GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Tecnología Blockchain aplicada a apuestas en juegos de habilidad

Autor: Mauricio Muñoz López

Director: Rafael Palacios Hielscher

Madrid 2019

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Tecnología Blockchain aplicada a apuestas en juegos de habilidad

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2018/19 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Mauricio Muñoz López Fecha: …12…/ …07…/ …2019…

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Rafael Palacios Hielscher Fecha: ……/ ……/ ……

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Mauricio Miñoz López

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Tecnología Blockchain aplicada a apuestas en juegos de habilidad, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción

de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 12 de Julio de 2019

ACEPTA

Fdo Mauricio Muñoz López

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Tecnología Blockchain aplicada a apuestas en juegos de habilidad

Autor: Mauricio Muñoz López

Director: Rafael Palacios Hielscher

Madrid 2019

Agradecimientos

En primer lugar, agradecer a mi familia el apoyo durante toda mi vida, especialmente a mis padres, quienes han hecho posible que esté hoy aquí (literalmente) y a mi tía Inma, quien me ha recordado la existencia del punto y coma.

También quisiera agradecer a mi tutor, Rafael Palacios, la ayuda suministrada a lo largo del proyecto, destacando su flexibilidad y comprensión en momentos de incertidumbre.

Por último, agradezco a Iñigo Sagredo Ruiz que me incluyera en los agradecimientos de su trabajo de fin de grado además de compartir conmigo su profundo conocimiento de Blockchain.

TECNOLOGÍA BLOCKCHAIN APLICADA A APUESTAS EN JUEGOS DE HABILIDAD

Autor: Mauricio Muñoz.
Director: Rafael Palacios.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

El presente trabajo aborda, como punto de inicio, el análisis de la tecnología Blockchain, los Smart Contract y la tecnología asociada a los mismos, así como toda la metodología necesaria para el diseño y desarrollo de una plataforma descentralizada que actúe de intermediaria en apuestas en juegos de habilidad, que recibe el nombre de GAINATGAME. Gracias a las posibilidades de los Smart Contracts [B] y a una lógica blindada en ellos se consigue un servicio de intermediación de apuestas que asegura el cumplimiento de la apuesta, así como la validación del resultado de la misma Esta validación se realiza mediante una audiencia externa de jueces quienes valorar y juzgan las pruebas aportadas por los usuarios que estén participando en la apuesta y designan al ganador de la misma.

Palabras clave: Blockchain, Apuestas, Smart Contract, Gainatgame.

1. Introducción

Blockchain es un ejemplo de tecnología que está revolucionando muchos sectores. Un Blockchain es una red descentralizada formada por diferentes integrantes (nodos) cada uno de los cuales conserva una copia de la información (almacenada en una cadena de bloques vinculados gracias a herramientas criptográficas) democratizando y verificando cualquier cambio posterior. Esta tecnología aporta cualidades muy importantes como confianza, seguridad, velocidad y transparencia las cuales impactarían y revolucionarían enormemente mercados como el de las apuestas, por lo que la elección de la tecnología como Blockchain se considera la más adecuada [A].

La elección de Blockchain como entorno para el desarrollo de la plataforma Gainatgame se basa también en la inmutabilidad que aporta y las garantías que ofrece. El uso de esta tecnología en un mercado emergente y con un gran potencial como el mercado de las apuestas hace de Gainatgame una solución innovadora y de valor real para los usuarios.

2. Definición del proyecto

El proyecto consiste en el análisis en profundidad de las tecnologías apropiadas para el desarrollo de una plataforma en tiempo real para apuestas en juegos de habilidad. Posteriormente se justifica la elección de cada elemento tecnológico basándose en la aplicabilidad que tiene para el prototipo que se desarrolla. Se pone especial interés en analizar la tecnología Blockchain, en qué consiste y que ventajas aporta su integración. Este análisis es clave para poder entender por qué Gainatgame es innovador y aporta valor real a los usuarios.

3. Gainatgame

Gainatgame tiene especial utilidad en apuestas en juegos de carácter privado y minoritario. Debido a su dificultad para ser validado, las casas de apuestas no suministran esta clase de servicios, dando pie a un nicho de mercado sin explotar.

Como los eventos privados y minoritarios comprenden un concepto muy amplio, se propone que la plataforma esté especializada en juegos (preferiblemente videojuegos debido a su mayor facilidad de validación) de suma cero con un único ganador. Estos son, por ejemplo, todos los juegos en los que dos o varios jugadores juegan entre ellos y únicamente existe un ganador que se lleva todo el dinero apostado.

El proceso de validación utilizado se basa en el dilema del prisionero, en el que, en caso de disputa, se celebra un juicio donde los "jueces" valoran una serie de pruebas aportadas por los usuarios que han disputado la partida (pruebas visuales como imágenes y vídeos) y designan al ganador de la misma. En este caso, el ganador será designado por mayoría (51%) y los jueces cuyos votos no pertenezcan a la mayoría no recibirán ningún tipo de compensación por sus servicios.

Además, como el fin de promover la honestidad dentro del sistema, se propone un sistema de incentivos tanto para usuarios como para jueces en el que ambos aportan una fianza (cantidades distintas para jueces y usuarios y que dependerán del tipo de partida) la cual perderán si no actúan honestamente.

A continuación, se expone el esquema de tecnologías implementado, destacándose el Smart Contract en el Blockchain de Ethereum el cual contiene toda la lógica del proceso de apuesta, votación y reparto de premios.

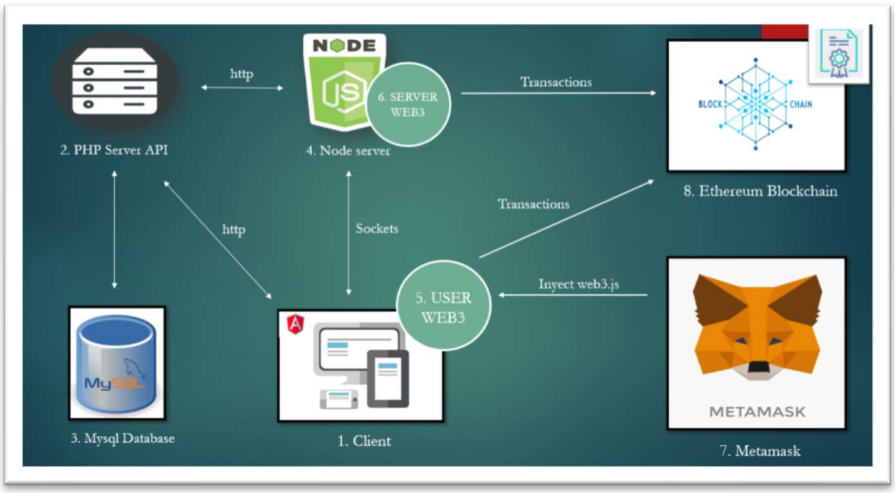


Ilustración 1 - Esquema de tecnologías de Gainatgame

Se ha optado por un diseño semidescentralizado en que se combina tanto Blockchain como tecnologías centralizada tradicionales. Estas incluyen: un servidor con PHP, un servidor de Node y un base de datos Mysql. Se ha elegido Angular 8 como framework para la interfaz, debido a su gran escalabilidad y compatibilidad con Blockchain haciendo de Gainatgame una SPA (Single Page Aplication).

4. Resultados

Gainatgame está completamente operativa, funcional y cumple con los objetivos previstos. Gracias al diseño del Smart Contract se han conseguido funciones cuyas comisiones (debido al gas utilizado) son razonables y permiten que la aplicación sea utilizada en tiempo real por los usuarios. Se muestra a continuación la interfaz de Gainatgame en navegador:

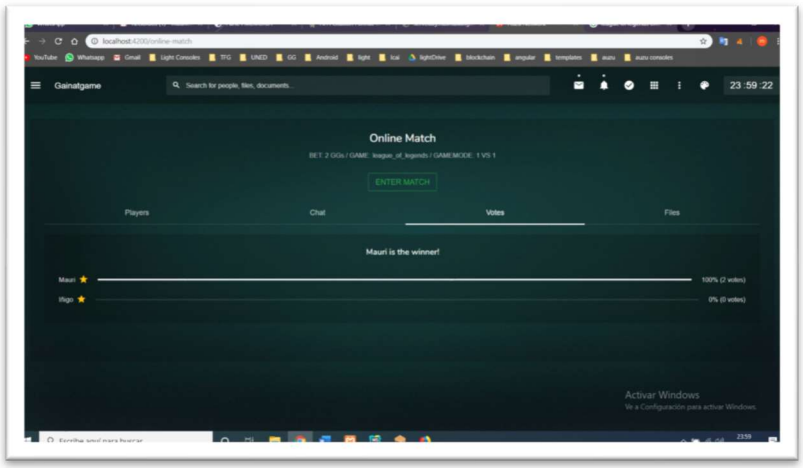


Ilustración 2 Pantalla en la que los jugadores votan y se muestra el resultado actual de la votación

5. Conclusiones

6. Se ha realizado una investigación y análisis extensivo sobre Blockchain con el fin de comprender qué supone la integración de esta tecnología en mercados tradicionales como el de las apuestas y el de los videojuegos. Demostrado así que Gainatgame aporta un servicio innovador en un nicho de mercado emergente además de ser completamente funcional y operativa.

7. Referencias

[A]/@VitalikButerin, Vitalik Buterin. "The Meaning of Decentralization - Vitalik Buterin." Medium, Medium, 6 Feb. 2017, medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274

[B] Antonopoulus, Andreas. Wood, Gavin.-Editorial: O'Reilly Media (2018)., Mastering Ethereum: Building Smart Contracts and Dapps

BLOCKCHAIN TECHNOLOGY APLLIED TO GAMBLING AND BETTING

Author: Mauricio Muñoz.
Supervisor: Rafael palacios.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

The following project studies and analyzes blockchain technology, Smart Contracts applications and all the associated technology owing to provide a decentralized solution to the lack of trust present in betting and gambling dynamics by acting as intermediary in bets on skill games. This platform is called GAINATGAME. Gainatgame ensures the fulfilment of the bet as well as the validation and verification of the result thanks to ironclad logics inside the Smart Contract. This validation is done by external and random judges who analyses and assess proofs provided by the users who participated in the bet with the objective of deciding who is the actual winner of the bet. This process of validation is call trial.[B]

Keywords: Blockchain, Betting, Smart Contract, Gainatgame.

1. Introduction

Blockchain is a good example of technology that is transforming many traditional markets. Blockchain is a decentralized network consisted of several entities(nodes), each of which keeps a copy of the information (in the form of blocks linked using cryptography) democratizing it and verifying any future change as consensus is required. This technology provides advantages in trust, security, speed and transparency. This characteristics impact and improve markets as the gambling and betting one. In consequence, Blockchain technology is the most suitable choice [A].

The application of Blockchain in a new and evolving market as the gambling and betting one is and innovative solution with applications that can actually be a source of value for users.

2. Project definition

The project consists of a deep analysis into all the possible technologies which can be applied in Gainatgame taking into account that the platform has requirements as live usage. The choices made will be justified and explained focusing specially on what are Blockchain integration advantages and consequences, in order to fully understand why Gainatgame is indeed a useful and innovative solution.

3. Gainatgame

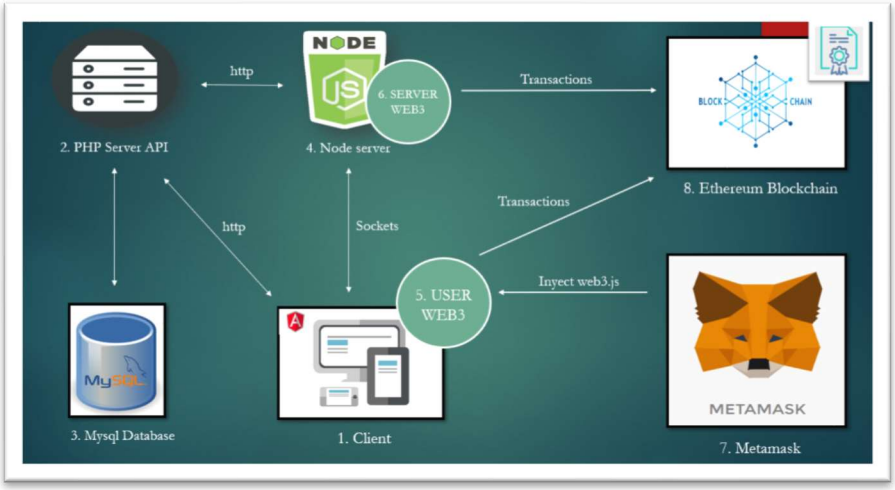
Gainatgame has useful applications in bets on private and minority games. Due to the difficulty of validations of this type of games, gambling and betting operators do not provide this class of services, giving birth to an untapped niche market.

As private and minority events are a really wide and general concept, Gainatgame will be specialize on zero-sum games with only one winner. An example of this games is every game in which two or more players play and only one of the wins all the money.

The validation process used in Gainatgame is based on the prisoner dilemma. In case that there is a dispute between the players while deciding who is the winner, a trial is held. In this trial, several judges verify and assess proofs (typically images or videos) provided by the users who participated in the bet aiming to decide who is the actual winner of the bet. In trials it is required a 51% majority in order to determine the winner and the judges who doesn't belong to this majority do not receive any economic reward in Exchange for their effort.

With the objective of promoting honesty and integrity among players and judges, a system of incentives is applied. User and judges will have to provide bail (different quantities for players and judges). They will lose this bail if they do not act with integrity.

In the following image, it is exposed Gainatgame technology diagram. Showing which technology have been finally chosen and how it is all ensembled together. It is especially important the role played by the Smart Contract which contains all the logic related with the bet.

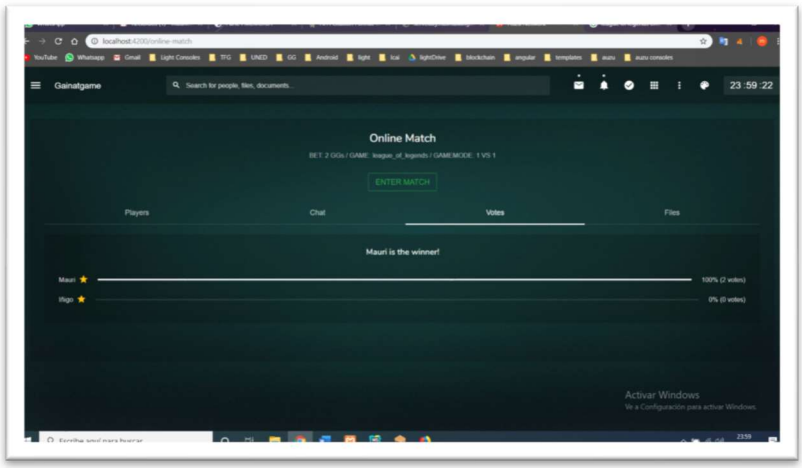


Ilustration 3 – Gainatgame technology diagram

As it can be seen in the image, a semi-decentralized system has been developed. combining Blockchain with traditional centralized technologies as PHP (server), Node (Server) or MySQL (Data base). Angular 8 framework has been the chosen technology to develop the client side. Thanks to Angular Gainatgame is a SPA (Single Page Application) as well as highly

8. Results

Gainatgame is developed and fully operative, functional and it satisfies all the objectives planned. Thanks to the design of the Smart Contract it has been achieved really competitive fees in the Smart Contracts functions allowing Gainatgame to be use live. The following image shows the Angular client in a browser:



Ilustration 4 Voting results screen

9. Conclusion

A Deep analysis and extense research it's been done in order to understand that consequences of integrating Blockchain in a traditional market. As a result, it has been proven that Gaintgame provide an innovative and useful service in an emerging market niche.

10. References

[A]/@VitalikButerin, Vitalik Buterin. "The Meaning of Decentralization - Vitalik Buterin." Medium, Medium, 6 Feb. 2017, medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274

[B] Antonopoulus, Andreas. Wood, Gavin.-Editorial: O'Reilly Media (2018)., Mastering Ethereum: Building Smart Contracts and Dapps

Índice de la memoria

Índice de figuras

Índice de tablas

Índice de ilustraciones

Capítulo 1. INTRODUCCIÓN

Mientras que para muchos las apuestas son un mal endémico de nuestra sociedad, para otros son una fuente enorme de ingresos. En la actualidad, las apuestas están presentes allí donde miremos. A pesar de que los anuncios de tabaco y productos alcohólicos hace tiempo fueron regulados, a día de hoy, las casas de apuestas son los principales patrocinadores de programas de televisión o equipos de futbol estando así muy presentes en nuestro día a día. Es evidente que estamos ante uno de los mercados más rentables de nuestra época con una capitalización global de más de $400 billones [1]. Pese a ser un mercado ya explotado sigue mostrando una gran capacidad de crecimiento y mejora [2]. La incorporación de nuevas tecnologías será clave para alcanzar todo su potencial.

En los últimos años hemos sido testigos de cómo prácticamente todos los mercados están pivotando hacia métodos y procedimientos más optimizados, inteligentes, eficientes y rápidos con el fin de aumentar la productividad y mejorar la experiencia de usuario. Esto se está consiguiendo aplicando tecnologías innovadoras a mercados obsoletos y tradicionales. Un gran ejemplo es el efecto de la inteligencia artificial y la robótica sobre sectores que hasta el día de hoy habían basado sus procesos de producción en la mano de obra. Progresivamente todos y cada uno de los sectores irán incorporando nuevos desarrollos con el fin último de aumentar sus beneficios y mejorar.

Blockchain es un gran ejemplo de tecnología que está revolucionando muchos sectores y mercados. Un Blockchain es una red descentralizada formada por diferentes integrantes (nodos) los cuales conservan una copia de la información (almacenada en una cadena de bloques) democratizando y verificando cualquier cambio que se desee realizar. Esta tecnología aporta cualidades muy importantes como confianza, seguridad, velocidad y transparencia [14]. Estas características impactarían y revolucionarían enormemente mercados como el de las apuestas, por lo que la integración de tecnologías como Blockchain es cuestión de tiempo.

El mercado de las apuestas es muy amplio y se extiende desde apuestas en juegos de azar hasta

apuestas deportivas. Sin embargo, el reciente aumento de la popularidad de las apuestas online está transformando el mercado en su totalidad. Las apuestas online son uno de los mercados que más rápido están creciendo en parte gracias a la mejora de accesibilidad a internet [2]. En la pasada década el crecimiento ha sido de un 9% anual lo cual equivale al triple del crecimiento del PIB global. Los operadores y casas de apuestas han tenido que adaptarse a esta nueva tendencia intentando conseguir la mejor experiencia de usuario posible y la mayor eficiencia para maximizar sus beneficios [3]. Sin embargo, existen muchos problemas por resolver y muchos procedimientos por optimizar:

1. Los sistemas de pago y de transacciones online son excesivamente lentos, tienen comisiones desproporcionadas y fallan habitualmente [7][8][9].
2. Existe una conciencia generalizada sobre las actividades fraudulentas [10][11], tanto por parte de los usuarios como por parte de los operadores. Mientras que los operadores tienen que invertir en sistemas de seguridad muy costosos, los usuarios no tienen confianza en los operadores ya que estos son la autoridad central y tienen todo el poder, pudiendo falsificar información en su propio beneficio. La desconfianza de los usuarios es más que justificable teniendo en cuenta que las apuestas son el origine de muchas estafas y engaños.
3. Los procesos de verificación y de validación son complejos y muy largos.
4. Los usuarios no tienen control sobre sus fondos depositados. El operador tiene control total sobre el dinero de todos los usuarios pudiendo hacer con ellos lo que desee.

Tras un profundo análisis de la situación actual del mercado de las apuestas [4][5][6] y de las ventajas y mejoras que aportan las nuevas tecnologías creemos que un sistema basado en Blockchain podría dar solución a los problemas antes mencionados beneficiando tanto a operadores como a usuarios.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Para poder entender qué supone la integración de Blockchain es necesario profundizar sobre qué es y qué ventajas aporta esta tecnología [12].

1. REDES CENTRALIZADAS VS REDES DESCENTRALIZADAS

Una red centralizada se caracteriza por almacenar la información en un único lugar. Este es el procedimiento tradicional. Por ejemplo, las bases de datos de los servidores de páginas webs. Los sistemas centralizados son los más populares y usados a día de hoy ya que son sencillos, rápidos y baratos. Esta tecnología está asentada, pero aún existen muchas formas y mecanismos para mejorar y eliminar sus debilidades. Sin embargo, hay otros problemas que son prácticamente irreconciliables. No solo debido a limitaciones prácticas, sino a que los intereses de las empresas no suelen mirar en la misma dirección [13].

Se han invertido miles de horas de investigación y miles de millones en la búsqueda de la descentralización. Al descentralizar un sistema, la información ya no solo se encuentra en un único lugar, sino que varias entidades, los denominados "nodos", poseen una copia y forman parte del sistema. Ya no hay un único dueño, por lo que el acceso y modificación de la información se democratizan. La descentralización soluciona los siguientes problemas:

- En las redes centralizadas se necesita depositar confianza en la autoridad central. Como la información la posee únicamente una entidad, ésta tiene el poder de manipular, distribuir e incluso borrar información.

- Existe un punto único de fallo. Si ocurre algún error el sistema al completo deja de estar operativo. Lo mismo ocurre con la información almacenada. Por supuesto existen soluciones como sistemas de back-up o redundancias en los sistemas para asegurar que, si un sistema queda inoperativo, habrá otro que pueda proporcionar ese servicio.

- Censura y manipulación. El dueño del sistema podría decidir qué está permitido y qué no. A quién mostrarle cierta información y a quién no. En empresas como Twitter o Facebook este problema cobra vida. Prohibir ciertas ideologías, promover un partido político, etc.

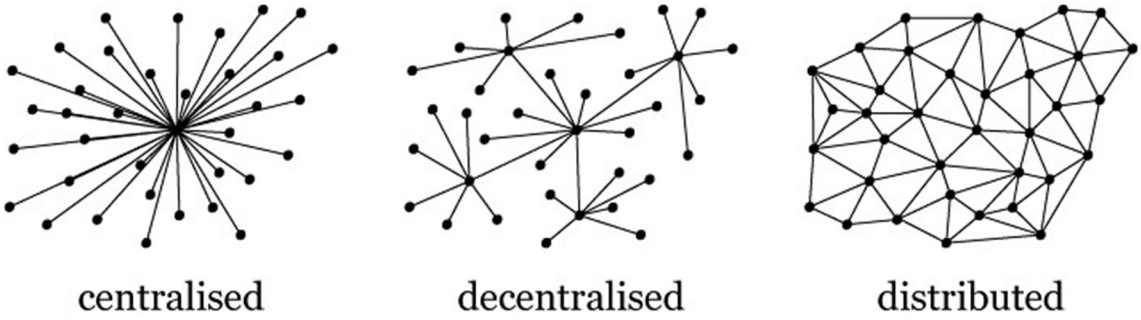


Figure 1 Topologías de red [14]

El término descentralización, pese a ser muy utilizado en la actualidad, es quizás uno de los más pobres en cuanto a definición. Cuando se habla de descentralización, en realidad, se está hablando de tres tipos distintos: estructural, política y lógica.

- Estructural. El sistema posee varios elementos. Si uno de ellos falla el sistema sigue operativo.
- Política. ¿Quién controla el sistema? ¿Es una única organización?
- Lógica. Se trata de varios elementos actuando como uno solo o unidades independientes. BitTorrent, la plataforma para descargar archivos localizados en muchos lugares en vez de únicamente en un servidor, es un ejemplo de descentralización lógica [14].

Las redes descentralizadas que vamos a tratar tienen descentralización política (nadie las controla) y estructural (no existe una autoridad central), pero no lógica ya que todos los nodos del sistema están en el mismo estado y se comportan como una solo entidad.

2. BLOCKCHAIN

Un ejemplo de red descentralizada y que será objeto de estudio en este trabajo es la red Blockchain. La primera mención a esta tecnología fue realizada por Stuart Haber y W. Scott Stornetta en 1991[12]. Una red Blockchain almacena la información en forma de bloques vinculados unos con otros como si de una cadena se tratase. La información almacenada se puede visualizar como una "base de datos descentralizada" la cual se almacena simultáneamente en los nodos del sistema. Para modificar el estado de la red se requiere del consenso y cooperación de los nodos. Este consenso tiene como objetivo validar y verificar la información incorporada. Los cambios e incorporaciones realizados se incluyen en un nuevo bloque que se añade a la cadena

La Seguridad de Blockchain se fundamenta fuertemente en la aplicación de la infraestructura criptográfica PKI (clave pública) y el uso de funciones resumen o hashes como SHA 256 y keccak256. Gracias al vínculo que existe entre un bloque y el hash del bloque anterior se garantiza la inmutabilidad de la red. La autenticación y "no repudio" (la realización de una transacción es vinculante y no es posible negar la autoría de la misma) se consigue mediante el uso de firmas electrónicas con clave privada, satisfaciendo los principios fundamentales de la seguridad [15][16][17].

El razonamiento matemático-económico detrás de la seguridad y fiabilidad del consenso para la creación de un nuevo bloque se basa en un modelo de elección descoordinada, en el que cada uno de los nodos toma decisiones independientes. Si más de un tercio de la red se sincronizase y tomase decisiones conjuntas podría aprovecharse del sistema. Un ejemplo es el "self-mining" en el que un grupo de "mineros" (los nodos del sistema) eligen cooperar para obtener ventajas sobre el resto [18]. Poco a poco más y más mineros se irán incorporando al grupo ya que les es más rentable que la actuación en solitario. Si este grupo consigue más de la mitad de la red podrían realizar "ataques de 51%" en los que al ser mayoría podrían aprobar transacciones unilateralmente [14]. El sistema dejaría de ser descentralizado.

En la práctica conseguir un gran porcentaje de proceso de la red es complicado. Sin embargo, Vitalik Buterin [19], cocreador del Blockchain Ethereum, en su artículo "The Meaning of descentralization" se hace esta pregunta. ¿Es de verdad realista decir que las decisiones de los nodos son completamente independientes y descoordinadas cuando en una misma foto encontramos al 90% del poder de procesamiento de Bitcoin?[14]

Figure 2 Dueños del 90% del poder de procesamiento de Bitcoin (mineros y pools) [14]

Asumiendo que las acciones de los nodos son tomadas de forma independiente, sigue siendo necesario un sistema de incentivos para promover la honestidad y castigar las acciones fraudulentas (ej. incluir información falsa en un nuevo bloque). Este sistema de incentivos está estrechamente relacionado con la verificación del nuevo bloque. A la hora de crear un nuevo bloque es crucial la validación y verificación de qué información se está incluyendo. Los encargados de realizar esta comprobación son los propios nodos (Mineros). A cambio de este servicio reciben una recompensa. Las dos metodologías de verificación más utilizadas en la actualidad son:

Proof Of Work (POW): Es el tipo de validación de bloques más conocido y utilizado. Consiste en buscar un hash de la cabecera del bloque que cumpla una serie de requisitos. Para encontrar este hash se requiere capacidad de procesamiento. Quién más procesamiento posea más probable es que sea el primero en encontrar la solución. A cambio, el sistema otorga una

recompensa, típicamente criptomonedas de la propia red. Si un minero decidiese actuar fraudulentamente y tratase de engañar a los demás nodos, estos finalmente lo detectarían y no aprobarían dicho bloque. Esto es debido a que, antes de recibir la recompensa, el resto de nodos debe dar su visto bueno al nuevo bloque creado. En consecuencia, todo el trabajo de procesamiento realizado por el minero fraudulento se perdería. Y con él, los costes asociados de energía, mantenimiento etc. Además, como la recompensa otorgada será en forma de criptomoneda de la propia red, a los mineros les conviene que la red incremente en popularidad para así que aumente su valor. De este modo, los intereses de la red y los de los mineros están alineados. Una práctica común entre los mineros es la de unirse para incrementar su capacidad de procesamiento con el fin último de aumentar las probabilidades de obtener la recompensa. Si ésta se consigue, se reparte proporcionalmente entre los participantes según el poder de procesamiento (medido en hashes por unidad de tiempo) aportado. Estas asociaciones de mineros se denominan "Pools".

