Universidad Zaragoza
1542

Proyecto Fin de Carrera

Aplicación web de póker Texas Hold'em multijugador

Autor

Rubén Serrate Pardo

Director/es y/o ponente

Director - Graham Hutton
Ponente - José Merseguer

Escuela de Ingeniería y Arquitectura

2015

Repositorio de la Universidad de Zaragoza – Zaguan http://zaguan.unizar.es

APLICACIÓN WEB DE POKER TEXAS HOLD'EM MULTIJUGADOR

RESUMEN

Puesto que soy jugador ocasional de póker, en todo momento tuve clara la funcionalidad básica de la aplicación. Sin embargo, tras realizar un pequeño análisis de productos similares, obtuve la conclusión de que el usuario siempre necesitaba ejecutar software adicional para acceder a las partidas, ya fuera en forma de *applets* o de *plug-ins* para el navegador. A partir de esta observación, y puesto que la tecnología actual lo permite, decidimos que la principal característica diferenciadora de nuestra aplicación iba a ser que se podría utilizar simplemente con un navegador web actualizado, sin necesidad de instalar ningún otro software.

Durante todo el ciclo de desarrollo de la aplicación se siguió una metodología ágil, dando lugar a varios prototipos que se iban mejorando en sucesivas iteraciones. El primero de estos prototipos que se programó fue una librería que ofrecía una serie de operaciones a través de las cuales era posible desarrollar una partida de póker. Siguiendo con las prácticas ágiles, se aplicó *TDD* (*Test Driven Development*, Desarrollo guiado por pruebas), es decir, primero se programaron los tests para las clases involucradas y posteriormente el código necesario para satisfacerlos, todo ello en C#. Esta primera versión fue después enriquecida con funcionalidad para soportar varias partidas simultáneas y con una base de datos que almacenaría las acciones de los usuarios. Se añadieron también operaciones para calcular estadísticas comunes a partir de éstas, jugadores de la IA (Inteligencia Artificial) y otras pequeñas funcionalidades que habían quedado fuera del ámbito de la primera iteración. Este conjunto de clases y sus tests dio lugar al núcleo de la aplicación, el cual se puede ver como una API que ofrece soporte para desarrollar partidas de póker.

El siguiente paso fue integrar este núcleo en el servidor web. Puesto que el patrón con el que se trabaja en el servidor web es MVC (Modelo Vista Controlador), las clases que conforman el núcleo de nuestra aplicación pasaron a ser los modelos, se crearon controladores para permitir la interacción con estos modelos a través de peticiones HTTP, y por último unas vistas muy sencillas que volcaban la información en forma de documento HTML. Por otro lado, se adaptó el sistema de identificación que ASP.NET MVC provee para integrarlo con nuestra aplicación, o en otras palabras, se estableció la relación entre un jugador en una mesa y un usuario identificado en el sistema.

Una vez se tuvo el sistema integrado en la aplicación web, se diseñó una primera interfaz a la que se añadió tanto la información como los controles necesarios para que los jugadores pudieran desarrollar partidas y ver sus estadísticas. Para implementar esta interfaz, se extendieron las vistas del modelo MVC y se ofrecieron nuevas operaciones en los controladores para ofrecer soporte a las peticiones AJAX por parte del cliente. Estas peticiones asíncronas son necesarias para actualizar el estado de la mesa y transmitir al servidor las acciones realizadas por los jugadores entre otras funciones. En este momento se adoptaron también una serie de medidas para permitir el acceso concurrente a la aplicación de varios jugadores.

Por último, una vez se tuvo una aplicación completamente funcional, se pidió a un grupo de usuarios que trataran de desarrollar una partida y consultar sus estadísticas y que después dieran sus opiniones sobre la interfaz y/o posibles problemas encontrados. Este último análisis de la aplicación dio lugar a un rediseño de la interfaz y a la implementación de nuevas funcionalidades que facilitaban el manejo de la misma.

TÁBLA DE CONTENIDOS

INTRODUCCIÓN

El proyecto se realizó en la Universidad de Nottingham durante mi segundo cuatrimestre de participación en el programa *Erasmus* y fue dirigido por Mr. Graham Hutton, profesor de *Computer Science* en la citada universidad. El ponente fue en primera instancia Jorge Júlvez, y después José Merseguer, ambos profesores de la Universidad de Zaragoza.

Mas que formar parte de una línea de trabajo, este proyecto es el comienzo de una. La Universidad de Nottingham tendrá acceso al código del proyecto y según el director, se pretende ofrecer futuros trabajos basados en la mejora y/u optimización de ciertas partes de éste, siendo de particular interés las partes dedicadas a la inteligencia artificial y a la comparación y asignación de rangos a las manos de póker.

Si bien se investigó y planteo la utilización de alguna plataforma o biblioteca de póker existente, su incorporación definitiva al proyecto fue descartada debido en parte a la escasa y deficiente documentación que ofrecían, a la diferencia de lenguajes de programación y sobre todo a que como se ha comentado, se pretende realizar un futuro trabajo de optimización sobre el código. Por lo tanto, con el fin de ofrecer una implementación clara junto con una documentación completa y precisa sobre el mismo, la implementación del núcleo de póker partió desde cero.

Durante la totalidad del ciclo de desarrollo del proyecto, éste estuvo gestionado acorde con las nuevas metodologías de desarrollo de software ágil, intentando adoptar en todo momento las prácticas que forman parte de esta filosofía. Al tratarse de un proyecto individual, aquellas prácticas cuyo objetivo es mejorar la comunicación y/o facilitar la interacción entre las diversas partes del equipo, como pueden ser ciertas partes de SCRUM [1], no fueron de aplicación. Sin embargo sí se sacó un gran provecho del resto de recomendaciones. Las principales implicaciones de adoptar una filosofía de desarrollo ágil para el proyecto son las siguientes:

- No existió una primera fase en la que exclusivamente se produce documentación para seguir a lo largo del proyecto, en cambio, se optó por dividir el desarrollo en diversas fases cada una de las cuales va mejorando iterativamente nuestro prototipo.

- Para cada una de estas fases se realiza una pequeña planificación al comienzo de la misma en la que se divide el trabajo en pequeñas tareas. A cada tarea se le asigna un valor según la cantidad de trabajo requerida para finalizarla. Tal y como se explica en el anexo I, las tareas serán luego incorporadas al flujo de trabajo (o *sprint* según la nomenclatura ágil) correspondiente. Para llevar el seguimiento de los distintos *sprints* que conforman una fase, las tareas se representarán con tarjetas que serán dispuestas y actualizadas en un "*Kanban Board*" de modo que en todo momento se tiene una visión clara del estado en el que se encuentra el flujo de trabajo o *sprint* actual.

- La mayoría del código de la aplicación fue escrito tras escribir las unidades de tests correspondientes. Siguiendo el modelo *TDD* o "*Test Driven Development*" [2].

- Al no partir de un diseño estricto, el proyecto se adaptó a las diversas conclusiones y problemas que se iban encontrando a lo largo del desarrollo del mismo, en lugar de seguir a pies juntillas un diseño que se había preparado antes de ni siquiera comenzar con la implementación, como predican los modelos de desarrollo tradicionales o "*Waterfall*".

En cuanto al proyecto en sí, el objetivo es ofrecer una interfaz web a través de la que se puedan desarrollar partidas de póker entre varios jugadores humanos. Para ello, además de dicha interfaz, es

necesario programar el servidor con el que los usuarios interaccionarán y que mantendrá el estado de las partidas.

La interfaz tendrá que mostrar toda la información necesaria para que el jugador pueda decidir su próximo movimiento, además de ofrecerle los controles para poder efectuarlo. Todo ello de la manera más clara posible. Entre otras cosas, el usuario necesitará conocer: qué cartas hay sobre la mesa, cuantos jugadores hay involucrados en la mano y las apuestas de los mismos, la cantidad de fichas de cada jugador, de quién es el turno y cuánto tiempo le queda por actuar… etc. Además, debido a la naturaleza del juego, esta información es muy dinámica y se encuentra constantemente cambiando, por lo que la interfaz deberá reflejar estos cambios y actualizarse continuamente.

Así como la posibilidad de desarrollar partidas, también se ofrecerán al usuario estadísticas relacionadas con su juego, las cuales vendrán acompañadas de gráficos animados para facilitar y hacer más inmediata la transmisión de la información. Estos gráficos serán incrustados en el código *HTML* mediante *SVG* y serán generados dinámicamente con los datos que el sistema ha almacenado de las partidas que ha desarrollado el jugador.

Tal y como se ha justificado en el resumen, esta interfaz será únicamente codificada en HTML y JavaScript, lo que además de hacerla mucho más ligera y compatible con prácticamente cualquier dispositivo capaz de acceder a la web, hace a nuestra aplicación única.

Si bien esto presenta numerosas ventajas, tiene también un principal inconveniente o hándicap, y es que el paradigma web y en concreto el protocolo HTTP no están orientados a ofrecer una conexión TCP continua como la que se puede establecer entre una aplicación cliente y un servidor comunes. Para superar esta limitación se implementará en el cliente un sistema de sincronización con el servidor mediante encuesta. Pese a no ser ideal en cuanto a su eficiencia, se decidió que éste era el modelo más adecuado debido a que otras tecnologías que pretender ofrecer una comunicación bidireccional sobre el protocolo HTTP, como por ejemplo *WebSockets*, no están todavía lo suficientemente maduras o soportadas. Para la implementación de esta sincronización además de otras operaciones necesarias para operar la interfaz, se optó por utilizar la librería *jQuery* debido a la simplicidad y potencia que aporta.

Una vez superado el hándicap de la sincronización y la comunicación con el servidor, la programación de la interfaz consiste principalmente en la codificación del diseño en HTML y CSS. Concretamente se optó por las versiones HTML5 y CSS3 de ambos, puesto que a pesar de no ser soportadas por versiones de Internet Explorer anteriores a Internet Explorer 9, ofrecen un abanico de posibilidades mucho más amplio y potente. Los últimos datos de uso de navegadores reportan que el porcentaje de usuarios que utilizan Internet Explorer 8 es inferior a 9% [3], una cifra de usuarios a la que es justificable renunciar, ya que las opciones elegidas ofrecen recursos de un gran valor para la construcción de nuestra aplicación, ya sean en forma de una mayor eficiencia o de características inexistentes en versiones anteriores.

Esta interfaz cliente se comunicará con un servidor web, el cual además tendrá integrado el núcleo de póker, por lo que también hará el papel de servidor del juego. El software en el que se basa el servidor web será *Internet Information Services* (IIS), de Microsoft, y concretamente nuestra aplicación se basará en el *framework ASP.NET MVC*, versión 4. Sobre este *framework* se ejecutará tanto el código necesario para servir las páginas web de la aplicación como el núcleo de póker en sí. Este núcleo de póker es el que se encargará de llevar el control de las mesas, las partidas y las estadísticas. Para permitir el desarrollo de partidas de póker, el servidor debe de llevar a cabo un gran número de funciones distintas, entre las que se puede destacar la clasificación y asignación de valores a las distintas manos, la gestión de botes divididos, y el manejo de temporizadores. Por otro lado, el núcleo de póker deberá tener en cuenta otros aspectos menos inmediatos como por ejemplo la gestión adecuada del acceso concurrente al servidor de todos los usuarios en la mesa.

Un nivel por encima, se encuentra el manejo del *Lobby* o sala, que consiste en que el servidor mantenga siempre una mesa vacía de cada tipo accesible a los jugadores. Los tipos de mesas varían en cuanto al número de jugadores y a la velocidad en la que se desarrollan las partidas. En nuestro caso, el término mesa es equiparable al término torneo, puesto que todos los torneos que se van a desarrollar

comprenden solamente una mesa. El máximo número de jugadores en un torneo por lo tanto será 6 ó 9, depende de cual sea el tamaño de la mesa correspondiente a dicho torneo.

Por último el servidor también se encarga de las estadísticas. Para ello, se guardará en una base de datos relacional cada una de los movimientos que el usuario realiza durante las partidas así como sus resultados en los diferentes torneos, y en base a estos datos, se calcularán dinámicamente las distintas estadísticas cuando el usuario lo solicite.

El mínimo número de jugadores requeridos en una mesa antes de comenzar el torneo es evidentemente dos, pero puesto que el número de usuarios reales utilizando el sistema no se prevé que sea elevado, el servidor ofrecerá la posibilidad de poblar los asientos restantes en una mesa con jugadores ficticios o *bots*, que contarán con una inteligencia muy limitada pero permitirán a los usuarios experimentar en mesas completas. De este modo, un solo jugador real puede sentarse en una mesa y jugar una partida contra la IA que si bien no es demasiado exigente en cuanto al nivel de habilidad requerido para derrotarla, permitirá la generación de estadísticas y ofrecerá al usuario novato una primera toma de contacto con el juego.

Las herramientas utilizadas durante el desarrollo del proyecto fueron principalmente Microsoft Visual Studio 2012 y el *framework* ASP.NET MVC versión 4. El diseño de la interfaz se realizó en un primer lugar con lápiz y papel y fue posteriormente finalizado en *Adobe Illustrator CS5*. Para la depuración del código que forma parte del cliente de nuestra aplicación se utilizó el navegador web Mozilla Firefox junto con la extensión *FireBug*.

Puesto que la aplicación consta de tres partes muy diferenciadas (núcleo de póker, integración en el servidor web y diseño e implementación de la interfaz), se va a dedicar una sección de la memoria al análisis de cada una de ellas. Posteriormente aparece también una pequeña mención a cómo la aplicación interacciona con el hardware que la ejecuta, y por último, otra sección dedicada a las conclusiones en las que aparecen además las posibles mejoras o ampliaciones.

La memoria cuenta además con tres anexos. El primero de ellos está dedicado a la gestión y organización del trabajo, y en él, además de explicar la metodología empleada, aparece la división del trabajo en iteraciones y *sprints* que se empleó en el proyecto. El segundo anexo recoge todas las pantallas de la interfaz de usuario y en el tercero aparece un resumen de la mecánica y las reglas de la variedad de póker *Texas Hold'em* sin límite, que es la que ofrece nuestra aplicación.

No se precisa de ningún conocimiento avanzado de póker para leer y entender la totalidad de la memoria, además, cuando es preciso diversas nociones básicas aparecen explicadas a lo largo de ésta, por lo tanto, la lectura de este tercer anexo no es necesaria si ya se es familiar con el juego de póker.

Cuando a lo largo de la memoria se haga referencia a alguna fuente externa, la referencia se acompañará de un número en superíndice el cual indicará a qué entrada de la sección bibliografía se refiere.

NÚCLEO DE POKER

DESCRIPCIÓN DEL PROTOTIPO

Al final de esta primera iteración se pretende ofrecer una API con las operaciones necesarias para mantener varias partidas de póker *Texas Hold'em* simultáneamente, tanto con jugadores humanos como de la IA. Además de llevar el control de las partidas, nuestro prototipo tendrá que almacenar cierta información acerca de los jugadores, como por ejemplo el número de créditos de los que disponen, así como modificar y almacenar persistentemente este balance acorde a sus ganancias y pérdidas.

Por otro lado, el prototipo almacenará también de manera persistente las acciones que los jugadores humanos realizan durante las partidas, con el fin de calcular a partir de las mismas las siguientes estadísticas:

- **Torneos jugados, ganados y perdidos.**
- **ITM**: Porcentaje que refleja el número de ocasiones que el jugador termina un torneo en una de las posiciones premiadas respecto al total de torneos jugados.
- **Posiciones en las que jugador ha finalizado los torneos que ha jugado.**
- **ROI**: Cociente entre los beneficios de un jugador y el dinero que ha gastado en jugar los torneos.
- **VPIP**: Porcentaje que representa el número de ocasiones en que el jugador, realiza o iguala una apuesta antes de ver el *flop*.
- **PFR**: Porcentaje representando el número de ocasiones que el jugador sube la apuesta antes de ver el *flop*.
- **Factor de agresividad**: Cociente resultante de dividir el número de ocasiones que el jugador sube la apuesta entre el número de ocasiones que el jugador iguala la apuesta.
- **WTSD**: Porcentaje que refleja la cantidad de ocasiones en que tras ver el *flop*, el jugador llegó a la fase de s*howdown*.

La interacción con el servidor se realizará simplemente mediante llamadas a los métodos públicos que ofrecen las clases que lo conforman, y para verificar el correcto funcionamiento del prototipo nos valdremos de una batería de tests, por lo que en este momento no es necesario contar con una interfaz gráfica.

ESTRUCTURA DEL SISTEMA

A continuación se presenta un diagrama de clases en el que se recogen todas las clases que forman parte de nuestro sistema con los métodos y atributos más representativos de cada una así como las relaciones entre las mismas. Posteriormente, cada una de estas clases y sus relaciones es justificada.

