EasyCard.js Framework

Ezequiel Ángel Gómez[1], Alejandro Sartorio[2], Marcelo Vaquero[3], Daniel Tedini[4]

[1] Estudiante, Ingeniería en Sistemas Informáticos, Universidad Abierta Interamericana Av. Ovidio Lagos 944, Rosario, Santa Fe, Argentina, Tel. 0341 435-6510 eze.angel.gomez@gmail.com

[2] Profesor, Ingeniería en Sistemas Informáticos, Universidad Abierta Interamericana Av. Ovidio Lagos 944, Rosario, Santa Fe, Argentina, Tel. 0341 435-6510 alejandro.sartorio@gmail.com

[3] Director, Ingeniería en Sistemas Informáticos, Universidad Abierta Interamericana Av. Ovidio Lagos 944, Rosario, Santa Fe, Argentina, Tel. 0341 435-6510 marcelo.vaquero@hotmail.com

[4] Director, Ingeniería en Sistemas Informáticos, Universidad Abierta Interamericana Av. Ovidio Lagos 944, Rosario, Santa Fe, Argentina, Tel. 0341 435-6510 Daniel.Tedini@uai.edu.ar

***Resumen*—** Para este trabajo se implementó un juego de cartas con tecnología JavaScript y HTML5. EasyCard.js brinda un diseño modular e implementación conceptual que permite adaptaciones y configuraciones para cualquier tipo de juegos de apuestas de estas características.

***Palabras Clave*—JavaScript, HTML5, framework,videojuegos, apuestas, online.**

I. INTRODUCCIÓN

Los juegos de apuestas online se están instalando cada vez más y en diferentes propuestas y formatos [1]: casinos, apuestas deportivas, y otras modalidades. La posibilidad de jugar en cualquier momento y lugar puede ser uno de los principales factores de atracción. Los avances tecnológicos y conectividad permiten instrumentar servicios de juegos en diferentes plataformas y estilos de participación. Por ejemplo: partidas de juegos individuales, en clubes, en las redes, etc. También se tienen en cuentan cuestiones de accesibilidad, seguridad y privacidad [2].

La industria de software de desarrollo de juegos online está en continuo crecimiento y estudio [1]. Del estado del arte de este trabajo se puede evidenciar fundamentos y lineamientos de los principales objetivos y propuestas de la comunidad de usuarios, agentes y desarrolladores.

Los cambios a lo largo del desarrollo de cualquier tipo de videojuego son inevitables. Los requerimientos pueden cambiar constantemente, donde se requiere cambios en el diseño original del proyecto. Teniendo en cuenta las características de los requerimientos funcionales, tecnológicos, estándares y legales, tiene sentido estudiar la posibilidad de

un diseño de software general utilizando buenas prácticas de Ingeniería de Software. En este sentido, esta propuesta se enmarca dentro del proyecto de I+D+i denominado: "*Hacia una infraestructura técnica, funcional y de negocio para aplicaciones de apuestas online utilizando técnicas de Ingeniería de Software y Gestión de Negocio Inteligente.*" perteneciente al Centro de Altos Estudios en Tecnología Informática (CAETI) de la Universidad Abierta Interamericana (UAI).

Se establecieron criterios de aplicación de patrones de diseños para maximizar la reutilización de las partes de nuestros proyectos. De esta manera, se pueda modificar de una forma más sencilla, y nos posibilita un mejor mantenimiento del código a lo largo de la vida del proyecto.

Ante la necesidad de establecer una base sólida sobre la cual desarrollar aplicaciones concretas surgieron los llamados "Frameworks" [3], con el propósito de normalizar y estructurar el código del sistema, facilitando un esquema (un patrón, un esqueleto) para el desarrollo y/o la implementación de aplicaciones. El uso de frameworks para este tipo de desarrollo mejora la calidad, el tiempo de elaboración e implementación y ayuda a hacer un trabajo mantenible y escalable, según las características del mismo [4]. Un framework agrega funcionalidad extendida a un lenguaje de programación, automatiza muchos de los patrones de programación para orientarlos a un determinado propósito, proporcionando una estructura al código, mejorándolo y haciéndolo más entendible y sostenible, donde también permite separar en capas la aplicación.