Proof of Stake (POS): Proof of work supone un gasto innecesario de procesamiento y electricidad al estar todos los nodos simultáneamente realizando la misma tarea. Una forma mucho más eficiente es establecer un sistema democrático de elecciones en el que se elige qué nodo será el siguiente en crear el bloque. Para poder participar en las elecciones se necesita realizar una inversión en la red. Cuanto mayor sea la inversión realizada más probable es que sea la red elija a ese inversor para la creación del nuevo bloque. La función de distribución usada para determinar la probabilidad con la que un minero crea el siguiente bloque es crucial. Si no es la adecuada, el sistema puede centralizarse al haber nodos que prácticamente siempre son los elegidos. Por ello se elige una función lineal la cual intenta luchar contra los grandes mineros que cuentan con ventajas como economías de escala, diferentes precios y gastos debido a la ubicación, etc. Una vez todos los componentes de la red han dado el visto bueno al nuevo bloque, el dinero invertido ("Stake") es devuelto. Si se detecta actividad fraudulenta, tanto el nodo responsable como los nodos que dieron su visto bueno a dicho bloque, son sancionados retirándoles parte de la inversión inicial. De esta forma se desincentiva tanto la creación de bloques fraudulentos como su validación por el resto de los nodos.

3. BITCOIN

En 2008 Satoshi Nakamoto, quien a día de hoy sigue en el anonimato, plasmó la idea en un sistema real y creó el protocolo Bitcoin de transferencia P2P (peer-to-peer) de dinero digital. El protocolo Bitcoin es la primera aplicación práctica del concepto de Blockchain. En los bloques de Bitcoin se almacena la contabilidad de la criptomoneda (Bitcoin). De este modo un movimiento de dinero digital se efectúa cuando éste se incorpora en un nuevo bloque en forma de transacción. Se trata de un "libro mayor descentralizado". En Bitcoin un nuevo bloque es creado cada 10 minutos

4. ETHEREUM

Tras la creación de Bitcoin en 2008 muchos otros protocolos han aparecido, incorporando modificaciones y mejoras a Bitcoin con nuevos enfoques y aplicaciones. Sin embargo, la lógica subyacente y las posibilidades de la misma seguían siendo muy pobres. Volviendo a la comparación entre un servidor y un Blockchain como representantes de redes centralizadas y descentralizadas, se hace patente rápidamente la enorme diferencia en cuanto a lógica, proceso y capacidades entre ambos. En 2013 Vitalik Buterin, de apenas 20 años, propuso a su grupo de trabajo integrar al protocolo Bitcoin una Máquina de Turing completa; el concepto es una idea original desarrollada por Sergio Demian Lerner. Esta mejora permitiría ejecutar cualquier lógica en el propio Blockchain y así poder proporcionar más funcionalidad. En 2014, gracias a una plataforma de financiación colectiva, nace Ethereum [22]. [La primera plataforma en permitir la ejecución de lógica programada en un entorno descentralizado.]

Ethereum amplía las posibilidades de anteriores Blockchain mediante el uso de su máquina virtual (Ethereum Virtual Machine, EVM en futuras referencias), con la que consiguen integrar satisfactoriamente una máquina de Turing completa [23][24]. En sus inicios Ethereum usaba Proof of Work para la validación y verificación de los nuevos bloques. Sin embargo, debido al aumento de popularidad y claras ventajas que supone, se está incorporando progresivamente el uso de Proof of Stake. Actualmente aún conviven POW y POS.

La moneda oficial en Ethereum es el Ether. Sin embargo, con el objetivo de tratar siempre con números enteros y no particionar las cantidades a la hora de realizar transacciones más pequeñas, suelen usarse unidades de medida menores como el Wei o el Gwei.

Unit	Wei Value	Wei
wei	1 wei	1
Kwei (babbage)	1e3 wei	1,000
Mwei (lovelace)	1e6 wei	1,000,000
Gwei (shannon)	1e9 wei	1,000,000,000
microether (szabo)	1e12 wei	1,000,000,000,000
milliether (finney)	1e15 wei	1,000,000,000,000,000
ether	1e18 wei	1,000,000,000,000,000,000

Figure 3 Unidades de magnitud del Ether [25]

2.1.1 SMART CONTRACTS

Gracias a la EVM es posible crear programas que se ejecuten en el Blockchain. Nace aquí el concepto de Smart Contract. Un Smart Contract asegura el cumplimiento de forma automatizada y digital de una serie de instrucciones (como si de cláusulas de un contrato se tratase). En este contrato pueden participar distintos usuarios y el contenido del mismo no tiene límites lógicos puesto que se cuenta con un lenguaje de Turing Completo. Las consecuencias de este contrato son irreversibles puesto que una vez incluidas en un nuevo bloque el propio Blockchain asegura su inmutabilidad.

Para definir la funcionalidad del Smart Contract se utiliza un lenguaje de programación de alto nivel. El más utilizado es Solidity. Una vez escrito el código, éste es traducido a una forma más primitiva y de bajo nivel conocido como "Bytecode"[27]. El Bytecode contiene las operaciones más elementales y de más bajo nivel posible tales como la modificación de registros u operaciones matemáticas básicas. Estas operaciones se conocen como "OpCodes". En todos

los nodos de Ethereum corre la misma versión de la EVM para poder asegurar mismos resultados [26].

2.1.1 ERC

Como en toda tecnología existen estándares y en Ethereum no es distinto. Los ERCs (Ethereum Request for Comments) definen reglas y funcionalidades a implementar mediante interfaces para permitir a los desarrolladores y usuarios predecir de forma precisa el comportamiento del contrato. Si un contrato implementa un ERC es posible asegurarse de cómo va a funcionar. Un ejemplo de ERC es el ERC-20 [27] para la implementación de token.

2.1.2 TOKENS, ICO E ITO

Un token es un Smart Contract que simula una contabilidad interna dentro del propio Blockchain, creando así una nueva moneda completamente independiente [23]. No se trata de una criptomoneda como típicamente se entiende ya que éstas son las monedas oficiales de los Blockchains como Bitcoin o Ether. Cualquier empresa o entidad puede crear su propia moneda creando un Smart Contract que implemente el ERC-20 para disponer de un token completamente operativo. Los tokens descritos en el ERC-20 [27] son completamente fungibles, es decir todo token es idéntico a cualquier otro. Sin embargo, existen tokens como los descritos por el ERC-721 [28] los cuales son no-fungibles. Esto hace que cada token sea único y completamente transferible.

Una ICO (Initial Coin Offer) o ITO (Initial token offer) hace referencia a la primera distribución de tokens que una empresa realiza con el fin de sacar al mercado sus tokens. Típicamente en startups, se genera un número limitado de tokens (para evitar inflación) y un porcentaje del mismo se vende a los usuarios. El dinero recaudado sirve como financiación para la empresa evitando así diluirse al vender porcentaje de la misma a cambio de financiación a inversores. Es también muy habitual conservar parte de los tokens generados para destinarlos a los fundadores, equipo principal etc.… con la intención de revenderlos en un futuro cuando el valor del token suba [1][31]. En una ICO el token se genera sobre una red Blockchain nueva mientras que en una ITO se genera en una red ya existente [30].

Los usuarios que compran tokens lo hacen por dos motivos:

- Especulación: Si la empresa sube en popularidad el token subirá de valor ya que sus servicios son más demandados y existe el mismo número de tokens.
- Utilidad: Muchos tokens sirven para comprar servicios o productos que la propia empresa ofrece [1].

Una vez que la ITO se ha realizado para poder adquirir o vender tokens a cambio de dinero real o por otra criptomoneda se acude a un "Exchange" donde, de forma análoga a la bolsa, cada token tiene un valor y se puede comprar y vender en tiempo real [32].

2.1.3 GAS Y COMISIONES.

Hemos determinado que la forma de realizar un cambio de estado en el Blockchain es mediante una transacción la cual se validará y se incorporará al nuevo bloque. Un bloque tiene un tamaño máximo dependiendo del Blockchain y en consecuencia un número máximo de transacciones que se pueden realizar. En Bitcoin se mina un nuevo bloque cada 10 minutos. En Ethereum Vitalik Buterin estimó que el tiempo de minado de un bloque seria de aproximadamente 12 segundos puesto que 12.6 segundos es el tiempo medio que tarda en propagarse un nuevo bloque a una mayoría de los nodos en una red P2P. Sin embargo, en la práctica, algunas dificultades hicieron que el tiempo medio real de minado de bloque rondase los 17 segundos [23].

Al existir un número máximo de transacciones por bloque y un tiempo de bloque, existe un máximo de transacciones por unidad de tiempo. Teniendo en cuenta estas limitaciones es coherente preguntarse cómo hace la red para no saturarse. Cómo se evita que, por ejemplo, un usuario con malas intenciones realice continuamente transacciones provocando que la red vaya más lenta e incluso quede inoperativa. La solución radica en las comisiones. Para evitar cantidades infinitas de transacciones por parte de los usuarios se determinó que, asociada a cada transacción, sería buena idea que hubiese una comisión. Esta comisión determinaría el orden en el que ésta es incluida en los bloques por parte de los mineros. Cuanto mayor sea la comisión, más rápido será la transacción validada puesto que los mineros la priorizaran sobre el resto

atendiendo a su propio beneficio. En consecuencia, si un usuario quiere enviar constantemente transacciones a la red tendrá que asumir el coste. Las comisiones están presentes en Blockchains como Bitcoin, Ethereum, EOS, etc.…

En Ethereum la ejecución de los Opcodes requiere procesamiento adicional. Por lo que, si seguimos la misma lógica que en el apartado anterior con las comisiones, es necesario establecer algún mecanismo para incentivar a los nodos y retribuirlos por su trabajo. Surge así la idea de "gas".

A cada OpCode se le asigna un coste en gas. Por ejemplo, sumar cuesta 3 de gas. De esta forma, toda transacción incluyendo aquellas que ejecuten código de un Smart Contract tendrán asociado un coste en gas. El gas no constituye ninguna unidad monetaria por lo que es necesario transformarlo a Ether. Esto se realiza multiplicando el gas utilizado por el precio del gas. Este precio no es fijo y es el usuario quien deberá elegirlo. Así, cuanto mayor sea el precio elegido, menor será el tiempo que tarde la transacción en ser incluida en un nuevo bloque. No siempre para el mismo precio se obtiene el mismo tiempo de espera. Este tiempo depende de muchos factores, pero sobre todo del nivel de saturación de la red.

Operation	Gas	Description
ADD/SUB	3	Arithmetic operation
MUL/DIV	5	
ADDMOD/MULMOD	8	
AND/OR/XOR	3	Bitwise logic operation
LT/GT/SLT/SGT/EQ	3	Comparison operation
POP	2	Stack operation
PUSH/DUP/SWAP	3	
MLOAD/MSTORE	3	Memory operation
JUMP	8	Unconditional jump
JUMPI	10	Conditional jump
SLOAD	200	Storage operation
SSTORE	5,000/ 20,000	
BALANCE	400	Get balance of an account
CREATE	32,000	Create a new account using CREATE
CALL	25,000	Create a new account using CALL

Figure 4 Opcodes de Ethereum con su coste en gas y descripción correspondiente [23]

2.1.4 ANÁLISIS DE LOS COSTES EN GAS EN UN SMART CONTRACT

Tomando como precio medio 1 Gwei se obtiene que el precio de una unidad de gas es 0.0001 céntimos. El mínimo uso de gas en una sola transacción está limitado a 21000. Por lo que el coste mínimo de una transacción será de 0.5 céntimos [33]. Las transacciones limitadas por esta cota inferior son típicamente transacciones que únicamente tratan de movimientos de Ether de una cuenta a otra.

Nunca antes había sido tan importante optimizar el código de un programa. En Ethereum literalmente cada línea de código cuesta dinero. En todo sistema que almacene información hay dos acciones principales: escribir y leer.

Existen dos formas principales de escribir o almacenar información: en "storage" o en "memoria". Guardar una variable en storage es guardar la variable de forma persistente en Ethereum. No se destruye entre distintas llamadas a funciones. Ésta es sin duda una de las acciones más costosas de todo Smart Contract (20,000 gas al crearla y 5,000 al modificarla). Guardar una variable en memoria consiste en almacenarla de forma temporal de forma que cuando la llamada a esa función termine la variable será destruida (4,000 gas)[23].

Los nodos tienen acceso a todo el histórico de transacciones del Blockchain por lo que son capaces de leer toda la información que desean y saber en cada momento el estado del Blockchain. De tal modo que, si necesitan el valor de una variable a la hora de ejecutar una función, tienen acceso a ella. El coste de leer una variable del storage o de memoria es 200 y 3 de gas, respectivamente. Pero surge aquí una nueva pregunta: ¿Es económicamente viable una aplicación que tenga que pagar cada vez que necesita leer el estado de una variable?

Pongamos un ejemplo: El dueño de un Smart Contract desea guardar el valor de una variable en él y crear una aplicación en la que los usuarios puedan acceder a dicha variable. Para ello diseña un Smart Contract que contenga esta lógica y crea dos funciones, una para escribir y otra para leer la variable. Tanto el despliegue y creación del Smart Contract como la primera llamada a la función escribir usan gas y le cuesta dinero al dueño en forma de comisiones. Una vez que la variable ya posee un valor, todos los nodos pueden acceder a ella. Una posibilidad para que

los usuarios puedan acceder a esta variable sería la de descargarse todo el Blockchain como si de un nodo se tratase y leerlo ellos mismos. Esta opción queda descartada debido a la dificultad y la poca escalabilidad de este método. Imagine a un smartphone teniendo que descargar Gigabytes de información para tener que ver una página web; por no hablar de la necesidad de actualizar la información. La otra posibilidad es asumir los costes que los nodos cobran por leer esa información en forma de gas. Pero si cada vez que recarga una página web el usuario tuviese que pagar nadie usaría esta tecnología. Por ello nacen los conceptos de funciones View [23][34] y JSON/RPC [35].

Para dar soluciones y conseguir que la tecnología aporte valor y sea integrable en sistemas reales existen nodos que de forma altruista mediante el protocolo JSON/RPC sobre HTTP ejecutan en lugar de los usuarios las funciones leer en el Blockchain (puesto que tienen toda la cadena de bloques ya descargada al ser nodos). El resultado es devuelto en una respuesta también sobre HTTP y todo ello sin costes asociados. Ethereum, para evitar que los usuarios abusaran de esta solidaridad, crearon el tipo de función VIEW. Este tipo de función solo permite funcionalidad similar a la lectura, pero no escritura puesto que se entiende que la lectura no supone un coste excesivo para los nodos mientras que guardar un nuevo valor en el storage sí. Cabe destacar que las funciones View únicamente conllevan cero costes cuando son ejecutadas por los nodos como respuesta a una petición JSON/RPC. Si dentro de la función escribir antes descrita se necesita llamar por cualquier motivo a la función leer, el gas usado por la función leer si tendrá coste asociado y formará parte del coste total de llamar a la función escribir. Así pues, las funciones VIEW son gratuitas solo bajo ciertas condiciones. La existencia de estas funciones es, sin duda, esencial. Pero ¿por qué debería importarle al dueño de un Smart Contract cuánto se pague en comisiones al interactuar con el mismo si, al fin y al cabo, son los usuarios quienes pagan las comisiones?

La razón radica en la experiencia de usuario. Estamos acostumbrados a que todas las aplicaciones sean de gran calidad, instantáneas y sobre todo gratuitas. Ethereum no es instantánea ni gratuita. Su uso conlleva un coste: las comisiones. Y estas comisiones, pese a ser pequeñas, pueden suponer sin duda una desventaja competitiva frente a proveedores de servicios y empresas tradicionales. Por ello existen actualmente muchas iniciativas

(ej. EOS Blockchain) que intentan solucionar estos problemas mediante nuevos enfoques. En EOS para poder tener un Smart Contract en vez de usar gas para desplegarlo y mantenerlo se debe tener CPU, ancho de banda y RAM. Para conseguirlos es necesario "alquilarlos" de forma parecida al concepto de Proof of Stake. En EOS para el uso de red se debe tener dinero invertido en la misma y si finalmente se decide cesar la actividad se podrá recuperar lo depositado. Un claro inconveniente son los altos precios altos mencionados. En EOS el coste en memoria y ancho de banda que resulta de desplegar y mantener un Smart Contract con el que interactúan 1000 usuarios durante un año es de $50.000, siendo íntegramente recuperable. Mientras que en ETH por creación, mantenimiento y 1 millón de transacciones que interactúen con el Smart Contract el coste sería de unos $30.000 de los cuales $5 serían lo asumido por el dueño del Smart Contract por la creación del mismo y $29.995 lo asumido por los usuarios en forma de comisiones. Ambos enfoques tienen sus ventajas y desventajas, como se ha explicado, pero, de momento, Ethereum es sin duda el más utilizado.

A corto plazo parece que las comisiones son un mal menor aceptable. Sin embargo, cabe plantearse qué pasará en un futuro cuando el valor del Ether aumente. El gas usado seguirá siendo el mismo mientras que el precio del gas habrá aumentado al estar éste vinculado al Ether. Conforme el Ether aumente en valor los precios de gas deberán ir bajando para que así las comisiones se mantengan coherentes y asumibles.

2.1.5 APLICACIONES DE TIEMPO REAL

Un caso concreto y que va a ser determinante a lo largo de todo este trabajo son las implicaciones de las comisiones en aplicaciones en tiempo real. ¿Es posible usar Blockchain en aplicaciones en tiempo real de forma rentable y con una experiencia de usuario competitiva?

Sabemos que el tiempo de bloque de Ethereum es de unos 15 segundos. Ésta es una limitación práctica y toda aplicación deberá lidiar con ello. Si una aplicación requiere de interacciones cada menos tiempo, Ethereum no es la tecnología apropiada. Aunque 15 segundos no son sinónimo de instantaneidad, es un tiempo de espera asumible por los usuarios. Sin embargo, que

un usuario esté dispuesto a esperar un par de decenas de segundos no significa que esté igual de dispuesto a pagar las comisiones necesarias como para que todas y cada una de sus transacciones realizadas al interactuar con el Smart Contract se incorporen al Blockchain en 1 tiempo de bloque. Para conseguir este logro habría que utilizar precios de gas muy elevados que pueden llegar a superar los 15 Gwei en ciertas horas del día. Siendo 80.000 de gas una buena aproximación del gas usado en una transacción media al interactuar con un Smart Contract y asumiendo que el usuario requiere realizar 5 transacciones para completar un servicio y todas y cada una de ellas deben realizarse en 1 tiempo de bloque, el coste asociado sería de: 15 céntimos * 5 transacciones = 75 céntimos [33].

Todos los servicios y funcionalidades que no aporten un valor superior a 15 céntimos por transacción no son económicamente rentables. Esta limitación, como veremos más adelante, juega un papel fundamental a lo largo de todo el proyecto, siendo uno de los mayores retos que afrontar.

5. APLICACIONES DESCENTRALIZADAS O DAPPS

Una Dapp o aplicación descentralizada es aquella cuyo back-end está descentralizado. Esta descentralización puede ser total o parcial. Las Dapps se basan habitualmente en la tecnología Blockchain para lograr la descentralización, aunque existen excepciones, como BitTorrent, que utilizan otros tipos de redes P2P [36]. Para poder albergar no solo información sino lógica es necesario el uso de Smart Contracts por lo que son necesarios Blockchains con máquinas de Turing completas. Algunos de los más utilizados son Ethereum y EOS. Sin embargo, cabe la posibilidad de que cada Dapp cree su propio Blockchain con sus propios nodos. Esta alternativa es sin duda poco escalable. Para que el usuario interactúe con el sistema y realice las transacciones pertinentes las Dapps también deben poseer una interfaz o front-end.

Cabe destacar que los usuarios de una Dapp no son nodos del Blockchain. Los nodos de un Blockchain poseen una copia de todos los bloques y participan activamente en los procesos de verificación de los nuevos bloques. Sin embargo, los usuarios simplemente publican nuevas transacciones que desean introducir en los nuevos bloques. Son los nodos quienes aceptan esas transacciones.

Una Dapp no tiene por qué estar completamente descentralizada, sino que puede combinar el uso de servidores, bases de datos y tecnologías centralizadas con Blockchain. Es más, esta práctica es muy utilizada ya, que el uso de Blockchain no solo es muy caro, sino que conlleva muchas limitaciones. Únicamente la información más esencial es la que se almacena en Blockchain.

2.1.6 COMUNICACIÓN CON UN BLOCKCHAIN

El front-end de las Dapps es donde los usuarios visualizan la información e interactúan con el Blockchain. Pero, ¿cómo se lleva a cabo la comunicación entre el usuario y el Blockchain? Lo primero que se necesita para poder interactuar con un Blockchain es una clave pública y una clave privada. Para almacenar estas claves se utilizan las wallets. Una wallet puede ser tanto un medio físico como digital para almacenar de forma segura dichas claves (las cuales son utilizadas para firmar digitalmente las transacciones que un usuario realiza y así poder autenticarle). Una wallet muy extendida y utilizada es MetaMask [37]. Ésta se encuentra en forma de extensión de navegador. Está disponible Google Chrome y Firefox en ordenador y solo para Firefox en móvil. MetaMask almacena las claves en el almacenamiento local del navegador. Una alternativa a MetaMask muy prometedora es Mist, un navegador que incluye toda la funcionalidad que MetaMask aporta por defecto [38][39].

Una vez que se tienen las claves es necesario crear la transacción que se desea incluir. Por supuesto en el formato correcto. Para que ni los usuarios ni los desarrolladores tuvieran que adentrarse en especificaciones y detalles de bajo nivel, nace la librería web3 desarrollada en JavaScript. Las comunicaciones con el Blockchain en entornos diferentes al web no son objeto de este trabajo.

La librería web3.js contiene la funcionalidad necesaria para poder crear transacciones que más tarde serán enviadas a través de HTTP a un nodo del Blockchain y así incluir la transacción en el siguiente bloque. Además, permite la llamada a funciones de tipo View mediante JSON/RPC para que los usuarios puedan realizar llamadas a funciones de forma gratuita [37].

Para poder utilizar la librería web3.js se requiere de un objeto web3, el cual se encuentra en el DOM del navegador. Este objeto contiene toda la funcionalidad necesaria. El objeto web3 variará según el Blockchain al que se desea enviar la transacción. Por ello para conseguir el objeto web3 correcto es necesario seleccionar el proveedor adecuado. Algo muy habitual es tener que elegir entre la red principal de Ethereum para el entorno de producción y un Blockchain (Ganache) local para el entorno de desarrollo. El objeto web3 con el que conectarse a la red principal de Ethereum usando MetaMask está siempre accesible, ya que MetaMask siempre lo inyecta en el DOM. El objeto web3 para conectarse al Blockchain local se necesitará indicar la dirección donde se encuentra este proveedor (típicamente es localhost ya que el Blockchain está corriendo en el propio equipo y el puerto 7545) [41]. Cabe destacar que, aunque el objeto web3 se encuentre en el cliente, éste no puede acceder a las claves privadas del usuario.

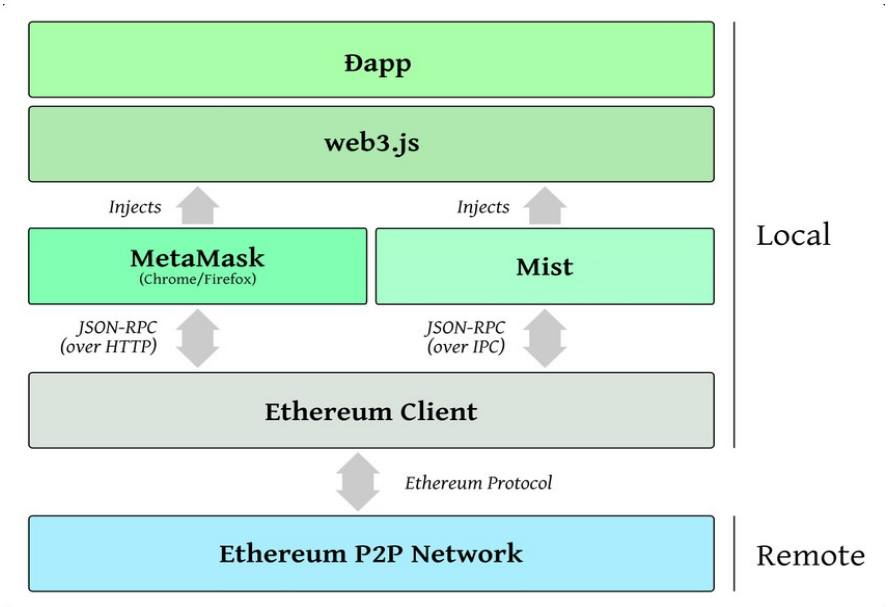


Figure 5 Diagrama de una Dapp [62]

2.1.7 GANACHE Y TRUFFLE

A la hora de crear una Dapp es posible generar un entorno de desarrollo adecuado evitando así utilizar la red principal de Ethereum para testear. Esto se consigue con Ganache, una herramienta que simula una red Ethereum de manera local. Esto permite probar y analizar el comportamiento de los Smart Contracts en los que se basará la Dapp.

Para interactuar con Ganache se puede utilizar tanto el objeto web3 que Ganache provee como MetaMask. MetaMask permite conectarse a otras redes que no sean la red principal de Ethereum, incluso redes locales. De esta forma, el objeto web3 sería el inyectado por MetaMask, pero se estaría conectando a Ganache. Si el lector alguna vez decide crear su propia Dapp, debe tener especial cuidado cuando se está conectado a Ganache mediante MetaMask ya que, si decide reiniciar Ganache, debe también notificárselo a MetaMask ya que si no MetaMask notificará errores debido a números de transacciones (nonces) repetidos. Esto se consigue fácilmente incrementando en uno el id de red en Ganache y reconectando MetaMask.

Para poder crear, desplegar y testear contratos en Ganache es necesario utilizar truffle.js (una librería que compila y migra contratos a Ganache). Además, proporciona una estructura de archivos correcta para la creación de la Dapp [41][42].

2.1.8 IDENTIFICACIÓN EN DAPPS

Una de las principales ventajas en cuanto a experiencia de usuario en las Dapps es que no es necesario autenticarse como en aplicaciones tradicionales. Como todas las transacciones están firmadas el proceso de autenticación concluye cuando el usuario posee sus claves. Sin embargo, cuando una Dapp no está completamente descentralizada sí es necesario autentificar al usuario para registrar las acciones que realiza fuera del Blockchain.

Si las acciones que realiza en los servicios centralizados son independientes de las que realiza en el Blockchain, no hay ningún problema. Se trata de realizar un proceso de autenticación como los habituales. Pero, ¿qué pasa cuando es necesario relacionar acciones del lado centralizado con el del descentralizado?

El proceso se complica. En el Blockchain a un usuario se le identifica por una "address" o dirección. Por lo que al usuario en la parte centralizada se le deberá identificar también por esa misma dirección. Por ello el usuario tiene que verificar que esa dirección le pertenece. Para ello basta con que el usuario firme (utilizando, por ejemplo, funcionalidad disponible en el objeto web3 suministrado por MetaMask) un mensaje cualquiera con su clave privada.

Una vez que el proceso de autenticación está definido se debe decidir si se permite que un mismo usuario utilice diferentes direcciones para interactuar con el Blockchain, o si a diferentes usuarios se les permite utilizar la misma dirección para interactuar con el Blockchain.