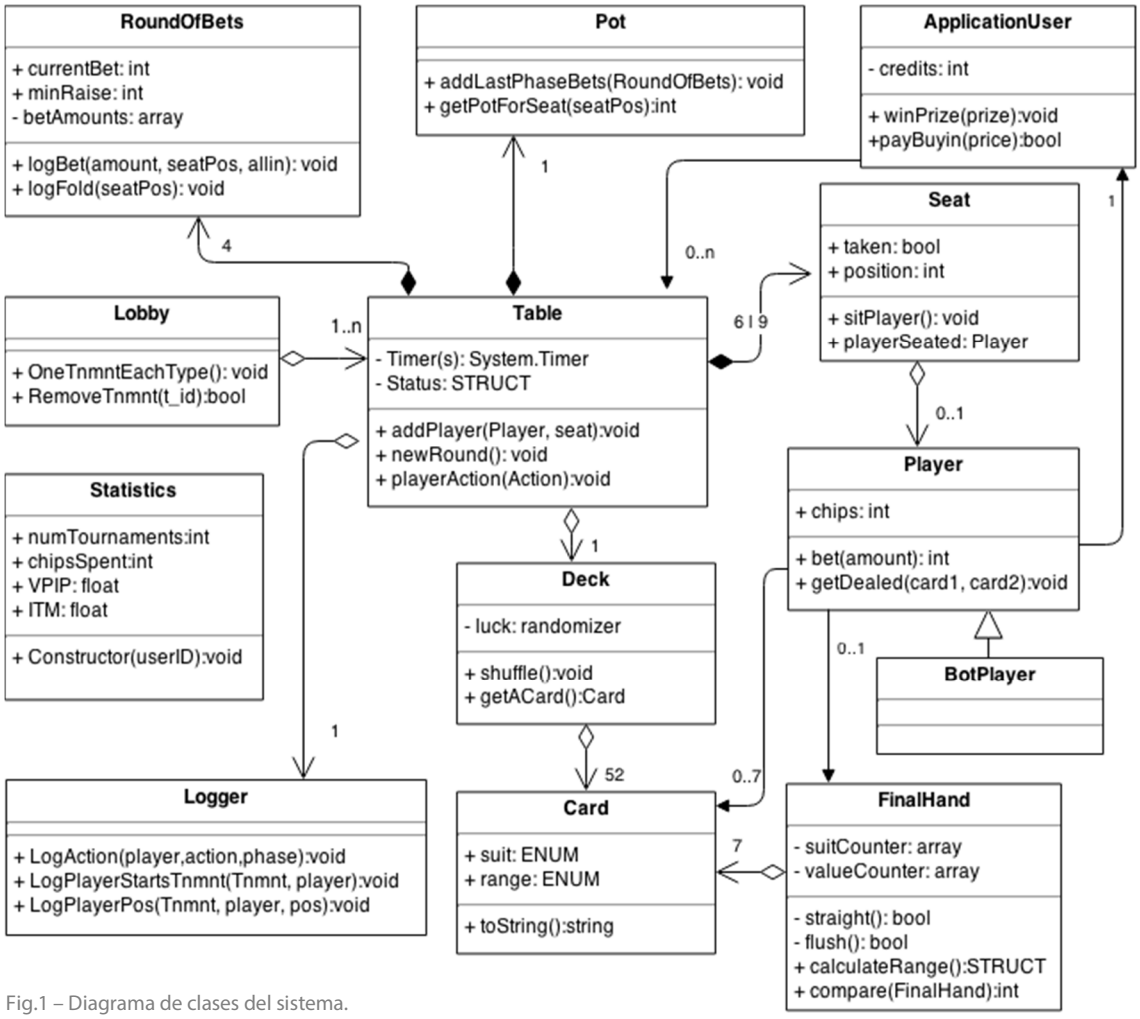


Fig.1 – Diagrama de clases del sistema.

Contemplando el diagrama se aprecia que la clase central que cuenta con más relaciones es la clase ***Table***, lo cual tiene pleno sentido, al ser esta la clase que representa los diversos torneos, que son el eje fundamental de nuestra aplicación. Conviene recordar en este punto que en nuestra aplicación, una mesa es equivalente a un torneo, ya que todos los torneos serán únicamente de una mesa, de ahí el nombre de la clase.

Como ya se ha comentado, se podrán jugar varios torneos al mismo tiempo, por lo que necesitamos una clase que mantenga y controle los diversos torneos. Para ello, utilizaremos la clase ***Lobby***, que como se puede apreciar en el diagrama, estará compuesta de un número de mesas o torneos. Nuestra aplicación contará pues con una única instancia de esta clase que mantendrá el registro de las diversas mesas.

Cada uno de estos torneos cuenta entre otros con un objeto de la clase ***Logger***, el cual se utilizará para almacenar en la base de datos los diversos torneos a los que los jugadores se unen, así como las acciones que éstos realizan a lo largo de los mismos. A partir de esta información, la clase ***Statistics*** calculará las

estadísticas cuando los jugadores las requieran. Esta clase accede directamente a la base de datos, por lo que no necesita interaccionar con ninguna otra clase del sistema.

Para llevar el control de las apuestas y los botes, la mesa cuenta con dos tipos de objetos. El primero de ellos es de tipo ***RoundOfBets*,** y se utiliza para registrar lo que cada jugador ha apostado en una ronda de apuestas (*preflop, flop*…). Al comienzo de cada una de las rondas se creará un nuevo objeto de este tipo, el cual al final de la misma, será agregado a un objeto ***Pot***, que se encarga de combinar las distintas rondas para obtener el bote final junto con la cantidad de bote a la que cada jugador aspira. Esta información será utilizada más tarde a la hora de repartir el bote al jugador o jugadores ganadores.

Evidentemente, para jugar al póker hacen falta cartas, por lo que cada torneo cuenta también con un objeto de tipo ***Deck***, que representa un mazo de cartas y contiene funciones como extraer una carta o barajear. Una baraja de póker está formada por 52 cartas, por lo que tal y como se puede leer en el diagrama, cada objeto de tipo *Deck* contiene 52 objetos de tipo ***Card***.

Volviendo a nuestro objeto mesa, éste contará con 6 ó 9 objetos de tipo ***Seat***, dependiendo de si se trata de un torneo de 6 ó 9 jugadores. Estos objetos representan los distintos asientos de la mesa, y podrán estar vacíos u ocupados por un jugador. En caso de que el asiento esté ocupado, el objeto *Seat* contendrá una referencia a un objeto ***Player***, que es la clase que representa a un jugador en un torneo. Este objeto por lo tanto contiene información y ofrece una serie de métodos relacionados con la participación del jugador en el torneo, como pueden ser las cartas que se le han repartido o la cantidad de chips que tiene el jugador en un determinado momento. La subclase ***BotPlayer*** se utilizará para representar a jugadores no humanos.

Cuando el jugador llega a la fase de *showdown*, el objeto *Player* correspondiente crea un objeto de tipo ***FinalHand***, el cual, a partir de las 7 cartas de las que el jugador dispone (las 5 de la mesa más las dos de mano), obtiene la mejor mano posible y la información precisa para que ésta pueda ser comparada con las de los demás jugadores con el fin de obtener al jugador ganador del bote.

Como ya se ha dicho, los objetos de tipo *Player* simplemente representan a un jugador en un torneo, y no almacenan nada de información relacionada con la existencia del usuario en el sistema, como puedan ser la cantidad de créditos que el jugador tiene para jugar torneos, sus estadísticas, su nombre de usuario o su contraseña. Toda esta información es accedida a través de un objeto de tipo ***ApplicationUser***.
Puesto que en todo momento es necesario identificar a qué usuario del sistema pertenece cada jugador en las mesas, cada objeto de tipo *Player* está relacionado con un objeto de tipo *ApplicationUser*. Sin embargo, como un usuario puede estar jugando en varios torneos al mismo tiempo, un mismo objeto de tipo *ApplicationUser* puede ser referenciado por más de un objeto de tipo *Player*.

ELECCIÓN DE LA MANO GANADORA

A lo largo de una partida, a menudo un jugador consigue con sus apuestas provocar que el resto de jugadores renuncie a sus manos y por tanto llevarse él el bote, sin embargo, si esto no ocurre, los jugadores realizan apuestas hasta la ronda final en la que se decide el ganador según sus cartas. Por lo tanto, una de las partes fundamentales de la aplicación y que va a ser utilizada constantemente, es la de elegir la mano ganadora de entre todos los jugadores que han llegado a la ronda final.

Como ya se ha comentado en la introducción, esta parte del proyecto en particular es de especial interés para el director, y probablemente sea abordada de nuevo en futuros proyectos. Mientras que en nuestro caso la implementación de la misma pretende ser clara y esquemática, poniendo énfasis en la mantenibilidad del código, en un posible proyecto a desarrollar posteriormente, este procedimiento se realizará de una manera más oscura pero radicalmente más eficiente. En concreto se pretende utilizar una variación del método de *Cactus Kev* [4], que emplea lógica combinatoria, árboles binarios, números primos y la representación de cada carta como un número entero para resolver el problema.

En mi particular implementación de este proceso existen dos partes diferenciadas, la primera de ellas consiste en obtener el valor de la mano del jugador, y la segunda, en comparar las manos entre sí para poder decidir cuál es la más fuerte de todas. Si bien la segunda fase no es especialmente compleja, ya que simplemente consiste en obtener el máximo o máximos de una serie de valores, la primera parte, en la que se obtenemos el valor a comparar con el del resto de jugadores, es más interesante y por tanto aparece descrita en más detalle a continuación.

Cuando los jugadores llegan a la ronda final, cada uno cuenta con sus dos cartas de mano, y además, las 5 cartas comunes que en ese momento se encuentran sobre la mesa. Los jugadores disponen por tanto de 7 cartas que pueden combinar para formar su mano. Sin embargo en la variedad de póker con la que estamos tratando en esta aplicación, las manos están formadas solamente por 5 cartas, por lo que el problema queda reducido a encontrar la mejor combinación de 5 cartas posible entre un conjunto de 7.

En la siguiente página se representa mediante un diagrama el razonamiento anterior.

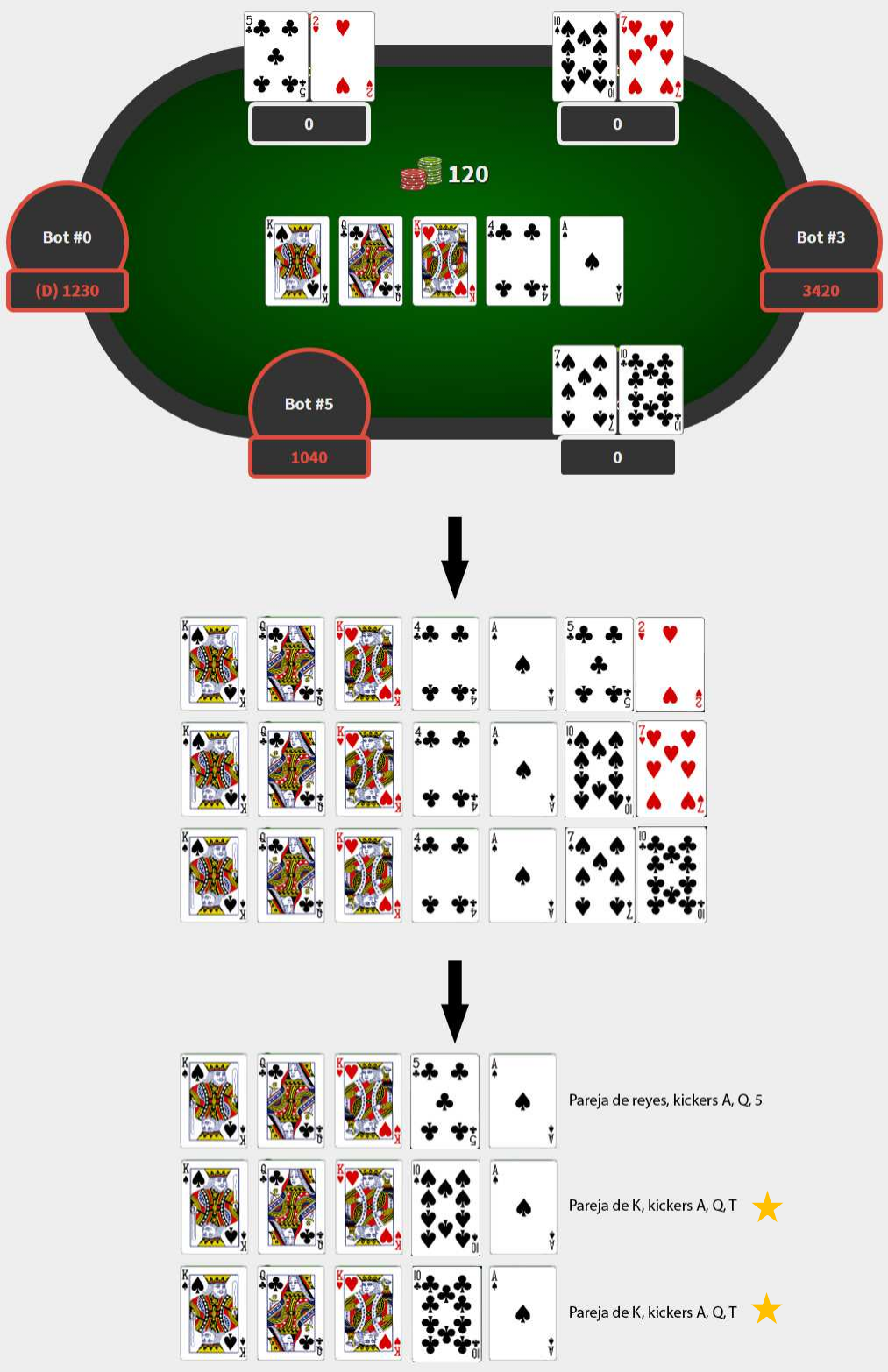


Fig.2 – Representación visual del proceso para seleccionar el jugador ganador.

El primer paso del diagrama es simplemente conceptual, por lo que no conlleva ningún cálculo, sin embargo para el segundo paso, consistente en encontrar la mejor combinación de 5 cartas de entre las 7 disponibles, se utilizará el siguiente algoritmo: primero se intenta encontrar entre las 7 cartas la mano de mayor valor, es decir una escalera real, si ésta existe, devolvemos ese valor, si no, pasamos a buscar en las 7 cartas la siguiente mejor mano, una escalera de color. Este proceso será repetido hasta que encontramos la mano que buscamos.

Una simplificación de dicho algoritmo aparece representada en el siguiente diagrama:

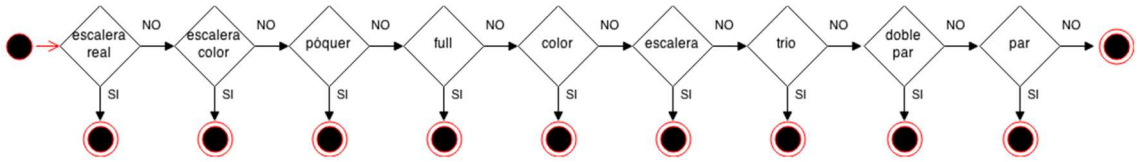


Fig.3 – Algoritmo de asignación de rango a conjuntos de 5 cartas.

Sin embargo, conocer el rango de la mano no es suficiente para comparar las distintas manos, puesto que por ejemplo, un póker de ases es mejor que un póker de reyes, y como siempre necesitamos considerar 5 cartas, un póker de ases acompañado de una J es superior a un póker de ases acompañado de un 3. Por lo tanto, nuestro algoritmo devolverá una estructura que además de contener el rango de la mano, contiene el resto de información que necesitamos para compararla con otras.

Para realizar cada una de las comprobaciones empleadas en las búsquedas de las distintas manos, se utilizarán dos contadores, uno de valores y uno de palos, que almacenarán el número de cartas que hay de cada valor y palo en cada conjunto de siete cartas en el que estemos buscando.

El estado de estos contadores si por ejemplo estamos operando con las siguientes siete cartas sería el siguiente:

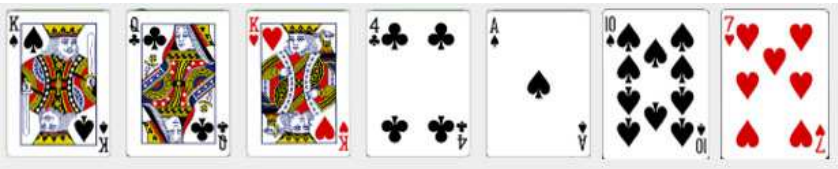


Fig.4 – Ejemplo de conjunto de 7 cartas.

PICAS	CORAZONES	DIAMANTES	TRÉBOLES
3	2	0	2

A	K	Q	J	T	9	8	7	6	5	4	3	2
1	2	1	0	1	0	0	1	0	0	1	0	0

Tablas 1 y 2 – Resultado de los contadores considerando las cartas anteriores.

Basándonos pues en estos dos vectores, la única operación que no es trivial es la de verificar si existe o no una escalera. Sin embargo, es información que necesitamos conocer desde el comienzo del algoritmo (la primera mano que comprobamos es una escalera real, por lo que necesitamos conocer si tenemos o no una escalera), por lo tanto, después de rellenar los contadores pasamos a verificar si podemos o no encontrar una escalera entre nuestras cartas.

Una vez contamos con esta información, la comprobación necesaria para cada una de las manos es muy simple:

- **Escalera real:** Existe una escalera que termina en A y todos sus miembros son del mismo palo.
- **Escalera de color:** Existe una escalera y todos sus miembros son del mismo palo.
- **Póker:** En el contador de valores, existe un 4.
- **Full:** En el contador de valores, aparece el valor 3 repetido o acompañado de un 2.
- **Color:** Existe un valor mayor o igual a 5 en el contador de palos.
- **Escalera:** Ya hemos calculado al comienzo del algoritmo si existe una escalera.
- **Trio:** En el contador de valores somos capaces de localizar un 3.
- **Doble pareja:** Somos capaces de encontrar 2 doses en el contador de valores.
- **Pareja:** Hay algún 2 en el contador de valores.
- **Carta alta: Puesto** que las comprobaciones se realizan en orden, siempre que lleguemos a este punto, podemos garantizar que tenemos Carta alta como mejor mano.

REPARTO DE BOTES

La principal complicación del reparto de botes a los jugadores ganadores reside en que en algunos casos, no todos los jugadores aportan la misma cantidad de fichas al bote, lo que provoca que no todos los jugadores aspiren a la totalidad del bote, creándose por tanto botes divididos.

Según las reglas del póker, un jugador siempre está obligado a igualar o superar la apuesta del jugador anterior si es que quiere seguir participando en la mano, sin embargo, si el jugador no tiene suficientes fichas, éste podrá ver la apuesta siempre y cuando utilice todas las fichas que le quedan. Es en ese momento cuando este jugador aspira a una cantidad de bote menor que el resto de jugadores, puesto que ha contribuido con menos fichas a la formación de éste.

Por ejemplo:

1. El jugador A apuesta 300 fichas.
2. El jugador B iguala la apuesta de 300.
3. El jugador C solo tiene 100 fichas, pero decide que también quiere ver la apuesta.
4. El resto de jugadores no van.

En este momento, el bote cuenta con una cantidad total de 700 fichas, sin embargo, sería injusto que el jugador C se llevara la totalidad de éste, puesto que los otros jugadores han arriesgado 300 fichas mientras que el solo ha arriesgado 100.
Lo que sucede es que se crean por lo tanto 2 botes, uno de ellos, compuesto por 100 fichas de A, 100 de B y 100 de C, y otro bote secundario, al que los jugadores A y B aportan 200 fichas cada uno (300 que han apostado menos las 100 que han puesto en el primer bote) y que solo pueden ganar ellos.

De este modo, si el jugador C resultara ganador, se llevaría el primer bote, de 300 fichas, mientras que el segundo bote, de 400 fichas, sería para el jugador que tenga mejores cartas de entre A y B.