En este trabajo se presenta EasyCard.js, un framework hecho con JavaScript y Node.js, que propone un diseño modular genérico para ser implementado dentro del contexto de juegos de apuestas online. Su diseño, está compuesto por módulos relacionados entre sí que pueden ser reutilizados y extendidos con facilidad, con el fin de reducir tiempo y costo en diseñar una nueva estructura para juegos. Además, utiliza patrones como *Singleton*, que se enfoca en asegurar la existencia de instancias únicas de una clase en memoria, y se tenga acceso global al objeto [5]. Este patrón se implementa para crear un objeto global que sirva de acceso a las propiedades y funciones de la estructura del framework. Otro patrón que es utilizado es *Strategy*, que permite a un algoritmo variar independientemente de quién lo utilice, y pueda ser intercambiado en tiempo de ejecución [5]. Este patrón es utilizado para resolver el problema de determinar diferentes comportamientos en tiempo real, basándose en las reglas del juego y las acciones de los jugadores.

A lo largo de este trabajo se incorporan definiciones e interpretaciones de este framework. En la sección 2 se describen trabajos previos utilizados como punto de partida y se introducen los primeros lineamientos conceptuales y de diseño que se tuvieron en cuenta. La sección 3 está enfocada en explicar cómo se resuelven los requerimientos y propiedades enunciados en esta introducción, partiendo de la propuesta de un diseño conceptual.

II. ESTADO DEL ARTE

A continuación, se describen partes de los trabajos utilizados como punto de partida e inspiración para el desarrollo de este trabajo.

En el trabajo "Rebel War" [6] se propone un modelo para la presentación y desarrollo de un juego multijugador de cartas. Toma varios capítulos donde cuenta la evolución y crecimiento de los juegos online, desarrolla el modelo teórico de juego, el proceso de adaptación y la parametrización de valores y finaliza con un ejemplo de implementación del modelo.

El trabajo "MMO de Navegador en Tiempo Real con Node.js y WebSockets" [7] plantea crear un juego de estrategia masivo en línea que se pueda jugar a través del navegador web y mejorar la interacción entre el cliente y el servidor mediante WebSocket, proporcionando una comunicación bidireccional, full-dúplex y en tiempo real. Esta tecnología está incluida en una librería de Node.js llamada "Socket.io", que es requerida por EasyCard.js, para poder crear juegos online. Es decir, si un jugador entra en una sesión de juego en donde participan otros jugadores, cualquier acción que realice se convertirá en información que será enviada al servidor para después ser procesada por el framework y luego enviada a los demás jugadores, y viceversa.

En el trabajo "Análisis, diseño e implementación de un software para un salón de póquer gratuito" [8] propone diseñar e implementar un prototipo de software para simular virtualmente un juego de póquer Texas Hold'em que incluye apuestas; con diagramas y especificaciones de casos detallados. En los trabajos [9] [10] [11] [12] se explican las ventajas del software reutilizable, así como sus metodologías y las aplicaciones más frecuentes; en donde la más recomendada y con mejores críticas es la aplicación de frameworks.

Un framework es un entorno o ambiente de trabajo para desarrollo; dependiendo del lenguaje normalmente integra componentes que facilitan el desarrollo y/o implementación de aplicaciones. En otras palabras, se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta [3].