2.1.9 EXPERIENCIA DE USUARIO EN DAPPS

Es evidente que las Dapps proponen soluciones a problemas sin resolver y que bien utilizadas pueden suponer una gran ventaja para los usuarios, pero la realidad es que la experiencia de usuario en las Dapps hoy en día no está al nivel de las aplicaciones tradicionales centralizadas. Esto provoca que únicamente un 4,5% de todas las Dapps activas tengan más de 2000 usuarios diario, siendo el máximo número de usuarios diarios 6200, cifras ridículas comparadas con cualquier aplicación media [44].

Los motivos por los que las Dapps no están siendo aceptadas por los usuarios son que se requiere un conocimiento mínimo sobre la tecnología para poder utilizarla y no se puede esperar que el usuario medio realice tal inversión de tiempo. Analicemos el on-boarding de una Dapp para ver dónde radican estas dificultades:

1. La primera dificultad se encuentra cuando para poder utilizar la Dapp se requiere de una wallet. Supongamos que al usuario se le propone el uso de MetaMask, entonces debe descargarse la extensión web de MetaMask, registrarse en la misma y entrar en un ecosistema completamente nuevo, así como entender cómo se usa y para qué sirve.

2. Una vez que se posee la wallet y se comprende cómo funciona, es necesario adquirir Ether si se quiere realizar cualquier transacción que no sea gratuita. Siguiendo con la wallet MetaMask, ésta sugiere automáticamente el uso de Coinbase una empresa reconocida y autorizada para la compra y venta de criptomonedas. Tras el registro en su sistema, el envío y autorización de DNI y de una factura del lugar donde el usuario para validar el domicilio ya es posible comprar mediante métodos de pagos tradicionales como PayPal la criptomoneda deseada.

3. Es muy probable que la Dapp opere con un "utility token" de los que ya se ha hablado anteriormente. Por ello se debe acudir a un exchange de tokens para comprar los tokens y poder usar los servicios de la Dapp.

4. En Blockchains como Ethereum en los que existen comisiones los usuarios deberán pagar comisiones por cada acción que realicen. Y además esperar a que las transacciones se completen. La comparación con la instantaneidad de las aplicaciones tradicionales hace que queden en peor lugar.

Sin duda es un on-boarding muchísimo más complejo y extenso que a los que estamos acostumbrados.

2.1.5 FORTMATIC

Recientemente están naciendo nuevas startups con el objetivo de reducir y simplificar el on-boarding que antes analizábamos. Un ejemplo es Fortmatic. Esta startup ha creado una wallet en la que únicamente se requiere el número de teléfono del usuario. Cada vez que se quiere realizar una transacción, el usuario recibe un SMS y debe aceptar la misma. A cambio de mejorar la experiencia de usuario Fortmatic sacrifica descentralización, ya que es necesario almacenar en un servidor (con una seguridad especialmente alta) las claves públicas y privadas asociadas a

cada número de teléfono con el fin de que los usuarios no tengan que almacenarlas [45]. Este servicio también lo ofrece Microsoft quien te permite almacenar las claves privadas de los usuarios de la Dapp [46].

2.6 BITCOIN RSK

Bitcoin fue la primera implementación real de la tecnología Blockchain. Sin embargo, al ser la primera, las posibilidades de la misma están limitadas. Por ejemplo, no posee una máquina de Turing completa para albergar Smart Contracts. Además, Blockchain se caracteriza por ser inmutable por lo que es imposible cambiar el protocolo una vez ya está en marcha. Se puede crear otro nuevo, pero no tendría la misma reputación e importancia que tiene Bitcoin actualmente. Por ello se crea RSK, una plataforma P2P sobre Bitcoin que permite la ejecución de contratos inteligentes [48].

RSK, a diferencia de Bitcoin, sí posee un lenguaje de Turing completo y permite la ejecución de Smart Contracts. Esta plataforma ha aprendido de los errores de todas los pasados Blockchains para intentar dar una solución más completa.

¿Qué se quiere decir con que RSK se crea sobre Bitcoin? RSK es un ecosistema completamente independiente con su propia cadena de bloques, máquina virtual y criptomoneda (el RBTC) [49]. Sin embargo, existe una relación bidireccional entre la cadena de bloques de RSK y la de Bitcoin. Esto conlleva que un BTC pueda abandonar su ecosistema para entrar en RSK en forma de RBTC siendo el BTC destruido. Esto asegura que exista una relación de 1 a 1 entre BTC y RBTC. Los encargados de realizar este cambio son una "federación" compuesta por empresas que actúan a modo de notario para validar el buen funcionamiento [50]. Para evitar que se centralice el sistema la federación deberá componerse de muchos miembros y todos de ellos independientes.

Capítulo 3. ESTADO DE LA CUESTIÓN

El uso e incorporación de tecnologías como Blockchain para mejorar mercados existentes como el de las apuestas ya se ha realizado. Es más, las características de Blockchain la hacen una tecnología muy adecuada para este tipo de dinámicas. Como ya se ha mencionado, el mercado de las apuestas y en concreto el mercado online está creciendo y expandiéndose rápidamente. Al ser un mercado muy rentable, nuevos competidores están intentando hacerse un hueco en el mercado y, utilizar tecnologías descentralizadas como Blockchain, es un modo de diferenciarse de la competencia. Sin embargo, no hay que olvidar el pasado infame que tienen criptomonedas como Bitcoin en cuanto a especulación, por lo que cabe preguntarse si de verdad el uso de Blockchain supone beneficios y ventajas reales para empresas y usuarios o es solo un intento más de obtener beneficios.

La adopción de Blockchain por los mercados no es tarea fácil. Se requerirá mucho tiempo y esfuerzo para que los sectores sufran una transformación real y que los usuarios puedan extraer valor de los servicios. Sin embargo, el mercado de las apuestas online es distinto. Se trata de un mercado cuyos problemas y desventajas son fácilmente resolubles utilizando Blockchain.

Algunos de los problemas actuales que sufren las empresas relacionadas con las apuestas online y sus usuarios son:

- La confianza: Existe una opinión generalizada de que las apuestas son inherentemente injustas. La casa siempre juega con ventaja. Y en efecto, los casinos ganan mucho dinero debido a que los usuarios pierden más veces que ganan. Podría considerarse que el control total de las condiciones de juego y la falta de transparencia, benefician enormemente a las empresas de juego y los casinos. El usuario puede llegar a sospechar amaño o manipulación y sólo puede confiar en que actúen justamente. Utilizando Blockchain los usuarios no empezarán a ganar más dinero mágicamente, pero podrán confiar en las acciones de los casinos. Esto se debe a que toda entrada y salida de dinero

estará reflejada en el Blockchain de manera inmutable por lo que no hay cabida para el engaño y el amaño.

- Los depósitos y retiros de fondos habitualmente sufren retrasos que pueden llegar a durar días o semanas. Incrementado por las comisiones que cobran las entidades bancarias y pasarelas de pago por las transacciones. Blockchain aporta una solución independiente de bancos y entidades, con transacciones prácticamente inmediatas y comisiones muy bajas comparadas con las tradicionales. Además, las transferencias no dependen del lugar geográfico donde el usuario se encuentre ni de la entidad donde tenga el dinero. La independencia con entidades bancarias atraerá usuarios que antes no podían utilizar los métodos de pagos tradicionales.

- La censura es un problema inherente a los sistemas centralizados. Y en los casinos, cuando un usuario consigue grandes beneficios cobra vida. Los casinos sospechan de fraude cuando hay grandes beneficios y prohíben la entrada a usuarios como medida preventiva. Utilizando Blockchain el usuario puede cambiarse de dirección y seguir apostando desde ahí evitando cualquier tipo de baneo o censura.

- En muchos lugares del mundo como en Estados Unidos apostar está prohibido excepto en ciertas ubicaciones legalmente habilitadas. Esto hace que muchos usuarios no puedan apostar. Blockchain da solución a este problema. Surgen nuevos problemas como prohibir la entrada a menores, combatir la alegalidad e ilegalidad de Blockchain; ello supone un gran problema para países y organismos y aunque ya se están viendo algunos indicios de regulación, la ley está aún muy atrasada.

Blockchain, pues, soluciona muchos de los problemas actuales en las apuestas online. Por lo que los motivos por los que una empresa decide utilizar Blockchain no son solo especulativos y de diferenciación. Sin embargo, existe una diferencia entre la utilidad y el valor que una nueva tecnología aporta y la que los usuarios y empresas deciden realmente extraer. Blockchain es un claro ejemplo de tecnología cuyas implicaciones y ventajas aportan muchísimo valor a los

usuarios y, sin embargo, el coste en cuanto a experiencia de usuario y sencillez de uso puede llegar a ser tan grande que los usuarios no estén dispuestos a asumirlo. Un usuario perfectamente puede llegar a la conclusión de que prefiere utilizar un sistema centralizado y asumir las desventajas que este conlleva a cambio de inmediatez y sencillez en su uso. El objetivo a largo plazo es crear aplicaciones y servicios cuyos niveles de calidad puedan competir con los ya existentes para así forzar a la masa de usuarios a pivotar a este nuevo tipo de aplicaciones. Si tanto empresas como usuarios al usar Blockchain solo tienen algo que ganar es claro que acabaran utilizando lo que más les beneficie.

Existen varias soluciones diferentes ofrecidas por los competidores:

- Casas de apuestas y casinos tradicionales: Estos son los operadores habituales a los que estamos acostumbrados. Gigantes de las apuestas como Bwin, Bet365, PokerStars, 888Poker etc. operan en entornos centralizados.
- Casas de apuestas y casinos basados en Blockchain: Con la llegada de Blockchain han aparecida algunas iniciativas
- Prediction market: Plataformas donde los usuarios apuestan a eventos futuros donde el porcentaje de dinero apostado a un resultado de un evento concreto muestra la probabilidad con la que ese evento ocurrirá. Se trata de un sistema que premia el conocimiento en el que las apuestas no son a eventos completamente aleatorios (como que salga rojo en la ruleta), sino que habitualmente es posible predecir el resultado del mismo. Por ejemplo, ¿quién ganará las primarias en el partido demócrata? Los competidores principales que operan en este mercado son Augur, Stox y Gnosis [51]. Tres startups basados en Blockchain que no solo se llevan comisiones por cada apuesta, sino que extraen valor de las predicciones hechas por los usuarios y utilizan dicha información para obtener más beneficios.

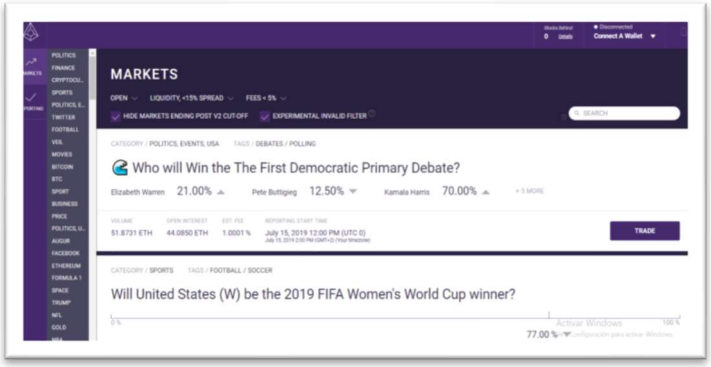


Figure 6 Página principal de Augur, especializada en mercados de predicción [51]

- Apuestas P2P (Peer-to-peer): Tradicionalmente las apuestas disponibles las deciden las casas y casinos. Los eventos, juegos, etc. a los que se puede apostar están delimitados y muy bien estudiados. Las probabilidades de los resultados de cada evento esta cuidadosamente elegidas para que la casa preferiblemente gane. El siguiente paso es que los propios usuarios puedan elegir a qué eventos y juegos apostar. Existen soluciones en las que los usuarios pueden decidir y publicar eventos y sus resultados correspondientes para que otros usuarios apuesten. Esto se conoce como apuestas P2P. Betterbetting es una startup basa en Blockchain que permite a los usuarios publicar apuestas. Decent.bet es otra startup nacida en marzo de 2019 que intenta integrar tanto apuestas P2P como prediction markets [52].

Existen muchísimos competidores y todos cuentan con gran financiación y apoyo debido al potencial que muestran estos mercados. Sin embargo, existe un tipo de apuesta que, por motivos que en breve trataremos, los grandes operadores no cubren: éstas son las apuestas P2P privadas y minoritarias. Las apuestas que las casas publican tienen que estar vinculadas a eventos públicos y fácilmente validables. Por ejemplo, la final de la Champions o el ganador del próximo gran premio de F1. Estos eventos se pueden hacer más específicos si se dispone de tecnología y recursos suficientes para validarlo. Por eso, algunas casas permiten apostar en cosas tan concretas como cuántos córners ha habido en un partido de futbol. Sin embargo, todo tiene un

límite. Apuestas como ¿quién ha ganado la partida de ajedrez entre un chico y su abuelo que juegan todos los jueves? son imposibles de validar. La casa de apuestas no tiene el alcance y la tecnología necesarias para poder saber quién ha sido el ganador. Para ello debería crear un entorno controlado para que el chico y su abuelo jueguen la partida y poder saber quién ha ganado. Esto se realiza para juegos como el póker donde la casa permite que jugadores jueguen a cambio de una comisión o rake. Además, debido al carácter minoritario de la apuesta, prácticamente nadie querría apostar por el chico o por el abuelo ya que no habría motivos para depositar confianza en ellos. Lo común sería que el abuelo apostase por sí mismo y el chico hiciera lo mismo. Cada uno con una probabilidad de ganar del 50%. Se trata de juegos de suma cero. Lo que alguien gana, alguien lo pierde. Todo esto hace que las casas de apuestas no puedan cubrir este tipo de apuestas.

Se ha razonado por qué una casa de apuestas no ve interesante cubrir eventos de este tipo. Pero ¿por qué querrían los propios apostadores que haya un intermediador, sabiendo además que ese intermediador posiblemente se lleve una comisión por sus servicios? ¿Por qué querrían el chico y el abuelo que haya un tercer integrante en su apuesta? La respuesta es confianza. O mejor dicho la ausencia de ella. Siempre hay un perdedor en los juegos de suma cero. Y a nadie le gusta perder. Por lo que es posible e incluso probable que, si los rivales son desconocidos entre sí, o no hay confianza entre ambos, el acuerdo al que previamente llegaron los participantes no se finalice. Es decir, el perdedor no pagará sus deudas al ganador. En el ejemplo mencionado, el chico está cansado de que su abuelo nunca le pague lo prometido cuando pierde, por lo que acude a un intermediario para que se asegure que la próxima vez que él gane su abuelo le pague. Tradicionalmente este intermediario en el que todos los participantes depositan confianza es una casa de apuestas debido a su reputación. Si el intermediario decidiese quedarse con el dinero y romper el contrato esto le supondría la pérdida total de reputación por lo que el castigo es tal que prácticamente siempre actuará de buena fe. Sabiendo de las posibilidades que ofrece Blockchain, es posible utilizar esta tecnología para crear una plataforma que suministre un servicio automatizado a todo aquel que quiera realizar y validar una apuesta tan arbitraria como se desee pese a no confiar en los demás participantes. Eliminando así intermediarios y asegurando que todas las partes cumplen lo acordado. Este tipo de apuesta es un tipo de apuesta

peer-to-peer, pero con la peculiaridad de que los eventos a los que es posible apostar no tienen límites.

El procedimiento para crear esta solución propuesta es crear un Smart Contract en un Blockchain como Ethereum que contenga la lógica de la apuesta, así como todos los términos del contrato. Una vez se conozca el resultado de la apuesta, el Smart Contract se ejecutará y repartirá los fondos según estaba programado y acordado con anterioridad. Hasta aquí la metodología es sencilla. El problema surge a la hora de determinar el resultado. Como los eventos que estamos tratando pueden ser tan privados y minoritarios como se desee, el problema a solucionar no consiste en un problema práctico sino de una limitación teórica intrínseca a las dinámicas que tratamos. Si nadie, aparte de los apostantes (en el ejemplo anterior el chico y el abuelo), ha estado presente durante el evento (la partida de ajedrez), la validación se complica. Dividamos los eventos en dos categorías:

- El primero se da cuando sí hay testigos que no sean apostadores. Estos testigos pueden ser quienes decanten la apuesta aportando información. Sin embargo, alguien solo se molestaría en ofrecer información si a cambio recibe una compensación. Aunque es preferible poder obtener esa información sin tener que compensar de forma económica o de cualquier otra forma, esta podría ser una forma de averiguar el ganador de la apuesta. Con el uso de Blockchain se consigue que el administrador del dinero total apostado no sea ninguno de los dos apostantes, ganando así fiabilidad en que la apuesta se va a ejecutar, sin embargo, si se quiere obtener información se necesita confiar en terceras personas (testigos).

- El segundo se da cuando no hay testigos. En este caso es imprescindible que los propios apostantes aporten pruebas de cualquier tipo que aseguren quién es el ganador de las apuestas. Estas pruebas serán juzgadas del mismo modo que en el primer caso salvo que la información proviene de los propios apostantes y no de los agentes externos. Esto conlleva un riesgo aún mayor. Existe un móvil claro para que los apostantes manipulen la información suministrada. Por ello de nuevo es necesario incentivar que los apostantes no manipulen las pruebas, castigarles cuando lo hacen e incentivar que los que juzguen esas pruebas actúen también de forma íntegra.

Independientemente de si hay o no testigos, es necesaria información con la que determinar el resultado del evento. Y una vez que se tiene información, es necesario juzgarla. Por ejemplo, el apostante aporta una prueba visual que determina que él es el ganador del evento. Es necesario saber si realmente esa prueba determina lo que el apostante asegura. Para ello se requiere que alguien juzgue esas pruebas. Siempre y cuando sean suficientes, y se vean incentivados a decir la verdad, y perjudicados si mienten, y por supuesto no tengan ningún tipo de relación con los apostantes, de media, la información que aporten los testigos será fiable y cierta, haciendo posible así la validación. Siguiendo con el ejemplo anterior, si cincuenta personas viesen la prueba visual que ha aportado el apostante y todas y cada una de ellas deliberasen que en efecto la prueba determina quién es el ganador del evento, se podría dar por finalizada la apuesta y repartir los premios según corresponda. En los eventos donde no existan testigos, serán los propios apostadores quienes suministrarán las pruebas y en eventos en los que haya testigo tanto apostadores como testigos podrán suministrar pruebas. Las pruebas aportadas por los apostadores tendrán que ser tratadas con más cuidado debido al más que evidente móvil para engañar y falsificar.

Este método de validación se basa en el dilema del prisionero (teoría de juegos) donde los jueces están incentivados a decir la verdad porque, si su decisión no coincide con la de la mayoría, no obtendrán ninguna recompensa. Es más, se puede hacer que los jueces apuesten una cantidad de dinero simbólica en forma de fianza la cual perderían si su decisión no corresponde con la decisión de la mayoría. La realidad es una sola y si las pruebas son determinantes todos los jueces deberán delibrar siempre lo mismo. En caso opuesto se deberá castigar a los jueces fraudulentos.

En el Whitepaper de Ethereum se sugieren posibles aplicaciones de la tecnología Blockchain, en concreto del uso de su propia red [23]. Entre ellas, se sugiere usar Ethereum para obtener información fiable que provenga de una masa de usuarios. El ejemplo que se aporta en el propio paper es el siguiente: Una empresa desea averiguar cuál es el valor del dólar. No tiene ninguna forma de saberlo salvo preguntar a sus usuarios. Sin embargo, estos usuarios pueden o no estar en lo cierto. Sin embargo, si se cambia el enfoque y se otorga una bonificación económica al 50% de los usuarios que más se han acercado a la mediana, es decir, a los cuartiles

2 y 3, del 25% al 75%, se consigue que lo usuarios se esfuercen por dar una información fiable. Este es un caso típico de teoría de juegos. Los usuarios tienen que responder lo mismo que van a responder los otros usuarios. Y la mejor opción es dar la solución real o la más próxima a ella.

Estas formas de validación no solo funcionan para averiguar el resultado de una apuesta. Se podría utilizar para validad cualquier cosa susceptible de ello. Pongamos un ejemplo: Una empresa en plena campaña publicitaria contrata una avioneta de anuncios, la cual recorre las playas en hora punta y lleva colgando una pancarta. La avioneta debe pasar por un lugar a una hora determinada tal y como indica el servicio por el que la empresa ha pagado. Para poder comprobar que en efecto la avioneta ha hecho el servicio en el lugar y hora determinados es necesario que alguien lo compruebe. La empresa contratante puede enviar a alguien a comprobarlo, pero se podría usar un sistema de validación parecida a lo explicado anteriormente. Las personas que casual o intencionadamente se encuentren en el lugar y hora correctos pueden verificar que en efecto ha pasado la avioneta. Si suficiente gente aporta información es muy probable que la opinión mayoritaria sea la cierta. Las personas que aporten información son conscientes que si la opinión mayoritaria es distinta a la suya no conseguirán ninguna compensación, luego están motivados e incentivados para decir la verdad. Se deberán usar métodos de incentivo y castigo similares a los propuestos con anterioridad para evitar que los usuarios respondan aleatoriamente o simplemente intenten engañar. Sería interesante desarrollar una plataforma que valide eventos de carácter genérico como el ejemplo mencionado, aunque esto no es objeto de este trabajo y se deja como recomendación a cualquier lector interesado.

Con el sistema que se acaba de describir los usuarios podrán realizar apuestas tan arbitrarias como deseen siempre y cuando sean validables. El proceso de validación depende de una audiencia externa de jueces quienes se basarán en las pruebas aportadas por los propios apostantes o testigos del evento. Hasta ahora todas las soluciones existentes que se encuentran en el mercado solo cubren eventos que no requieren de pruebas, sino que al ser eventos públicos el resultado es conocido. Proof Of Toss es sin duda la iniciativa que más se asemeja al sistema propuesto en este proyecto [1]. En febrero de 2019 propuso una plataforma completamente

descentralizada cuya lógica se encuentra completamente en Smart Contracts del Blockchain RSK. Proof of Toss propone una solución tanto para usuarios como para casas de apuestas y operadores tradicionales. Estos podrán crear sus propios eventos, asignarlos probabilidad y determinar su resultado usando una audiencia de jueces. Proof Of Toss finalizó la ICO de su utility token "Toss" en marzo de 2019, recaudando más de un millón de dólares con solo marketing y la promesa de que crear la plataforma propuesta. Una curiosidad digna de mención es que Proof Of Toss en su White paper asegura poder dar un servicio de apuestas en tiempo real. Sin embargo, si se analizan los costes en gas de las funciones de su Smart Contract rápidamente se llega a la conclusión de que las comisiones asociadas son exageradamente altas. Apostar 1 euro a 2 Gwei (para que la transacción se valide en menos de 5 min) con un coste de 250,000 de gas (tal y como se indica en su whitepaper) conllevaría una comisión de 15 céntimos [33]. Un 15% de lo apostado. Es la comisión más alta en el mercado. Como ya hemos explicado, las aplicaciones de tiempo real y Blockchain tienen una relación especial y para poder usar Proof Of Toss en tiempo real de manera rentable habrá que apostar cantidades superiores a los 10 euros.

A pesar de existir muy poca competencia y que además la que existe aún no tiene soluciones desarrolladas íntegramente, desarrollar un sistema que permita toda clase de eventos es demasiado genérico y abarca muchas posibilidades. Por ello es necesario concretar más sobre qué tipo de eventos se quiere cubrir. Una industria que llama enormemente la atención por su crecimiento es la industria de los videojuegos, en particular la industria de las competiciones y torneos en videojuegos. Estos son los famosos E-sports. La capitalización de este sector el último año es de más de $100 Billones y se espera que siga en aumento [57]. Pese a ser un gran mercado, solo unas pocas empresas privilegiadas son las que reciben todos los ingresos. Sin duda uno de los mercados más monopolizados hoy en día. La empresa china Tencent Holdings es un ejemplo [53]. Tencent es propietaria de: Riot (League Of Legends), Epic Games (Fortnite), Supercell (Clash Of Clans), Activision Blizzard (World of Warcraft, Call Of Duty) y King of Glory (de los juegos más jugados en Asia y pese a ser desconocido en Europa supone el 50% de los beneficios de Tencent) [54].

Cuando los E-sports ganaron popularidad y tuvieron la suficiente expectación, las casas de apuestas empezaron a cubrir estos eventos, permitiendo apostar a competiciones y torneos de videojuegos [55][56][57]. Pese a ser reciente, esta industria es también una de las que más rápidamente está creciendo, llegando a alcanzar en 2018 más de $1 Billón de ingresos totales [58]. Las apuestas en E-sports son, sin duda, un mercado emergente y con mucho potencial.

Del mismo modo que tradicionalmente las casas de apuestas solo cubrían eventos públicos y mayoritarios y dejaban de lado los privados y minoritarios por ser muy difíciles de validar, ocurre lo mismo con los E-sports. Solo es posible apostar en las grandes competiciones, a los grandes eventos. Las apuestas entre particulares no se permiten. Siguiendo con el ejemplo del chico y el abuelo, supongamos que ahora en vez de querer jugar una partida de ajedrez se quiere jugar una partida a un videojuego. La dinámica es la misma. Se necesita de un sistema que recoja los términos y condiciones en un Smart Contract y más tarde una audiencia externa juzgue las pruebas que ambos jugadores han aportado, decidiendo quien es el ganador y repartiendo consecuentemente el dinero apostado. Reduciendo el número de eventos posibles a únicamente partidas en videojuegos se está aplicando una tecnología disruptiva como Blockchain a dos mercados en auge como son las apuestas y los videojuegos. De esta forma se simplifica suficiente el sistema como para que aporte valor real y se pueda definir con precisión.

Capítulo 4. DEFINICIÓN DEL TRABAJO

6. JUSTIFICACIÓN

El mercado de las apuestas está en pleno auge. Tecnologías como Blockchain están revolucionando el sector aportando ventajas tanto a empresas como a usuarios. Ofreciendo tanto los servicios tradicionales como los más recientes e innovadores, incluyendo apuestas P2P y mercados de predicción. Sin embargo, existe un nicho aún sin explotar: las apuestas a eventos de carácter privado y minoritario.

Por ello, en este proyecto se propone una plataforma que actúe de intermediaria entre dos o más apostantes asegurándose del cumplimiento de la apuesta, así como la validación del resultado. La validación se realizará mediante una audiencia externa de jueces (también usuarios) quienes valorarán las pruebas aportadas por los usuarios que estén participando en la apuesta y deliberarán quién es el ganador.

Como los eventos privados y minoritarios comprenden un concepto muy amplio, se propone que la plataforma esté especializada en juegos (preferiblemente videojuegos debido a su mayor facilidad de validación) de suma cero con un único ganador. Estos son, por ejemplo, todos los juegos en los que dos o varios jugadores juegan entre ellos y únicamente existe un ganador que se lleva todo el dinero apostado. Prácticamente todos los juegos son susceptibles de este tipo de apuestas.

A pesar de que se ha limitado los tipos de eventos permitidos a juegos de suma cero con único ganador la plataforma sigue mostrando una potencia de propósito general para validar cualquier tipo de evento, no solo en apuestas. Las aplicaciones de este sistema de validación son prácticamente infinitas. Cualquier cosa susceptible de validación se podría validar utilizando el concepto de audiencia externa de jueces incentivados y movidos por el dilema del prisionero. Queda para desarrollos futuros el ampliar y generalizar las funcionalidades de esta plataforma.