Para hacer posible este reparto, es necesario mantener un registro de la cantidad que cada jugador ha aportado al bote, lo cual complica lo que a priori parecía una operación sencilla. Tal y como se comentó en el apartado ESTRUCTURA DEL SISTEMA, existen dos clases que colaboran entre sí para hacer esto posible. Una de ellas es la clase *RoundOfBets*, la cual gestiona las apuestas que los jugadores realizan durante una determinada ronda de apuestas (*preflop, flop*…). Para ello, estos objetos utilizan un vector en el que almacenan la cantidad que cada jugador ha apostado. Por ejemplo, para nuestro ejemplo anterior, ese vector presentaría el siguiente aspecto al final de la ronda de apuestas.

A	B	C
300	300	100

Tabla 3 – Vector con la cantidad de fichas apostada por cada jugador.

Una vez la ronda de apuestas finaliza, este objeto obtendrá a partir de nuestro vector los distintos botes resultantes de la ronda, y los almacenará en un diccionario en el que las claves serán los jugadores que aspiran al premio, y los valores la cantidad de fichas a la que aspiran.

Para ello, utiliza el siguiente algoritmo:

Mientras el vector no esté vacío:

1. Obtener la menor cantidad presente en el vector.
2. Restar dicha cantidad todas las cantidades presentes en el vector.
3. Crear un bote al que aspiran todos los jugadores del vector con un valor igual al producto de dicha cantidad por el número de jugadores en el vector.
4. Añadir bote al diccionario de botes.
5. Eliminar del vector aquellos jugadores con una cantidad de fichas apostadas igual a 0.

Si seguimos con nuestro ejemplo, tras aplicar el algoritmo obtendremos el siguiente diccionario:

A,B,C	A,B
300	400

Tabla 4 – Diccionario con las cantidades y los jugadores que aspiran a ellas

Este diccionario es entonces procesado por un objeto de la clase *Pot* que representa el bote general de la mano, es decir, la agregación de los botes de todas las rondas, y que es el que será posteriormente repartido al jugador o jugadores ganadores. Este objeto *Pot* contará con una seria de botes que han sido generados por las rondas anteriores (o estará vacío si es la primera ronda de apuestas) con los que combinará botes de la nueva ronda. Este proceso simplemente consistirá en sumar las cantidades de los botes con la misma clave y añadir los botes con claves nuevas a su lista de botes. Si por ejemplo nuestro objeto *Pot* contaba con un único bote de 1000 fichas al que aspiraban los jugadores A, B y C, tras agregar nuestra ronda de apuestas, los botes resultantes serán:

A,B,C	A,B
1300	400

Tabla 5 – Aspecto del diccionario con los botes tras la agregación.

A partir de esta estructura, cuando haya que pagar los premios simplemente tenemos que iterar sobre los distintos botes y seleccionar al ganador o ganadores de entre los jugadores que forman parte de cada bote y pagar la cantidad asignada a dicho bote.

BASE DE DATOS

La función de la base de datos en la aplicación es principalmente la de almacenar información acerca de la actividad de los jugadores, la cual será más adelante utilizada para calcular las estadísticas descritas en el apartado "Descripción del prototipo".

Concretamente, las estadísticas se van a calcular según las fórmulas utilizadas en el software *PokerTracker*[5], las cuales son de dominio público y para cuyo cálculo precisan de la siguiente información: número de torneos que se ha jugado de cada tipo y posición en la que se ha finalizado, fichas gastadas en unirse a los torneos, fichas ganadas en premios, número de ocasiones que el jugador ha llegado a la ronda de *showdown* y la cantidad de veces en la que éste ha realizado una determinada acción en una determinada ronda.

Toda esta información será almacenada y actualizada en una base de datos modelada según el siguiente esquema entidad-relación.

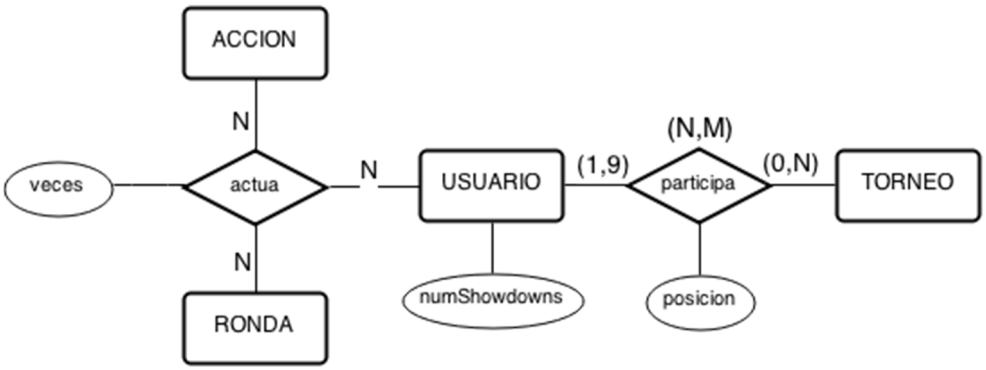


Fig.5 – Esquema entidad-relación simplificado de la base de datos del sistema.

Al cual le corresponde el siguiente modelo de datos:

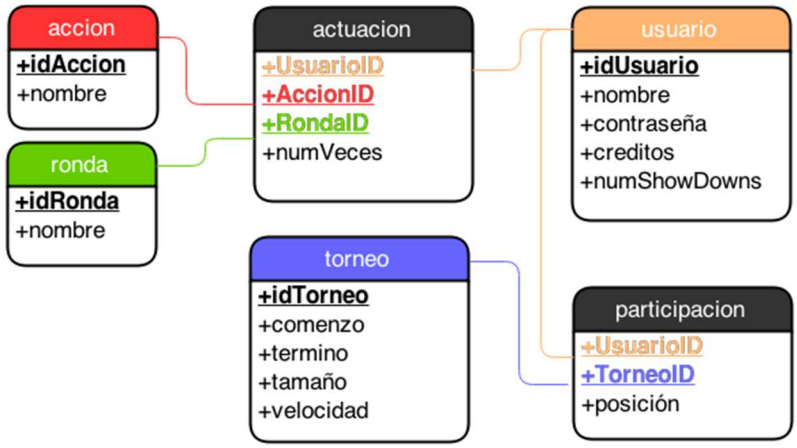


Fig.6 – Modelo de datos de la base de datos del sistema.

Una vez se obtuvo el modelo de datos, se adaptó al formato de "*Entity Framework*", una herramienta de Microsoft que ofrece una capa entre la base datos y la aplicación, y que además de facilitar las consultas e inserciones, se encargó de traducir nuestro modelo de datos a una estructura física. Habiendo creado la base de datos, las operaciones de inserción y consulta necesarias para el cálculo de las estadísticas resultan triviales. Éstas serán llevadas a cabo por un objeto de tipo *Logger*, tal y como se recoge en el apartado "Estructura del sistema", en la página 5.

INTELIGENCIA ARTIFICIAL

El papel de la inteligencia artificial en el proyecto es únicamente el de brindarle al usuario humano la posibilidad de jugar en mesas completas y no el de poner a prueba sus habilidades como jugador, por lo tanto, la finalidad de la IA en el proyecto es simplemente la de permitir o facilitar la utilización completa de la aplicación, lejos de ser el principal foco de atención o característica de ésta. Este hecho nos permite ciertas licencias en cuanto al nivel de complejidad requerido a la IA.

Para contar con una IA exigente, sería necesaria la aplicación de algoritmos de IA que funcionan según heurísticas para cuya obtención es necesario involucrar información como las fichas apostadas, las fichas de otros jugadores, y sobre todo, las cartas de las que el jugador dispone en un determinado momento y las posibilidades combinatorias de estas. Es éste último tipo de información el que supone un mayor problema, puesto que tal y como se explica en el apartado ELECCIÓN DE LA MANO GANADORA, toda la lógica dedicada a la comparación y obtención de valores de manos de póker se basa en conjuntos de siete cartas, lo cual implica que un porcentaje muy pequeño de dicho código podría ser reutilizado y por tanto una gran cantidad de nuevo trabajo sería necesaria para proveer esta heurística. Además por supuesto, deberían implementarse los complejos algoritmos requeridos para la utilización de esta heurística y consecuente toma de decisiones.

Se decidió por tanto que la IA ignoraría las cartas y basaría la toma de decisiones en otra información más fácilmente accesible. En una primera aproximación se consideró utilizar únicamente la probabilidad como estrategia decisiva por parte de la IA, lo cual nos permitiría modelar a los jugadores artificiales de modo que sus estadísticas sean idóneas (VPIP, PFR, Factor de agresividad…). Sin embargo, la excesiva simplicidad de este modelo dio lugar a una actitud repetitiva y predecible por parte de los jugadores artificiales, según la que éstos tomaban a menudo decisiones que un jugador real nunca tomaría, incluso si como los jugadores artificiales, éste no tuviera constancia de sus cartas. Y es que si bien la frecuencia de las decisiones tomadas era ideal, estas eran tomadas de manera independiente al estado de la mano.

Como consecuencia, se decidió pasar a incorporar información del estado del juego a la toma de decisiones. Concretamente, se consideraron la magnitud de la apuesta a la que el jugador se enfrenta, la cantidad de fichas del bote a las que aspira y la ronda de apuestas en la que se encuentra, información toda ella inmediata o fácilmente accesible.

Concretamente, a partir de la información que se comenta en el párrafo anterior se van a obtener dos valores, el primero de ellos, al que denominaremos “b” (de bote), representa la cantidad de fichas que el jugador se podría llevar en caso de resultar ganador de la mano, y el segundo, denominado “a” (de apuesta) que es la cantidad de fichas que el jugador debe de poner en el bote para igualar la apuesta anterior y seguir por tanto en el juego. Una vez calculados, estos coeficientes se emplean según se puede observar en el árbol de la derecha, el cual concretamente se utiliza para la ronda de *preflop*.

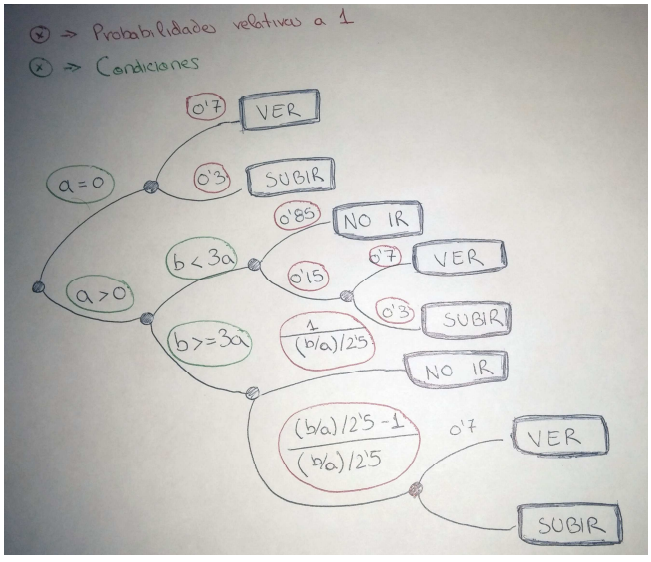


Fig.7 – Árbol de decisión de la IA.

Como se aprecia en el árbol, a no ser que el jugador pueda pasar (no necesita pagar más fichas para continuar en el juego; a=0), la probabilidad de que un jugador renuncie a su mano dependerá de a y b. Concretamente, en el caso de la ronda *preflop* representada en el árbol:

- si **b<3a** la probabilidad de es de 0.85
- si **b >= 3a** la probabilidad es $\frac{1}{(b/a)/2.5}$

Donde 3 y 2.5 son constantes que varían dependiendo de la ronda en la que nos encontramos.

Expresado verbalmente, la aplicación de la ecuación implica que cuantas más fichas se puede llevar el jugador en relación a su apuesta, es menos probable que éste renuncie a su mano.

Los árboles de decisión utilizados para las otras rondas son análogos, sin embargo, como ya se ha comentado, se utilizarán distintas constantes con el objetivo de representar la tendencia del jugador a resistirse a renunciar a su mano conforme más avanzada se encuentra la ronda. Concretamente, las constantes utilizadas en las distintas rondas fueron:

RONDA	CONSTANTE COMPARACION	CONSTANTE ECUACIÓN
PREFLOP	3	2.5
FLOP	2.7	2.25
TURN	2.5	2.1
RIVER	2.35	2

Tabla 6 – Constantes utilizadas para las distintas rondas de apuestas.

Estas constantes fueron obtenidas por aproximación con el objetivo de conseguir un comportamiento similar al de un jugador humano.

Una vez tomada la decisión de si renunciar o no a la mano, en caso de que se haya decidido seguir en el juego todavía se debe decidir si simplemente se igualará o se subirá la apuesta.

En una partida de jugadores humanos, éstos jugarán más agresivamente cuanto más avanzada se encuentre la mano, por ejemplo, en la ronda de *preflop* algunos jugadores igualan las apuestas anteriores para poder ver el *flop*, mientras que en rondas posteriores subirán las apuestas para intentar asustar a sus rivales y obligarles a renunciar, en otras palabras, conforme avanzan las rondas los jugadores tienden a jugar más de una manera más agresiva (subir más e igualar menos).

Para plasmar este comportamiento en la IA, nos basaremos simplemente en probabilidades de modo que si se ha decidido que el jugador no renuncia a su mano, este igualará la apuesta o la subirá según las probabilidades reflejadas en la siguiente tabla:

RONDA	% VER	% SUBIR
Preflop	70	30
Flop	50	50
Turn	45	55
River	35	65

Tabla 7 – Posibilidades de que un jugador vea o suba en cada ronda.

A continuación se aplican los conceptos explicados anteriormente en la siguiente tabla, en la que se recogen una serie de valores para los coeficientes a y b junto con las posibilidades de que el jugador tome una determinada acción. Si se es familiar con el juego de póker se puede apreciar que estos valores son bastante coherentes con las decisiones que un jugador real tomaría si contara con la misma información que la IA.

A – Cantidad de fichas que el jugador debe de apostar para seguir optando al bote.
B – Cantidad de fichas que el jugador se llevaría en caso de resultar ganador de la mano en ese momento.

RONDA	A	B	NO IR	VER	SUBIR
Preflop	20	30	**85 %**	**10.5 %**	**4.5 %**
	20	70	**71.42 %**	**20 %**	**8.57 %**
	20	150	**33.33 %**	**46.66 %**	**20 %**
Flop	20	80	**56.25 %**	**21.87 %**	**21.87 %**
	60	300	**45 %**	**27.5 %**	**27.5 %**
	100	800	**28.12 %**	**35.93 %**	**35.93 %**
Turn	20	200	**21 %**	**35.55 %**	**43.45 %**
	80	200	**84 %**	**7.2 %**	**8.8 %**
	240	1000	**50 %**	**22.32 %**	**27.28 %**
River	20	150	**26.66 %**	**25.66 %**	**47.66 %**
	100	800	**25 %**	**26.25 %**	**48.75 %**
	300	1200	**50 %**	**17.5 %**	**32.5 %**

Tabla 8 – Valores para A y B junto con las probabilidades que la IA tiene de tomar cada decisión en cada ronda.

Tal y como se observa en la tabla, se ha conseguido una IA aceptablemente competitiva que simula adecuadamente el comportamiento de un jugador humano y que puede poner a prueba a los jugadores menos experimentados, lo cual es más que suficiente para el papel de ésta en nuestra aplicación.

INTEGRACIÓN EN EL SERVIDOR WEB

ADAPTACIÓN AL PATRÓN MVC

Hasta ahora toda la interacción con nuestra aplicación se había venido realizando simplemente mediante llamadas por parte del *framework* de tests a los métodos que exponen las diversas clases. En este paso sin embargo, se va a posibilitar la interacción de usuarios reales con ésta, para ello, será necesario integrar nuestro núcleo en un servidor web de modo que nuestra aplicación sea accesible a través del navegador, que es tal y como los usuarios interaccionarán con nuestro programa.

Para ello, debemos de integrar nuestro núcleo de póker en el software del servidor web (Microsoft IIS con ASP.NET MVC4), el cual ofrece un cómodo *framework* basado en el patrón Modelo Vista Controlador (MVC). La aplicación concreta de este patrón que realiza ASP.NET consiste en que el controlador recibe peticiones HTTP, éste interacciona con el modelo para ejecutar la operación requerida y posteriormente, inyecta cierta información en las vistas, lo cual resulta en el documento HTML que se sirve al cliente web.

Así pues, aplicando esto a nuestro caso, vemos que nuestro núcleo de póker va a pasar a formar parte del modelo, mientras que necesitaremos crear los controladores y vistas apropiados para posibilitar la interacción con éste. La interacción con nuestro modelo tendrá lugar pues según el siguiente diagrama:

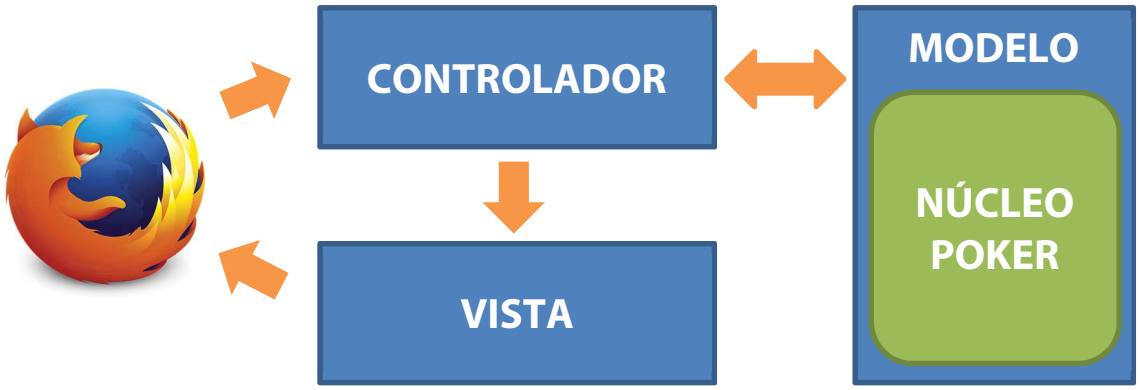


Fig.8 – Modelo de interacción entre los componentes del sistema.

De las operaciones necesarias para completar la integración en el servidor web, la más sencilla fue la de la inclusión de nuestro núcleo en el modelo de la aplicación, de hecho no requirió ningún desarrollo adicional, si no que bastó con indicar mediante la configuración donde se encontraban estos modelos.

En cuanto a los controladores, se decidió crear cuatro distintos, uno para cada parte diferenciada de nuestro sistema.