Dentro de los frameworks más utilizados se mencionan los que se utilizaron de referencia para el desarrollo de EasyCard.js:

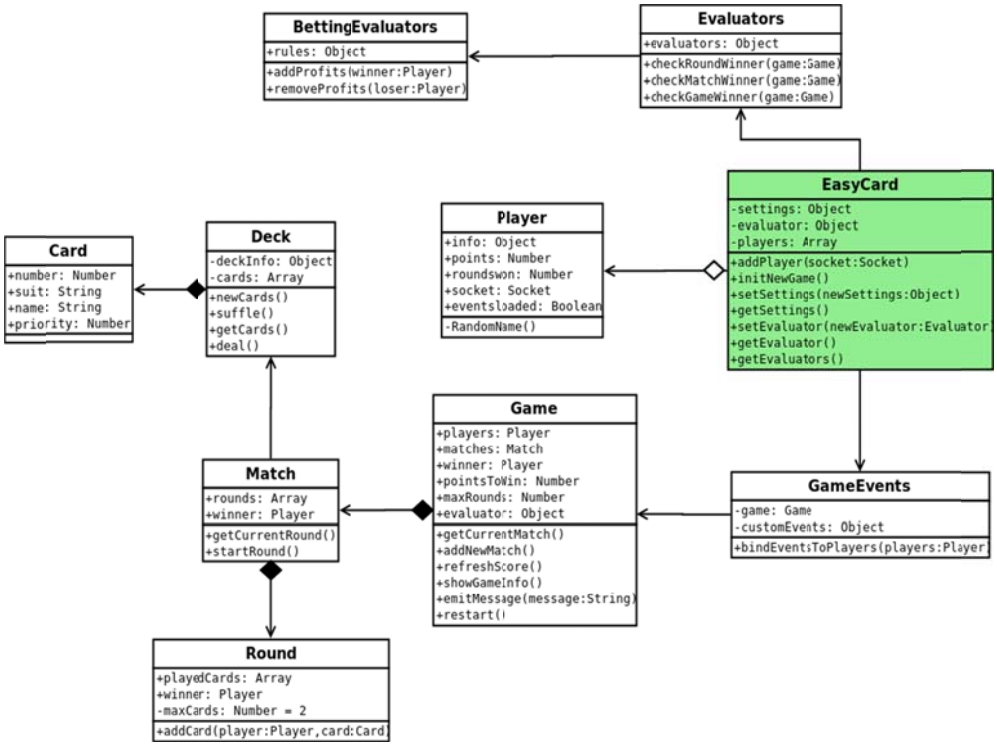

Fig1. Diseño de Easycards.js

Phaser (http://phaser.io) es Open Source y utiliza JavaScript. Está diseñado para que los juegos se puedan ejecutar tanto en ordenadores como en dispositivos móviles, siendo este último su principal foco. No sólo soporta Canvas sino también WebGL y puede pasarse de uno a otro automáticamente según la compatibilidad del navegador. Eso le da un punto de ventaja en cuanto al rendering y la velocidad de respuesta. También maneja física, colisiones, animaciones, sistema de partículas, mapas de patrones, sonidos y permite escalar el juego para que se ajuste a la resolución de cualquier dispositivo sin alterar la relación de aspecto. Impact.js (http://impactjs.com) maneja un sistema de plugins que permite extender sus funcionalidades. Esta flexibilidad de poder extender es uno de los puntos claves que EasyCard.js toma de referencia para incorporar nuevos módulos.

Construct 2 (https://www.scirra.com/construct2) es un framework en el cual se puede realizar videojuegos en formato HTML5 sin tener conocimientos de programación gracias a un intuitivo entorno de desarrollo basado en la filosofía “drag and drop”, es decir, que la mayoría de sus funcionalidades se pueden utilizar desde una interfaz visual sin tener que escribir ni una línea de código. Está orientado a la creación de videojuegos

en 2D, e incluye un motor físicas; también la posibilidad de agregar archivos multimedia. Es un framework muy robusto con muchas herramientas, en donde el usuario no tiene que escribir ninguna línea de código para desarrollar su juego, pero esto lo limita a trabajar con eventos ya preestablecidos sin poder modificarlos, provocando cierta frustración para aquellos casos en el que no exista un evento para algo específico. EasyCard.js soluciona estas limitaciones ya que permite acceder a su código fuente y modificarlo con total libertad.