7. GAINATGAME

La plataforma a desarrollar se denominará Gainatgame. Esta plataforma deberá:

1. Ser una versión mínima viable (MVP) que cuente con las funcionalidades aquí descritas.
2. Contar con una interfaz que permita a los usuarios y a los jueces realizar todas las acciones que se acaban de describir, de un estilo atrayente y responsive, de modo que se pueda usar tanto en navegador como en teléfono móvil.
3. Contener toda la lógica de apuestas, validaciones, y movimientos de dinero en Smart Contracts. A nivel lógico el Smart Contract tiene que estar blindado. No puede haber ningún error o caso sin contemplar.
4. Ofrecer servicios en tiempo real.
5. El uso de la plataforma tiene que ser tanto viable en términos de uso y experiencia de usuario como en términos económicos. Prestando especial atención a las comisiones que cobra Blockchain a los usuarios y a los términos y condiciones de las apuestas para que éstas sean justas para todas las partes.
6. Permitir encontrar rivales adecuados con los que un usuario pueda apostar.
7. Permitir crear y configurar apuestas personalizadas.
8. Asegurarse que toda apuesta es segura y fiable y que el dinero quede intacto hasta que la apuesta finalice.
9. Ser capaz de determinar el resultado de la apuesta una vez que esta finalice mediante el uso de una audiencia externa de jueces elegidos al azar entre un pool de jueces y, dependiendo del veredicto, repartir el dinero consecuentemente entre los participantes. Los jueces, a cambio de su aportación, recibirán una compensación que saldrá de los participantes.
10. Los usuarios podrán, una vez acabada una partida, aportar las pruebas que más tarde los jueces utilizarán para determinar quién es el ganador.
11. Permitir apuestas en la criptomoneda del Blockchain o en un utility token creado específicamente para ello.
12. Se deberá asegurar la honestidad y buena fe de usuarios y jueces mediante un sistema de incentivos y castigos que esté diseñado a prueba de trampas.

13. Contar con una seguridad de calidad excepcional ya que las actividades a realizar conllevan movimientos de dinero. Blockchain es esencial para lograrlo.

Para lograr estos objetivos el sistema no tiene por qué ser completamente descentralizado. Es más, debido a que los usuarios requieren subir pruebas que típicamente serán fotos o videos, se requiere de un servidor donde subirlos. Blockchain se trata de una tecnología de bajo nivel donde la información a guardar debe ser lo mínima posible. Subir megabytes de video a Blockchain es impensable.

8. ¿QUÉ APORTA BLOCKCHAIN A GAINATGAME?

Una de las cualidades más importantes del sistema a desarrollar es la seguridad. Tradicionalmente sería inviable que, sin contar con grandes presupuestos y equipos de desarrollo, se pueda alcanzar una seguridad competente y fiable que asegure que el dinero de los usuarios está a salvo. Sin embargo, gracias a Blockchain esta seguridad se puede "subcontratar". Toda la lógica se deberá encontrar en el Blockchain de forma que si el Smart Contract está bien programado y la lógica reflejada es correcta la seguridad de los fondos ya dependerá únicamente del Blockchain.

La confianza es crucial, no solo entre usuarios, sino entre usuarios y el sistema. La lógica plasmada en un Smart Contract es inmutable. Es decir, los términos y condiciones que los usuarios aceptan al realizar la apuesta son intocables, por lo que se puede predecir el resultado y no hay que confiar en ninguna parte. La autoridad central no existe, por lo que el dueño del Smart Contract no tiene ningún poder sobre los fondos depositados en él.

Los usuarios manejan su propio dinero. El dinero ya no lo controla la autoridad central, quien en sistemas centralizados tiene todo el poder. Ahora, los usuarios manejan su propio dinero (criptomonedas y tokens) y pueden moverlo e intercambiarlo como deseen.

Al realizar todas las apuestas en un token que se puede controlar, los beneficios no vienen solo de las comisiones que se cobra como intermediador, sino del propio incremento del valor del token. Éste iría a la par con el valor de la empresa.

9. ¿ES LEGAL?

Es imprescindible que todos los juegos o eventos en los que se vaya a realizar apuestas sean juegos de habilidad. Legalmente no es equivalente apostar a los dados que a una partida de ajedrez. Las apuestas en juegos de azar están muy reguladas y requieren licencias muy caras mientras que los jugos de habilidad no [61]. Existen antecedentes. Skillz es una empresa con sede en San Francisco que realiza torneos on-line en videojuegos [60]. Skillz permite a empresas desarrolladoras de videojuegos implantar un sistema para que en el propio juego haya una sección de competición con apuestas económicas. Esta actividad es lícita únicamente debido a que se trata de juegos de habilidad. En Estados Unidos la normativa en contra de las apuestas es muy estricta. El juego es legal en los de estados de Nevada, Nueva Jersey y Nueva York [61]. Otro ejemplo sería de nuevo una empresa estadounidense: Player´s Lounge [59]. Nacida en 2015, su actividad principal es muy similar a la detallada en este proyecto. Permiten a personas desconocidas entre si realizar apuestas en videojuegos. Aunque de forma centralizada. Su método de validación es mediante pruebas visuales juzgadas por la propia administración y no por los propios usuarios en forma de jueces. Aunque parece prometedora, aún no cuenta con apenas tráfico en su plataforma.

10. ÉTICA Y JUEGO RESPONSABLE

El propósito de la plataforma no es solo permitir las apuestas con dinero real, sino también ofrecer un entorno para la competición. Por ello, la plataforma deberá contar con un modo en el que los usuarios puedan jugar y apostar con monedas virtuales sin valor, de manera simbólica. Existen muchas formas de motivar, y por supuesto el dinero real no es la única. Sistema de puntuación como el ELO, rankings, y todo tipo de gamificación son una parte fundamental de las aplicaciones y de sistemas de hoy en día. La plataforma contará con un tamaño máximo de apuesta. De esta forma, se evitan inconvenientes y enfrentamientos entre los usuarios.

11. MODELO DE NEGOCIO

El modelo de negocio de esta plataforma es el siguiente: se trata de un modelo B2C (Bussiness to Consumer). A cambio de suministrar un servicio de intermediación entre usuarios, la plataforma cobrará una comisión. Esta comisión podrá variar dependiendo del tipo de partida o evento, del número de jugadores, etc. Se puede incluso llegar a no cobrar comisiones en ciertos casos. Se realizará en secciones futuras un análisis detallado de las comisiones y de la rentabilidad de la plataforma.

Uno de los procedimientos más habituales en la actualidad realizados por startups para conseguir financiación son las ITO [1]. Ya no es necesario vender porcentaje de una empresa a inversores para adquirir financiación. Solo se necesita crear un token y sacarlo al mercado. Si se realiza una buena campaña de marketing no es necesario tener aún el producto o servicio que se esté anunciando. Con Gainatgame ocurre igual. Sería posible crear un token (GaG token) el cual serviría para apostar. En vez de utilizar criptomonedas se usaría el token. Todos los usuarios deberán comprar el token si quieren utilizar cualquiera de los servicios de la plataforma.

Es clave que la empresa al realizar la ITO se reserve un porcentaje de los tokens para sí misma. De forma que, cuando se haya concluido la ITO, la empresa haya ganado popularidad y la demanda de tokens haya subido, la empresa pueda conseguir beneficios gracias a la revalorización de los tokens que se reservó para sí misma. Además, es común que la comisión a cobrar a los usuarios sea en tokens, por lo que cuanto más suba el precio del token, más beneficios obtendrá la empresa.

La diferencia entre usar tokens o cripto monedas y dinero real radica no solo en el aspecto legal, sino en el simbólico. Parece que, de media, las personas son mucho más reticentes al gasto en internet que en la vida real. Esto se apreció cuando WhatsApp intentó poner su aplicación de pago. A pesar de que el precio era de tan solo 1 euro, muchísima gente se indignó y estaba dispuesta a cambiarse a Telegram. Teniendo en cuenta que el gasto de 1 euro en metálico no supone un gran problema para la gran mayoría de la población este hecho es muy interesante. Esta diferencia quizás radique en la desconfianza e inseguridad a la hora de ingresar dinero en las plataformas y en el desconocimiento tecnológico de lo que realmente está ocurriendo. Sin embargo, es una realidad. Es por ello que cuando en vez de utilizar la palabra euros se usa el

concepto moneda virtual, psicológicamente, esta actividad parece causar menos presión. Parece no ser lo mismo perder 1 euro que perder 1 moneda virtual, aunque su valor sea idéntico. Además, se pueden generar tantas monedas virtuales como se desee siempre que se esté dispuesto a soportar las consecuencias de la inflación.

12. CICLO DE USO

Muchas de las casas de apuestas que operan en la actualidad dan bonos de bienvenida y en ocasiones hasta permiten realizar las primeras acciones sin necesidad de pagar ni de ingresar dinero. Esto se debe a que existe la creencia de que eventualmente estos usuarios generarán más beneficios a la casa de los que ésta le dio con intención de introducirle y fidelizarle a sus servicios.

En esta plataforma se puede hacer lo mismo. Especialmente, si se trata de tokens. Cuando un nuevo usuario ingrese se le pueden otorgar los tokens necesarios para que realice la primera apuesta o juegue su primera partida. Sin embargo, existe una posibilidad mucho más interesante. Dado que en la plataforma que se quiere desarrollar los usuarios juegan dos papeles importantes (apuestan y juzgan), es posible diseñar la plataforma de modo que un nuevo usuario que ingrese por primera vez pueda juzgar partidas hasta que haya conseguido lo suficiente como para jugar su primera partida como apostante. Con ello se crea así un ciclo en el que todo usuario pueda probar la aplicación sin necesidad de ingresar dinero y sin necesidad de que la propia aplicación le ofrezca ofertas o descuentos.

El procedimiento mediante el cual un juez juzga una prueba visual y recibe una compensación económica está arraigado y se lleva realizando mucho tiempo en páginas webs y juegos En éstas, a cambio de ver anuncios ofrecían dinero o monedas virtuales del propio juego, todos ganan. La empresa anunciante gana, ya que su publicidad está siendo visualizada. La empresa que permite a sus usuarios visualizar publicidad gana, ya que cobra a las demás empresas anunciantes por publicar sus anuncios. Y los usuarios ganan, ya que pueden transformar su tiempo en dinero real. En este ejemplo se vuelve apreciar de nuevo cómo el concepto de moneda virtual es muy potente. Un usuario puede estar dispuesto a invertir cierto tiempo viendo anuncios a cambio de 1 euro, pero puede no estar dispuesto a ingresarlo él mismo.

Capítulo 5. GAINATGAME

Gainatgame es el sistema propuesto y desarrollado a lo largo de este trabajo. En este capítulo se expondrá el análisis realizado a la hora de elegir qué tecnologías utilizar en el desarrollo que mejor satisfacen los objetivos y funcionalidad a conseguir. Finalmente se analizará Gainatgame como plataforma totalmente construida y ya desarrollada, su funcionamiento, y sus características.

Con el fin de desarrollar una plataforma de la mayor calidad posible, se deben elegir las tecnologías que más se adapten a los objetivos descritos en el capítulo anterior. Se quiere construir una plataforma de fácil acceso, con apariencia profesional, mucha funcionalidad y que se pueda conectar con Blockchain Se deben elegir:

- Front-end
- Back-end
- Blockchain

13. FRONT-END

Lo primero que se debe decidir es si será una plataforma basada en web, una aplicación para móvil o ambas. Los videojuegos, por lo general, se juegan en ordenador y en consola. Mientras que para los de ordenador una plataforma web sería lo deseable, para los de consola, tener el ordenador abierto supondría una molestia, por lo que una plataforma móvil sería preferible. Una plataforma web responsive que se pueda utilizar en móvil satisface ambos casos, siendo posible en un futuro, si es necesario, crear una aplicación de móvil.

La mayoría de Dapps desarrolladas están basadas en Web ya que Web3.js y MetaMask aportan mucha funcionalidad y simplifican el trabajo. Algunas iniciativas crean sus propias wallets y servicios de comunicación con Blockchain, sin embargo, ésto no es alcanzable sin un equipo de

desarrollo y financiación. En ocasiones algunas empresas incluso crean sus propios Blockchains privados.

Para realizar el front-end de la aplicación que es con lo que van a interactuar los usuarios existen diversas opciones. Por supuesto opciones como WordPress y otros CMS no son viables ya que queremos un nivel de especificación y de detalle que no se puede conseguir con módulos prediseñados. Por lo que es necesario realizar la interfaz en HTML. En concreto se usa HTML5 con CSS3 y JavaScript para añadirle funcionalidad y dinamismo. Crear una interfaz de nivel profesional desde cero no es tarea recomendable. Como se suele decir, no es necesario reinventar la rueda. Por ello es común el uso de plugins y de templates. En nuestro caso una plantilla HTML5 únicamente con Bootstrap para conseguir que sea responsive es una opción bastante adecuada. Para cualquier funcionalidad extra se recurre a plugins específicos de JQuery. JQuery es una librería programada en JavaScript con una gran variedad de funcionalidades que sirve de gran ayuda en desarrollos web y que es bastante popular hoy en día. En startups o en empresas consolidadas sería muy común contratar a un diseñador gráfico, experto en front-end para diseñar toda la interfaz y no requerir de templates, sin embargo, existen limitaciones en el proyecto y debemos ser conscientes de las mismas y actuar en consecuencia.

Otra posibilidad muy popular en la actualidad es desarrollar una SPA (Single page application o aplicación de una sola página). Mediante el uso de Frameworks como React o Angular, basados en MVC (Model View Controller) lo cual permite separar la lógica, datos e interfaz, se pueden desarrollar aplicaciones cuya actividad transcurra siempre en la misma página, sin navegación. Esto se logra usando el concepto de módulo. En vez de navegar a una dirección para acceder a una funcionalidad, se despliega un módulo. Angular, por ejemplo, está desarrollado en TypeScript, un superconjunto de JavaScript, aportando tipado estático y objetos basados en clases.

5.1.1. Angular 8

Angular se trata de un framework desarrollado por Google el cual está desarrollado en un TypeScript (un superconjunto de JavaScript, es decir, un lenguaje que mejora y amplía JavaScript

añadiendo más funcionalidades y características). Angular aporta características muy ventajosas que encajan perfectamente con los requisitos de nuestra plataforma.

Esta plataforma tiene que poder suministrar servicios en tiempo real. La manera tradicional es utilizando JQuery y Ajax. Ajax permite realizar peticiones http de forma asíncrona sin que el usuario cambie de página. Sin embargo, aunque es efectivo, este método cuando se tiene mucha funcionalidad es poco escalable y muy difícil de mantener. Gracias a angular, y a los conceptos de detección de cambio en variables, servicios, proveedores y sockets, se puede lograr la misma funcionalidad de una forma mucho más limpia, escalable, y con división de código.

5.1.1.1 Detección de cambio en variables

En angular cuando una variable que se está mostrando por pantalla cambia de valor, está se actualiza automáticamente por pantalla. La manera tradicional de conseguir esto es o recargando la página o mediante JavaScript manual. Sin duda es una funcionalidad muy útil si el estado de una variable está continuamente cambiando, sobre todo, si ese estado se obtiene del servidor mediante http.

5.1.1.2 Servicios y proveedores

Los servicios son clases que permiten compartir información y funcionalidad entre clases que normalmente no podrían. Cuando un servicio es una clase de tipo Singleton, es decir, solo existe una instancia de la misma, ese servicio actúa como proveedor. Esta cualidad es especialmente útil cuando tienen variables que se quiere que sean accesibles desde muchos puntos de la aplicación. Por ejemplo, en el caso de nuestra plataforma, el estado de la apuesta. Se puede crear un servicio encargado de informar del estado de la apuesta del que todas las demás clases obtengan la información.

5.1.1.3 Sockets

La comunicación en http es unidireccional. El cliente envía una petición y el servidor responde. Existen casos concretos donde se necesita que el servidor contacte con el cliente sin que este le contacte previamente. Por ejemplo, en nuestra aplicación se necesita suministrar un servicio de búsqueda de rivales en el que a un usuario se le encuentra otro usuario con los mismos intereses.

Cuando un usuario comunica a la plataforma que quiere empezar a buscar a otro usuario, ¿cómo puede saber el usuario que está buscando cuándo se ha encontrado a otro usuario? Un procedimiento habitual y sin duda válido es que el usuario que está buscando envíe peticiones cada cierto tiempo al servidor preguntando: ¿se ha encontrado rival para mí? Sin embargo, este método es muy poco eficiente ya que el cliente no solo está constantemente mandando peticiones, sino que, entre peticiones, el servidor no puede contactar por lo que es posible que haya encontrado un rival y el usuario no lo sepa hasta que realice otra petición. Una solución a este problema es el uso de sockets (canales de comunicación bidireccional). Con ellos, tanto el usuario como el servidor, pueden enviar mensajes en todo momento. Angular cuenta con la librería socket.io la cual permite crear sockets entre cliente y servidor de forma sencilla y rápida. Esto es perfecto para los objetivos de la plataforma.

5.1.1.4 Ventajas

Una cualidad muy ventajosa de Angular es la de división de código. En ocasiones es necesario reutilizar código en muchas partes diferentes de la aplicación, por ejemplo, la cabecera. La cabecera va a estar presente en todas las páginas de la aplicación y es necesario que este código sea fácilmente modificable. De forma que si hay que realizar un cambio no haya que ir una por una cambiando todas las páginas. Para conseguir esto, habitualmente se utilizan lenguajes de servidor como PHP o JAVA los cuales ayudan de forma dinámica a incluir código de un archivo en otro archivo. Esto, pese a ser muy útil, tiene una desventaja: el desarrollo del cliente está unido al del servidor, son interdependientes. Si el desarrollador sabe de ambos entornos, tanto cliente como servidor, no hay problema. Sin embargo, normalmente, en las empresas grandes, existen expertos del entorno visual en cliente que no dominan el del servidor y viceversa. Angular aporta la funcionalidad necesaria como para solucionar estos problemas estando completamente en el lado del cliente y sin depender de la tecnología back-end que se utilice. Angular divide el código en componentes los cuales se despliegan en cualquier lugar de la web. De esta forma se puede reutilizar código y conseguir una estructura muy organizada y limpia.

Otra funcionalidad interesante es la de direccionamiento. Angular permite direccionamiento con urls parametrizados del tipo www.gainatgame.com/profile/100 donde se está accediendo al perfil del usuario con id 100. Este tipo de direccionamiento se consigue normalmente

configurando correctamente el servidor. En PHP, por ejemplo, se consigue configurando el .htaccess. Sin embargo, como Angular está completamente desvinculado del lado del servidor, el redireccionamiento se puede realizar en cliente. Esto aporta mayor flexibilidad y sencillez.

Angular al estar escrito en TypeScript (JavaScript a efectos prácticos) es muy compatible con lenguajes de servidor como Node.js. Node.js es un lenguaje de servidor análogo a JavaScript en el lado del cliente. Se creó con la intención que desarrolladores de JavaScript en cliente no tuvieran que cambiar de lenguaje cuando desarrollaban la parte de servidor. La mayoría de Dapps operativas actualmente utilizan tanto Angular como React (competidor directo de angular administrado por Facebook) con Node.js con el fin de poder usar Web3.js tanto en cliente como en servidor.

14. BACK-END

Para la tecnología a utilizar en el lado de servidor tenemos que tener en cuenta una serie de requisitos que nuestra plataforma debe satisfacer:

- Seguridad: dentro de los servicios que ofrecemos está la fiabilidad, confianza, manejo de dinero etc. Se requiere un nivel alto de seguridad.

- Tiempo real: En la plataforma estarán simultáneamente multitud de usuarios buscando rivales con los que apostar e interactuando entre sí. Tiene que ser capaz de soportar múltiples conexiones simultáneas. Por lo general la mayoría de los lenguajes y servidores son capaces de satisfacer este requisito, pero sin duda existen algunos mejores que otros.

- Uso eficiente de sockets. Para una conexión bidireccional.

- Librerías Web3 para conectarse con Blockchain.

- Comunidad de desarrolladores: Se trata de un proyecto bastante ambicioso. Se necesita una comunidad que facilite la solución de errores y minimice el tiempo de desarrollo.

- Lenguaje robusto: Para este tipo de proyectos es esencial que el lenguaje sea robusto, no tenga fallos, este actualizado, proporcione mucha funcionalidad etc.

- Hostings: Es necesario poder encontrar un hosting de precio asequible donde desplegar la aplicación.

Lenguajes como JAVA EE, PHP 7 con su versión orientada a objetos y NODE.JS se consideraron como posibles candidatos.

5.1.2 JAVA EE

En la actualidad el lenguaje preferido por los bancos debido a su robustez y la seguridad que aporta. Tiene una comunidad de desarrolladores muy amplia, quizás la más grande de todos, lenguaje de tipado fuerte. Aunque sea menos común estos días seguramente el tipado fuerte ofrece robustez que el tipado débil no puede superar. Esto implica sacrificar, por supuesto, sencillez en el lenguaje y en el código. Se trata de un lenguaje compilado y por ello los hostings que permiten instancias de Java son más limitados que los de PHP. Posiblemente la única opción rentable sean los servidores de Amazon que ofrecen un año de uso gratis. Cuenta con una librería Web3 para Blockchain y una librería socket.io para sockets, lo cual facilita ambas tareas. Angular también dispone de un sistema de actualización de variables. Si el valor de una variable cambia en tiempo de ejecución, no es necesario volverla a imprimir por pantalla, angular lo realiza automáticamente. Este servicio es de suma importancia ya que usar otros métodos como JQuery para actualizar las variables es mucho más rudimentario.

5.1.3 PHP 7

En sus últimas versiones es orientado a objetos. Tiene una gran comunidad desarrolladores, sin embargo, no goza de buena reputación entre ellos. Las anteriores versiones de PHP en nivel de robustez y limpieza de código distaban mucho de lo que ofrecen otros lenguajes. En las nuevas versiones se ha reinventado y se ha convertido en una más que razonable alternativa. Cuenta con librerías de web 3 para Blockchain y elephant.io para sockets, aunque menos estables y

consolidadas. Sin duda está por detrás de Java y Node en este aspecto. La seguridad radica en el uso del mismo. Buenos hábitos y estilos de programación son lo que hacen a la plataforma segura. Es posible usar PHP junto con alguno de sus bien conocidos frameworks como Laravel o Symphony.

Estos frameworks no solo cuentan con un enfoque MVC (Model View Controller) sino que facilitan mucho el desarrollo comparado con PHP nativo y aportan robustez y seguridad. Es posible realizar un buen proyecto sin usar ningún plugin o framework adicional sin embargo no es recomendable.

5.1.4 NODE JS

Se trata de un lenguaje orientado a eventos. Es muy similar a JavaScript y se creó para que los desarrolladores no tuvieran que cambiar de lenguaje entre Front-end y Back-end, homogeneizando así el desarrollo. Es muy popular en la actualidad. Es un entorno en tiempo de ejecución. Estas características lo hacen un lenguaje altamente escalable. Es el lenguaje por excelencia si se quiere conectar con Blockchain mediante librerías Web3. También es muy eficiente en la manipulación de sockets gracias a Socket.io. Presenta mayor dificultad para encontrar un hosting que para un servidor PHP. Igual que con JAVA EE, Amazon es la opción más indicada.

Tras este análisis de las opciones en tecnologías de back-end se ha decidido implementar la plataforma con un servidor de Node para el uso de sockets y para la conexión con el Blockchain y un servidor de PHP 7 para toda la funcionalidad. Actuando como una Api-Rest a la que, tanto el servidor de Node como el cliente, podrán llamar.

15. BLOCKCHAIN

Para la elección de Blockchain existen las siguientes alternativas:

- Ethereum
- Alastria

- Hyperledger
- Bitcoin RSK
- EOS

ICAI Comillas cuenta con un nodo de Alastria lo que podría ser de gran ayuda. Sin embargo, no se puede obviar la gran predominancia de Ethereum hoy en día. La mayoría de los proyectos e iniciativas profesionales, así como las ICOs e ITOs se están desarrollando en Ethereum. Por lo que Ethereum es probablemente la mejor opción. En un futuro si Bitcoin RSK gana popularidad y se consolida se podría barajar el cambiar de Blockchain. Tanto RSK como Ethereum son compatibles en cuanto a Smart Contracts. Ambos permiten Solidity como lenguaje, por lo que el cambio de uno a otro sería parecido a cambiarse de hosting de páginas web. Otros cambios como de Ethereum a Hyperledger serían más complicados. Recientemente, muchas startups están optando por Hyperledger ya que ésta es la mejor opción para empresas que necesitan un mínimo de privacidad y confidencialidad.

Figure 7 Logo de Ethereum

16. DIAGRAMA DE TECNOLOGÍAS

Finalmente se concluye que realizar una SPA para el front-end, usando el framework Angular, HTML5, CSS3, Templates y Plugins y para el back-end, Ethereum, Node, librerías Web3, MySQL como gestor de base de datos y PHP como Rest-api es la mejor opción para conseguir las funcionalidades y características propuestas.

17. DESARROLLO

Se divide el desarrollo en dos partes: Apostar y jugar y validación de partida. Un usuario que ingrese a la plataforma podrá elegir entre dos opciones, buscar partida con el fin de apostar o validar una partida actuando como juez. Para analizar ambas opciones se expondrá el diagrama de flujo que seguirá el usuario en la plataforma. Cabe destacar la diferencia entre las acciones de un usuario cuando este necesita que su partida previamente jugada se valide y las acciones de un usuario que está validando partidas de otros usuarios como juez.

