. Fiabilidad de las estadísticas

Las estadísticas del modo contra otros jugadores es calculada íntegramente en el servidor con datos extraídos del propio servidor. La seguridad comentada en el punto 4.4.5. (Seguridad) es la misma seguridad que se aplica a este modo.

Las estadísticas del modo contra otro jugador no son fiables. El control del juego, desde que se reparten las cartas hasta que termina la partida se hace mediante javascript en el cliente. Ya que el cliente tiene acceso a los cálculos realizados y puede modificarlos a su favor provocando la realización de trampas, la fiabilidad de las estadísticas para este modo es nula. Para que este modo tuviera estadísticas fiables sin recurrir a una falsa fiabilidad mediante el oscurecimiento del código javascript, haciendo difícil su comprensión (falsa seguridad), sería necesario que el servidor llevara el control de la partida al igual que se hace con el modo contra otros jugadores, para lo cual sería necesario portar la parte del objeto IABrisca de javascript que contiene el patrón para calcular jugadas para la IA y los objetos que engloban a jugadores a php para que estos cálculos sean realizados desde el servidor, dejando que el usuario únicamente lance cuando se le sea solicitado y recibiendo datos desde el servidor que le indiquen qué debe mostrar en la pantalla.

5. Conclusiones

Los mayores retos de este proyecto han sido la realización de la inteligencia artificial y la sincronización de la partida en el modo entre varios usuarios entre los jugadores teniendo en cuenta todas las posibilidades que había. La mayor parte del tiempo ha sido dedicado a estos dos puntos entre el planteamiento y desarrollo de los mismos, además de una gran cantidad de tiempo realizando pruebas y arreglando fallos hasta que pudo considerarse estable.

La realización de este proyecto ha resultado de gran utilidad sobre todo por la introducción al mundo de la inteligencia artificial. Ha resultado muy interesante su realización y fue divertido comprobar las decisiones que realizaba al igual que modificarlas para tratar de conseguir que realizara mejores decisiones.

Un hosting gratuito es capaz de aguantar la web pero siempre que la cantidad de usuarios simultáneos se mantenga en un número muy reducido ya que la cantidad de llamadas que se realizan al servidor en una partida online es elevada y debe multiplicarse por cada usuario que se encuentre jugando en el modo contra otros usuarios.

HTML5 ha resultado tener muy pocas utilidades. El doctype de los html está en html5, se han usado características de html5 como inputs con label, etiquetas scripts sin descripción del lenguaje, se han dejado etiquetas abiertas como p ya que no son necesario cerrarlas y se ha evitado el uso de aquellas etiquetas como center que han recibido el valor deprecated. La utilidad no ha sido muy grande y por ello no se ha mencionado anteriormente al no ser relevante para el proyecto.

6. Medios/recursos utilizados

Para la realización de este proyecto se ha hecho uso del programa notepad++ y Aptana para la edición de código, los navegadores web internet explorer, firefox y chrome para las comprobaciones de la web, el programa git para el control de versiones mediante el programa SourceTree y el programa xampp para el servidor (php y base de datos MySQL).

Nota: El proyecto puede encontrarse en la siguiente página de GitHub: https://github.com/forestrf/Brisca-online

7. Posibilidades para futuros desarrollos

La mayor parte del código ha sido escrito pensando en la futura reutilización del mismo. En el caso de la clase IABrisca en javascript es independiente de otras librerías para evitar conflictos con cualquiera de ellas y es fácil de implementar en cualquier web que quisiera disponer el juego la brisca con enemigos con inteligencia artificial. Ello se ha conseguido gracias a introducir todo el código de la clase dentro de un closure.

El código del servidor dedicado a la sincronización de la partida entre varios usuarios es menos portable pero en un commit del historial de git se puede encontrar una versión básica del chat con salas que no requiere de usuarios muy fácil de ampliar e implementar ya que se trata de la base y además es capaz de funcionar de forma estable.

La interfaz de la web ha sido desarrollada mediante gran cantidad de css por lo que la modificación de esta es en gran parte la modificación del css. Esto hace que la interfaz no sea un obstáculo.

El sistema de administración de usuarios también es posible portarlo con facilidad gracias a que el código se encuentra agrupado en una clase que además contiene todo lo necesario para conectarse a la base de datos y realizar consultas ya formuladas en la clase mediante llamadas a funciones del objeto que retornan arrays o booleanos.

Todo el código se encuentra documentado por lo que incluso sin esta memoria sería sencillo realizar modificaciones del código siguiendo los comentarios.

8. Bibliografía

Wikipedia

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

https://es.wikipedia.org/wiki/Brisca

https://es.wikipedia.org/wiki/Inteligencia_artificial

https://es.wikipedia.org/wiki/Inyecci%C3%B3n_SQL

StackOverflow

http://stackoverflow.com/questions/11131875/what-is-the-cleanest-way-to-disable-css-transition-effects-temporarily

http://stackoverflow.com/questions/122102/most-efficient-way-to-clone-an-object

http://stackoverflow.com/questions/3629183/why-doesnt-indexof-work-on-an-array-ie8

http://stackoverflow.com/questions/122102/most-efficient-way-to-clone-an-object

http://stackoverflow.com/questions/2665984/is-subtracting-zero-some-sort-of-javascript-performance-trick

http://stackoverflow.com/questions/15615552/get-div-height-with-plain-javascript

http://stackoverflow.com/questions/9713058/sending-post-data-with-a-xmlhttprequest

http://stackoverflow.com/questions/951021/what-do-i-do-if-i-want-a-javascript-version-of-sleep

http://stackoverflow.com/questions/5767325/remove-specific-element-from-an-array

php.net

http://www.php.net/manual/en/mysqli.persistconns.php

http://www.php.net/mysql_real_escape_string

http://www.php.net/mysql_real_escape_string

http://www.php.net/manual/es/security.database.sql-injection.php

http://www.php.net/manual/es/language.oop5.static.php

http://www.php.net/manual/en/function.sqlite-busy-timeout.php

http://www.php.net/manual/es/function.usleep.php

http://www.php.net/manual/en/memcache.examples-overview.php

MeyerWeb.com

http://meyerweb.com/eric/tools/css/reset/

Andylanfton.co.uk

http://andylangton.co.uk/blog/development/get-viewport-size-width-and-height-javascript