Kiwi.js es un framework que apunta a ser una herramienta amigable para crear juegos. Incluye soporte para animaciones, sprites, cámaras, sonidos, texturas y un útil módulo para crear interfaces de usuario. Sin embargo, carece de motor de física y colisiones. A partir de estas propiedades, EasyCard.js toma de referencia su diseño sencillo y de fácil implementación.

III. DISEÑO

En esta sección se presenta el diseño de EasyCard.js como se muestra en la Fig. 1, para poder explicar cómo fue diseñado a partir de los requerimientos propuestos. El diseño establece una separación entre módulos determinando jerarquías, donde cada uno se encarga de la implementación del otro. Por ejemplo, el módulo EasyCard implementa GameEvents, y éste a su vez, implementa el módulo Game, etc. Con el requerimiento de realizar un diseño modular genérico, cada módulo cuenta con funciones diseñadas de manera que no tengan un solo propósito y puedan ser reutilizadas para diferentes casos.

El módulo EasyCard es el principal del framework debido a que se inicializa a sí mismo, se encarga de detectar la conexión de jugadores y tiene las funciones setSettings y setEvaluator. Ambas funciones son las únicas que se deben invocar para crear un nuevo juego.

La función setSettings, se utiliza para configurar las propiedades del juego que se va a desarrollar, tales como la cantidad máxima de jugadores, puntos para ganar el juego, etc.

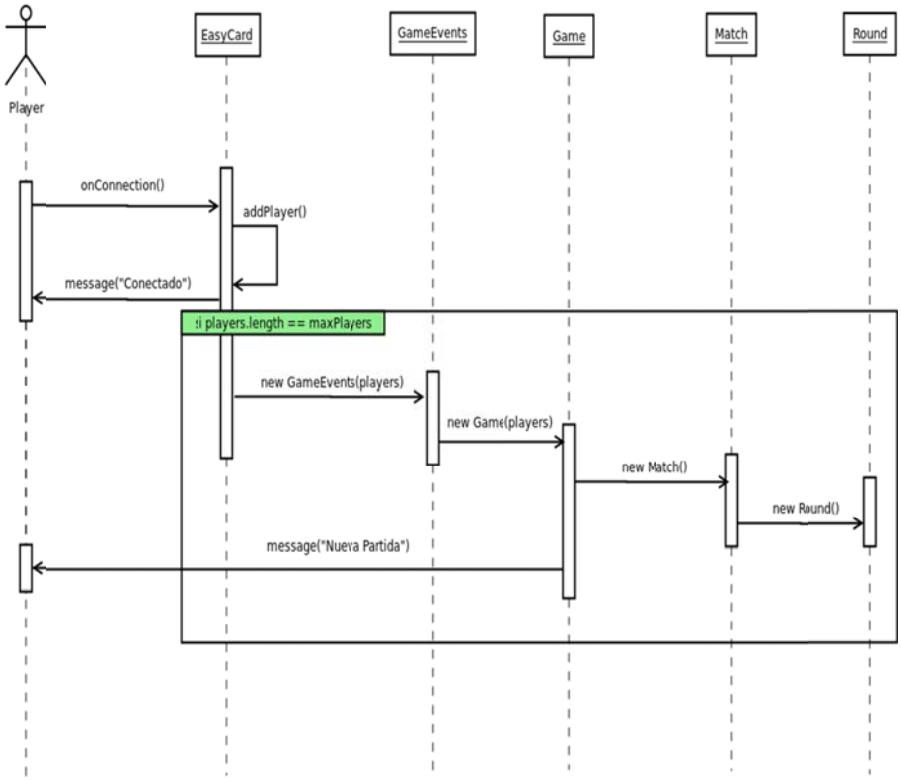

Fig. 2. Diagrama de Secuencia de iniciar partida.