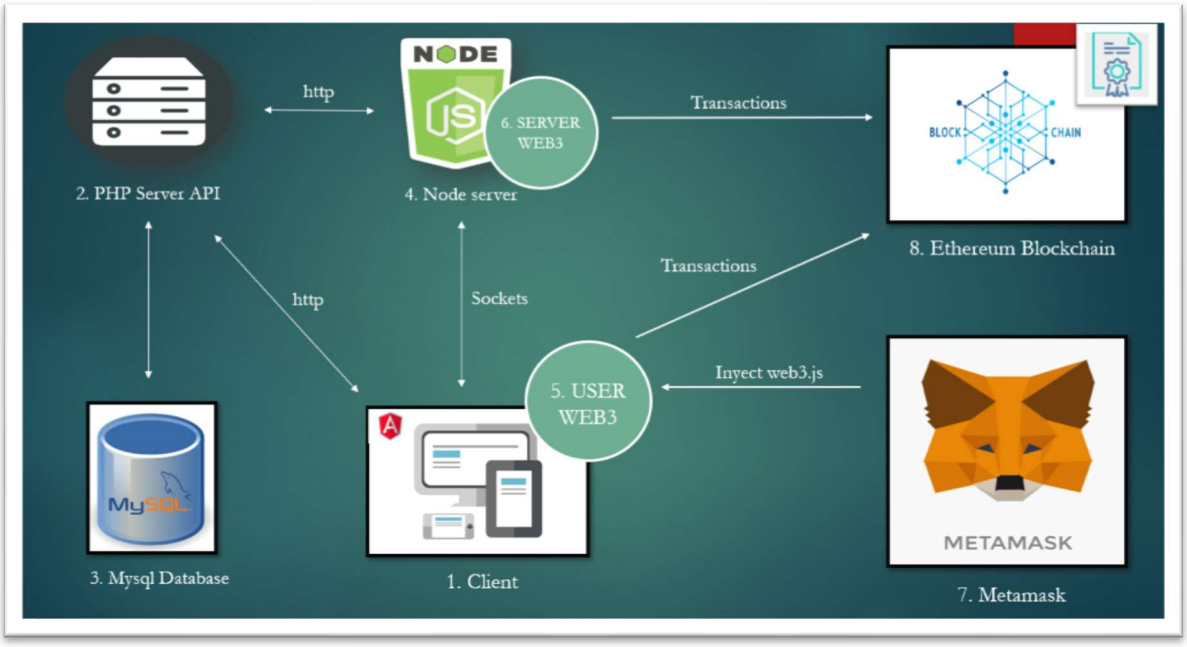


Figure 8 Diagrama de tecnologías de gainatgame

5.1.5 APOSTAR Y JUGAR PARTIDA

El principal servicio que Gainatgame aporta a los usuarios es la posibilidad de apostar y jugar en una partida con otros jugadores. El flujo de acciones (detalladas en los puntos siguientes) que un usuario debe seguir para poder completar una apuesta satisfactoriamente en es el siguiente:

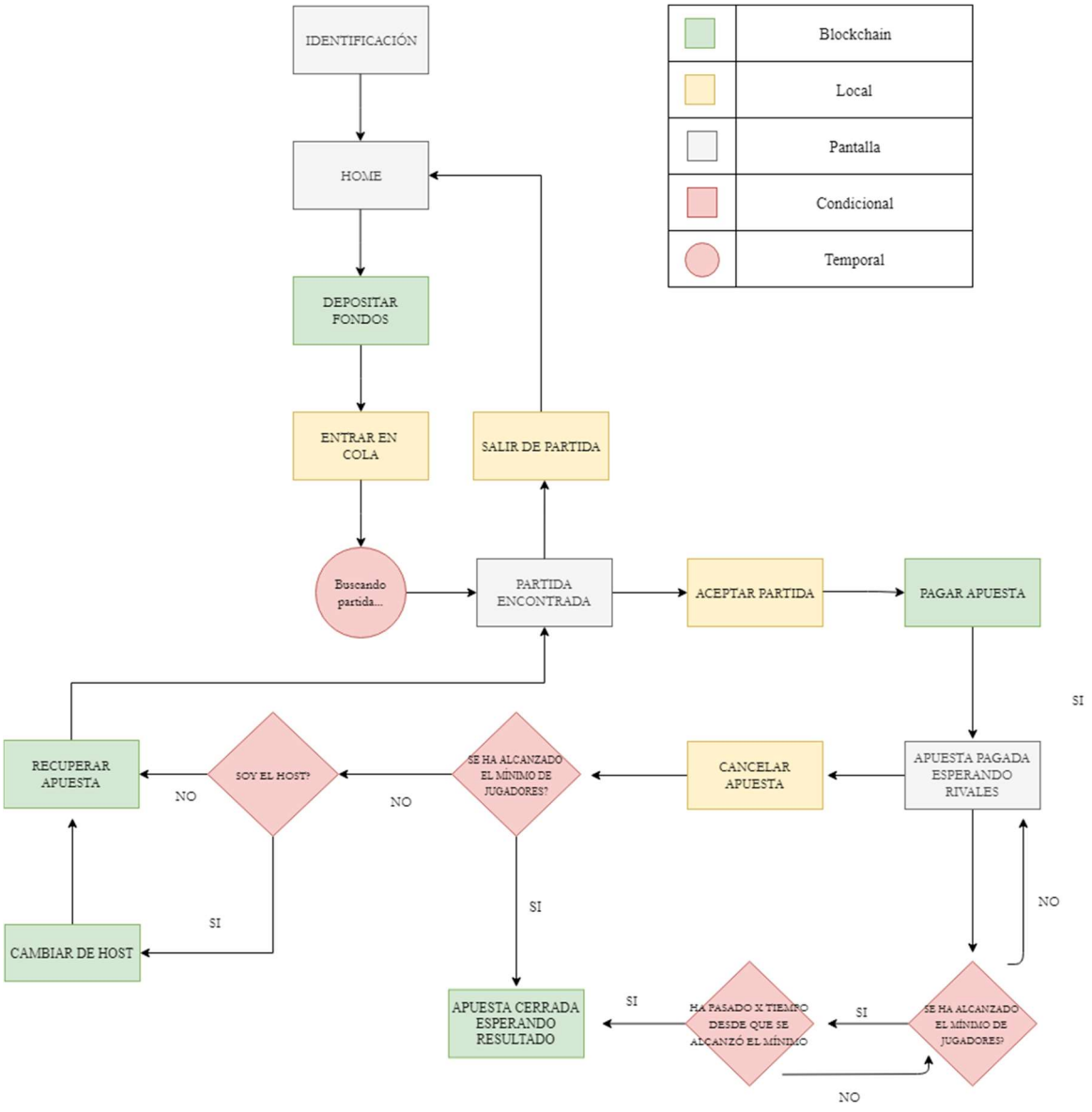


Figure 9 Flujo de acciones de un usuario para comenzar partida

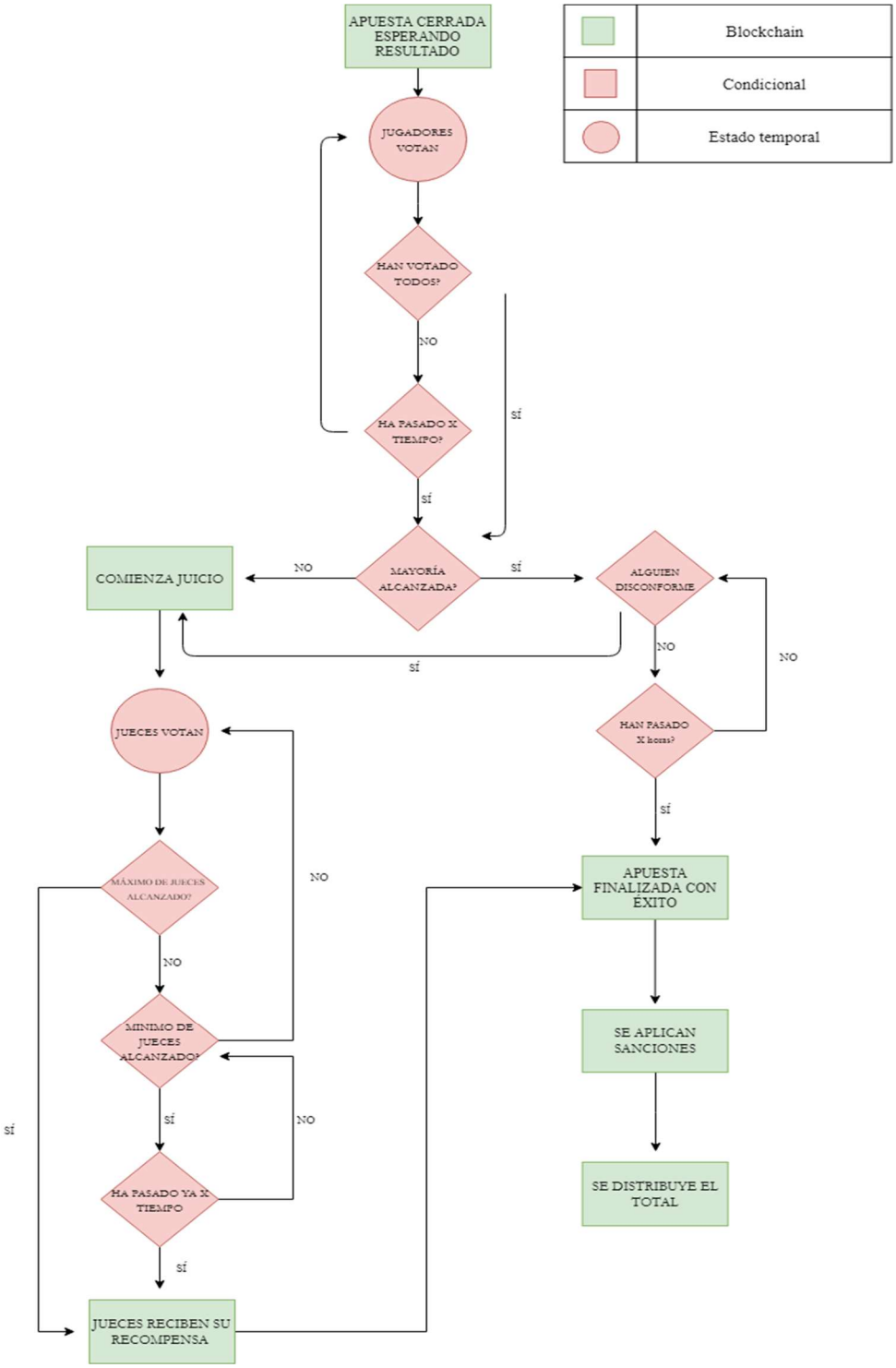


Figure 10 Flujo de acciones de usuarios para que su partida se valide

5.1.5.1 Identificación de los jugadores

Toda aplicación debe tener un log in para poder identificar a los usuarios. En Gainatgame, al tener parte centralizada y parte descentralizada, este proceso es más complejo. Habitualmente, en entornos centralizados, se identifica a los usuarios con un id el cual se incluye en una fila en la base de datos cuando este se registra. El problema viene a la hora de relacionar acciones locales con acciones en el Blockchain. Por ello se opta por asociar cada usuario con una dirección pública del Blockchain de forma unívoca. Es decir, un usuario solo podrá tener una dirección asociada, y una dirección solo podrá estar asociada a un usuario. Como veremos más adelante, la dirección será necesaria en funcionalidad a nivel local. Para asegurarse que un usuario no miente a la hora de asociarse con una dirección es necesario un proceso de autenticación. Para autenticarse, el usuario deberá firmar un mensaje con su clave privada que el servidor autenticará. Se trata de una firma digital común. Una vez se haya asegurado que ese usuario dispone de esa dirección, se les relacionara mediante un campo en base de datos. La tabla resultante es la siguiente:

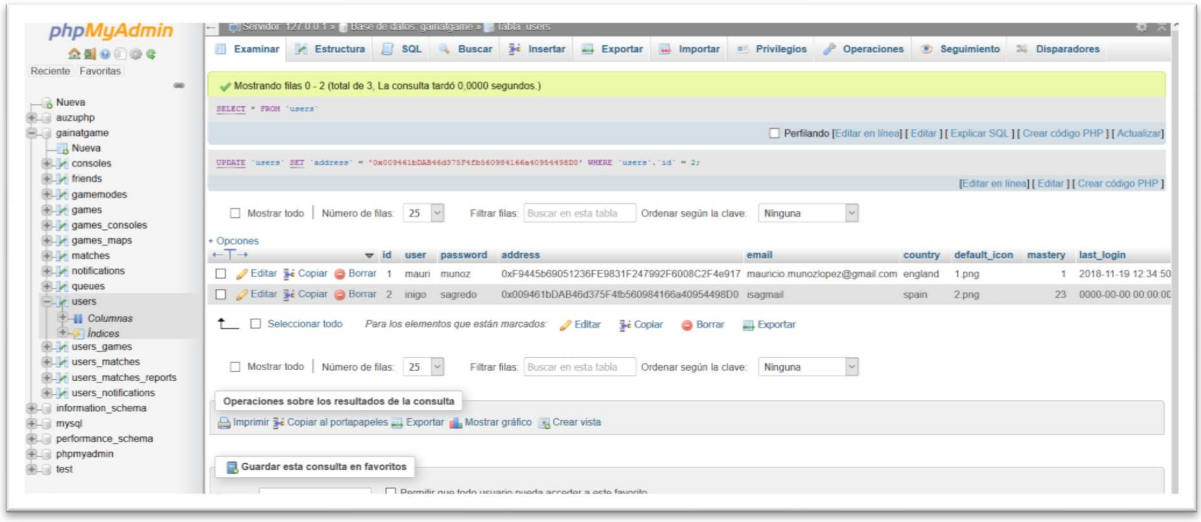


Figure 11Tabla de usuarios en base de datos

5.1.5.2 Home

Una vez hecho el log-in el usuario se encontrará en una página principal donde podrá elegir entre las dos opciones antes mencionadas: comenzar partida o validar partida. Esta pantalla es donde el usuario decide si quiere ser jugador o juez.

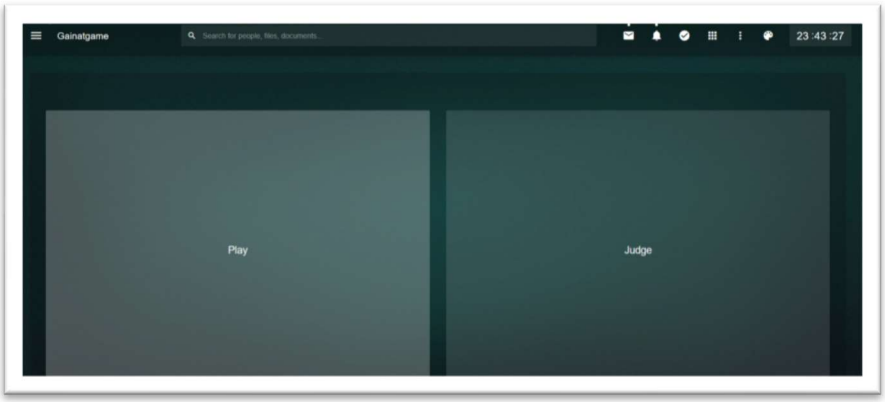


Figure 12 Pantalla Home de Gainatgame en navegador

Además de estas dos opciones, un usuario tiene la posibilidad de unirse a una partida en la que un amigo suyo (entendiéndose amigo como usuario al que ha agregado previamente a su lista de contactos) ya esté unido, pero aún no haya comenzado.

5.1.5.3 Depositar fondos

Antes de poder comenzar partida es necesario que el usuario ingrese fondos, ya sean tokens o Ether (la criptomoneda de Ethereum) en su address de Ethereum. Para ello se recomienda usar proveedores que MetaMask recomienda por defecto, por ejemplo, Coinbase.

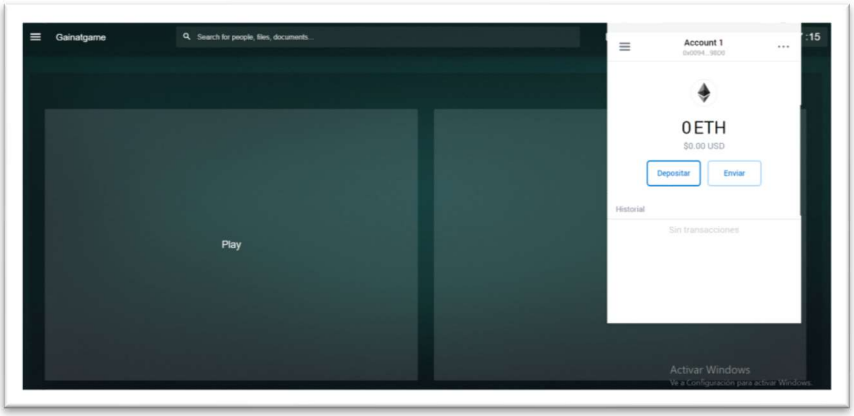


Figure 13 Cómo depositar fondos en tu cuenta utilizando MetaMask

5.1.5.4 Entrar en cola

Un usuario que desee buscar un rival lo primero que debe hacer es seleccionar la configuración de su apuesta. Ésta es: consola, juego, modo de juego y cantidad a apostar. Todas estas posibilidades se encuentran ya determinadas dentro de la base de datos. De forma que, una vez que selecciona una consola, al usuario le aparecen únicamente los juegos disponibles en esa consola. Una vez el juego esté seleccionado puede elegir el modo de juego y la cantidad. Finalmente, el usuario entra en cola y se mantiene a la espera. (El concepto de cola no es más que muchos usuarios esperando a que se le asigne un rival).

Existen dos formas de crear una cola de búsqueda:

- El servidor es quien mediante sockets o mediante un programa que se ejecuta constantemente va asignando a cada usuario su rival

- Son los propios usuarios quienes activan la funcionalidad con llamadas periódicas a una función. Esta opción es menos flexible, pero evita tener un programa ejecutándose eternamente en el servidor.

Se ha elegido la primera opción por ser mucho más flexible, aunque el coste de desarrollo sea mayor. Cuando el usuario ingresa en cola, este avisa al servidor de Node, mediante un evento por los sockets, de que cada cierto tiempo debe ejecutar una función de búsqueda. Esta función busca rivales para los usuarios que estén simultáneamente en cola y conectados al socket. Cuando el servidor detecta que hay un rival, éste crea la partida, ingresa a los dos jugadores (o a todos si hubiera más de dos) en partida y se lo comunica a los jugadores mediante un evento enviado por los sockets. El jugador, al recibir la confirmación, actualiza su información y si, en efecto, es cierto que su estado ha cambiado y ahora se encuentra en partida, este es redirigido a una pantalla de partida. En esta pantalla aún no se ha realizado ningún pago.

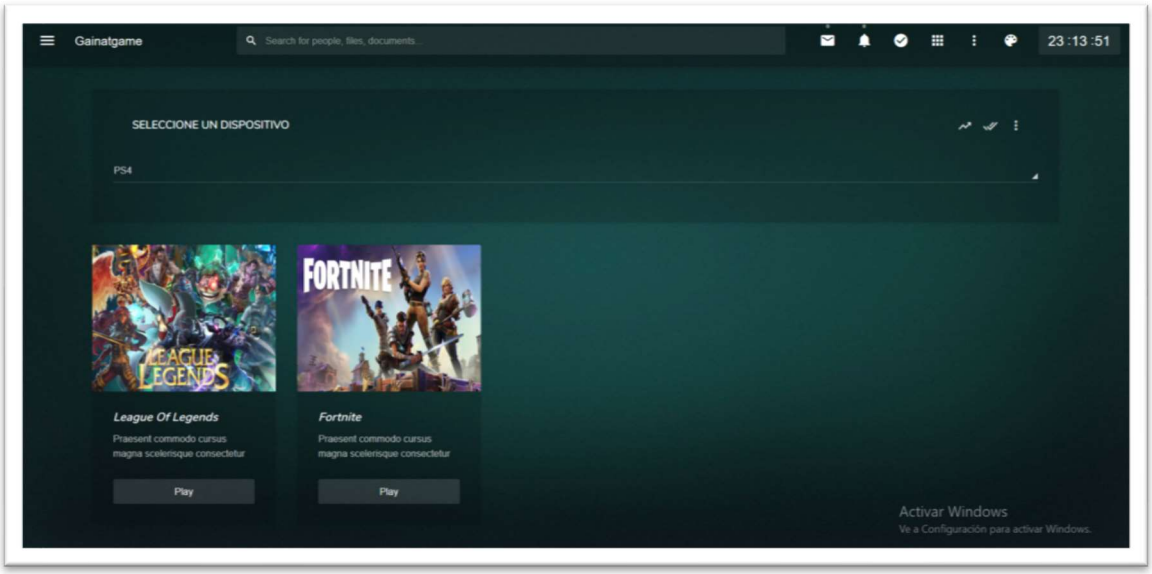


Figure 14 Pantalla de selección de consola y de juego

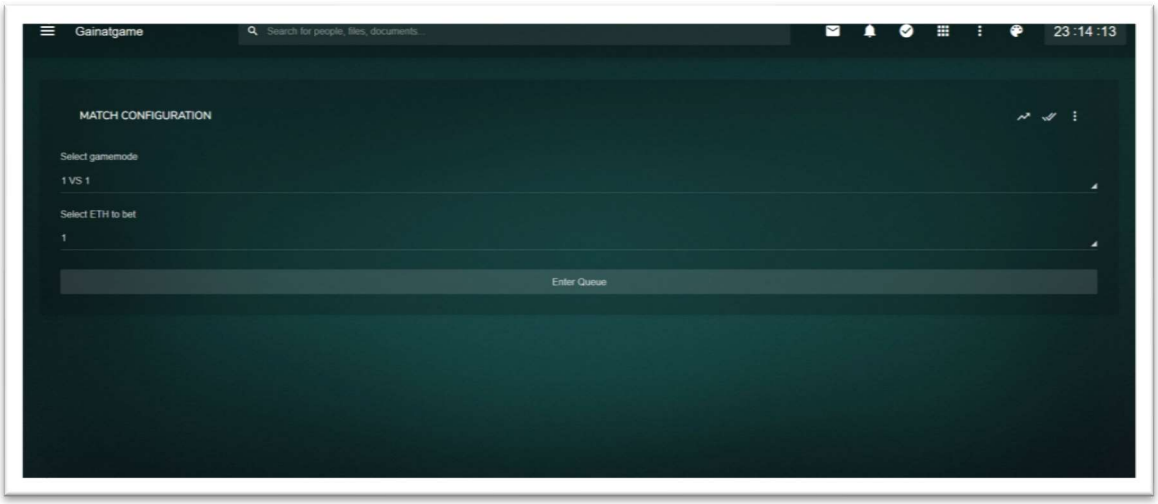


Figure 15 Pantalla de configuración de partida. Selección de modo de juego y cantidad a apostar

Figure 16 Pantalla de espera. Gainatgame está buscando rivales adecuados.

5.1.5.5 Partida encontrada(local)

Cuando se ha encontrado rival para el usuario, éste es redirigido a una pantalla donde se le muestra la información del rival. Es ahora momento de realizar los pagos correspondientes o salirse de partida. En esta pantalla, en todo momento, los jugadores sabrán quién ha realizado ya el pago de la apuesta y quién no. Si se sale de partida, se volverá al Home, donde podrá volver a entrar en cola. Si decide aceptar partida, el usuario realizará una transacción a Ethereum del pago correspondiente Una vez un jugador haya realizado el pago, quedará esperando a que su rival o rivales acepten también la partida y realicen el pago. Esta espera termina cuando se alcanza un mínimo de jugadores que hayan hecho el pago. Si aún no se ha llegado al mínimo de jugadores, cualquiera puede echarse atrás realizando otra transacción (cancelar apuesta) y recuperar todo el dinero apostado. Sin embargo, si ya se ha alcanzado el mínimo de usuarios que hayan pagado, ya no se permite a ningún usuario que cancele su apuesta. Para permitir que haya usuarios que se unan a la partida y realicen el pago una vez se llega al mínimo, las apuestas están permitidas un determinado tiempo (que depende del juego y de la situación) desde que se alcanzó el mínimo. Una vez pase ese tiempo, la partida se cierra, no se permiten más apuestas y se da por comenzada la partida.

Un caso extremo digno de estudio se da cuando un usuario acepta partida realiza la apuesta y su rival abandona partida. En esta situación el usuario ha pagado la apuesta y las comisiones correspondientes por la transacción. Mientras la apuesta es completamente recuperable, sin embargo, las comisiones no. El usuario ha incurrido en unos gastos que no tendría por qué haber tenido. Mientras, el rival, quien ha abandonado partida no ha sido perjudicado en absoluto. Este problema es intrínseco a Blockchain. La forma de solucionarlo sería que la plataforma uniese a los dos jugadores a la vez a la partida y a la apuesta, pero eso es imposible. Cada jugador es responsable de sus transacciones. Una forma de solucionar este problema es la de encontrar al usuario que ya ha realizado el pago a otro rival. Y así hasta que se encuentre un rival que en efecto pague la apuesta. Esta solución satisface al usuario que iba a perder su comisión, sin embargo, el rival que dejó la partida a medias y perjudicó al usuario, sale intacto. Para poder aplicar una sanción al rival es necesario que éste haya dado una fianza previamente. Es una práctica posible la de pedir a todos los usuarios una pequeña fianza a la hora de registrarse para pagar futuros imprevistos y perjuicios. Sin embargo, esto puede desmotivar y desincentivar

a algunos usuarios a usar la plataforma. Todo tiene sus pros y sus contras. Otra solución es castigar al rival localmente baneándole o restringiéndole temporalmente el uso de la plataforma. Este castigo no es equivalente al económico pero seguro que desincentiva el mal comportamiento.

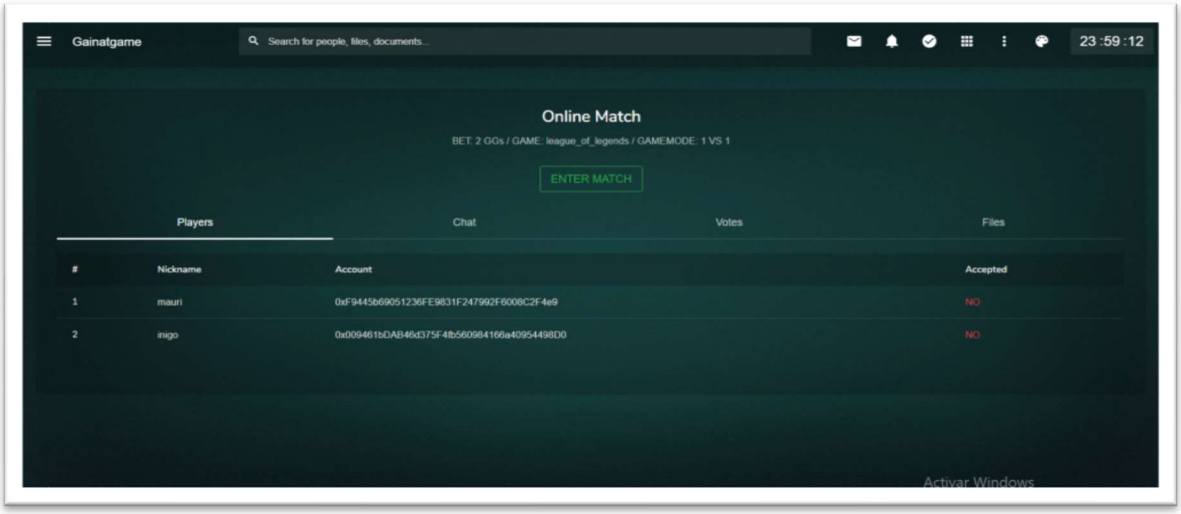


Figure 17 Pantalla de partida encontrada. Esperando a que el usuario acepte partida.

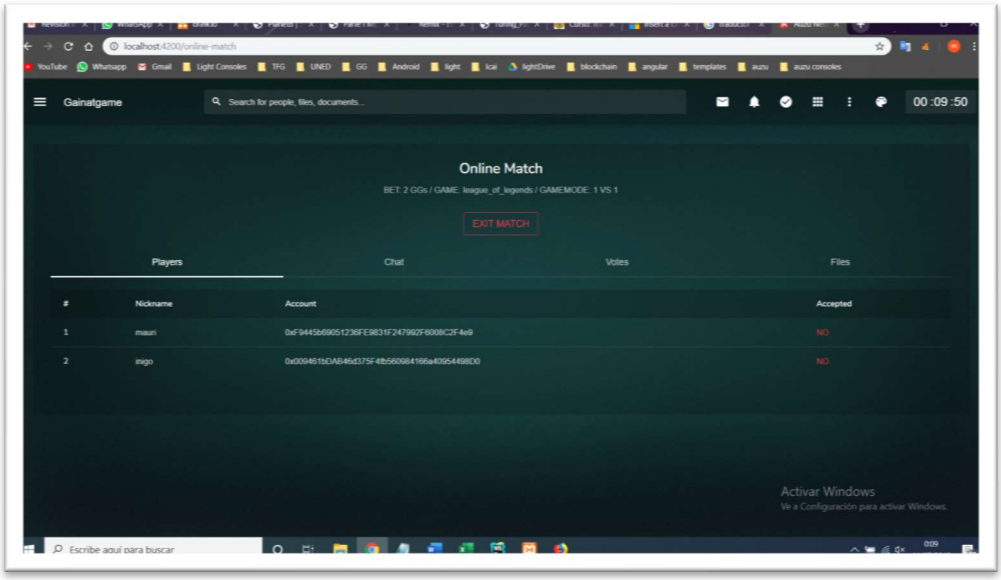


Figure 18 Pantalla de partida encontrada y aceptada. Esperando a rivales o a cancelar apuesta.

5.1.5.6 Host

En la mayoría de los juegos existe la necesidad de que alguien cree la partida, sea el host. Este usuario deberá ser el responsable de invitar a los demás usuarios. Sin duda, un usuario que sea host incurre en más esfuerzo que alguien que no lo es, por lo que debe ser recompensado. Por ello, se crea el concepto de host y de comisión de host. El primer usuario en unirse a partida será nombrado host y cuando la apuesta finalice se llevará una pequeña comisión del bote total. El hecho de que el host sea el primero en realizar la transacción y el pago incentiva a los usuarios a unirse a partida. Esto puede evitar situaciones incómodas y de desconfianza en la que varios usuarios se encuentran en partida esperando a que alguien realice el pago. Sin embargo, nadie lo realiza. De esta forma, a todos querer ser el host, comienza una carrera para ver quién es el primero.