CONTROLADOR DE USUARIOS
A través del cual se realizan todas las operaciones relacionadas con el manejo de los usuarios del sistema. Entre otras cosas ofrece *URLs* a través de las cuales identificar a los usuarios en el sistema, permitir su registro, cambio de contraseñas etc.

Proporciona por tanto, accesibles mediante peticiones HTTP GET, *URLs* que generan vistas con los formularios requeridos, y por otro lado, *URLs* a las que enviar la información que los usuarios introducen en los formularios, las cuales son accedidas mediante peticiones HTTP POST.

CONTROLADOR DE SALÓN
Este controlador únicamente ofrece dos *URLs*. La primera de ellas genera una página web con todos los torneos disponibles en este momento junto con controles para acceder bien como jugador o espectador, mientras que la segunda URL, utilizada para actualizar la lista de torneos en los que el jugador está participando mostrada en la interfaz gráfica, simplemente genera un objeto JSON con esta información, y por tanto, no está prevista ser accedida directamente, sino mediante peticiones asíncronas de las que el usuario no será consciente.

CONTROLADOR DE ESTADÍSTICAS
La única función de este controlador es la de ofrecer una dirección a la que el usuario pueda acceder para consultar sus estadísticas. No será necesario pues más que una URL accesible mediante HTTP GET que genere la vista adecuada.

CONTROLADOR DE TORNEOS
Se trata del controlador principal a través del cual tiene lugar casi toda la interacción con el sistema. En concreto, se encarga de presentar la página web con toda la información y los controles que el usuario necesita del torneo, de ofrecer información para mantener ésta actualizada y también de proporcionar *URLs* para comunicar al servidor las acciones tomadas por los jugadores.

Las operaciones ofrecidas por este controlador se dividen pues según estas estén dedicadas a proporcionar información del torneo o posibilitar la interacción del jugador con el servidor.

Aquellas que están dedicadas a proporcionar información serán accedidas mediante peticiones HTTP GET, y de entre ellas, podemos destacar las siguientes:

- ***Index*** – Generará un documento HTML con la información acerca del torneo necesaria para el jugador. Está previsto por tanto que la URL correspondiente a esta operación sea accedida directamente.
- **Estado** – Produce un objeto JSON con la representación del estado del juego en un determinado instante. Destinado por tanto a ser accedido a través de peticiones AJAX.

En el siguiente apartado de esta sección, dedicado a "Asincronía", se explica en profundidad la necesidad de la operación **Estado** y cómo ésta se utiliza en el sistema.

Por otro lado, tenemos aquellas operaciones que ofrecen al usuario la posibilidad de interaccionar con el servidor, la mayoría de las cuales son accedidas mediante peticiones HTTP POST. Debido a la naturaleza interactiva de las mismas, estas operaciones requieren ser accedidas por un usuario identificado en el sistema. Dentro de este grupo, cabe destacar las siguientes:

- **Unirse** – Como su nombre indica, permite al usuario unirse a un determinado torneo en una determinada posición. Se servirá de las cookies para identificar de qué usuario se trata y recibirá el torneo y el asiento al que asignar al jugador como argumentos de la petición HTTP.
- **Salir** – Permite a un usuario dejar un torneo siempre y cuando éste esté todavía en la fase de esperar a más jugadores. Al igual que la anterior, se sirve de las cookies para identificar al usuario en cuestión y recibe el torneo como parámetro.
- **Votar** - Antes de comenzar un torneo, los jugadores que en ese momento están registrados en el mismo pueden decidir si comenzar ya, esperar a más jugadores a añadir *bots* a los asientos vacíos. Esta operación es utilizada para registrar estás decisiones por parte de los usuarios. Para ello, además de las cookies para saber de qué usuario se trata, recibe un parámetro con la decisión del usuario.
- **No Ir** – Sirve para indicar al servidor que el jugador al que le toca actuar ha decidido renunciar a su mano. Como en los casos anteriores, comprueba mediante las cookies de qué usuario se trata, y tras verificar que efectivamente dicho usuario es el siguiente en actuar, pasará la petición al modelo.

- **Ver** – Análoga en su funcionamiento a la operación anterior, indica al servidor que el jugador ha decidido ver la apuesta. Tampoco necesita ningún parámetro ya que el servidor tiene constancia de cuál es el valor de la apuesta actual.
- **Subir** – Similar a las dos anteriores, sirve para enviar la acción de subir al servidor. En este caso, es necesario enviar también un parámetro que contiene la cantidad a la que el jugador ha decidido subir.

Así pues, a través de estas operaciones, ofrecemos al usuario toda la información e interacción necesaria para utilizar la aplicación de forma completa, lo cual concluye el desarrollo de los controladores.

La tercera parte del patrón se trata de las vistas, o en otras palabras, de cómo el controlador devuelve al navegador el resultado de la operación. Muchas de ellas, como ya se ha comentado, simplemente generan pequeños fragmentos de JavaScript representando objetos JSON, sin embargo, hay otras que generan código HTML que será posteriormente interpretado por el navegador, el cual lo mostrará tras combinarlo con el código CSS.

Acerca de éstas últimas, de la información de contienen, de cómo la muestran, de su diseño etc., se habla en la tercera sección de la memoria dedicada a la interfaz gráfica.

ASINCRONÍA

En este apartado se va a reflexionar acerca de cómo mantener la interfaz gráfica actualizada de modo que cuando un usuario esté en un torneo, éste cuente en todo momento con una representación fiel del estado actual del juego.

Tal y como se ha explicado en el apartado anterior, la interfaz a través de la que un usuario interacciona con el servidor para jugar un torneo es generada en forma de documento HTML cuando visitamos la URL que desencadena la llamada *Index* en el controlador de los torneos. Este documento HTML contendrá una interfaz que representa el estado del torneo en el momento en el que se visitó la URL, sin embargo, debido a la naturaleza dinámica del juego de póker, el estado representado en la interfaz pronto diferirá del estado real del juego.

Evidentemente, una representación del juego diferente a la real impide el desarrollo de la partida, por lo que la interfaz debe de ser actualizada tan pronto como se produzca un cambio en el estado del juego.

Si bien hay ciertas propiedades del estado del juego que varían cada segundo (por ejemplo, el contador de tiempo restante para actuar), la mayoría de las variaciones en este suceden una vez que el jugador al que le toca actuar ha realizado su jugada, es decir, de manera totalmente asíncrona e impredecible. Por lo tanto, el modelo ideal de sincronización sería uno en el que el servidor informará a los clientes conectados a una partida (navegadores de los usuarios) cuando se ha producido un cambio, es decir, un modelo de sincronización por interrupción del cliente por parte del servidor como el que se muestra a continuación.

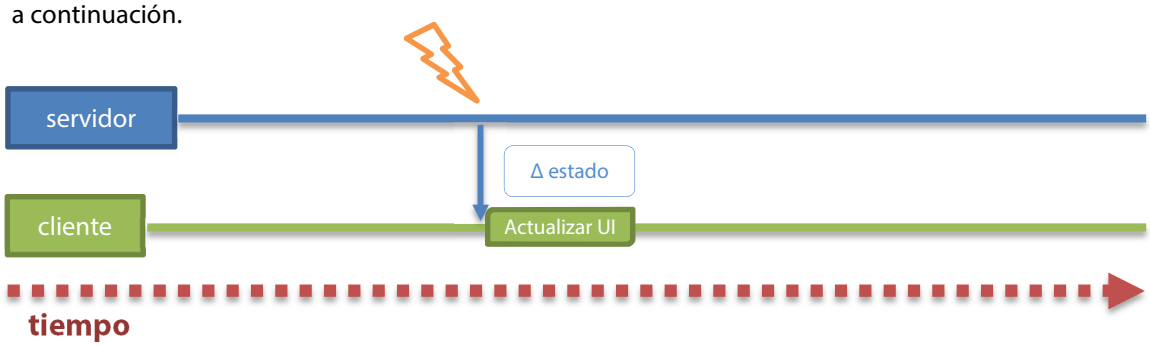


Fig.9 – Modelo de comunicación ideal entre cliente y servidor.

Sin embargo, en el momento de desarrollar el proyecto todavía no existía ninguna tecnología lo suficientemente desarrollada como para ofrecer soporte a este modelo de comunicación sobre HTTP, por lo que se tuvo que renunciar a este modelo y pasar a implementar un modelo de sincronización por encuesta. Así pues, la sincronización consistirá en el cliente periódicamente preguntando al servidor si ha habido algún cambio, y en caso de que lo haya habido, actualizar la interfaz acordemente.

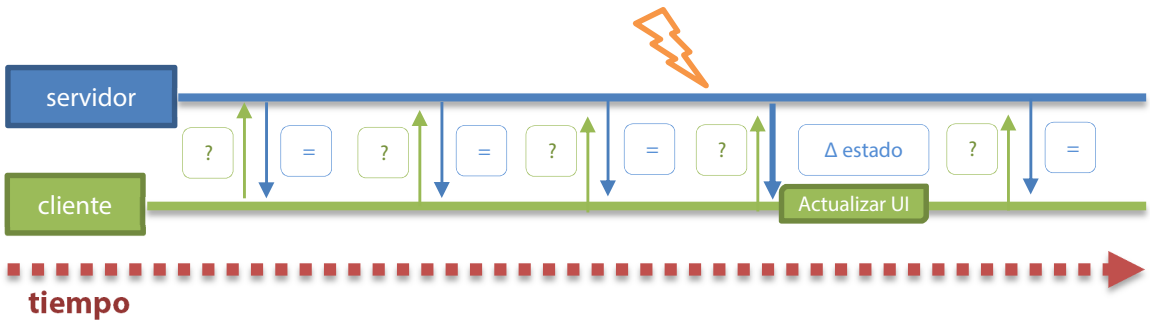


Fig.10 – Modelo de comunicación alternativo entre cliente y servidor.

En este momento nos encontramos con otro problema derivado de la naturaleza sin estado del protocolo HTTP. Puesto que el servidor no mantiene un estado de las distintas conexiones, para preguntar si ha habido algún cambio es necesario enviar desde el cliente el estado respecto al que esperamos los cambios, para que entonces el servidor conteste con los diversos cambios con respecto al estado proporcionado en caso de haberlos. Éste modelo por lo tanto implica que cada vez que el cliente pregunta por cambios, una representación completa del estado debe de ser enviada al servidor, el cual responderá con los cambios respecto a esta:

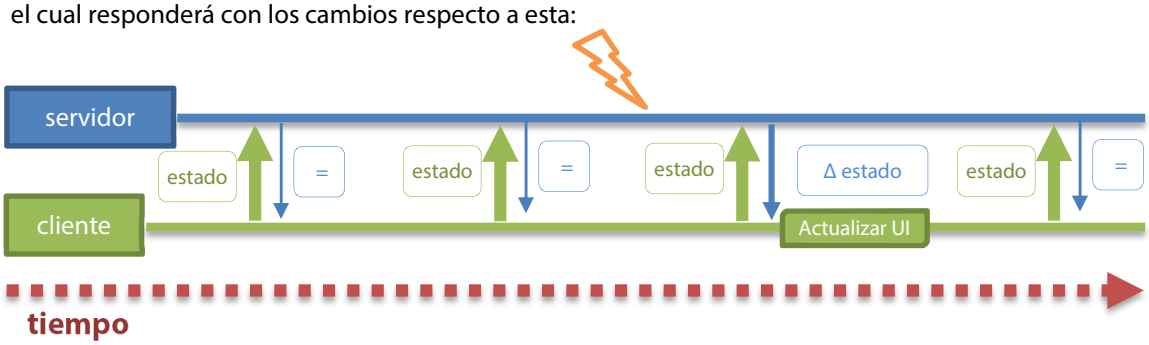


Fig.11 – Modelo de comunicación alternativo y sin estado entre cliente y servidor

A partir de este modelo, cuya implementación es técnicamente posible, se decidió realizar la siguiente optimización. Puesto que el estado debe de ser enviado por parte del cliente en cada ocasión, y en caso de haber cambios, éste recibe también los cambios sobre el mismo, se decidió que el cliente dejaría de enviar el estado y de preguntar por cambios, y pasaría simplemente a pedir una copia del estado actual al servidor.

Esto provoca que haya menos intercambio de información en caso de que se produzcan cambios, y además, la información es ahora descargada por el cliente, en lugar de enviada por éste al servidor, lo cual suele resultar más rápido. Además, la implementación de este modelo resulta más sencilla tanto en el servidor como en el cliente. Representado gráficamente, el modelo utilizado es el siguiente:

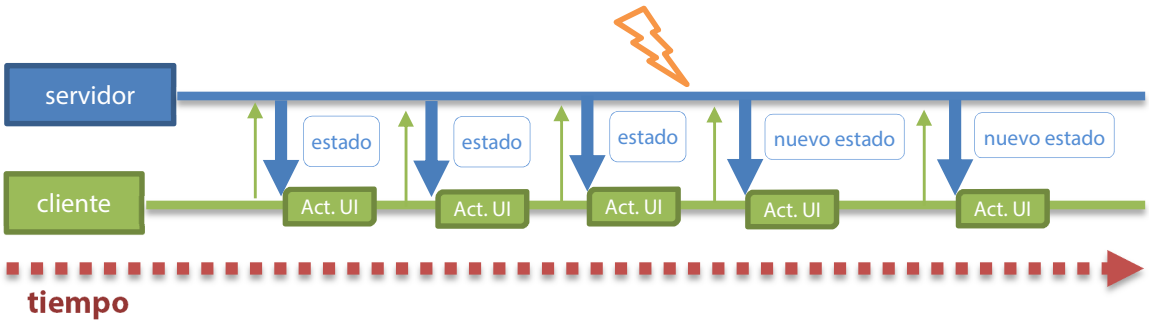


Fig.12 – Modelo de comunicación entre cliente y servidor finalmente implementado.

Si bien según el diagrama puede parecer poco eficiente que el estado de la interfaz se actualice para cada petición, a la hora de la implementación real, esta actualización es necesaria, ya que las peticiones son realizadas cada segundo, por lo que como mínimo el contador de tiempo deberá de ser actualizado cada vez que un nuevo estado es recibido.

Además de este mecanismo de sincronización, para evitar retrasos y ofrecer una interfaz lo más responsiva posible, cuando el usuario envíe al servidor la acción que desea realizar, éste le responderá con la representación del estado de la mesa tras ejecutar dicha acción, de modo que la interfaz puede ser actualizada inmediatamente en lugar de esperar a que la siguiente petición de estado sea efectuada.

Una vez el cliente ha recibido el objeto que representa el estado de la mesa, simplemente se debe de acceder al DOM de nuestra página web y actualizar la información que sea precisa y mostrar u ocultar ciertas partes según corresponda. Estas tarea así como la de encuestar al servidor periódicamente son llevadas a cabo por un objeto JavaScript que contiene métodos como: *ComenzarEncuesta*, *ActualizarDesdeJSON*… etc.

CONCURRENCIA

Al hacer nuestra aplicación accesible desde un servidor web, estamos permitiendo el acceso concurrente a la misma de los usuarios, lo cual puede provocar ciertas condiciones de carrera que deben de ser tenidas en cuenta.

Tal y como se comenta en el apartado "Estructura del sistema", la manera en la que se controlarán los diversos torneos es a través de un objeto de la clase *Logger*, la cual está programada según el patrón *Singleton*. Así pues, todos los usuarios que acceden a la aplicación van a tener acceso a la misma instancia del objeto, y a través de ésta, todos los usuarios involucrados en un torneo, tendrán pues acceso a la misma instancia de objeto *Table*.

Imaginemos pues que tal y como se representa en el siguiente diagrama, un usuario A ha enviado una petición para consultar el estado de la mesa (necesaria como se ha explicado en el apartado anterior para mantener la interfaz sincronizada), y que mientras el método correspondiente está ejecutándose construyendo el objeto con la información del estado, otro usuario B realiza una acción, modificando consecuentemente el estado de la mesa. El método que genera el objeto representando el estado que el usuario A ha solicitado ha empezado pues su ejecución en un estado y habrá finalizado con la mesa en otro estado distinto, habiendo construido un objeto en el que algunas propiedades, las que se procesan primero, tendrán valores pertenecientes al estado anterior a la acción de B, y en el que otras, aquellas que fueron procesadas después de que B realizara su acción, pertenecerán al estado posterior, o quizás a uno intermedio que nunca existió realmente.

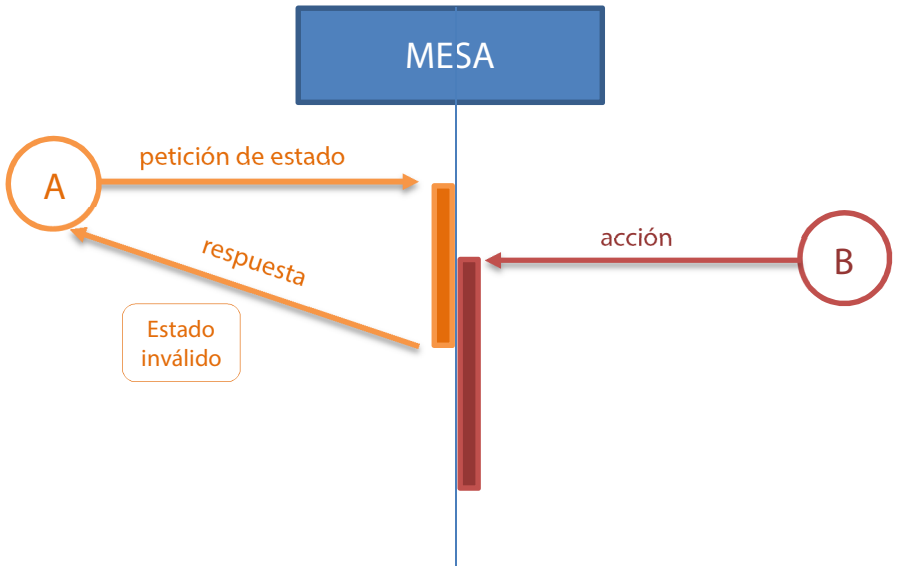


Fig.13 – Representación de condición de carrera.

Si bien ambas operaciones se efectúan en distintos *threads* en el servidor, ambas utilizan la misma instancia del objeto *Table*, por lo que estaremos en una situación conflictiva que debemos de evitar. Si tenemos en cuenta además que cada jugador en la mesa va a realizar una petición de estado cada segundo, es bastante probable que en algún momento dado una situación como esta vaya a suceder.