Dentro de la misma se encuentra otra función llamada defineCustomEvents, que define los eventos personalizados que serán propios del juego y estarán ligados al jugador. En la sección 4 se hablará en profundidad de su funcionalidad. Por otro lado, la función setEvaluator, es utilizada para definir los evaluadores que se encargarán de determinar el ganador del juego. Se pueden usar evaluadores que ya están incluidos dentro del módulo Evaluators, o crear uno nuevo respetando la interfaz para después poder incluirlo.

Cuando el módulo EasyCard detecta que se ha llegado al límite de jugadores conectados, instancia GameEvents y con la función bindEventsPlayers se agregan a los jugadores todos los eventos que provee el framework por defecto y los personalizados. Después de ese proceso instancia al módulo Game, como se muestra en la Fig. 2, pasando como parámetro a todos los jugadores conectados.

El módulo Game contiene todos los módulos que estarán interactuando con las reglas de juego y con los jugadores. Éste mismo en el momento de iniciar una partida, en cada enfrentamiento se implementará una instancia del módulo Match, y éste a su vez, implementará una colección de instancias de Round, donde la cantidad de instancias dependerá de las reglas del juego. En el caso de que se esté desarrollando un juego de cartas, se puede implementar el módulo Deck, que provee una interfaz para gestionar una colección de objetos de tipo Card, lo cual simula la estructura de un mazo de cartas. La estructura deberá ser definida con la propiedad deckInfo en setSettings. El módulo Card provee una interfaz para crear diferentes estructuras de cartas.

Durante la partida, el módulo Evaluators recorre toda la estructura de Game para acceder a las propiedades de los módulos que tiene implementados. Sus tres funciones principales checkRoundWinner, checkMatchWinner y checkGameWinner, son utilizadas en cada jugada para detectar si ya existe un ganador. Primero se revisa si existe un ganador en la instancia actual de Round, y si ya se jugaron la cantidad máxima de rondas, se determina el ganador de la instancia actual de Match. Este proceso se repite hasta que un jugador llegue a la cantidad de puntos para ser el ganador. Después Evaluators crea una instancia del módulo BettingEvaluators para invocar la función gestoreProfits que se encargará de enviar el resultado de las apuestas a cada jugador.

Como se ha mencionado anteriormente, EasyCard.js ofrece eventos incluidos por defecto que serán nombrados a continuación.

El evento sendbet, se invoca cuando un jugador realiza una apuesta a través de un campo de texto que es incluido automáticamente por el framework en el HTML del juego. Por defecto este evento está habilitado y el programador lo podrá posicionar donde quiera usando CSS o incluso deshabilitarlo. Como se muestra en la Fig. 3, la apuesta es recibida por el módulo Game y éste solicita a BettingEvaluator que sea almacenada con la función saveBetting.

El evento sendchat, es similar al anterior, pero en vez de enviar apuestas, solamente envía mensajes de textos para que los jugadores puedan conversar entre sí durante la partida. Y por último el evento disconnect, que será invocado cada vez que un jugador se vaya de la partida, para detener y reiniciar el juego, hasta que se cumpla la cantidad requerida de jugadores conectados.

IV. UN EJEMPLO DE UTILIZACIÓN DEL FRAMEWORK

Si bien el framework fue desarrollado principalmente para juegos de cartas, se puede utilizar su diseño modular para implementarlo en juegos sencillos sin tener que realizar modificaciones en los módulos.

Se utilizará de ejemplo un juego de "Piedra, Papel o Tijera" online para dos jugadores, debido a la sencillez de sus reglas, para explicar la implementación del framework. Este juego se ha creado a partir de un proyecto vacío de EasyCard.js compuesto por una estructura de archivos ya preparada, que sirve de esqueleto para desarrollar juegos con funcionalidades ya implementadas. Se puede descargar el repositorio de este proyecto vacío en https://github.com/ezequiel-gomez/easycard- quickstart.