5.1.5.7 Cancelar apuesta

Si se ha realizado la transacción con el pago de la apuesta y no se ha alcanzado el mínimo de jugadores, es posible que el usuario cancele su apuesta. No se le cobrará ningún tipo de comisión por daños. El usuario recibirá su cantidad abonada íntegramente. Una dificultad técnica se da cuando cancela la apuesta el host. En el Smart Contract a cada jugador se le representa como una estructura, que se guarda dentro de un array de estructuras. Cuando un jugador normal abandona la apuesta, no hay mayor problema que cambiar una de sus variables a 0 indicando que ya no está en partida. Sin embargo, el host es siempre el jugador en la posición 0 del array. Esto requiere que si el jugador 0 abandona partida todos los jugadores se muevan una posición a la izquierda. Esto no conlleva demasiado procesamiento, pero sin duda aumenta el gas utilizado y la comisión. Por lo que la comisión que le cobra el Blockchain al host cuando cancela la apuesta es mayor que la que le cobra al resto de usuarios.

El motivo por el que el array tiene que contener únicamente jugadores en partidas y por lo que hay que desplazarlo una posición a la izquierda cuando alguien cancela partida es el ahorro de gas. Si en el array hubiera tantos jugadores en partida como no en partida, cada vez que se requiere iterar el array sería muy ineficiente. Y en Blockchain la ineficiencia se paga literalmente.

Al ser el array siempre de jugadores en partida, el host siempre se encuentra en la posición 0 por lo que no es necesario crear otra variable para indicarlo.

5.1.5.8 Evaluación de la partida

Una vez que la partida ha terminado, los jugadores podrán votar a quien ha ganado. Es en esta sección donde se ve claramente la necesidad de tener una relación usuario-address. El sistema de votación y toda la lógica se encuentra en el Smart Contract. Por lo que es necesario que cada jugador realice una transacción indicando quién ha ganado. El problema es que los usuarios solo conocen los nicknames de los demás jugadores. Una posibilidad sería crear relaciones nickname-address en Blockchain y que los jugadores votasen por nickname, pero esto encarece mucho el uso. Además, es preferible no hacer más difícil la lógica de los Smart Contracts. Por ello, se utilizan las address con las que los usuarios se autenticaron. De esta forma se vota a la address asociada al nickname que el usuario elija. El usuario en ningún momento verá nada en hexadecimal ni nada relacionado con Blockchain. Es posible que un usuario use una address distinta a la indicada. En este caso, en el Blockchain el jugador que esté jugando será diferente a la address que se tiene asociada a ese jugador en la base de datos. Esto perjudica gravemente al usuario, ya que, aunque él haya sido el ganador, ningún jugador le votará, ya que todos los votos irán para la address de la base de datos en lugar de la que realmente usó. Es crucial que se utilice la misma address con la que el usuario se identifica.

El plazo de votar para los jugadores finaliza cuando, o bien han votado todos o cuando ha pasado un tiempo predefinido. Una vez ha concluido el plazo hay que analizar si se ha alcanzado una mayoría lo suficientemente significativa como para poder asegurar el ganador real. En caso de falta de consenso se inicia un proceso al que llamaremos juicio.

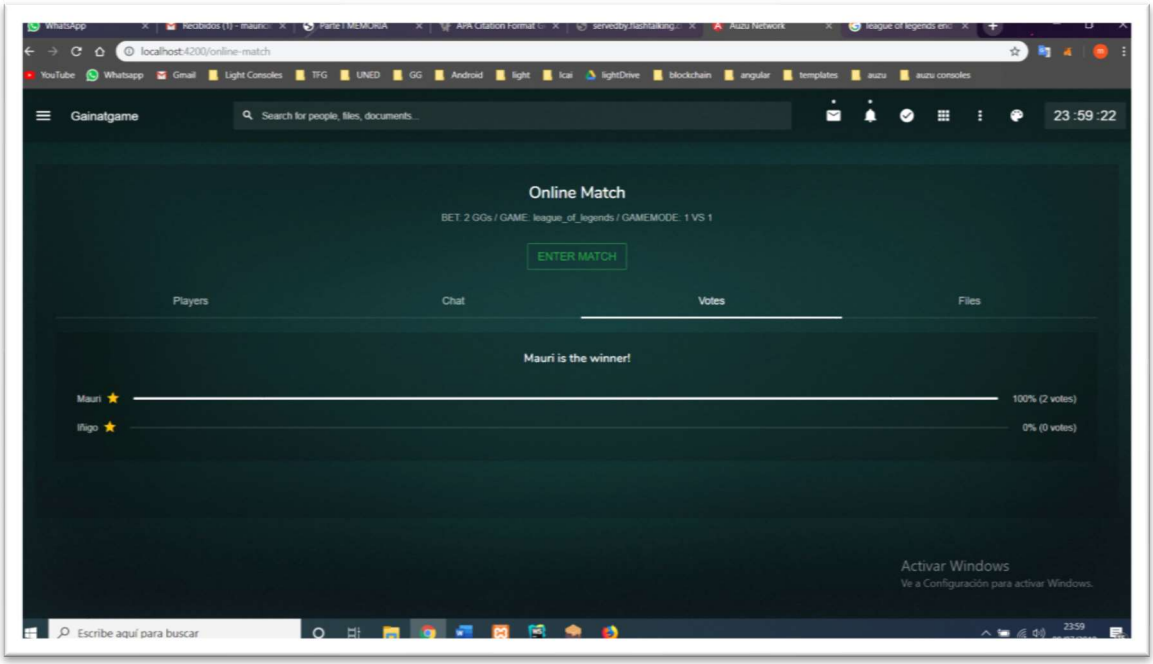


Figure 19 Pantalla en la que los jugadores votan y se muestra el resultado actual d la votación. En este ejemplo ambos jugadores han votado de forma unánime y se ha determinado un ganador.

5.1.5.9 Juicios

Un juicio es el proceso en el cual otros usuarios que no hayan participado en una partida y que hayan sido elegidos al azar juzgan unas pruebas aportadas por los jugadores para delibrar y elegir al ganador de la partida. Los jueces, a cambio de este trabajo, reciben una recompensa económica. Para poder decidir si es necesario ir a juicio es necesario analizar los perjuicios y consecuencias que estos suponen tanto para jueces como para jugadores.

Si el dinero para pagar a los jueces saliese del bote total de la apuesta, los jueces recibirían su parte, pero el ganador, quien ha actuado honestamente, estaría pagando todos los costes. Además, los jugadores que no hayan actuado honestamente, que los hay ya que no ha habido unanimidad, saldrían impunes. La solución más justa es la siguiente: todos los jugadores en el momento de realizar el pago aportan la cantidad acordada de apuesta junto con una pequeña fianza en concepto de validación. Imaginemos que en un juicio hay 5 jueces y cada uno cobra 5 céntimos. El precio total de la fianza son 25 céntimos. Cada jugador deberá realizar un pago de 1 euro (cantidad a apostar) y 25 céntimos (fianza).

Gracias a esta fianza, a los jugadores honestos les da igual si se va a juicio o no, ya que una vez que termine el juicio, si el ganador coincide con quien ellos votaron, recibirán de vuelta la fianza. Si en una partida de 4 personas una de ellas discrepa con el resto, no hay problema, se va a juicio. La consecuencia es que el jugador deshonesto perderá su fianza mientras que los demás no verán alterados sus beneficios o pérdidas. Si más de un jugador miente, los jueces recibirán más dinero por el mismo trabajo, ya que la fianza de todos los jugadores se reparte entre los jueces.

Este modelo de incentivos por fianza es perfecto para partidas pequeñas y medianas. Sin embargo, en partidas de muchos jugadores, como en una de 100, el precio del juicio es tan alto que es inviable que cada jugador aporte una fianza que pague íntegramente el juicio. Es por ello que se debe buscar otro sistema para partidas cuyos juicios sean demasiado caros.

En este último tipo de partidas (partidas con muchos jugadores) existe una ventaja: el premio total es muy grande. En el ejemplo de la partida de 100 jugadores el premio por ganar son 100€. Existe mucho margen. Al ganador es posible que no le importe que 5 euros (un 5%) de su premio vaya destinado a validación. Al fin y al cabo, no hay mucha diferencia entre cobrar 100€ y cobrar 95€ habiendo apostado 1 euro. Con esos 5€ se puede pagar a los jueces, pero sigue habiendo un problema en tanto a la justicia. Los jugadores que mintieron no se ven perjudicados. Para ello se crea un modelo mixto, donde los jugadores siguen aportando una fianza en concepto de validación. Como esta fianza no es suficiente para financiar el juicio, el restante se obtiene del premio del ganador. La cantidad aportada como fianza por los jugadores es meramente simbólica y tiene como único propósito promover la honestidad.

El porcentaje de mayoría necesaria en partidas de muchos jugadores puede oscilar entre 80-100% dependiendo del porcentaje del bote total que se esté dispuesto a pagar. Hay que tener en cuenta que las fianzas de todos los jugadores que discrepen se destinaran al juicio por lo que el porcentaje a sacar del premio del ganador es menor. Es posible que con una elección correcta del porcentaje de mayoría necesaria no sea necesario utilizar dinero del bote, ya que las fianzas de los jugadores que discrepen serán suficientes para pagar el juicio.

Del mismo modo que los jugadores pueden mentir, los jueces también. Para evitarlo y promover la verdad y la honestidad, los jueces aportarán también una fianza. Igual que con los jugadores, si el voto de los jueces coincide con el ganador, el juez recuperará su fianza junto con su compensación económica por haber ayudado a validar una partida. La fianza será exactamente el doble en valor que la compensación económica que el juez recibiría al terminar el juicio con éxito. Si la fianza fuera del mismo valor que la compensación, en partidas de dos jugadores donde un juez solo tiene que elegir entre dos opciones, las posibilidades de acertar serían del 50%. Si la fianza es igual a la compensación, si el juez decidiese juzgar aleatoriamente de cada dos juicios ganaría uno y perdería otro. Tras los dos juicios el juez quedaría igual que empezó ya que cuando acertó ganó la misma cantidad que perdió cuando falló. Esto no se puede permitir ya que los jueces podrían votar aleatoriamente y perjudicar al sistema sin verse ellos perjudicados. Por eso la fianza debe ser mayor que la compensación. El doble es un valor bastante aceptable.

Cuando un juez pierde su fianza, ésta es repartida entre los otros jueces, igual que pasaba con los jugadores. Una diferencia de los juicios con la votación de los jugadores es que en los juicios la mayoría necesaria para nombrar a alguien ganador es del 51%. Siendo los jueces impares, siempre existe un ganador. Existe un mínimo y un máximo de jueces tal que el juicio debe contar con un número de jueces que se encuentre entre ese rango. Una vez que se alcanza el mínimo hay un tiempo determinado para que nuevos jueces se incorporen. Al finalizar ese tiempo, el juicio se inicia y dependiendo del veredicto se reparte el dinero consecuentemente. Si el juicio nunca alcanza el mínimo, se quedaría en ese estado eternamente. Como la plataforma es la encargada de ofrecer los juicios a los jueces, si existe un juicio que lleva demasiado tiempo sin validarse es por un fallo en la plataforma o por un problema excepcional del juicio.

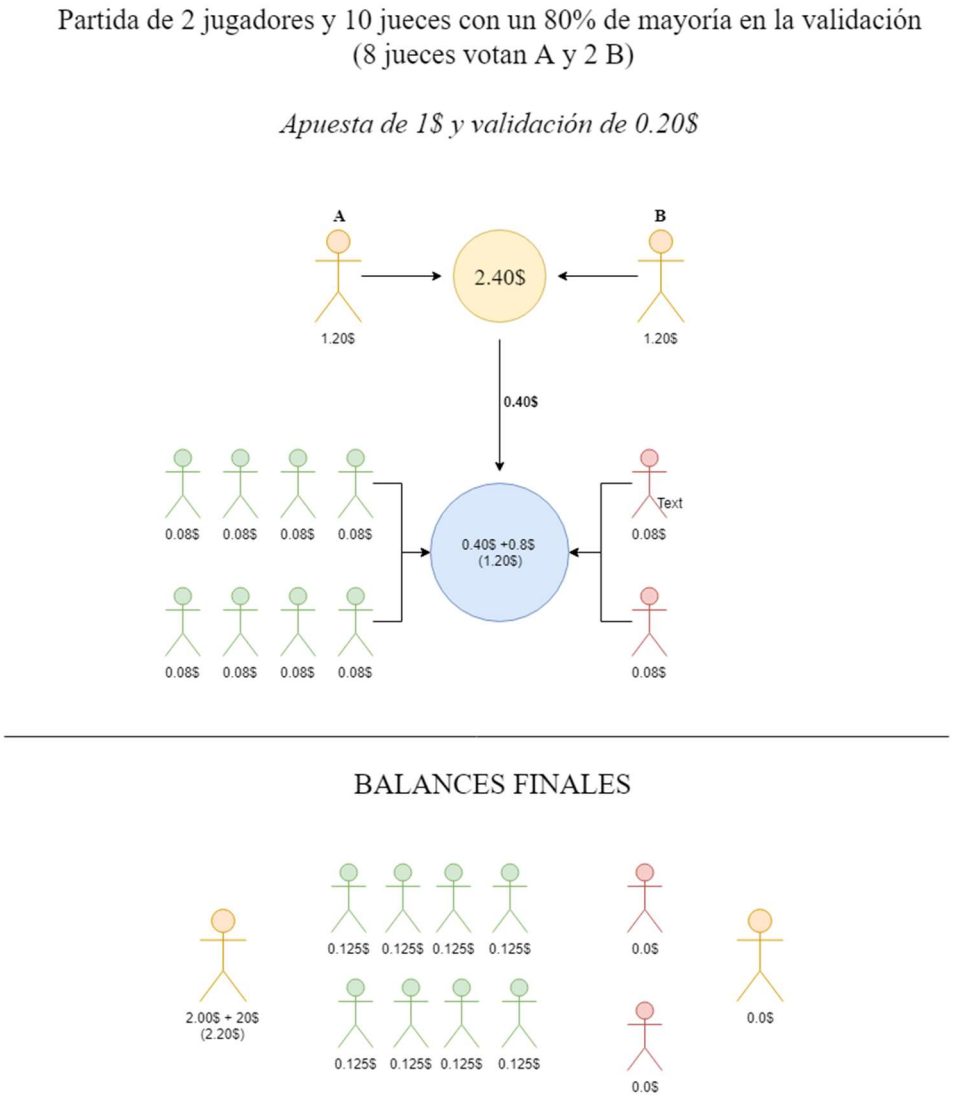


Figure 20 Esquema y balances finales de jugadores y jueces.

5.1.5.10 Número de jueces

El número de jueces necesario en un juicio depende de la dificultad que haya en la validación. Esta dificultad depende del número de pruebas a juzgar y por consiguiente del número de jugadores que haya en la partida. Sin embargo, es importante tener en cuenta que el número de pruebas a juzgar no es proporcional al número de jugadores, sino al número de jugadores que haya en disputa. Es decir, si en una partida de 10 jugadores, 5 jugadores votan al jugador 1, 3

jugadores votan al jugador 2 y 2 jugadores no votan, lo más probable es que únicamente los jugadores 1 y 2 decidan realizar el esfuerzo adicional de aportar pruebas. Al fin y al cabo, ¿por qué el jugador 3, que ha votado a otro jugador que no es él, o el jugador 10 que no ha votado iban a subir pruebas si ya han dado la partida por perdida? Por ello, para determinar el número de pruebas que se esperan hay que tener en cuenta el número de jugadores distintos que han recibido algún voto.

La función que seguirá el número mínimo de jueces será:

- Jueces = (jugadores votados*6) – 7

La función que seguirá el número máximo de jueces será:

- Jueces = ((jugadores votados + 1) *6) – 7

Nótese que la función da siempre resultados impares con el fin de que siempre exista mayoría en los jueces. Sería coherente dividir los juicios que requieran únicamente imágenes con los que requieran de videos para la validación, ya que los videos son más costosos de validar para los jueces por lo que deberían recibir una mayor compensación. Sin embargo, con el fin de generalizar el código y debido a la dificultad de clasificar los juegos según el tipo de prueba que se requiere (siendo lo más habitual que tanto imágenes como videos sean válidos) se decide no realizar esta división.

5.1.5.11 Fianza de jugadores y jueces

Se determina que la fianza a aportar por los jugadores será en todo tipo de partidas del 20% de la apuesta. Es decir, si se apuesta 1€, la fianza será de 20 céntimos. Además, se determina que en toda partida se deberá destinar un 5% del bote total de la apuesta a validación. Si este 5% es en cantidad igual o menor al 20% de la apuesta, significa que cada usuario puede financiar íntegramente el juicio con su fianza. En cambio, si es mayor, es necesario acceder a parte del bote para que la cantidad total sume el 5% del bote necesario. Esto no significa que siempre se utilice un 5% del bote, sino que, la suma de la fianza por jugador y lo utilizado del bote tiene

que ser igual en cantidad al 5% del bote. Por ejemplo, en una partida de 10 jugadores que apuestan 1€ la fianza por jugador es de 20 céntimos (20%). El dinero que se debe destinar al juicio es de 50 céntimos (5% de 10€). Sin embargo, la fianza de cada jugador es menor. (20 céntimos < 50 céntimos). En consecuencia, si únicamente un jugador difiere del resto y se quiere ir a juicio, es necesario utilizar 30 céntimos (el restante para alcanzar el objetivo del 5% del bote) del bote. El ganador en vez de recibir 10€ recibirá 9,70€ ya que 30 céntimos se han destinado a validación.

Sin embargo, es posible regular las partidas para que no sea necesario un 100% de unanimidad en los votos de los jugadores.

5.1.5.12 Partidas de pocos jugadores y muchos jugadores.

El punto de inflexión para diferenciar estos dos tipos de partidas se encuentra cuando el 20% de la apuesta es igual al 5% del bote. Es decir, en partidas de 4 jugadores. Por ejemplo, 4 jugadores apuestan 1 €. La fianza por jugador será de 20 céntimos y el 5% del bote es de 0,20€. En partidas de más de 4 jugadores, la fianza de un jugador ya no cubre el juicio.

El motivo por el que el dinero destinado a validación tiene que ser proporcional al tamaño de partida es debido a que la dificultad de validación aumenta cuantos más jugadores participan.

Por ello, las partidas entre 1 y 4 jugadores serán denominadas partidas de pocos jugadores (donde no se accede al bote de la apuesta y se requiere de unanimidad en los votos de los jugadores) y las partidas con más de 4 jugadores serán partidas de muchos jugadores (donde si se accederá a parte del bote de la apuesta y no se requiere de unanimidad en los votos de los jugadores). Al tratarse de partidas con muchos jugadores, cuando éstos votan quién ha sido el ganador una vez que la partida finaliza, no es necesario que haya unanimidad en los votos para asegurar con una confianza significativa que dicho jugador es el verdadero ganador. En partidas de pocos jugadores es necesario que todos voten lo mismo, sin embargo, en este tipo de partidas porcentajes de mayoría que ronden el 80% son completamente aceptables.

Cabe destacar que en partidas de muchos jugadores aún sigue siendo posible que el ganador reciba el 100% del bote. Esto se daría cuando más de un jugador ha sido deshonesto. De esta

forma la fianza de todos los jugadores deshonestos iría destinado a validación y no sería necesario acceder al bote.

5.1.5.13 Mayorías necesarias

En partidas de pocos jugadores se necesita que todos los jugadores voten de forma unánime para que se determine un ganador. En caso contrario habrá juicio.

En partidas de muchos jugadores la mayoría necesaria para que no haya juicio en las votaciones de los jugadores es del 80%. Esto significa que, por ejemplo, en una partida de 5 jugadores, si 4 jugadores votan a un jugador este jugador es automáticamente el ganador de la apuesta.

Tanto en partidas de pocos jugadores como en partidas de muchos jugadores que lleguen a juicio se necesitará un 51% de votos de jueces para que exista ganador. Motivo por el cual siempre hay jueces impares.

5.1.5.14 Ausencia de voto

Es muy posible que en dinámicas reales los jugadores al perder la partida decidan desconectarse y no seguir con el proceso de votación. Esto puede suponer un problema puesto que si se requiere unanimidad en ciertos tipos de partida es muy probable que no se alcance. Por ello, como se ha razonado en el punto anterior, se decide que la mayoría a alcanzar en los votos únicamente será sobre el total de votos realizados no sobre el total de jugadores.

Cuando haya jugadores que decidan no votar, en el caso en el que se realice un juico, estos no recuperarán su fianza para validación. Ya que, en parte, es su responsabilidad el que haya sido necesario un juicio.

En caso de que no sea necesario un juicio, los jugadores que no hayan votado sí recuperan su fianza ya que está acción no ha supuesto ningún perjuicio. Igual ocurre cuando un jugador vota a otro jugador diferente al que ha votado la mayoría. Al no requerirse juicio este jugador (quien probablemente no ha sido honesto) recibe igualmente su fianza.

Esta metodología crea un claro incentivo para los jugadores a la hora de votar, ya que, si no votan, en el caso de que haya juicio, éstos no recuperarán su fianza.

5.1.5.15 Pruebas

Cuando la partida va a juicio, los jugadores deben subir sus propias pruebas que aporten información sobre quién es el ganador. Las pruebas se subirán al servidor. Los jugadores disponen de un tiempo determinado para subir sus pruebas. Después de ese tiempo, los jueces empezarán a juzgar. Las pruebas pueden ser tanto fotos como videos. Una vez subidas las pruebas solo queda esperar al veredicto.

5.1.5.16 Challenge

Cuando no existe unanimidad en los votos de los jugadores, pero sí se alcanza la mayoría mínima un jugador es seleccionado como ganador. Existe la posibilidad de que algún jugador quiera reclamar esta decisión, paralizando así el reparto de premios y llevando a juicio la partida. El juicio lo deberá financiar íntegramente el jugador en caso de partidas pequeñas y parcialmente en caso de partidas grandes con juicios caros. El challenge está disponible por un tiempo determinado. Después, se repartirán los premios asumiendo el ganador que los jugadores decidieron. Esta funcionalidad se tiene en cuenta a la hora de diseñar el sistema, sin embargo, por motivos de simplificación se decide no implementar en el prototipo desarrollado. Se deja para futuros desarrollos.

5.1.6 VALIDAR PARTIDA

Cuando un usuario ingresa a la plataforma puede elegir entre comenzar partida y validar partida. Si elige validar partida, el usuario deberá realizar una transacción en la que abone la fianza antes mencionada. Una vez realizada, la plataforma le asigna al juez un juicio de forma aleatoria asegurando que los intereses del juez son independientes del de los jugadores de la partida que el juez está juzgando.

5.1.6.1 Jugador y juez simultáneamente

Es posible que un usuario decida utilizar otra address y se haga pasar por juez para validar su propia partida. Esto solo funcionaría en el caso que hubiera muy pocas partidas para ser validadas en ese momento. Si hay pocas partidas es probable que la plataforma le asigne su partida para validar. Esto sigue sin ser un problema ya que un solo juez corrupto no puede vencer al sistema.

Lo mismo ocurriría si varios amigos buscan partida a la vez y hay poca gente buscando partida. Es probable que los amigos sean emparejados en la misma partida. Estos podrían mentir en las votaciones. Como no hay 100% de mayoría, el juicio aclararía lo sucedido por lo que esta estrategia tampoco es efectiva.

El problema existe cuando se realizan ambas estrategias a la vez. Un grupo de amigos es emparejado en la misma partida y más tarde todos se hacen pasar por jueces para juzgar su partida. Este problema se da cuando hay pocos usuarios en la plataforma. La única forma de solucionarlo es haciendo una plataforma atrayente y de gran calidad para que nunca haya pocos usuarios.

5.1.6.2 Visualizar y deliberar

Una vez el juez tiene asignado juicio, deberá ver las pruebas y deliberar. El voto se realiza mediante una transacción indicando el address del ganador. Una vez más este address se saca de la relación user-address y user-nickname. En realidad, el juez está votando por un nickname, que es lo que se puede apreciar en las pruebas aportadas por los usuarios.

5.1.6.3 Reparto de premios

Una vez termina el juicio y el juez ha sido honesto, recibe su recompensa. Si no lo ha sido, pierde su fianza. El ganador del juicio recibe el total de la apuesta menos la suma de lo usado en validación si procede y la comisión del host.

5.1.6.4 Juegos disponibles

Gainatgame es de propósito general por lo que se puede utilizar para que los usuarios apuesten a cualquier juego. Sin embargo, es necesario que, de cada juego, se conozcan sus diversos modos

de juegos, dinámicas, etc. con el fin de asegurar que se trata de un juego de suma cero con un único ganador y que es posible validarlo. Por ello, los juegos a los que los usuarios pueden jugar se determinan a nivel local. Estos juegos estarán disponibles únicamente después de un análisis en el que se determine que un juego cumple con las características necesarias ya expuestas.

5.1.6.5 Responsive

Como se especificó en el diseño, Gainatgame es responsive tal y como se aprecia en la siguiente captura:

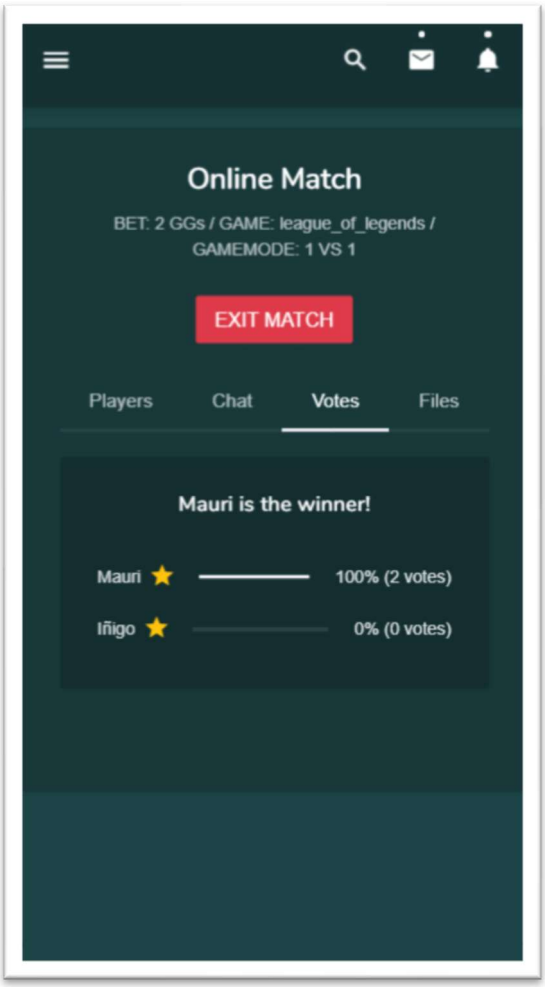


Figure 21 Pantalla de partida finalizada visto desde el móvil

Capítulo 6. ANÁLISIS DE GAINATGAME

18. ANÁLISIS DE VIABILIDAD ECONÓMICA

Como se ha explicado anteriormente, toda interacción con el Blockchain cuesta dinero. Estas son las comisiones que cobran los mineros. Las comisiones son del orden de céntimos por lo que normalmente no son de gran importancia. Sin embargo, en esta aplicación estamos tratando con transacciones en tiempo real y con tamaños de apuesta pequeños. Una comisión de 15 céntimos en una apuesta de 100 euros es perfectamente asumible teniendo en cuenta el valor añadido en seguridad. Mientras que 15 céntimos en una apuesta de 1 euro sería excesivo. Por ello, se procede a realizar un análisis de la rentabilidad y viabilidad económica de esta plataforma.