Necesitamos por lo tanto algún mecanismo que nos permita controlar el acceso de los diferentes *threads* a los recursos compartidos. Para conseguirlo, vamos a añadir a cada mesa un semáforo que gestione el acceso concurrente a dicha mesa. Particularmente, se utilizará un objeto de la clase *ReaderWriterLockSlim,* que forma parte de la librería de C#.

Este semáforo reconoce dos tipos de acceso a nuestros recursos, de lectura y de escritura, lo cual permitirá la ejecución en paralelo de varias operaciones de lectura, mientras que bloqueará las

operaciones de escritura hasta que no haya ningún *thread* leyendo. Del mismo modo, si hay un *thread* que tiene reservado el semáforo en modo escritura, ninguna operación de lectura podrá comenzar hasta que ésta no haya concluido.

Para conseguir este comportamiento, rodearemos cada operación de lectura de nuestra aplicación de las consiguientes llamadas al semáforo (obtención del permiso de lectura y liberación del mismo una vez hayamos finalizado), al igual que haremos con las operaciones de escritura (obtención del permiso de escritura y liberación del mismo una vez hayamos finalizado).

En caso de no poder conseguir el permiso inmediatamente, el *thread* correspondiente se bloqueará hasta que la operación que estaba impidiendo la obtención del permiso deseado termine, como ocurre en cualquier implementación de semáforo.

Si aplicamos estos cambios a nuestra aplicación, el ejemplo representado en el diagrama ya no se podría dar, puesto que la operación de consultar estado habría reservado el permiso de lectura, y por tanto, la operación mediante la cual el jugador B realiza su acción (escritura o modificación del recurso) quedaría bloqueada hasta que el permiso de escritura pueda ser obtenido.

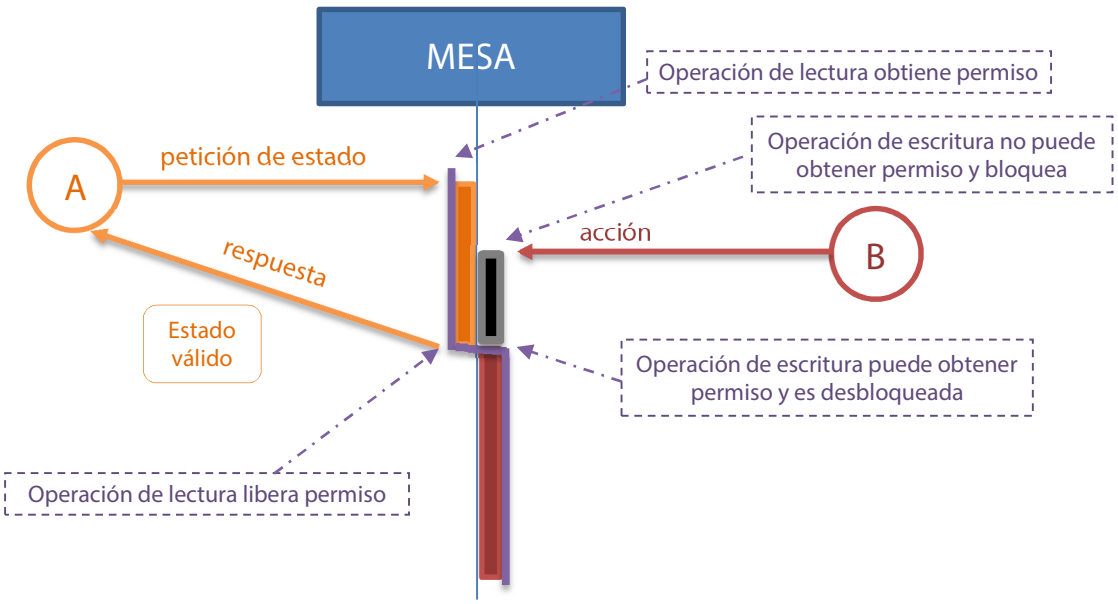


Fig.14 – Representación del uso de semáforos para evitar condición de carrera.

INTERFAZ DE USUARIO

PAUTAS GENERALES Y ESTRUCTURA

Tal y como se comenta en la introducción al proyecto, la característica que diferencia a nuestra aplicación del resto es que puede ser operada simplemente a través de un navegador web. Por lo tanto, además de ofrecer al usuario de una manera clara y concisa la información y controles que necesita como cualquier interfaz gráfica, también debe de observar una serie de pautas y/o buenas prácticas de diseño web. De este modo los usuarios, a los que se les asume familiaridad con páginas web, podrán disfrutar de una experiencia similar a la de otras páginas y por tanto manejarse por nuestra aplicación de una manera más intuitiva.

Así pues, teniendo en cuenta estas prácticas de diseño web, se construyó la siguiente estructura general para nuestra interfaz, la cual será utilizada en todas las pantallas de la aplicación:

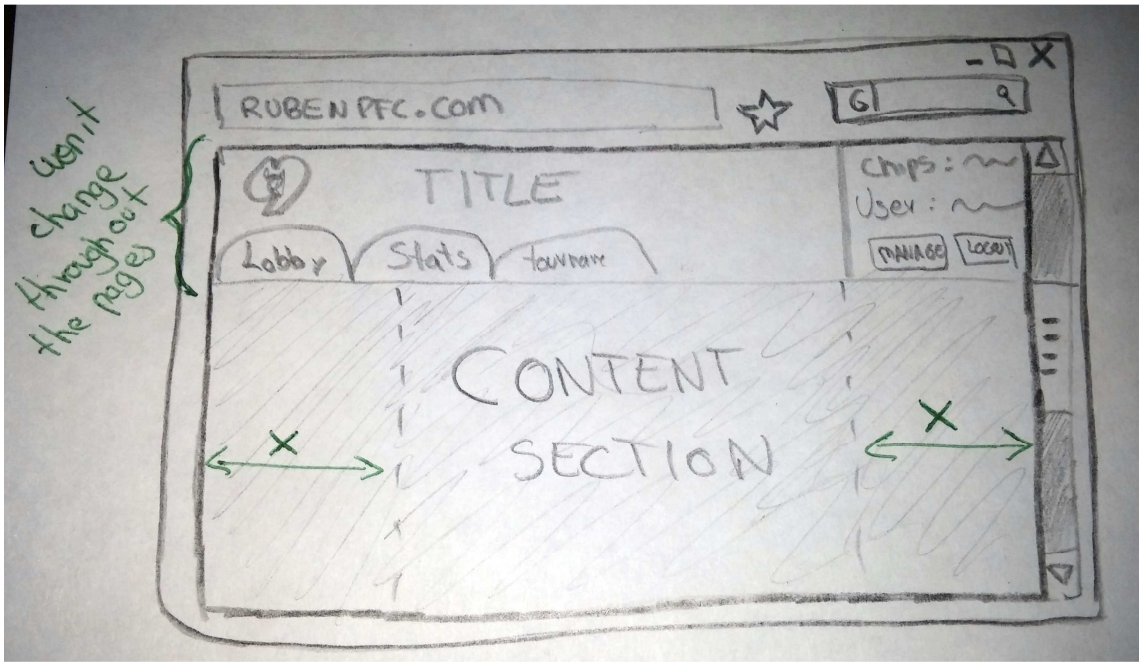


Fig.15 –Boceto de la estructura general de la interfaz.

Como se puede observar, la interfaz consta de una cabecera o encabezamiento el cual se mantendrá constante en las distintas pantallas. En éste, además del logotipo de la aplicación y un pequeño texto a modo de créditos, aparece un menú vertical con las diversas páginas a las que el usuario puede acceder. Por último se destina una pequeña porción del encabezamiento a ofrecer información y controles relacionados con el usuario.

El diseño de este encabezamiento fue inspirado por el de la página de apuestas *betfair*, en cual como se puede apreciar tiene una estructura similar:

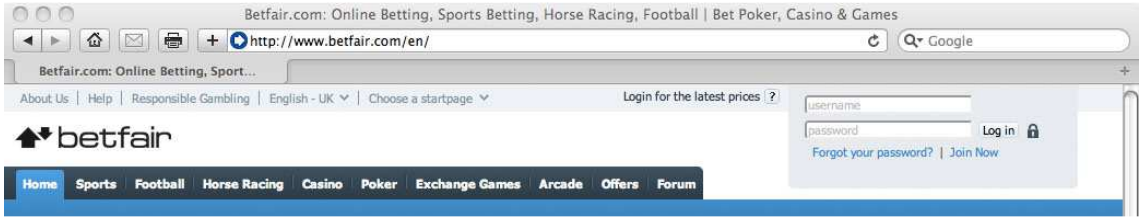


Fig.16 –Cabecero de betfair que sirvió como inspiración.

Debajo del cabecero aparece la sección principal de la página, la cual contará con el título y el contenido principal de esta.

Al contrario que la mayoría de sitios web, se decidió renunciar a añadir un pie de página a nuestro diseño debido a que la información que normalmente se muestra en éste no era de aplicación en nuestro caso al tratarse de un proyecto académico.

Puesto que según las estadísticas solamente un 1% por ciento de usuarios de ordenadores utiliza una resolución inferior a 1024x768 [6], se decidió que ésta sería la mínima resolución necesaria para una adecuada visualización de nuestra aplicación. Para resoluciones superiores, nuestra interfaz se extenderá verticalmente hasta rellenar todo el espacio disponible, variando el tamaño de sus elementos proporcionalmente. Sin embargo, para evitar tamaños demasiado grandes de algunos elementos, como por ejemplo la mesa de póker, se limitará la anchura máxima de la parte dedicada al contenido, de modo que si la página es más grande que dicho límite, ésta quedará centrada verticalmente.

A través de toda la interfaz, al igual que en el logo, se utilizarán únicamente tres colores, rojo y negro, en analogía con los colores de la baraja de póker, y gris claro. Además para aquellos elementos que requieran especial atención por parte del usuario se empleará también el color verde. De este modo, contamos un número suficiente de colores para dotar a nuestra interfaz de la profundidad necesaria a la vez que proporcionamos una experiencia agradable y uniforme al usuario.

Fig.17 – Logotipo

Además se emplearán botones y tamaños de fuente relativamente grandes para mejorar en la medida de lo posible la accesibilidad de nuestra aplicación.

PANTALLA DEL SALÓN

Esta página se utiliza para mostrar al usuario los distintos torneos disponibles en la aplicación. En ella aparecerán tanto los torneos en curso como los torneos que todavía no han comenzado y se encuentran a la espera de más jugadores. El usuario podrá por tanto acceder a un torneo en curso como espectador, o registrarse como jugador en uno de los que todavía no ha comenzado.

Para que éste puede decidir qué torneo desea presenciar o en cuál desea participar, tendrá disponible la siguiente información de cada uno:

- Precio de la entrada
- Número de asientos
- Velocidad del torneo (duración de los turnos y frecuencia de los incrementos de ciegas)
- Jugadores registrados y fichas de éstos en caso de que el torneo esté en proceso.
- Posiciones premiadas y cantidad pagada a cada una

Puesto que todos los torneos van a tener la misma entrada el precio de ésta se mostrará únicamente una vez en la interfaz, en lugar de repetirlo para cada torneo. En cuanto al resto de la información, se organizó de la siguiente manera:

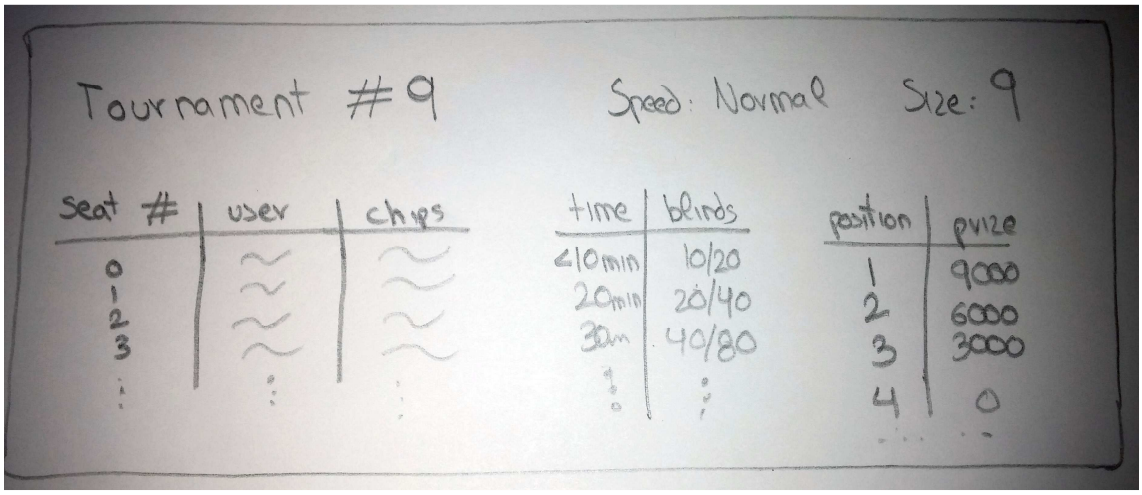


Fig.18 –Boceto de bloque de información de un torneo.

De modo que el contenido de la página del *Lobby* contendrá una representación como la anterior para cada torneo disponible. Debido a la gran cantidad de información que se muestra para cada torneo, se decidió que éstas ocuparían la totalidad de la anchura de la parte de la interfaz dedicada al contenido y que por tanto cada una de estas representaciones aparecería una encima de la otra según orden cronológico, de modo que aparezcan más arriba los torneos más antiguos.

La versión final de esta página tal y como se ve en el navegador del usuario se puede consultar en el anexo II, en el que aparecen además el resto de pantallas de la aplicación.

Esta página es la única que puede ser accedida por usuarios sin identificar en el sistema, por lo tanto, cuando los usuarios seleccionen un torneo, serán automáticamente dirigidos a la página de identificación en caso de que no se encuentren ya identificados en el sistema.

PANTALLA DE TORNEO

Esta pantalla se trata de la pantalla principal de la aplicación en la que se desarrollarán los torneos. Tendrá por tanto que mostrar al jugador toda la información necesaria para que este pueda seguir la partida y elegir cómo actuar, además de proveer los controles necesarios para que el jugador pueda llevar a cabo dicha actuación.

El póker es un juego muy dinámico que se encuentra cambiando constantemente, y en el que además hace falta estar al tanto en todo momento de los movimientos de los rivales, por lo tanto gran cantidad de información necesita estar disponible para el usuario en todo momento. Sin embargo, la manera en la que mostrar esta información al usuario ha sido ya explorada en multitud de ocasiones por los equipos de diseño profesionales que trabajan para las aplicaciones comerciales de póker. Por lo tanto tiene sentido basar nuestro diseño en estos modelos desarrollados por profesionales.

Concretamente vamos a basar nuestro diseño en la versión de la aplicación comercial *Pokerstars*, que se muestra a continuación.

Fig.19 –Diseño de la aplicación de póker *Pokerstars*.

A partir de este modelo se creó el prototipo de nuestra aplicación:

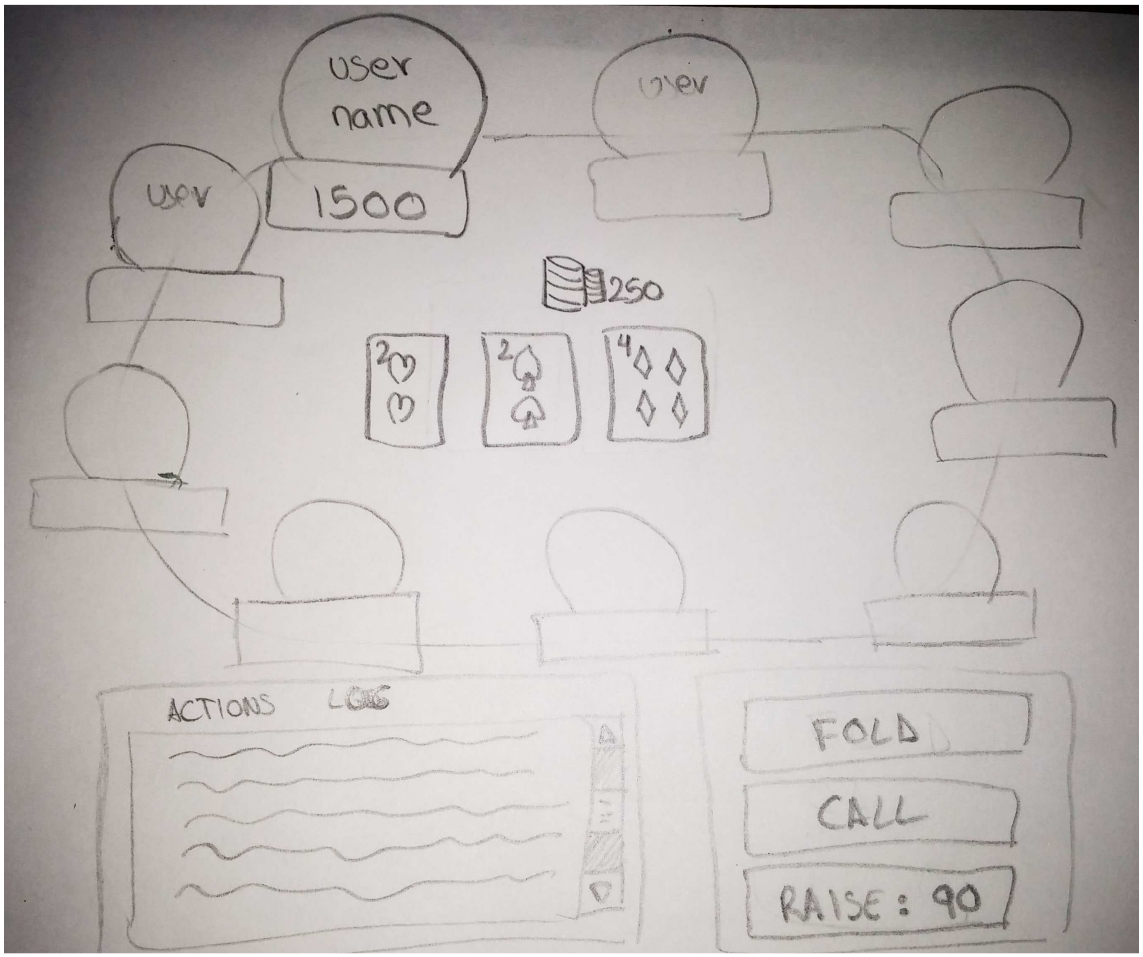


Fig. 20–Primer prototipo del diseño de la mesa.