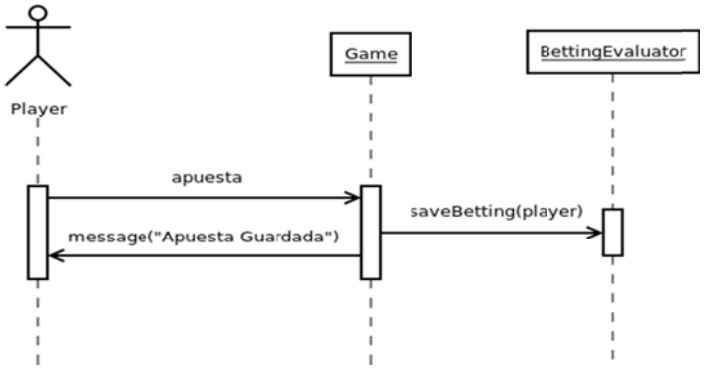

Fig. 3. Diagrama de Secuencia de realizar una apuesta.

```
$ ls easycardjs-rps/ -R
easycardjs-rps/:
client  lib  node_modules  package.json  server.js

easycardjs-rps/client:
css  images  index.html  js

easycardjs-rps/client/css:
main.css

easycardjs-rps/client/images:
mano.png  piedra.png  tijera.png

easycardjs-rps/client/js:
client.js

easycardjs-rps/lib:
betting-evaluators.js  deck.js      evaluators.js  game.js        match.js   round.js
card.js                easycard.js  game-events.js  interfaces.js  player.js  utils.js
```

Fig. 4. Estructura de archivos.

La Fig. 4 muestra la estructura de archivos anteriormente mencionada, donde la carpeta lib contiene todos los módulos de EasyCard.js, y la carpeta client posee todos los archivos necesarios para crear la parte visual del juego. Cada uno de los módulos serán gestionados por el módulo EasyCard.

El primer paso consiste en descargar los paquetes de Node.js para que luego sean instalados localmente en el proyecto. Estos paquetes son librerías externas que serán utilizadas por el framework y están referenciadas en el archivo package.json.

Para descargar e instalar todos los paquetes, solamente hay que posicionarse desde la terminal en la carpeta del proyecto e invocar el comando “npm install”. Una vez instalados todos los paquetes requeridos, será posible importarlos como se muestra en las líneas 2, 3, 4 y 5 en el Fragmento de código 1; para que después sean utilizados en la configuración del servidor en donde será ejecutado el juego, como se muestra en el resto de líneas.

Fragmento de código 1. Javascript de la creación del servidor del juego.

```
1   'use strict'
2   const http = require('http');
3   const express = require('express');
4   const socketio = require('socket.io'); 5
6   //Inicio de la configuración del servidor
7   const app = express();
8   const server = http.createServer(app);
9   const io = socketio(server); 10
11  app.use(express.static(  dirname + '/client'));
12
13  var port = Number(process.env.PORT || 8080);
14  server.listen(port, () => console.log('Ready
to work!.'));
15  //Fin de la configuración del servidor
```

El segundo paso es realizar la llamada a EasyCard.js en el archivo server.js, como se muestra en el Fragmento de código
2. La llamada se realiza de la misma manera en que se importaron los paquetes de Node.js, con la diferencia de que el framework ya está incluido en la carpeta “lib” del proyecto. Después de realizar la llamada, se instancia de manera global el objeto principal llamado easyCard, dentro del cual están todas las configuraciones y funciones que forman parte de la lógica del juego.

Fragmento de código 2. Javascript de la llamada al framework

```
17  require('./lib/easycard');
```

El tercer paso es configurar el objeto easyCard, definiendo las propiedades del juego utilizando la función setSettings, y dentro de la misma se definen los eventos que interactúan con el jugador con la función defineCustomEvents, como se muestra en el Fragmento de código 3. Además, en este paso se elige el evaluador que se utilizará en el juego. En este juego se define con la función setEvaluator, un evaluador ya existente dentro del módulo Evaluators del framework.