Como ya se ha visto, cuanto mayor es la comisión de una transacción, menor es el tiempo que un usuario espera hasta que su transacción es aceptada por el Blockchain. Dependiendo de la velocidad con la que se requiere que una transacción se acepte se seleccionará un precio de gas u otro. Estos son los tiempos de espera estimados para que se acepte una transacción en Ethereum para cada precio de gas

precio	2 Gwei	5 Gwei	10 Gwei
tiempo	180s	30s	20s

Tabla 1 Tiempos de espera para que se valide una transacción en función del precio del gas

Estas son todas las funciones con las que se puede interactuar en el Smart Contract y sus correspondientes comisiones a diferentes precios de gas:

Transaccion	Gas	2 Gwei	5 Gwei	10 Gwei
Unirse a partida	80.000	0,05	€0.108	$0.2512
Salir de partida	25.000	0,02	€0.03375	€0.0675
Votar	30.000	0,02	€0.04725	€0.0945
Distribuir premios 1	60.000	0,04	€0.081	€0.162
Disputar partida	70.000	0,04	€0.0945	€0.189
Juzgar partida	35.000	0,02	€0.04743	€0.09485
Distribuir premios 2	100.000	0,05	€0.1355	€0.271

Tabla 2 Funciones del Smart Contract y sus comisiones en € según el precio del gas

Los costes en comisiones conseguidos en las funciones del Smart Contract son, sin duda, un éxito rotundo. En un principio la función Unirse a partida llegó a costar 200,000 de gas. Los 80,000 de gas alcanzados finalmente, son un coste más que razonable, lo que supone una comisión asumible por los usuarios. Si se comparara el gas utilizado en este Smart Contract con el de algún competidor como Proof Of Toss rápidamente se puede observar en cuál de los dos sí se ha tenido en cuenta que la plataforma debe ser rentable en tiempo real.

19. CASOS DE USO

Gainatgame es una plataforma cuyo objetivo es proporcionar y suministrar un servicio de intermediación en apuestas a los usuarios. Gracias a esta nueva plataforma, los usuarios pueden apostar en partidas con otros usuarios, aunque estos sean desconocidos, y confiar en que la apuesta se finalice correctamente. Con el fin de demostrar la viabilidad y operatividad de Gainatgame en este apartado se expondrán diversos casos de usos que han sido implementados y analizados.

Los casos de uso a analizar se pueden dividir en dos tipos: las partidas de pocos jugadores (1-4) y las partidas de muchos jugadores (>4). En el primer tipo, la fianza aportada por el jugador cubre íntegramente el juicio. En el segundo caso, no es así (el dinero necesario para cubrir el juicio sale parcialmente del bote de la apuesta). En consecuencia, el ganador obtendrá un premio de menor en cantidad (máximo un 5% del bote).

Se recuerda que cuando se habla de mayoría en las votaciones se habla en términos de porcentaje de votos. Es decir, si hay 100 jugadores y solo han votado 50, una mayoría del 90% sería 45 votos y no 90 votos como se puede pensar. No votar equivale a no participar. Todos los casos tienen dos análisis, uno con comisiones y otro sin ellas.

6.1.1 CASOS DE USO CON POCOS JUGADORES (1-4)

En este tipo de partidas, la fianza que aporta cada jugador en concepto de validación (con el objetivo de pagar un juicio si esté llegase a realizarse) paga completamente el juicio. Esto se debe a que, al ser pocos jugadores en la partida, los jueces necesarios para validar la partida son también relativamente pocos, y en consecuencia el coste total del juicio (la suma de lo que cobran todos los jueces) queda asumido por la suma de las fianzas de los usuarios.

6.1.1.1 Dos jugadores sin juicio

El primer caso de uso a analizar es una partida entre dos usuarios. Cada usuario desea apostar 1 € a una partida de, por ejemplo, FIFA 18 (videojuego de fútbol). Como ya se ha explicado, para poder financiar un posible juicio (el cual tendría lugar si no existe unanimidad en las votaciones de los jugadores) es necesario que cada jugador aporte una fianza. En este ejemplo, cada jugador aporta 20 céntimos. 20 céntimos en comparación con 1 euro (la apuesta a realizar) es un 20%.

Una vez los jugadores realizan sus transacciones y ambos aportan el 1,20€ al Smart Contract la partida comienza. Ambos jugadores deben jugar la partida. Una vez la partida finaliza, deben votan al ganador de la misma. En este caso, el jugador 1 vota al jugador 1 y el jugador 2 vota al jugador 1. Existe unanimidad. Los dos jugadores han sido honestos, concluyéndose que el ganador de la partida es el jugador 1. El Smart Contract distribuye los premios consecuentemente. Como el jugador 1 ha ganado y ha sido honesto recibe 2,20€ (1€ de su apuesta, 1€ de su rival, 0,20€ de su fianza para validación). El jugador dos ha perdido, pero ha sido honesto por lo que recibe 0,20€ recuperando así su fianza para validación puesto que no ha sido necesario realizar un juicio.

El jugador 1 contaba al inicio de la partida con un saldo inicial de 10€. Al finalizar cuenta con 10,93€ (11€ - 0,07€ de comisiones cobradas por Ethereum). Las comisiones se cobran al ejecutar las funciones del Smart Contract. Estas funciones se ejecutan al aceptar la apuesta (0,05€) y al votar (0,02€).

- Jugadores: 2
- Apuesta: 1€
- Fondos iniciales de jugadores: 10€
- Juicio: no
- Fianza por jugador para validación: 0,20€
- Fianza/apuesta * 100: 20%

Sin comisiones (SC):

		n1 entra	n2 entra	partida	n1 vota n1	n2 vota n1	premios
jugador 1 SC	**10**	**8,8**	**8,8**	**8,8**	**8,8**	**8,8**	**11**
jugador 2 SC	**10**	**10**	**8,8**	**8,8**	**8,8**	**8,8**	**9**

Tabla 3 Saldos de jugadores 1 y 2 en una partida de dos jugadores sin juicio y sin comisiones

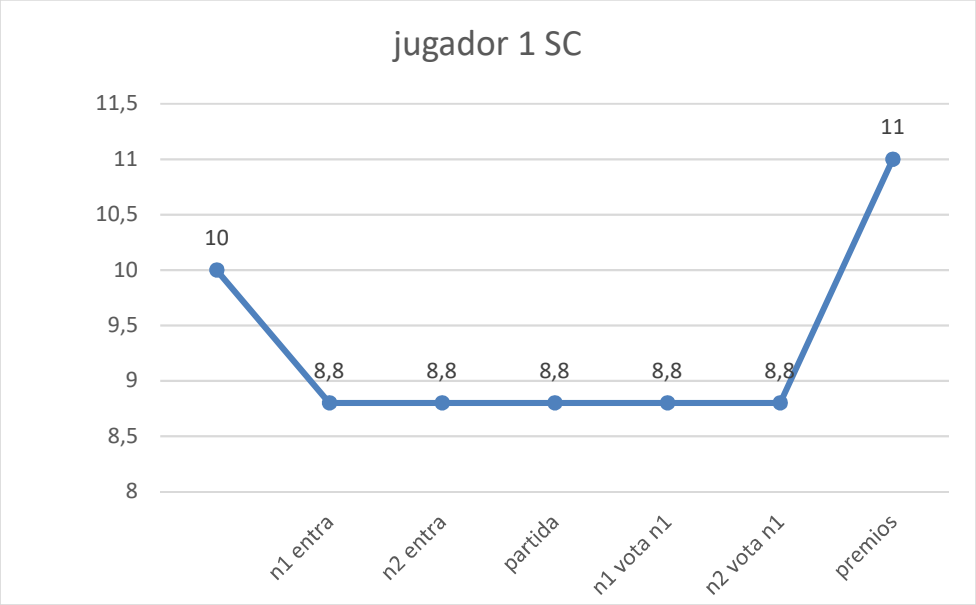


Figure 22 Gráfica jugador 1 en una partida de dos jugadores sin juicio sin comisiones

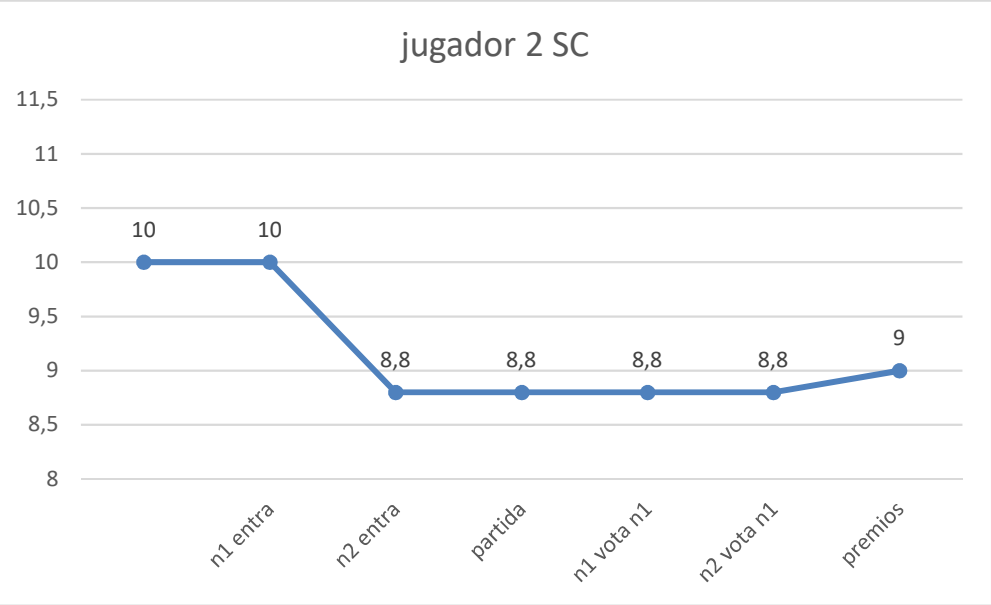


Figure 23 Gráfica jugador 2 en una partida de dos jugadores sin juicio sin comisiones

Con comisiones (CC):

		n1 entra	n2 entra	partida	n1 vota n1	n2 vota n1	premios
jugador 1 CC	10	8,75	8,75	8,73	8,73	8,73	10,93
jugador 2 CC	10	10	8,75	8,75	8,75	8,73	8,93

Tabla 4 Saldos de jugadores 1 y 2 en una partida de 2 jugadores sin juicio con comisiones

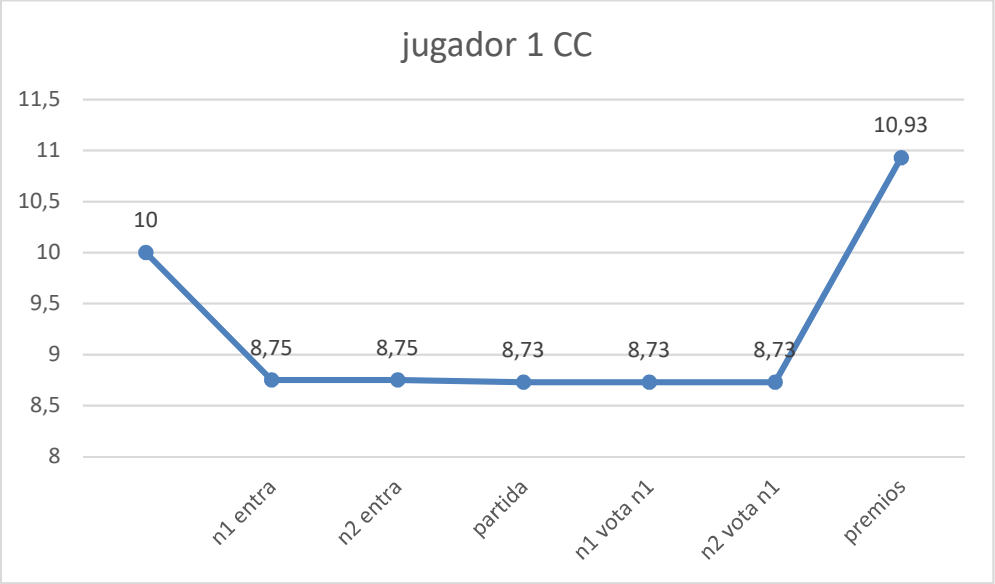


Figure 24 Gráfica jugador 1 en una partida de dos jugadores sin juicio con comisiones

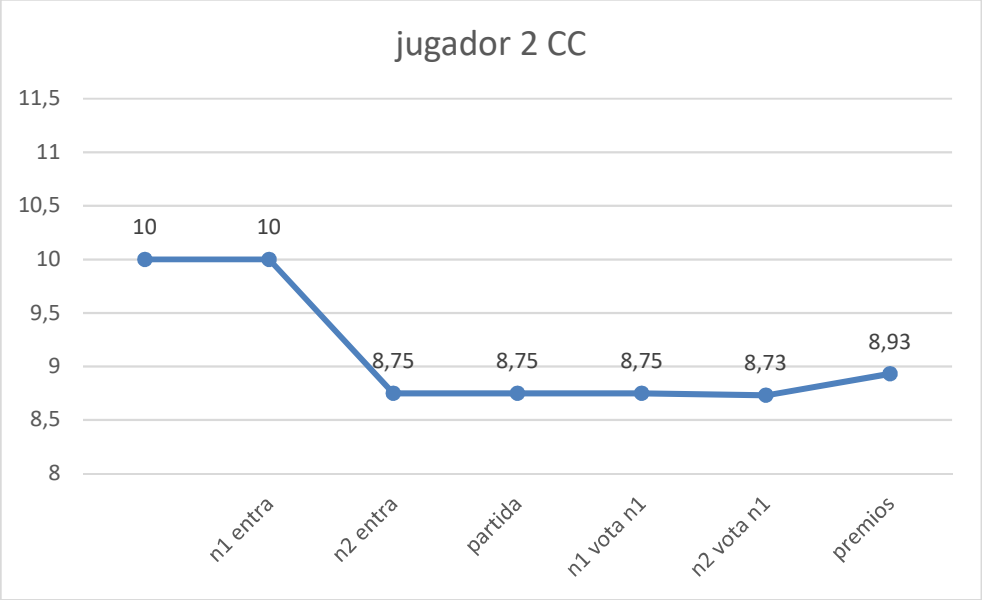


Figure 25 Gráfica jugador 2 en una partida de dos jugadores sin juicio con comisiones

6.1.1.2 Dos jugadores con juicio, todos los jueces votan unánime.

Se trata de un caso idéntico al anterior. Se parte de un saldo inicial de 10€, se apuesta 1€ y se aporta una fianza del 20% de la apuesta, 20 céntimos. Sin embargo, esta vez el jugador 1 vota al jugador 1 y el jugador 2 vota al jugador 2. No existe unanimidad. Es necesario realizar un juicio para determinar quién ha sido el verdadero ganador de la partida. Para ello se eligen aleatoriamente 5 jueces. Estos jueces, tras juzgar las pruebas visuales que tanto el jugador 1 como el jugador 2 han podido aportar, deliberan y determinan quién es el ganador. Cada juez debe aportar, del mismo modo que los usuarios, una fianza para incentivar su honestidad. La fianza de los jueces será en cantidad el doble de lo que estos pueden ganar a cambio de juzgar la partida. Lo que puede ganar cada juez como mínimo es la fianza de un jugador (0,20€) dividido entre el número de jueces (5). Siendo 0,04€ la recompensa para cada juez. Por lo que la fianza será de 0,08€. Esta fianza les será devuelta si su voto pertenece al de la mayoría. Nótese que son un número impar de jueces con el fin de que siempre exista una mayoría. A la hora de juzgar los cinco jueces determinan ganador de forma unánime al jugador 1. El jugador 1 al ser el ganador

y haber actuado con honestidad, gana el bote y recupera su fianza. El jugador 2 es el perdedor y no ha actuado de modo honesto por lo que pierde su apuesta y su fianza. Los jueces han actuado de forma unánime y honesta por lo que ganan su recompensa (financiada por la fianza del jugador 2) y recuperan su fianza.

- Jugadores: 2
- Juicio: sí
- Apuesta: 1€
- Fondos iniciales de jueces y jugadores: 10€
- Fianza por jugador para validación: 0,20€
- Fianza por juez para validación: 0,08€
- Recompensa para cada juez: 0,04€

Sin comisiones:

		n1 entra	n2 entra	partida	n1 vota n1	n2 votan2	juicio	j1 juzga n1	j2 juzga n1	j3 juzga n1	j4 juzga n1	j5 juzga n1	premios
jugador 1	10	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	11
jugador 2	10	10	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8
juez 1	10	10	10	10	10	10	10	9,92	9,92	9,92	9,92	9,92	10,04
juez 2	10	10	10	10	10	10	10	10	9,92	9,92	9,92	9,92	10,04
juez 3	10	10	10	10	10	10	10	10	10	9,92	9,92	9,92	10,04
juez 4	10	10	10	10	10	10	10	10	10	10	9,92	9,92	10,04
juez 5	10	10	10	10	10	10	10	10	10	10	10	9,92	10,04

Tabla 5 Saldos de jugadores 1 y 2 y jueces 1-5, en una partida de 2 jugadores sin juicio sin comisiones

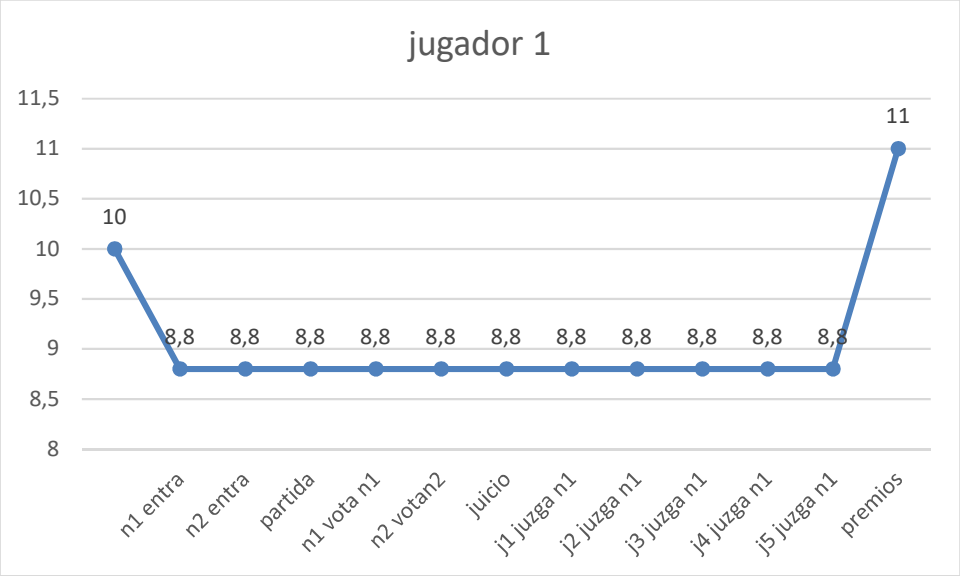


Figure 26 Gráfica del juez 1 en partidas de dos jugadores con juicio unánime sin comisiones

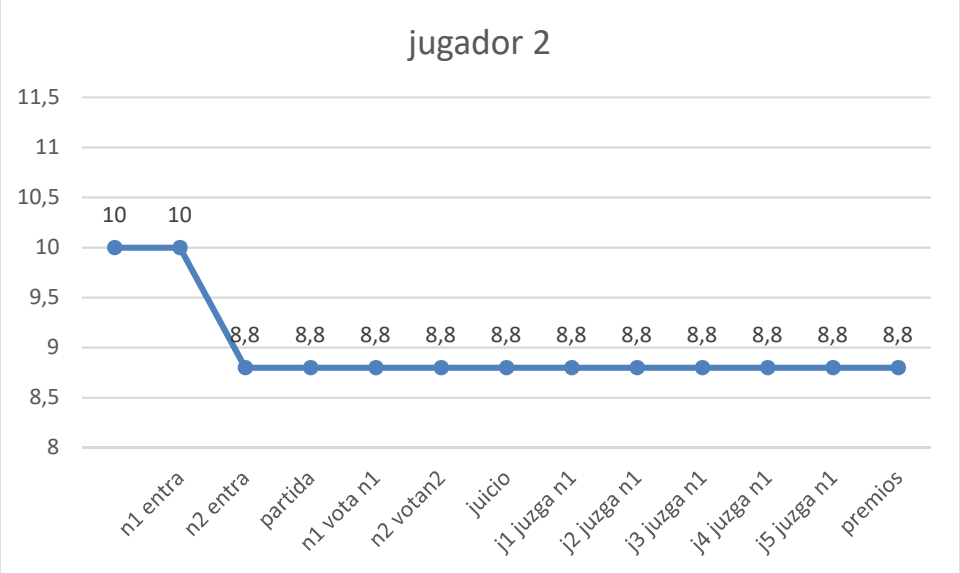


Figure 27 Gráfica del jugador 2 en partidas de dos jugadores con juicio unánime sin comisiones

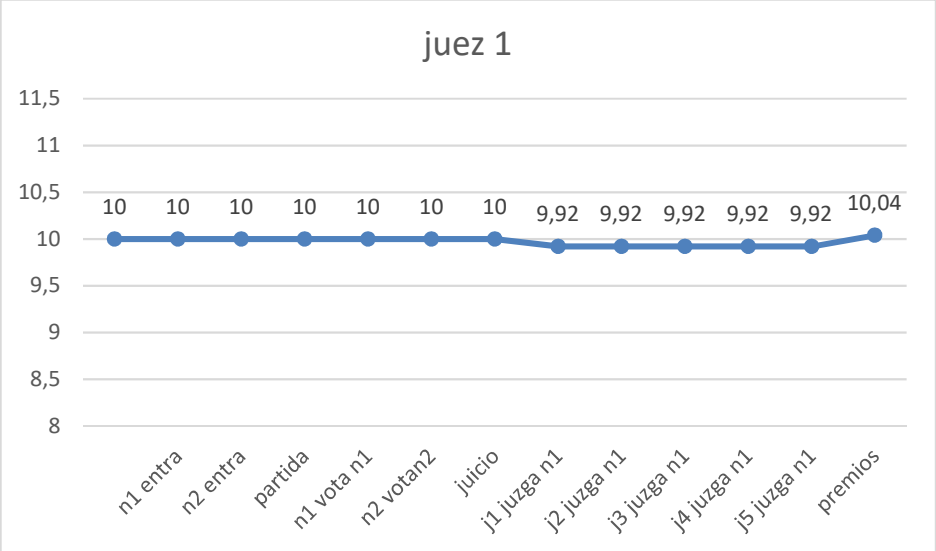


Figure 28 Gráfica del juez 1 en partidas de dos jugadores con juicio unánime sin comisiones

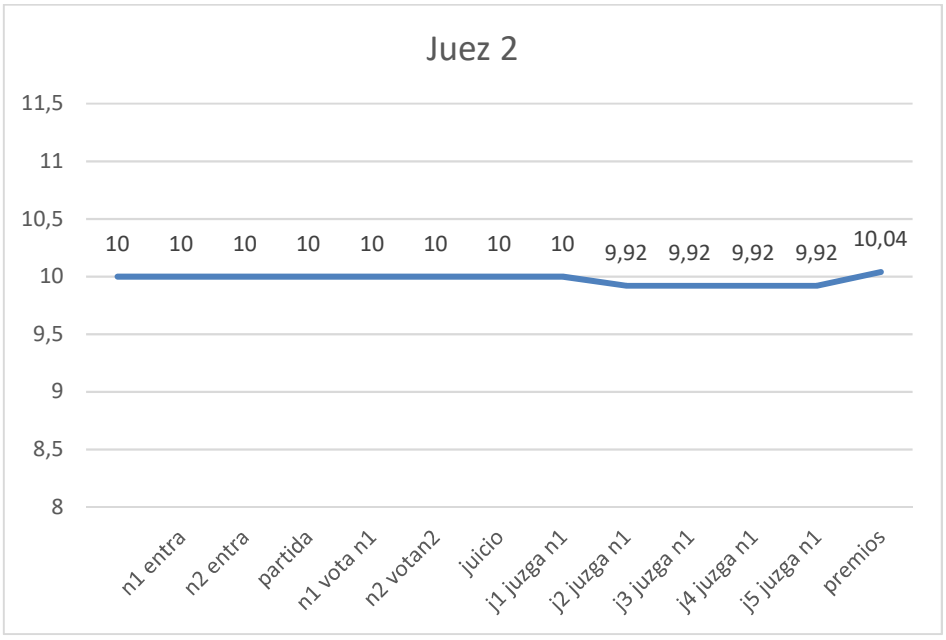


Figure 29 Gráfica del juez 2 en partidas de dos jugadores con juicio unánime sin comisiones

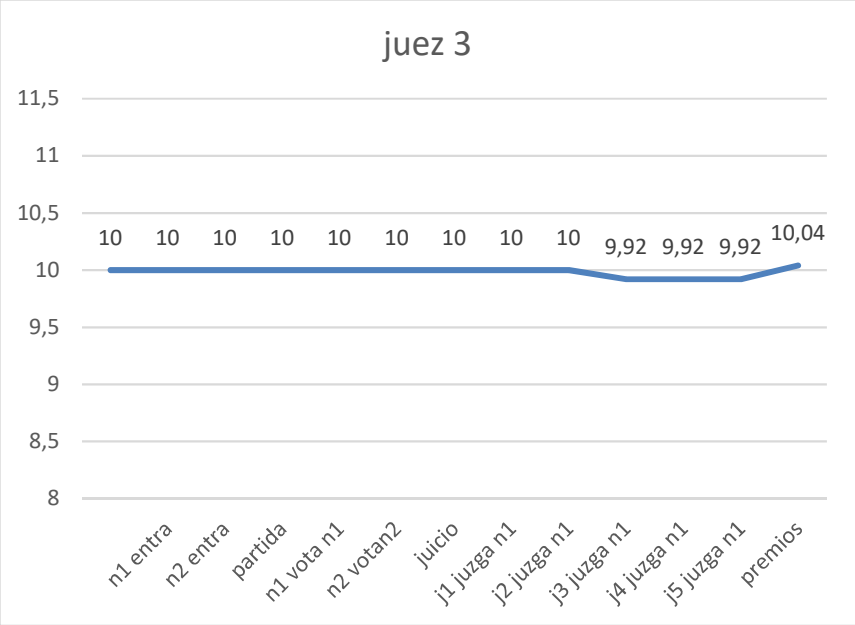


Figure 30 Gráfica del juez 3 en partidas de dos jugadores con juicio unánime sin comisiones

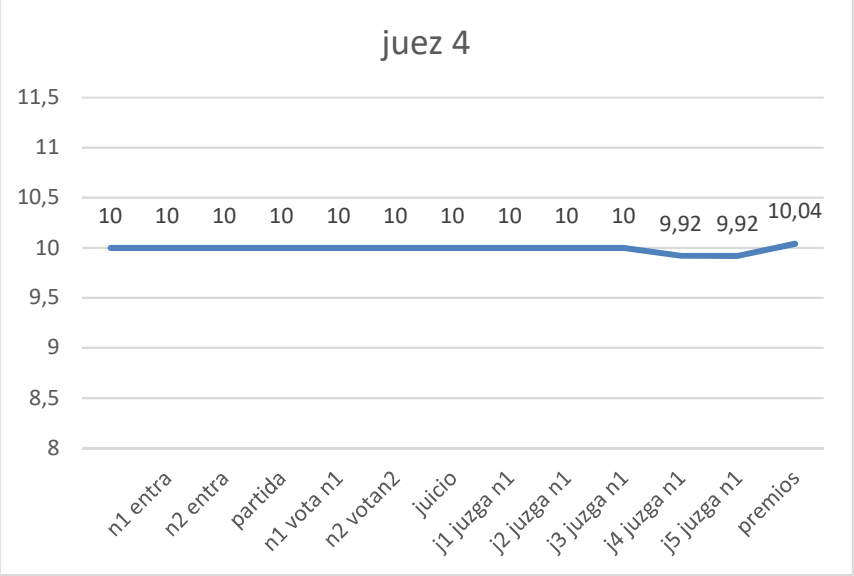


Figure 31 Gráfica del juez 4 en partidas de dos jugadores con juicio unánime sin comisiones