Como se puede apreciar se ha mantenido la misma estructura, con una parte central en la que aparece la mesa con los diversos asientos repartidos a lo largo del perímetro de esta, y una parte inferior en la que aparece a la izquierda el log de los eventos ocurridos en la partida y a la derecha las diferentes acciones disponibles.

Sin embargo, debido entre otras cosas a que nuestro diseño debe aparecer correctamente integrado en una página web y a que como ya hemos comentado soportamos resoluciones de 1024 pixeles de anchura, no contamos con demasiado espacio en el interior de la mesa. Por lo tanto, si como en el diseño de *Pokerstars* quisiéramos mostrar las cartas comunes, las apuestas y el botón de *dealer* sobre la mesa, tendríamos que emplear representaciones bastante pequeñas de estos elementos, lo cual va en contra de una de las pautas que se siguen a lo largo de la aplicación que aboga por facilitar la accesibilidad en la medida de lo posible mediante una interfaz con elementos grandes y bien visibles.

Por este y otros motivos, como por ejemplo que nuestros usuarios no cuentan con avatares, se decidió aplicar una serie de modificaciones a la manera en la que aparecen representados los jugadores en la mesa.

Primero, puesto que nuestros jugadores no cuentan con avatares, se decidió mover el nombre de usuario al círculo en el que *Pokerstars* muestra los avatares, para poder de este modo eliminarlo del contador de fichas y poder utilizar por tanto un contado más grande y legible.

Por otro lado, el cambio más significativo consistió en cuando un jugador no ha renunciado a sus cartas, en lugar de la cantidad de fichas que posee, aparecerá en su contador la cantidad de fichas apostada, de este modo, evitamos tener que mostrar dicha cantidad en la mesa. Al hacer esto estamos ocultando la cantidad de fichas total que un jugador posee, sin embargo si durante una ronda el jugador desea

conocer este dato, bastará con que coloque el ratón sobre dicho usuario para ver la misma información que vería si el usuario hubiera renunciado a sus cartas.

En otras palabras, tenemos dos representaciones distintas de los jugadores dependiendo de su estado:

- **si esté ha renunciado a sus cartas,** aparecerá su nombre en la parte superior junto con la cantidad de fichas total en la parte inferior
- si por el contrario este **sigue involucrado en la mano actual**, se mostrarán sus cartas (cubiertas, a excepción de las propias del jugador) junto con la cantidad de fichas apostada en la ronda de apuestas en curso. En este caso se podrá ver la representación correspondiente al estado anterior al colocar el cursor sobre el jugador en cuestión.

En la figura de la derecha aparecen de izquierda a derecha un jugador que ha renunciado a sus cartas y dos que todavía están involucrados en la partida, en el caso del jugador de en medio se muestran las cartas descubiertas al tratarse del jugador correspondiente al usuario

Fig.21– Las tres maneras posibles de representar un jugador en la mesa.

Otra modificación similar que se realizó fue la de retirar el botón de repartidor de la mesa y pasar a indicar qué jugador realiza ese papel mediante una (D) (de *dealer*) en su contador de fichas.

Por otro lado, debido a que contamos con jugadores que forman parte de la IA cuyas decisiones son tomadas automáticamente, si hay varios de estos jugadores seguidos, el segundo de ellos actuará inmediatamente después del primero, probablemente sin dar tiempo a que la interfaz haya sido actualizada, de modo que cuando finalmente esta actualización se produzca, la actuación del primer *bot* puede no estar demasiado clara. Para evitar esto y mantener a los jugadores humanos informados en todo momento de lo que ocurre en la partida, cada una de las acciones tomadas por los jugadores así como los resultados de estas aparecerá reflejados en la parte inferior izquierda de la interfaz, la cual actuará como un log de los diversos eventos sucedidos en la mesa.

Así pues, gran cantidad de información será añadida a este log durante el transcurso de la partida. Por lo tanto, para facilitar su lectura y evitar que las entradas ocupen varias líneas sin tener que reducir excesivamente el tamaño de la fuente, se le dio la mayor anchura posible, confinando los controles de las acciones a una reducida porción de la parte inferior de la interfaz.

Fig.22– Campo de texto integrado en el botón de subir.

El hecho de que las acciones cuenten con poco espacio en la interfaz provocó que la utilización de una barra deslizadora para seleccionar la cantidad que se desea subir tal y como se utiliza en la interfaz de *Pokerstars* no fuera una buena idea, pues en nuestro caso contaríamos con una barra demasiado pequeña en la que el usuario no sería capaz de seleccionar con precisión la cantidad deseada. Por lo tanto, tal y como se muestra en la figura de la derecha se decidió incrustar un campo de texto en el botón correspondiente, el cual utilizará el usuario para introducir la cantidad deseada. Esta cantidad debe además de estar dentro de unos límites fijados por las reglas del juego, por lo que en caso de no estarlo, la interfaz informará de ello al usuario.

Al igual que con la página anterior, la versión final de esta aparece en el anexo II.

PANTALLA DE ESTADÍSTICAS

La página de estadísticas albergará una seria de coeficientes acompañados de una pequeña descripción que ayude al usuario a interpretarlos, la mayoría de ellos además irán acompañados de una gráfica que ayuda a expresar el valor del coeficiente.

Todas las estadísticas van a estar recogidas en una única página, sin embargo, puesto que se consideran buen número de estadísticas, la página resultante es bastante larga y por tanto puede dificultar al usuario encontrar la estadística que está buscando.

Así pues, para ayudar al usuario a navegar entre los distintos coeficientes, se creará un pequeño menú dentro de la sección de contenidos en el que habrá un elemento para cada estadística proporcionada. El usuario podrá entonces hacer clic en el elemento del menú correspondiente a la estadística que desea consultar para que la página se desplace automáticamente hasta la posición en la que aparece la información solicitada. Este menú se mantendrá fijo en la parte izquierda de la sección de contenido a lo largo de toda la página para permitir al usuario desplazarse rápidamente entre las distintas estadísticas en todo momento.

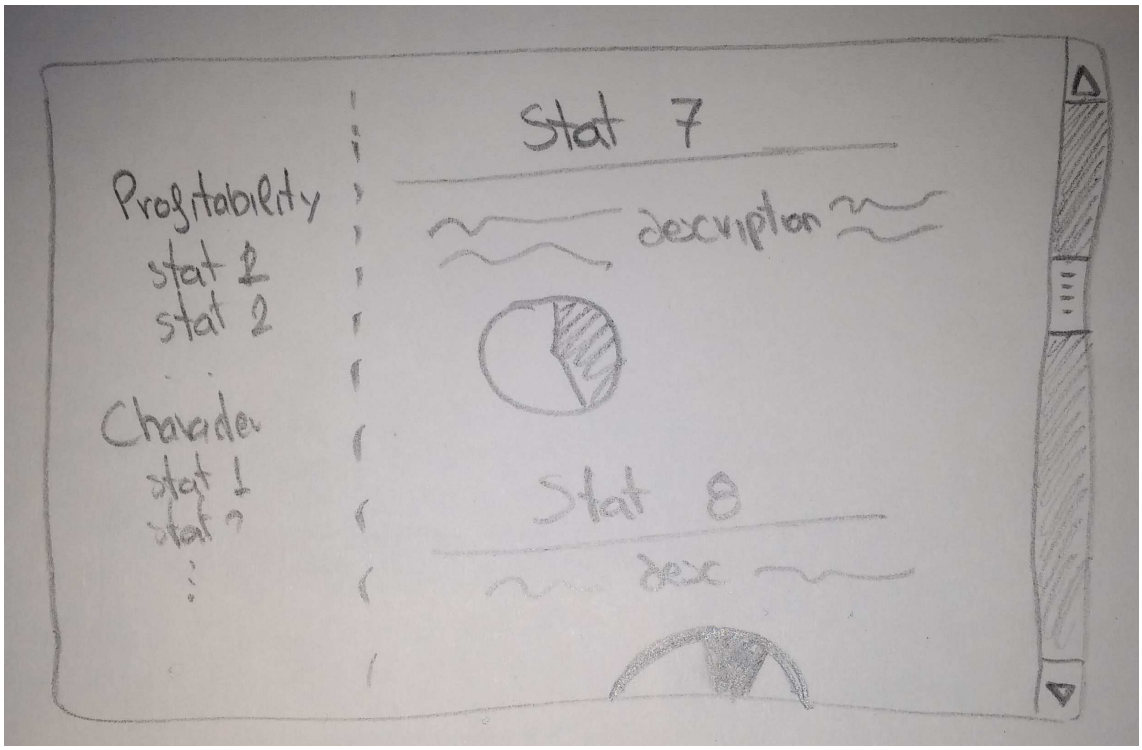


Fig.23– Boceto de la página de estadísticas.

Para hacer la lectura de las estadísticas más interactiva y expresiva, las gráficas serán animadas en el momento en el que están pasen a ser visibles por el usuario.

En el anexo II se puede consultar la versión final de la página con las estadísticas y el menú tal y como lo verá el usuario final.

FEEDBACK Y MODIFICACIONES

Una vez implementada la versión de la interfaz descrita en los capítulos anteriores, la utilización completa de la aplicación era posible ya que toda la información y controles necesarios para navegar por la misma estaban disponibles. Sin embargo, la interfaz que se ha generado es la mejor interfaz posible considerando los recursos disponibles según mi perspectiva, pero debido a mi visión distinta de la aplicación y a los conocimientos que me aporta haberla programado en su totalidad, es muy probable que el usuario final tenga una visión distinta acerca de la misma.

Se decidió por tanto someter la aplicación a la utilización por parte de diversos usuarios que no habían estado en contacto con la misma hasta el momento de realizar la prueba. Concretamente se les pidió que utilizaran la aplicación para jugar uno o varios torneos y consultaran sus estadísticas. Durante el proceso, los usuarios anotarán los posibles problemas que puedan encontrar o sugerencias que puedan tener para mejorar la interfaz.

De estas anotaciones, se eligieron las más importantes o aquellas que aparecían con mayor frecuencia y se realizaron ciertas modificaciones en el sistema para adecuar la aplicación a lo que los usuarios desean o esperan de ella.

A continuación se analizan algunas de estas anotaciones aportadas por los usuarios junto con las modificaciones aplicadas a la interfaz para solventarlas:

PESTAÑAS DEL MENU EN DOS LÍNEAS

Tal y como se explica en el primer apartado de esta sección en el que se describe la estructura del encabezamiento, en el menú aparecerán todas las páginas disponibles a las que un usuario puede acceder. Por lo tanto, cuando un usuario se une a un torneo, una nueva pestaña es añadida con un link a la página de dicho torneo. Sin embargo, cuando un jugador se ha unido a varios torneos, la cantidad de elementos del menú es superior a la que en un primer momento el menú fue diseñado para albergar, por lo que los elementos aparecen en dos líneas, rompiendo la analogía de pestañas con la que trabaja el menú.

Fig.24– Demasiados elementos en el menú rompen con nuestro diseño.

Para solucionar este problema, se decidió eliminar del menú principal las pestañas correspondientes a los torneos y agruparlas bajo una única pestaña en el menú principal, la cual mostrará en un segundo menú desplegable todos los torneos en los que el usuario está jugando al colocar el ratón sobre ella.

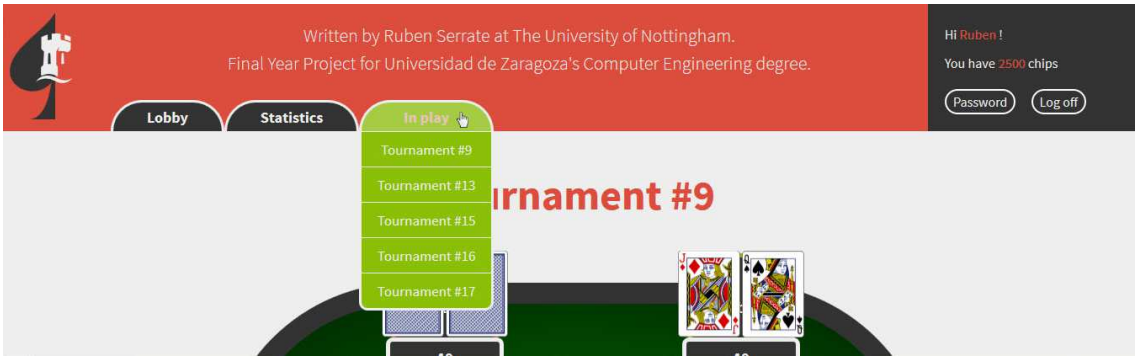


Fig.25– Una única pestaña contiene un submenú con los diferentes torneos.

PÁGINA DE LOBBY DEMASIADO LARGA

Debido a que la información que debemos mostrar de cada torneo en la página del *Lobby* es bastante extensa, cada torneo ocupa un espacio considerable en dicha página, por lo tanto, al colocar los torneos uno sobre otro, estamos creando una página demasiado larga en la que el usuario debe de desplazarse verticalmente una gran cantidad de píxeles para ver todos los torneos.

Para solucionar este problema, se decidió ocultar la mayoría de la información de los torneos y simplemente mostrar para cada uno de ellos un pequeño cabecero con las características principales, como son el número de jugadores, la velocidad y el enlace a la página del torneo. El resto de información aparecerá permanecerá oculta hasta que el usuario coloque el ratón sobre el cabecero del torneo correspondiente.

INFO DE TORNEO NO VISIBLE EN LA PANTALLA DEL TORNEO

Este caso es un perfecto ejemplo de algo que el usuario considera necesario y que sin embargo, debido a la dificultad por mi parte de utilizar la aplicación desde el punto de vista de un usuario tras haberla programado, me pasó desapercibida. Concretamente el problema consiste en que la información del torneo que aparece en el *Lobby* no se encuentra disponible al usuario en la propia página del torneo.

Parte de esta información, como el tamaño de la mesa, los jugadores y las fichas de cada uno ya aparecen representados en la interfaz, sin embargo, el resto de información como por ejemplo la velocidad del torneo y los premios pagados no lo hacen. Así pues, para ofrecer esta información al usuario sin ocupar demasiado espacio en la interfaz y poder mantener la mesa en la parte central de la pantalla, se optó por no mostrarla directamente y añadir un botón el cual el usuario podrá pulsar para generar un cuadro de diálogo superpuesto al resto de la interfaz que contendrá la información en cuestión.

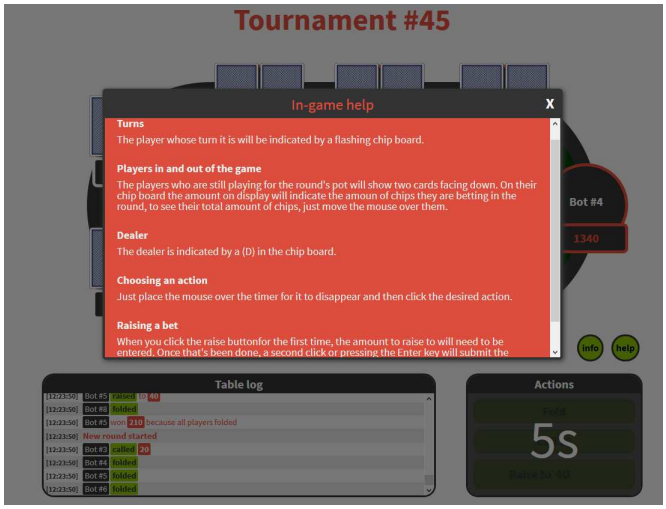


Fig.26– Cuadro superpuesto con ayuda al usuario.

INSTRUCCIONES Y DOBLE REPRESENTACIÓN DEL JUGADOR

Para un jugador que se enfrente a nuestra interfaz por primera vez, la distinta representación de los jugadores que se explica en el apartado anterior puede no resultar inmediata, por lo tanto, se añadirá un pequeño texto explicativo en un cuadro de diálogo similar al explicado en el punto anterior.

IMPLEMENTACIÓN

Ya se ha hecho hincapié en varias ocasiones a lo largo del proyecto en el hecho de que nuestra aplicación será ejecutada a través del navegador sin la necesidad de instalar ningún otro software adicional. Por lo tanto, la totalidad de la interfaz de la aplicación ha sido programada utilizando únicamente tecnologías web, concretamente HTML y CSS se encargan del aspecto visual y JavaScript del funcional.

El papel de JavaScript en nuestra interfaz consiste básicamente en escuchar una serie de eventos que pueden ser desencadenados por el usuario y procesarlos adecuadamente, además de, como se explica en otros apartados, mantener la interfaz sincronizada con el estado del servidor.
Para ello se crearon distintos objetos JavaScript que se sirven de la librería *jQuery* para llevar a cabo las funciones explicadas anteriormente, la implementación de las mismas no representa especial complejidad y por tanto no son analizadas en mayor profundidad.

En cuanto al aspecto visual, cabe destacar también la utilización de código SVG incrustado en HTML para la representación de las gráficas correspondientes a las estadísticas.

La recreación de la mesa sin embargo fue implementada utilizando puro HTML y CSS en sus versiones 5 y 3 respectivamente. La decisión de optar por esta implementación en lugar de utilizar el elemento *canvas* que proporciona HTML5 específicamente para la representación de elementos visuales es que éste es un mapa de bits, lo cual presenta numerosas desventajas.

La primera de ellas y más inmediata es que el mapa de bits debe de ser recreado completamente cada vez que una parte de la interfaz sea actualizada, mientras que con la solución de HTML y CSS podemos localizar el elemento en cuestión y modificarlo individualmente. Por otro lado, el generar eventos a partir de objetos presentes en el elemento *canvas* no cuenta con soporte nativo por parte del navegador y debe de desarrollarse código adicional para imitar dicho comportamiento.

Así pues, teniendo en cuenta la elevada frecuencia con la que vamos a actualizar nuestra interfaz y que vamos a hacer uso de distintos eventos, es por tanto mucho más eficiente el aplicar una seria de reglas CSS a elementos HTML que implementar una solución basada en *canvas* pese a que a primera vista pueda parecer esta es la solución idónea.

Como último detalle relativo a la implementación, mencionar que con el fin de facilitar la escritura, legibilidad y mantenibilidad del código CSS se utilizó el preprocesador *Sass* [7], el cual permite la utilización de una sintaxis más avanzada y compacta respecto al CSS puro. Este código "meta-css" se almacena en ficheros con extensión *scss*, los cuales serán compilados en CSS normal antes de ser subidos al servidor web.