Dentro de la función defineCustomEvents, se agregan todos los eventos que serán ejecutados por el jugador durante el juego. Por otra parte, siempre recibe dos parámetros, donde el primero es el jugador que ejecute algún evento y el segundo parámetro es el objeto Game que contiene toda la información actualizada de la

partida. Estos parámetros son indispensables para ser utilizados en los eventos personalizados. En este juego de “Piedra, Papel o Tijera”, es necesario crear un solo evento llamado optionselected, que será ejecutado en el momento en que un jugador tome una decisión. Cuando todos los jugadores hayan elegido sus opciones, se invocan las funciones del módulo Evaluators para comprobar quién es el ganador. En el caso de que se quiera crear otro tipo de juego, en este paso se podrían agregar todas las propiedades y eventos personalizados que se necesiten.

Fragmento de código 3. Javascript de la configuración del objeto easyCard antes de ser inicializado.

```
19  //Definición del evaluador del juego
20  easyCard.setEvaluator(easyCard.getEvaluat
ors().RPSEvaluator);
21
22  //Definición de las propiedades del juego
23  easyCard.setSettings({
24    maxPlayers: 2,
25    maxRounds: 3,
26    pointsToWin: 2,
27    sendChatEnabled: false,
28    defineCustomEvents: function(player, game) {
29      player.socket.on('optionselected', function
(option) {
30        var currentRound =
game.getCurrentMatch().getCurrentRound() || '';
31        currentRound.addCard(player, option);
32        if
(game.evaluator.checkRoundWinner(game)) {
33          game.emitMessage(currentRound.pla
yedCards[0].player.info.name + ': ' +
               currentRound.playedCards[0].ca
rd +
34              ' - ' +
currentRound.playedCards[1].player.info.name
+ ': ' +
               currentRound.playedCards[1].ca
rd);
35          player.socket.emit('enableoptions ');
36          player.socket.broadcast.to(player
.info.room).emit('enableoptions');
37          if
(game.evaluator.checkGameWinner(game)) {
38            game.emitMessage('Partida reiniciada');
39          game.restart(); 40   }
41        } else { player.socket.emit('disableoption s');
```

```
42          game.emitMessage('Falta que elija el otro
jugador', 'Ya elegí', player);
44        }
45      });
46    }
47  });
```

El cuarto paso consiste en agregar las funciones que escucharán el DOM de la página del juego, dentro del archivo client.js. En este juego como se ha creado solamente el evento optionselected, solo es necesario detectar en qué opción el jugador hizo click en el momento de elegir, para después enviar la información al evento, como se muestra en la función optionsClickHandler en el Fragmento de código 4.

Fragmento de código 4. Javascript de las funciones dentro de la página del juego.

```
123  //Función que detecta si alguna de las opciones fue
elegida
124  function optionsClickHandler (e) {
125    if (e.target.id) {
126      //Se envía la opción elegida al evento
personalizado
127      socket.emit('optionselected', e.target.id);
128    }
129  }
```

En el quinto paso hay que volver al archivo server.js y crear el objeto myGame, como se muestra en el Fragmento de código 5, que es básicamente la instancia del nuevo juego que contendrá la estructura del framework con todas las propiedades definidas de los pasos anteriores.

Fig. 5. Juego de “Piedra, Papel o Tijera” ya funcionando.

Fragmento de código 5. Javascript de la inicialización de un nuevo juego.

```
49  var myGame = easyCard.initNewGame(); 50
51  //Se crea un evento en el servidor para que detecte la
conexión de nuevos jugadores
52  io.on('connection', onConnection); 53
54  function onConnection(socket) {
55  myGame.addPlayer(socket); 56 }
```

La función onConnection se ejecutará cada vez que Socket.io detecte la conexión de un nuevo jugador. Recibirá como parámetro un objeto WebSocket que es generado en el momento de conectarse el jugador, para después ser agregado dentro del juego.