REFLEXIÓN SOBRE HARDWARE

Tal y como se ha programado e integrado en el servidor web el núcleo de póker, la aplicación almacena toda la información relacionada con el estado de la misma en la memoria de la máquina que la ejecuta. Si bien almacenamos en la base de datos qué torneos hay en ejecución y qué jugadores se han registrado en los mismos, gran cantidad de información vital para el funcionamiento de nuestra aplicación como por ejemplo toda la relacionada con la situación de cada torneo (nivel de ciegas, turnos, apuestas, fichas de cada jugador) está representada únicamente en la memoria del servidor.

Por lo tanto, nuestra aplicación confía en que la máquina que la ejecuta sea un servidor dedicado que vaya a estar en marcha por lo menos todo el tiempo que dura un torneo, en otras palabras, el servidor no podrá ser desconectado o nuestra aplicación no podrá ser movida a otro servidor mientras haya torneos en juego. Si esto pasara, esta memoria evidentemente se perdería y sería imposible restaurar el estado de los torneos en juego.

Esto representa un problema si por ejemplo queremos ejecutar nuestra aplicación en la nube, donde las aplicaciones se ejecutan en instancias virtuales que son creadas y destruidas a menudo y donde es frecuente mover las aplicaciones entre distintas instancias. Como se explica en el párrafo anterior esto provocaría que todos los torneos en curso fueran irrecuperables al haberse perdido el estado de la memoria.

Por lo tanto, para hacer posible el despliegue de nuestra aplicación en la nube y poder garantizar el funcionamiento de la misma en todo momento habría que almacenar en la base de datos toda la información necesaria para que nuestra aplicación sea capaz de reconstruir el estado de la misma en caso de que ocurra una de las situaciones mencionadas anteriormente.

Sin embargo, en caso de que como comúnmente sucede en la nube quisiéramos extender nuestra aplicación y pasar a ejecutarla en más de una instancia, la solución anterior dejaría de ser viable ya que cada una de ellas actuaría como un servidor distinto y contarían por tanto con estados distintos. En este caso nos veríamos obligados a utilizar productos específicos para resolver este tipo de problemas como pueden ser caches compartidas entre las distintas instancias (que ofrecen la mayoría de proveedores de computación en la nube).

CONCLUSIONES

La realización del proyecto se extendió a lo largo de 4 meses durante los cuales he tenido que enfrentarme a gran cantidad de retos nuevos, tanto técnicos como de organización del trabajo o incluso de diseño. La mayoría de estos retos son derivados de ser la primera vez que me enfrentaba al desarrollo de una aplicación tan grande.

En una primera instancia la cantidad de trabajo era abrumadora y a veces parecía inabarcable, sin embargo, la insistencia de la universidad de Nottingham en la adopción de las prácticas ágiles y la consiguiente división del trabajo en pequeñas tareas agrupadas en *sprints* ayudó en gran medida a lidiar con el gran volumen de trabajo. Por otro lado, herramientas como *Trello* fueron también de gran ayuda a la hora de ayudar a mantener en todo momento una visión global y de ser consciente del punto concreto del desarrollo en el que nos encontramos.
Así pues, pese a las reticencias personales que en un primer momento despertaron en mí estos procedimientos, tras finalizar el proyecto puedo confirmar que han sido de gran ayuda y que sin duda van a formar parte de mis herramientas en mi futuro profesional.

Técnicamente, el principal reto fue adaptarme a trabajar con la plataforma de Microsoft ASP.NET y C#, ya que mi experiencia en cuanto a programación web en el lado del servidor se limitaba a PHP. Sin embargo, la elección de esta nueva plataforma ha resultado ser una gran idea puesto que me ha proporcionado abundantes conocimientos acerca de un lenguaje y de una plataforma hasta entonces desconocidas y que son muy demandadas profesionalmente. Además, el desarrollo de la aplicación ha contribuido a aumentar mi familiaridad con lenguajes que ya conocía como JavaScript, HTML y CSS.
En un nivel un poco más abstracto, me resultó particularmente grato el poder aplicar ciertos patrones y modelos que había estudiado a situaciones reales.

En cuanto a la usabilidad y la interacción con el usuario, resultaron ser de gran utilidad las opiniones que estos ofrecieron acerca de la primera versión de la interfaz, pues en numerosas ocasiones apuntaron problemas que yo había pasado por alto debido a mi conocimiento intrínseco de la aplicación. Por otro lado, en ocasiones lo que para mí era una representación ideal de la información, para ciertos usuarios resultaba confuso o no idóneo, dejando en evidencia la gran dificultad de producir una interfaz que resulte clara e intuitiva para todos los usuarios, lo cual me hizo pasar a valorar la necesidad de un diseño profesional que anteriormente cuestionaba.

A lo largo de la memoria se hace hincapié en las distintas partes diferenciadas que forman parte del proyecto. Si bien estas partes no han sido optimizadas y/o depuradas hasta la excelencia, cada una de ellas realiza su función de manera más que suficiente, y la integración de todas ellas resulta en una aplicación compleja que funciona de acuerdo a todos los requisitos que se le exigieron en un primer momento. Este hecho casa a la perfección tanto con las expectativas que el director puso en el proyecto como a las mías personales.

Personalmente, el objetivo del proyecto no fue el de obtener una manera súper eficiente de comparar manos de póker o el de proveer un mecanismo de sincronización web bidireccional óptimo, si no el de obtener una visión global del funcionamiento de las aplicaciones de póker, el cual es una de mis grandes aficiones, aprendiendo cuales son los distintos retos y problemas que deben de resolverse. Por otro lado, debido a que profesionalmente estoy muy orientado al mundo de las páginas web, la elaboración de una interfaz web además de resultarme muy útil e interesante, en un determinado momento puede ser utilizada como muestra de mis aptitudes de cara a un posible empleador.

En cuanto a Mr. Graham Hutton, director del proyecto, él deseaba contar con una aplicación de póker que poder utilizar como esqueleto a la que aplicar diversas mejoras y optimizaciones como se ha explicado en varias ocasiones a lo largo de la memoria.

Puesto que el proyecto consta de varias partes diferenciadas comprendiendo una gran cantidad de tareas distintas, existen numerosas posibilidades de mejora o ampliación sobre el conjunto de éste.

En cuanto al núcleo de póker, la principal ampliación que se contempla es la de implementar una manera más eficiente de relacionar conjuntos de cartas con las distintas manos, de hecho, como ya se ha explicado en el apartado "Elección de la mano ganadora" de la primera sección de la memoria, se ofrecerá explícitamente realizar trabajo sobre esta parte de la aplicación.

Parte también del núcleo de póker es la inteligencia artificial, de la cual, como ya se ha explicado, se ha programado una versión mínimamente competitiva, por quedar fuera del alcance de este proyecto una versión mejor. Por lo tanto, el crear unos jugadores artificiales más inteligentes representa también un reto muy atractivo, a cerca del cual es muy posible que se realice trabajo en forma de futuros proyectos.

Relativa a la integración en el servidor web de nuestra aplicación, la principal mejora que se puede aplicar es la de realizar el proceso de sincronización de la interfaz con el servidor a través de un mecanismo de comunicación bidireccional, en lugar de la implementación por encuesta que se explica en el apartado dedicado a "asincronía". La maduración y adaptación por parte de los navegadores de tecnologías como *WebSockets* [8] abrirá un nuevo abanico de posibilidades en lo relativo a este apartado.

Por último, en cuanto a la interfaz gráfica, dejando de lado las pequeñas correcciones estéticas que se puedan aplicar a la misma, el principal proyecto de continuación que se puede plantear es el de aplicar una serie de *media queries* a nuestro CSS para crear una interfaz responsiva que se comporte de manera distinta según el tamaño de la pantalla del dispositivo cliente, en especial, adaptar la interfaz a tamaños de pantalla pequeños. De este modo abriríamos nuestra aplicación a un mercado cada vez mayor como es el de usuarios de dispositivos móviles y *tablets*.

ANEXO I
ORGANIZACÓN DEL TRABAJO
PROCEDIMIENTO

Al igual que el resto de facetas del proyecto, la planificación, seguimiento y organización del trabajo ha sido basada en metodologías ágiles. Concretamente la metodología seguida es una adaptación de SCRUM a mi caso particular, en el que el equipo de desarrollo está compuesto por una única persona. Además se han empleado otras técnicas como *Kanban Board* y *Burndown Charts*.

Una parte fundamental de la gestión del trabajo a lo largo de todo el proyecto será *Trello*, una implementación digital y "en la nube" de un *Kanban Board*, el cual en pocas palabras permite la creación de tareas y su distribución en distintas columnas. Para cada una de las iteraciones se creará un tablón o pizarra digital en el que se dispondrán y actualizarán las distintas tareas.

En concreto, el procedimiento a seguir para cada iteración es el siguiente:

1. **División del trabajo en tareas con una granularidad adecuada.**

   El primer paso al comenzar cada una de las fases o iteraciones consiste en analizar el prototipo final que se desea conseguir al final de la misma, con el fin de dividir el trabajo necesario para alcanzarlo en distintas tareas.

   La primera división suele ser de un nivel demasiado alto, dando como resultado tareas demasiado extensas y con poco nivel de detalle, por lo que a menudo es necesario subdividir estas tareas hasta tres veces más, hasta que se da con tareas de un tamaño manejable y con un nivel de detalle suficiente como para poder referirse a ellas y estimarlas.

   La subdivisión de tareas se realizará con lápiz y papel debido a la alta variabilidad de las tareas hasta que se da con el conjunto final. Una vez éste ha sido obtenido, todas las tareas que forman parte del mismo son añadidas a la columna "Iteración" en *Trello*.

2. **Estimación de las tareas.**

   Cada una de las tareas obtenidas en el apartado anterior es estimada.

   La estimación consiste en la asignación de un número según la tarea comprenda más o menos trabajo. Una vez el número ha sido elegido, éste es añadido a la tarea correspondiente en *Trello*.

   Para hacer más fácil la elección de este número, se restringe el dominio de estimaciones disponibles a un conjunto de números similar a la secuencia de Fibonacci.

   Las unidades empleadas en la estimación no se corresponden con ninguna magnitud física real, si no que se utiliza una medida abstracta de la cantidad de trabajo que representan. De este modo se aporta más flexibilidad a la estimación sin perder expresividad.
   Sin embargo, en ocasiones resulta útil establecer una relación entre los puntos asignados a cada tarea y el tiempo necesario para completarla, en mi caso un valor aproximado de esta relación es el de 3 o 4 puntos de dificultad por día de trabajo.

3. Agrupación de las tareas en "*Sprints*" o "flujos de trabajo".

En mi particular implementación de SCRUM, un sprint o flujo de trabajo es simplemente una manera de agrupar tareas para contar con bloque lógicos de trabajo. En un sprint se agrupan tareas que están relacionadas entre sí, normalmente mediante la funcionalidad a la que contribuyen, aunque también se incluirán algunas tareas cuya ejecución sea más conveniente en el sprint en cuestión por otro motivo (similitud con otra tarea, utilizan el mismo test…).

Además de las ventajas obvias que residen en agrupar tareas relacionadas entre sí, los *sprints* también facilitan el seguimiento y la planificación del trabajo al contar con un número de tareas manejable e ideal para la aplicación de las diversas técnicas de organización y seguimiento utilizadas.

Por otro lado, permiten subdividir el trabajo en bloques de un tamaño variable que cada uno puede ajustar según sus circunstancias. En este caso habrá *sprints* de una y de dos semanas. Esta elección debe ser tenida en cuenta a la hora de elegir qué tareas pertenecen al sprint, puesto que el sumatorio de puntos de estimación del conjunto de tareas en el sprint debe de estar dentro de un límite razonable que permita ajustar el sprint a la longitud deseada.

Una vez se han creado los distintos *sprints*, se selecciona aquel en el que se desea trabajar y se mueven en *Trello* las tareas que lo integran desde la columna "Iteracion" a la columna "Sprint", en la que permanecerán hasta que sean seleccionadas para comenzar a trabajar sobre ellas. El orden de las tareas en la columna sprint debe de corresponder con el orden en el que se espera trabajar sobre cada una de las tareas, de modo que la tarea sobre la que se espere trabajar primero deberá de aparecer en la primera posición de la columna.

4. Trabajo y seguimiento de cada una de las tareas.

Este paso consiste simplemente en seleccionar la tarea en la que se considera que es más conveniente trabajar, llevar a cabo las operaciones precisas sobre la tarea y al finalizar la sesión de trabajo, actualizar la tarea en *Trello* consecuentemente, de modo que contemos en todo momento con una representación fiel del estado actual del proyecto.

Las columnas a las que una tarjeta puede pertenecer son las siguientes:

- **Iteración** – Tareas pertenecientes a la actual iteración que no se encuentran incluidas en el actual sprint
- **Sprint** – Tareas pertenecientes al sprint actual sobre las que no se ha comenzado a trabajar todavía.
- **En proceso** – Actualmente se está realizando trabajo en la tarea o se encuentra bloqueada en espera de otra tarea.
- **En test** – Se está comprobando que la tarea funciona como se espera o se está esperando a realizar la comprobación.
- **Completa** – La tarea está finalizada correctamente.
- **No implementada** – Por algún motivo se ha decidido que la tarea no es necesaria y por tanto no ha sido implementada.

Cada una de las tareas debe de pasar por las distintas columnas hasta que una vez ha sido finalizada y testeada adecuadamente, se coloca en la columna "Completa". En ese momento, otra tarea puede ser extraída de la columna "Sprint".
La prioridad por tanto en todo momento es coger la tarjeta que se encuentra más a la derecha y trabajar en ella para poder llevarla hasta la columna "Completa", aunque en ocasiones se producen dependencias que provocan que más de una tarea estén siendo trabajadas al mismo tiempo.

El momento en el que todas las tarjetas que estaban en la columna "Sprint" se encuentran en la columna "Completa", (o "No implementada" si por algún motivo dicha tarea ha pasado a no

ser relevante) marca el fin del "sprint" actual, y debería de estar cercano a la fecha de finalización de sprint deseada. Al finalizar un sprint, se vuelve a seleccionar un sprint como al final del paso 3 y se repite el paso 4 hasta que todas las tareas en la columna "iteración" han sido agrupadas en "*sprints*" y posteriormente completadas.

Este conjunto de pasos, que representan mi particular implementación de SCRUM fueron muy útiles en cuanto a la división y organización del trabajo, y además permiten obtener fácilmente información acerca del estado del proyecto para ser utilizada en otras técnicas.

Por ejemplo, si hay más de una tarea en "En proceso" significa que la primera de ellas se encuentra bloqueada por la segunda o existe algún tipo de dependencia entre ambas. Es también muy fácil apreciar dependencias entre tareas que dependen de otras para ser testeadas.

Otra información muy útil que nos aporta este proceso es que en todo momento sabemos la cantidad de trabajo que nos falta por realizar (suma de las estimaciones de las tareas en la columna "Sprint"), lo cual puede ser comparado con la cantidad de trabajo al comenzar el sprint, o la cantidad de trabajo que comprenden las tarjetas en la columna "completa" para obtener gráficas de velocidad y de seguimiento muy útiles y expresivas, como por ejemplo *Burndown Charts*. Esta información puede ser también utilizada para establecer predicciones bastante precisas de la fecha de finalización del sprint, de la iteración o de todo el proyecto, dependiendo qué tareas consideremos a la hora de sumar sus estimaciones.

DIVISIÓN DEL TRABAJO

A continuación se presentan, según el orden cronológico en el que fueron abordados, los distintos *sprints* en que se dividió el trabajo desarrollado a lo largo del proyecto junto con la iteración a la que pertenecen.

Primera iteración.
Librería para desarrollar una partida de póker entre jugadores humanos.

1. Asignación de rangos y valores de manos a conjuntos de 7 cartas, selección del mejor conjunto.
2. Sistema de apuestas, turnos y acciones de los jugadores.
3. Repartición de botes al jugador o jugadores ganadores y eliminación de jugadores sin fichas.

Segunda iteración.
Finalización del núcleo de póker.

4. Temporización y jugadores de la IA.
5. Registro de acciones en la base de datos, cálculo de estadísticas y varias partidas al mismo tiempo.

Tercera iteración.
Integración del núcleo de póker en el servidor web.

6. Adaptación de las clases que forman parte de este núcleo a modelos del sistema MVC de ASP.NET y creación de los puntos de entrada apropiados en los controladores de modo que la aplicación pueda ser accedida a través de *URLs*.
7. Relación entre usuarios y jugadores, sistema de *login*.
8. Creación de vistas HTML, por el momento simplemente volcar información.

Cuarta iteración.
Interfaz web basada en texto.

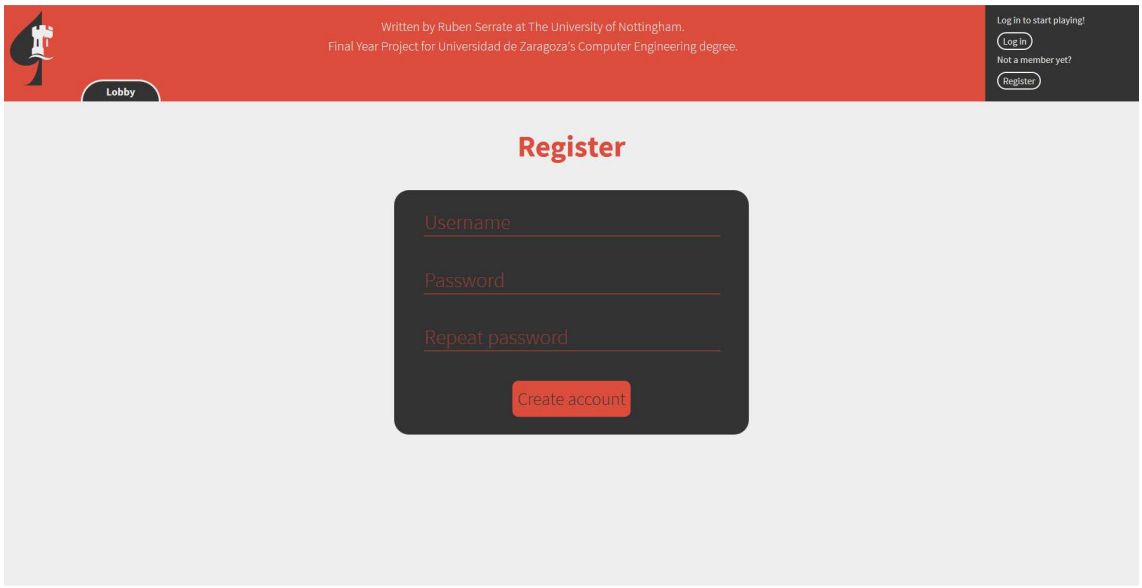
9. Actualización automática de las vistas mediante peticiones AJAX.
10. Adición a las vistas de los controles y la interactividad necesaria para desarrollar una partida completa.
11. Concurrencia.

Quinta iteración.
Diseño e implementación de la interfaz gráfica.

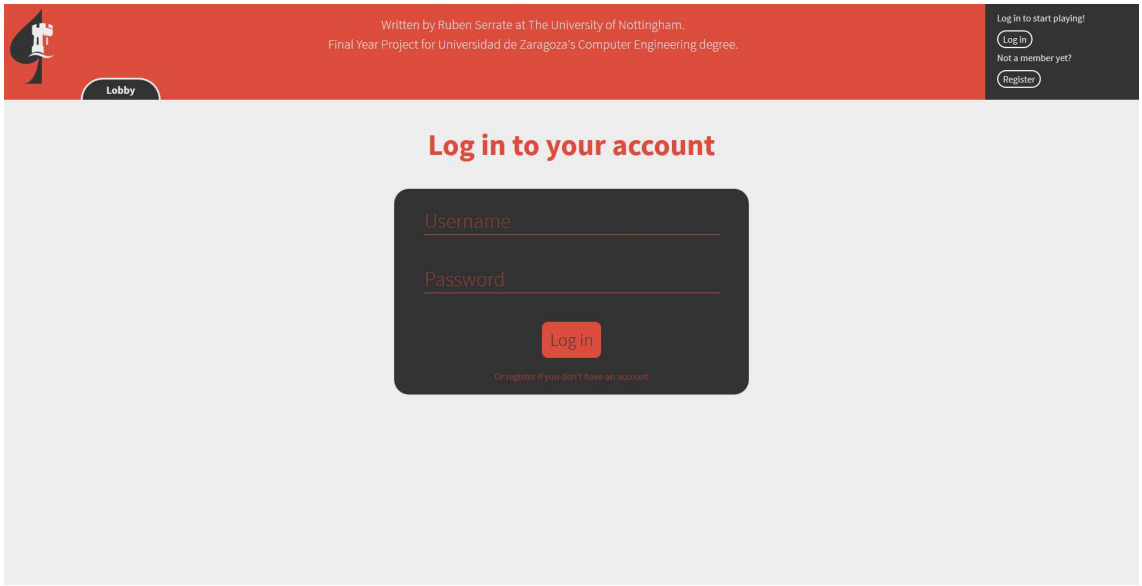
12. Diseño de la interfaz y codificación en CSS y HTML.
13. Rediseño y retoque de la interfaz en base al *feedback* obtenido en las pruebas realizadas a los usuarios.

ANEXO II
PANTALLAS DE LA INTERFAZ

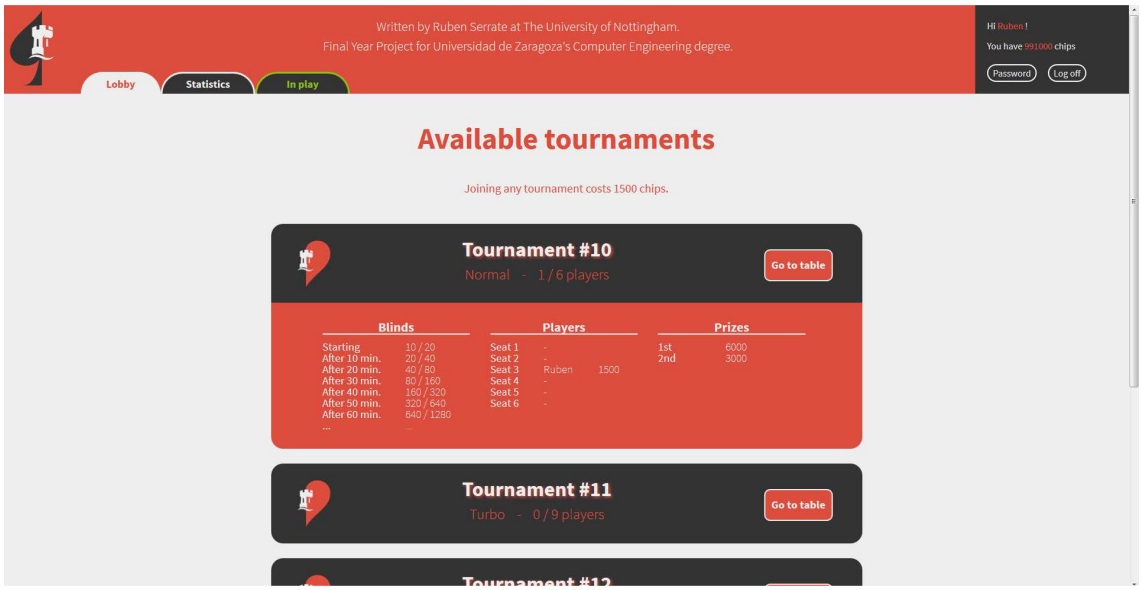
PANTALLA DE REGISTRO



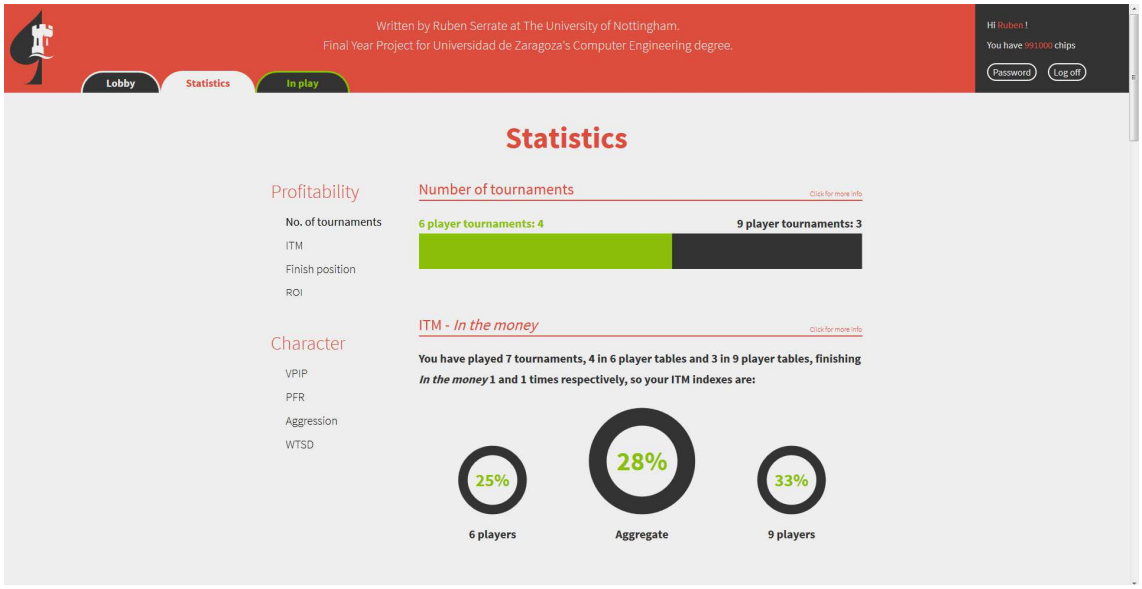
PANTALLA DE IDENTIFICACIÓN



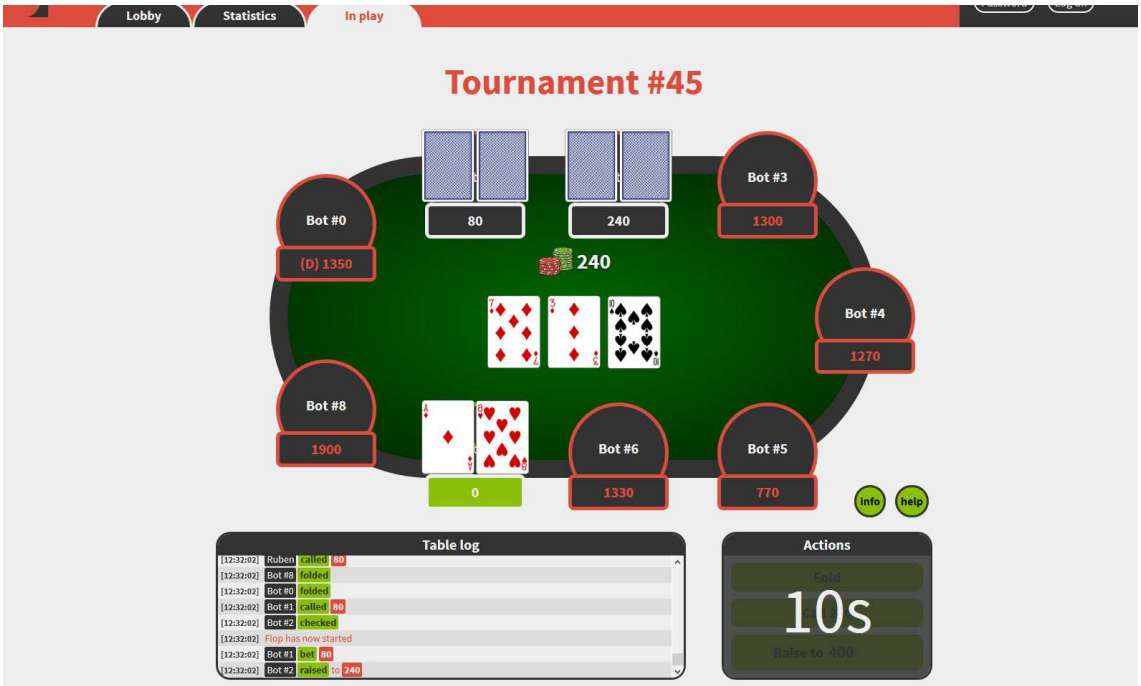
PANTALLA DE LOBBY



PANTALLA DE ESTADÍSTICAS



PANTALLA DE TORNEO



ANEXO III
MECÁNICA DEL JUEGO DE POKER

La información en este apartado se basa en las referencias bibliográficas 9, 10 y 11.

Cada mano de Hold'em comienza con dos ciegas. Las ciegas son apuestas preliminares hechas por dos jugadores antes de que se repartan las cartas, con el objetivo de estimular la acción. La posición del repartidor se indica con una D, y se denomina el botón del repartidor, o simplemente, el botón. Esta es la posición desde la que el repartidor distribuiría las cartas si fuera uno de los jugadores. Antes de repartir las cartas, el jugador a la izquierda del botón pondrá una cantidad de fichas igual a la mitad de la apuesta mínima para la partida (conocida como la ciega pequeña). El jugador a la izquierda de ese jugador pondrá una cantidad de fichas igual a la apuesta mínima para la partida (conocida como la ciega grande).

Cuando las ciegas están en su sitio, el repartidor da primero una carta y luego otra boca abajo a cada jugador, comenzando con la ciega pequeña. Estas dos cartas de inicio se llaman cartas de mano, y son solo visibles para el jugador al que se le han repartido.

Una vez todos los jugadores han sido repartidos sus cartas de mano, comienza la primera ronda de apuestas. Las apuestas de la primera ronda siempre comienzan por el jugador que se encuentra justo a la izquierda de la ciega grande.

El primer jugador, tiene tres posibilidades. Puede:

- no ir, retirarse
- igualar la ciega grande
- subir la apuesta

Si no va, sus cartas se retirarán del juego y estará fuera hasta la siguiente mano y hasta entonces no se jugará nada en el bote. Ahora será el turno del siguiente jugador, que tendrá las mismas opciones. Si todo el mundo se retira, incluida la ciega pequeña, el bote será para la ciega grande, y se repartirá la siguiente mano.

Cuando un jugador iguala la ciega o sube la apuesta, cada jugador después de él tendrá tres opciones:

- no ir
- igualar la apuesta anterior
- subir la apuesta

Cada jugador tiene las mismas opciones en su turno. Si ha habido una subida, el jugador que elija continuar deberá igualar la apuesta total que haya hasta ese momento o subir. Cuando la apuesta (denominada también acción) llegue a las ciegas, éstas tendrán las mismas opciones. No obstante las ciegas ya tienen fichas en el bote, y estas fichas cuentan para su apuesta. Si no ha habido subidas cuando las apuestas lleguen a la ciega grande, ese jugador tiene lo que se llama la opción. Podrá optar por subir; en cuyo caso cada jugador activo tendrá la opción de igualar la apuesta, subir o no ir. La ciega grande puede elegir también no subir, lo que terminaría las apuestas de la primera ronda.

Una vez que las apuestas de la ronda se hayan igualado, es decir, que todos hayan tenido la oportunidad de retirarse o de igualar la apuesta total, el repartidor colocará tres cartas boca arriba en el centro de la mesa. Estas tres primeras cartas comunitarias se denominan el *flop*.

Ahora tendrá lugar la segunda ronda de apuestas. En esta ronda, las apuestas comienzan por el primer jugador activo (uno que todavía tenga cartas) a la izquierda del botón. En todas las rondas posteriores a la primera, el primer jugador tendrá dos opciones:

- pasar, es decir, no apostar
- apostar

Si nadie apuesta, todos los jugadores tienen las mismas opciones cuando les llegue el turno. En todas las rondas, excepto la primera, es posible que no haya apuestas.

Si alguien apuesta, cada jugador después de él tendrá tres opciones:

- no ir
- igualar la apuesta anterior
- subir, es decir, elevar la apuesta anterior

El jugador que pasa conserva sus cartas. Si alguien apuesta, cuando le vuelva tocar actuar, el jugador que ha pasado tiene las tres opciones anteriores.

Una vez que las apuestas de la segunda ronda se hayan igualado, es decir, que todos hayan tenido la oportunidad de pasar o de igualar la apuesta total de esta ronda, el repartidor colocará otra carta boca arriba en el centro de la mesa. A esta cuarta carta comunitaria se la denomina el *turn*.

Ahora tendrá lugar la tercera ronda de apuestas. De nuevo, las apuestas comienzan por el primer jugador activo a la izquierda del botón. Las apuestas se desarrollan del mismo modo que en la segunda ronda. Una vez que las apuestas de la tercera ronda se hayan igualado, el repartidor colocará una quinta y última carta boca arriba en el centro de la mesa. A esta última carta comunitaria se la denomina el *river*. Ahora tendrá lugar la cuarta y la última ronda de apuestas. De nuevo, las apuestas comienzan por el primer jugador activo a la izquierda del botón. Las apuestas se desarrollan de igual modo que en las dos rondas anteriores.

Una vez igualadas las apuestas de la cuarta ronda, se terminan las apuestas, y se muestran las cartas. Los jugadores que queden activos muestran sus cartas y la mejor mano, que consiste en las mejores cinco cartas de entre la combinación que tenga cada jugador de dos cartas de mano más las cartas comunitarias, gana. Al jugador que posea la mano ganadora se le otorga el bote. Si hay un empate para la mejor mano, el bote se dividirá equitativamente entre los jugadores empatados. Al determinar la mano ganadora, la combinación de las cinco mejores cartas a veces incluye las dos cartas de mano del jugador. A veces solo incluye una de las cartas de mano del jugador. A veces, en raras ocasiones, no se usa ninguna carta de mano. En ese caso, la mesa incluiría alguna combinación mejor que cualquiera que pudiera hacerse con las cartas de mano del jugador.

Si no se igualan las apuestas en la ronda final, es decir, un jugador iguala o sube y nadie ve la apuesta, no habrá confrontación final, y el bote será para el jugador que haya hecho la apuesta no igualada. En cualquier ronda previa se seguirá el mismo proceso. Si esto ocurre en rondas anteriores, no se repartirán más cartas, ya que la mano habrá terminado.

A veces, un jugador se queda sin fichas antes de que acaben todas las apuestas. En ese caso, se creará uno o más botes secundarios, que serán concedidos equitativamente.

Si al terminar la ronda final de apuestas y repartir los botes un jugador no tiene fichas, éste será eliminado del torneo.

En la siguiente página se pueden apreciar las distintas manos que los jugadores pueden formar con sus cartas, ordenadas de mayor a menor.

	Jugada	*Jugada en inglés*	Descripción	Ejemplo
1	**Escalera real** o **Flor Imperial**	*Royal flush*	Cinco cartas seguidas del mismo palo del 10 al As.	
2	**Escalera de color** o **flor corrida**	*Straight flush*	Cinco cartas seguidas del mismo palo, pero que no tiene As como carta alta.	
3	**Póquer**	*Four of a kind*	Cuatro cartas iguales en su valor.	
4	**Full**	*Full house*	Tres cartas iguales (trío), más otras dos iguales (par).	
5	**Color**	*Flush*	Cinco cartas del mismo palo sin ser seguidas.	
6	**Escalera**	*Straight*	Cinco cartas seguidas de palos diferentes.	
7	**Trío, Tercio, Pierna o Trucha**	*Three of a kind*	Tres cartas iguales en su valor.	
8	**Doble par**	*Two pair*	Dos pares de cartas.	
9	**Par**	*Pair*	Dos cartas iguales y tres diferentes.	
10	**Carta alta o "Carta Mayor"**	*High card*	Gana quien tiene la carta más alta.	

BIBLIOGRAFÍA

1. "Scrum: The Art of Doing Twice the Work in Half the Time", Jeff Sutherland, Crown Publishing Group, 2014
   http://es.wikipedia.org/wiki/Scrum

2. “Test Driven Development: By Example”, Kent Beck, Pearson Education, 2003
   http://www.jamesshore.com/Agile-Book/test_driven_development.html
   http://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas

3. http://www.theie8countdown.com/

4. http://www.suffecool.net/poker/evaluator.html

5. https://www.pokertracker.com/guides/PT3/general/statistical-reference-guide

6. http://www.w3schools.com/browsers/browsers_Display.asp

7. “Instant SASS CSS How-to”, Alex Libby, PACKT Publishing, 2013
   http://sass-lang.com/

8. https://developer.mozilla.org/en/docs/WebSockets

9. http://www.fulltilt.com/es/poker/games/texas-holdem

10. http://www.fulltilt.com/es/poker/games/texas-holdem/comp

11. http://www.ipoquer.com/wp-content/uploads/2009/10/jerarquia.jpg