Una vez realizados todos los pasos anteriores, en la terminal y en la posición en donde está guardado el proyecto, se escribe el comando “node server.js”. Este comando hará correr el juego en un servidor local para poder verlo funcionando. En el caso de que el servidor local funcione sin problemas se mostrará en la terminal un mensaje diciendo “Ready to work!”. Finalmente, en el browser, se podrá ver el juego funcionando en la URL https://localhost:8080, como fue configurado previamente en server.js. Este juego se puede ver online en https://easycard-rps.herokuapp.com y su código fuente en https://github.com/ezequiel-gomez/easycard-rps.

V. CONCLUSIONES

En este trabajo se ha presentado EasyCard.js, un framework que cuenta con un diseño modular genérico que permite generar juegos de apuestas online de manera más rápida y sencilla.

Se ha implementado exitosamente un juego de “Piedra, Papel o Tijera” en unos pocos pasos, con el fin de demostrar la versatilidad de poder crear juegos reutilizando el diseño del framework.

El desarrollo de EasyCard.js ha tenido como premisa, los procesos de mejora continua, la capacitación, para hacer uso de buenas prácticas profesionales; y también el esfuerzo de realizar comparaciones con otros frameworks identificando sus cualidades y deficiencias, para crear una plataforma versátil y eficiente.

REFERENCIAS

[1] Hermo, E. M. (2016, Abril). La expansión y crecimiento del juego y las apuestas online. Recuperado de https://www.azarplus.com/opinion- plus.php?idopinionplus=63

[2] Ramirez, J. A. (2016, Mayo 12). El asombroso crecimiento de los juegos de apuestas y casino online para móviles. Recuperado de https://www.ngeeks.com/el-asombroso-crecimiento-de-los-juegos-de-apuestas-y-casino-online-para-moviles

[3] ¿Qué es un framework? Recuperado de http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf

[4] Larman, C. (2003). UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al

proceso unificado.
[5] Valderrama, R. P. et al. Una arquitectura para una aplicación inteligente de evaluación basada en patrones de diseño, componentes y agentes bajo el paradigma de WBE. Recuperado de http://www.iiis.org/CDs2012/CD2012SCI/SIECI_2012/PapersPdf/XA717 IJ.pdf
[6] Delgado, E. H. et al. (2011). Revel War, Juego online multijugador de cartas, pp.37-54. Recuperado de http://eprints.ucm.es/16075/1/Memoria_RebelWar.pdf
[7] Cid, A. C. (2014, Junio 20). MMO de Navegador en Tiempo Real con Node.js y WebSockets, pp. 7-27. Recuperado de http://diposit.ub.edu/dspace/bitstream/2445/59452/1/memoria.pdf
[8] Mosquera, R. S. y Herrera, J. A. (2007). Análisis, diseño e implementación de un software para un salón de póquer gratuito, pp. 43-107. Recuperado de http://repositorio.utp.edu.co/dspace/bitstream/handle/11059/790/0051M8 43ad.pdf
[9] Cuadrado, J. S. y Molina, J. G. (2008). Approaches for Model Transformation Reuse: Factorization and Composition. Recuperado de http://link.springer.com/chapter/10.1007/978-3-540-69927-9_12
[10] Jalender, B. et al. (2012, Enero 1). Code Level Reusable Software Components. International Journal of Software Engineering & Applications (IJSEA), Vol.3, pp. 219-227. Recuperado de http://airccse.org/journal/ijsea/papers/3112ijsea16.pdf
[11] Jalender, B. et al. (2010, Junio). A Pragmatic Approach To Software Reuse. Journal of Theoretical and Applied Information Technology (JATIT), Vol 3, pp. 87-96.
[12] Singh, S. et al. (2010, Octubre 14). Reusability of the Software. International Journal of Computer Applications (0975 – 8887) Volume 7, pp .38-41. Recuperado de http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.206.4847&rep =rep1&type=pdf