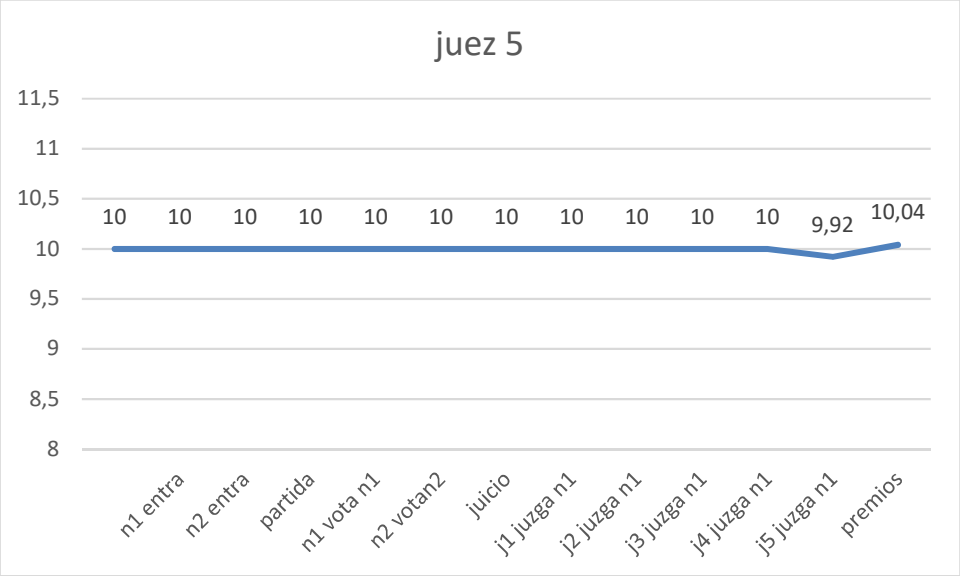


Figure 32 Gráfica del juez 5en partidas de dos jugadores con juicio unánime sin comisiones

Con comisiones:

		n1 entra	n2 entra	partida	n1 vota n1	n2 votan2	juicio	j1 juzga n1	j2 juzga n1	j3 juzga n1	j4 juzga n1	j5 juzga n1	premios
jugador 1	10	8,75	8,75	8,75	8,73	8,73	8,73	8,73	8,73	8,73	8,73	8,73	10,93
jugador 2	10	10	8,75	8,75	8,75	8,73	8,73	8,73	8,73	8,73	8,73	8,73	8,73
juez 1	10	10	10	10	10	10	10	9,9	9,9	9,9	9,9	9,9	10,02
juez 2	10	10	10	10	10	10	10	10	9,9	9,9	9,9	9,9	10,02
juez 3	10	10	10	10	10	10	10	10	10	9,9	9,9	9,9	10,02
juez 4	10	10	10	10	10	10	10	10	10	10	9,9	9,9	10,02
juez 5	10	10	10	10	10	10	10	10	10	10	10	9,9	10,02

Tabla 6 Saldos de jugadores 1 y 2 y jueces 1-5, en una partida de 2 jugadores con juicio unánime con comisiones

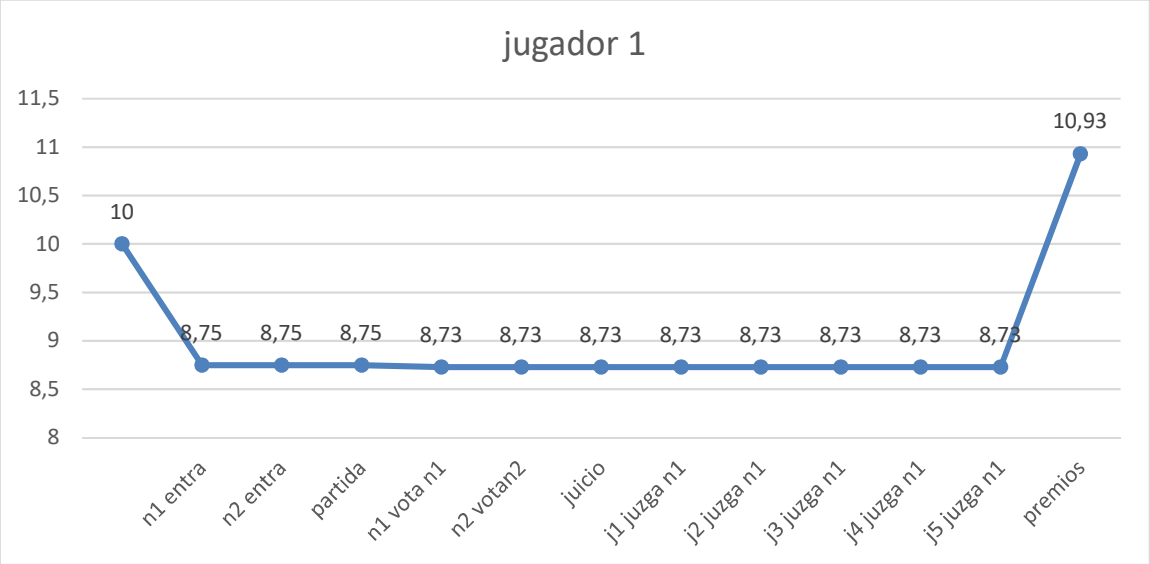


Figure 33 Gráfica del jugador 1 en partidas de dos jugadores con juicio unánime con comisiones

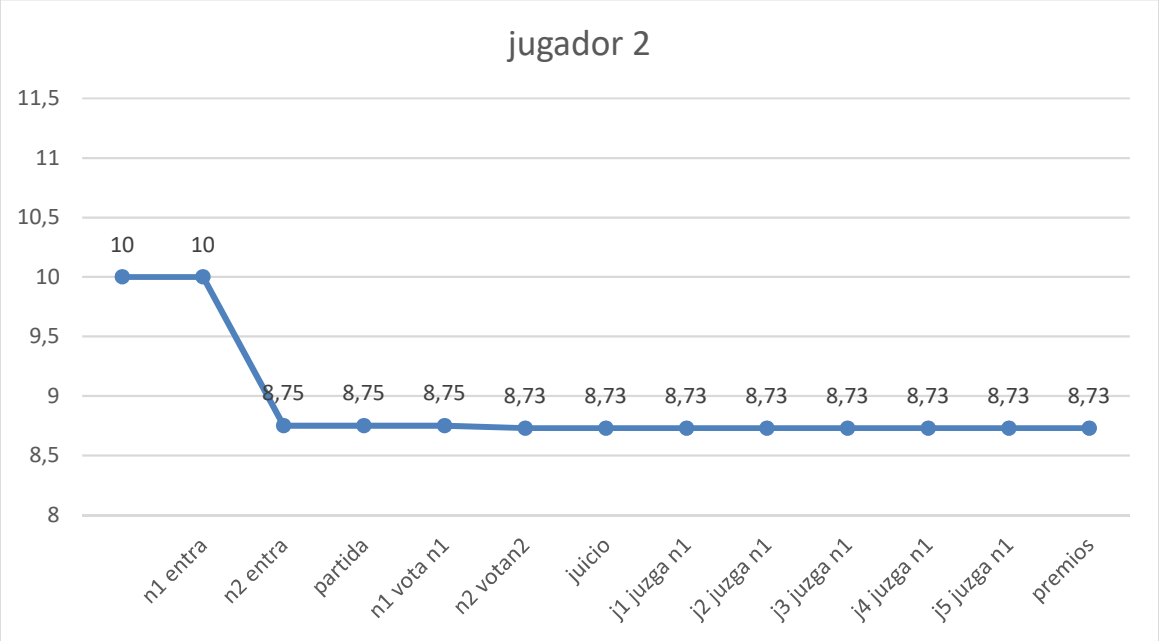


Figure 34 Gráfica del jugador 2 en partidas de dos jugadores con juicio unánime con comisiones

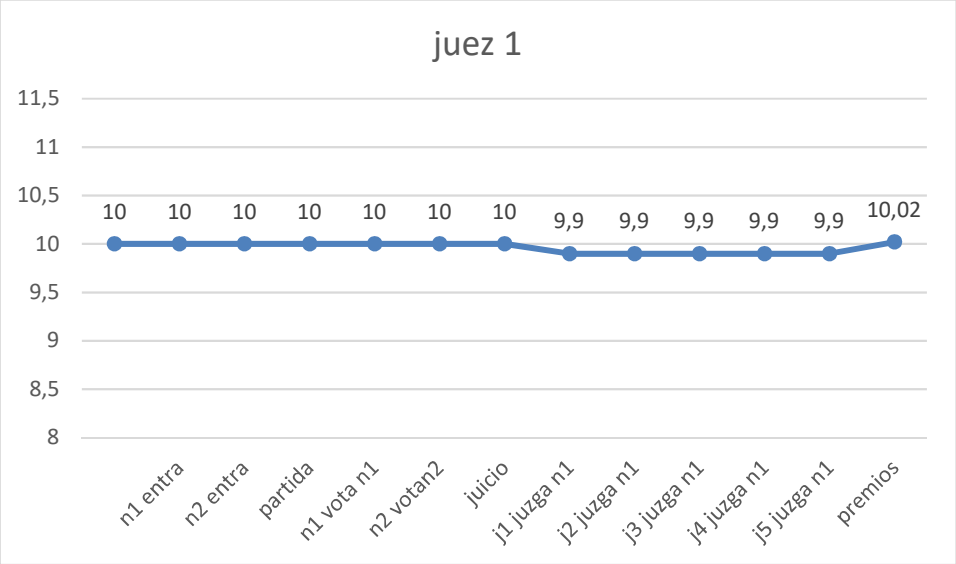


Figure 35 Gráfica del juez 1 en partidas de dos jugadores con juicio unánime con comisiones

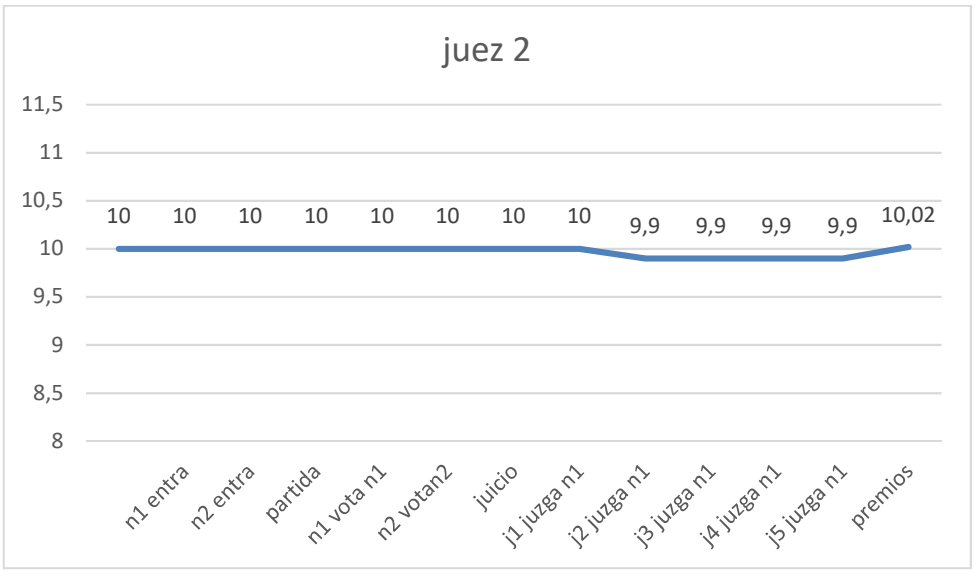


Figure 36 Gráfica del juez 2 en partidas de dos jugadores con juicio unánime con comisiones

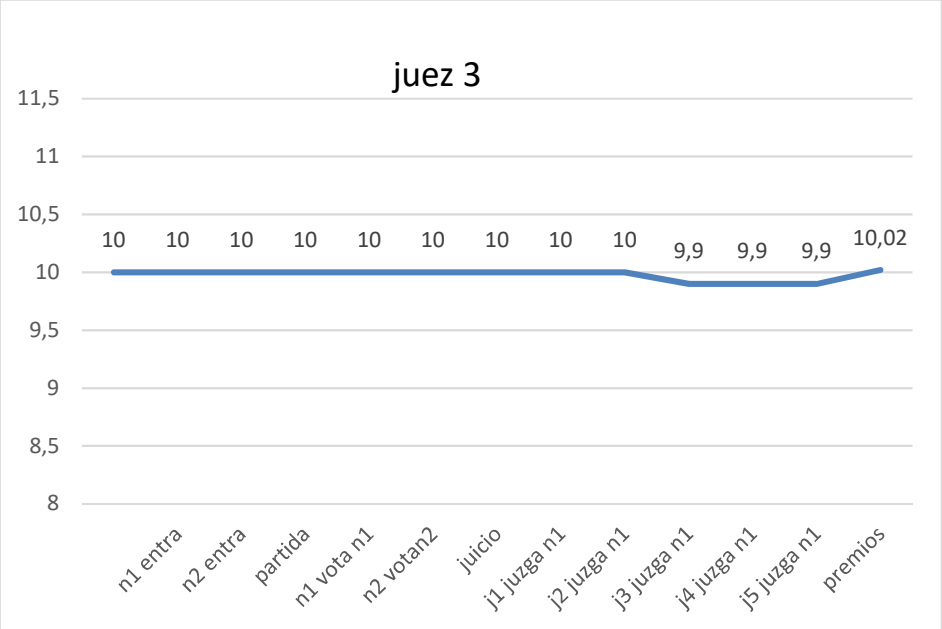


Figure 37 Gráfica del juez 3 en partidas de dos jugadores con juicio unánime con comisiones

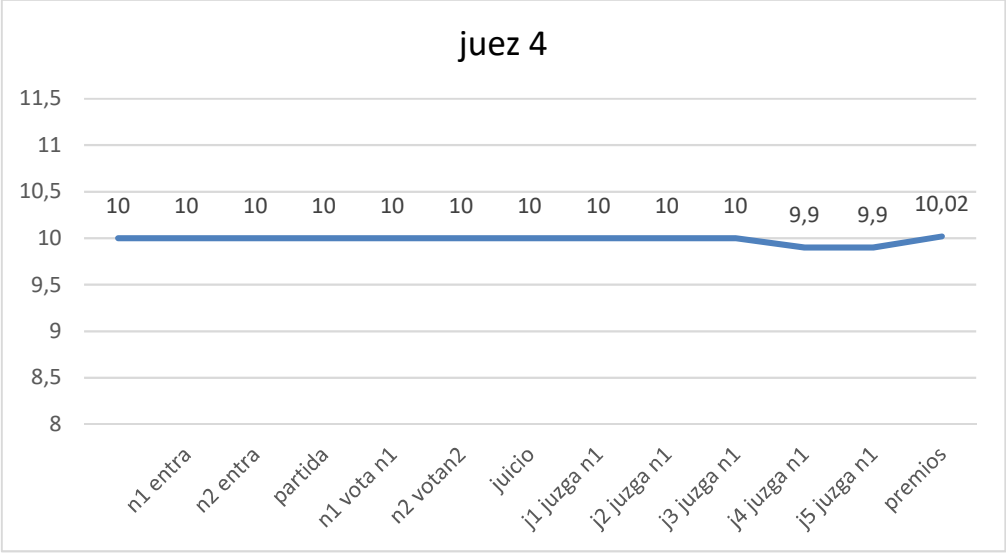


Figure 38 Gráfica del juez 4 en partidas de dos jugadores con juicio unánime con comisiones

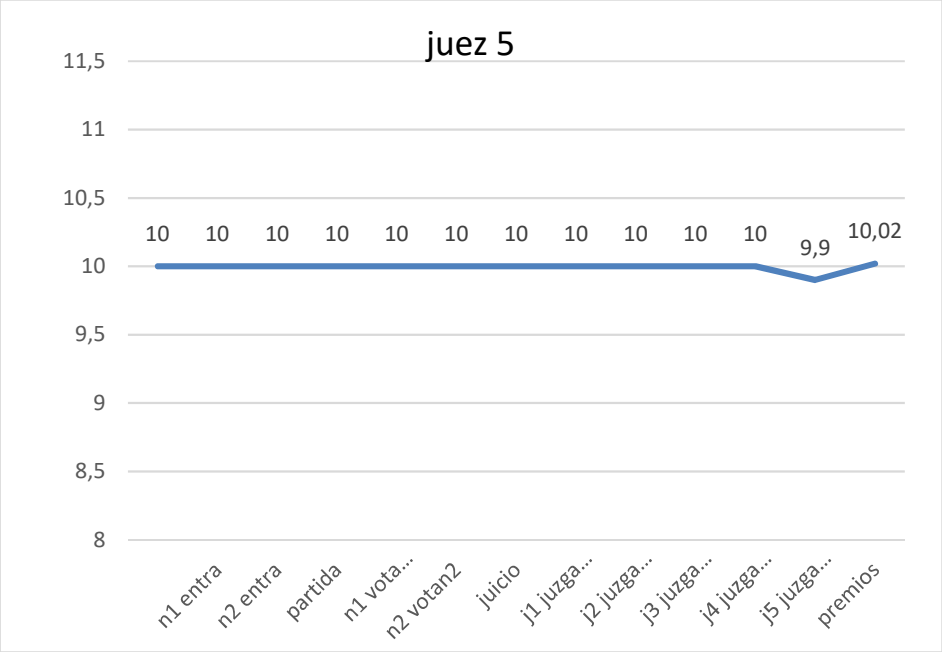


Figure 39 Gráfica del juez 5 en partidas de dos jugadores con juicio unánime con comisiones

6.1.1.3 Dos jugadores con juicio, con conflicto de jueces.

También idéntico a los dos casos anteriores. Se parte de un saldo inicial de 10€, se apuesta 1€ y se aporta una fianza del 20% de la apuesta, 20 céntimos. El jugador 1 vota al jugador 1 y el jugador vota al jugador 2. Es necesario realizar un juicio para determinar quién ha sido el verdadero ganador de esta partida. Sin embargo, en este caso una vez se ha realizado el juicio, los jueces no votan de forma unánime. Existe un conflicto entre los jueces. Los jueces 1, 2, 3 y 4 votan al jugador 1 y el juez 5 votan al jugador 2. La mayoría de jueces han votado al jugador 1 por lo que el jugador 1 es el ganador de la partida. El juez 5 al haber votado al jugador 2 pierde su fianza puesto que no pertenece a la mayoría. Los jueces 1, 2, 3 y 4 ganan 0,07€ cada uno ya que se reparten entre 4 jueces (y no entre 5) la fianza del jugador 2 (0,20€) y la fianza del juez 5 (0,08). Es decir 0,28 a repartir entre 4. El jugador 1 recupera su fianza y gana el bote, un total de 2,20€. El jugador 2 pierde la apuesta y su fianza su fianza.

- Jugadores: 2
- Juicio: sí
- Apuesta: 1€
- Fondos iniciales de jueces y jugadores: 10€
- Fianza por jugador para validación: 0,20€
- Fianza por juez para validación: 0,08€
- Recompensa para cada juez: 0,28€ / 4 = 0,07€

Sin comisiones:

		n1 entra	n2 entra	partida	n1 vota n1	n2 vota n2	juicio	j1 juzga n1	j2 juzga n1	j3 juzga n1	j4 juzga n1	j5 juzga n2	premios
jugador 1	10	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	11
jugador 2	10	10	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8	8,8
juez 1	10	10	10	10	10	10	10	9,92	9,92	9,92	9,92	9,92	10,07
juez 2	10	10	10	10	10	10	10	10	9,92	9,92	9,92	9,92	10,07
juez 3	10	10	10	10	10	10	10	10	10	9,92	9,92	9,92	10,07
juez 4	10	10	10	10	10	10	10	10	10	10	9,92	9,92	10,07
juez 5	10	10	10	10	10	10	10	10	10	10	10	9,92	9,92

Tabla 7 Saldos de jugadores 1 y 2 y jueces 1-5, en una partida sin comisiones de 2 jugadores con juicio no unánime

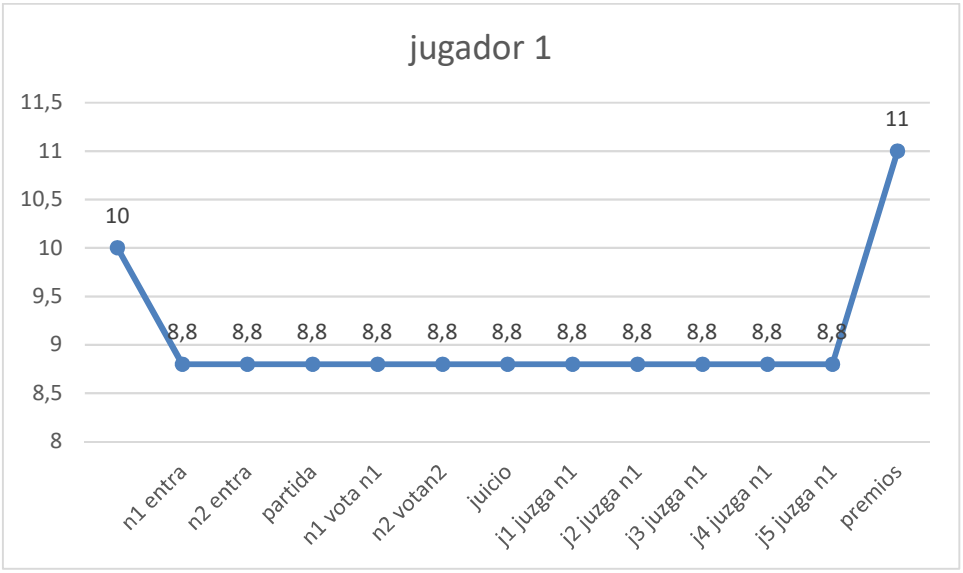


Figure 40 Gráfica del jugador 1 en partidas de dos jugadores con juicio y disputa entre jueces sin comisiones

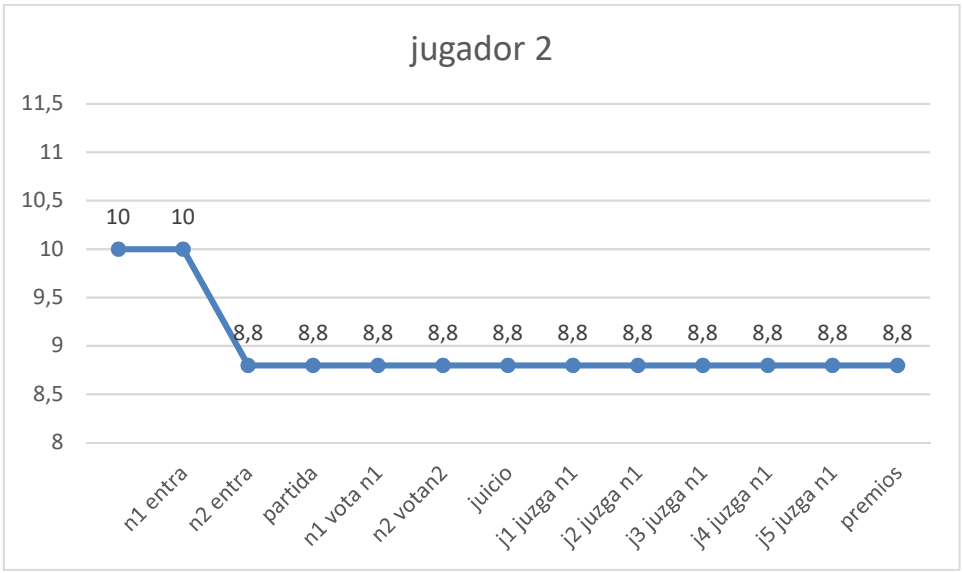


Figure 41 Gráfica del jugador 2 en partidas de dos jugadores con juicio y disputa entre jueces sin comisiones

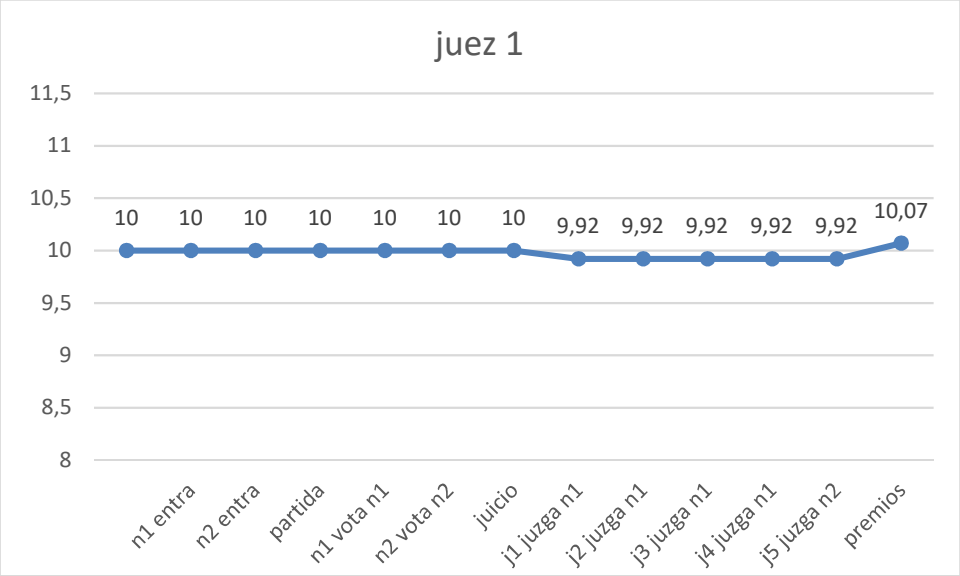


Figure 42 Gráfica del juez 1 en partidas de dos jugadores con juicio y disputa entre jueces sin comisiones

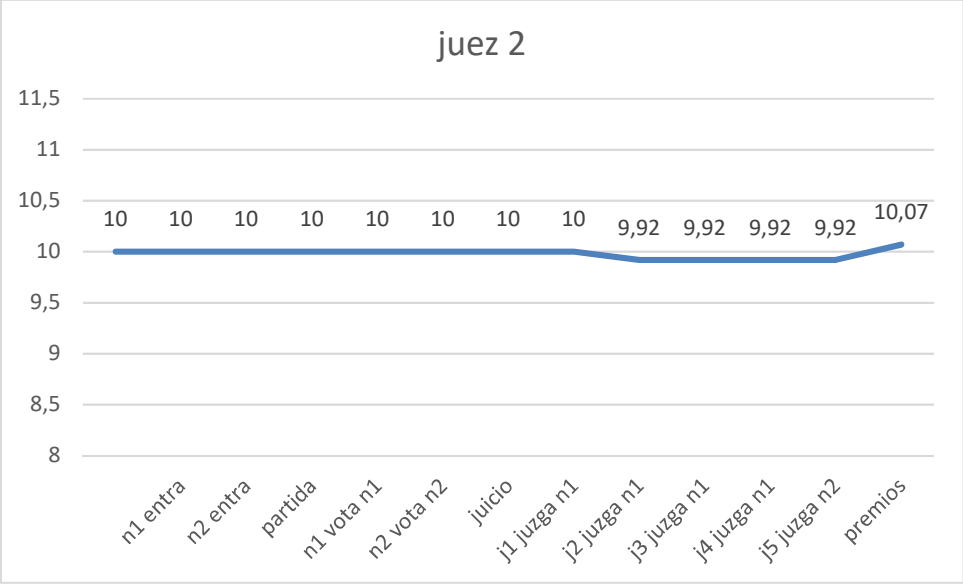


Figure 43 Gráfica del juez 2 en partidas de dos jugadores con juicio y disputa entre jueces sin comisiones

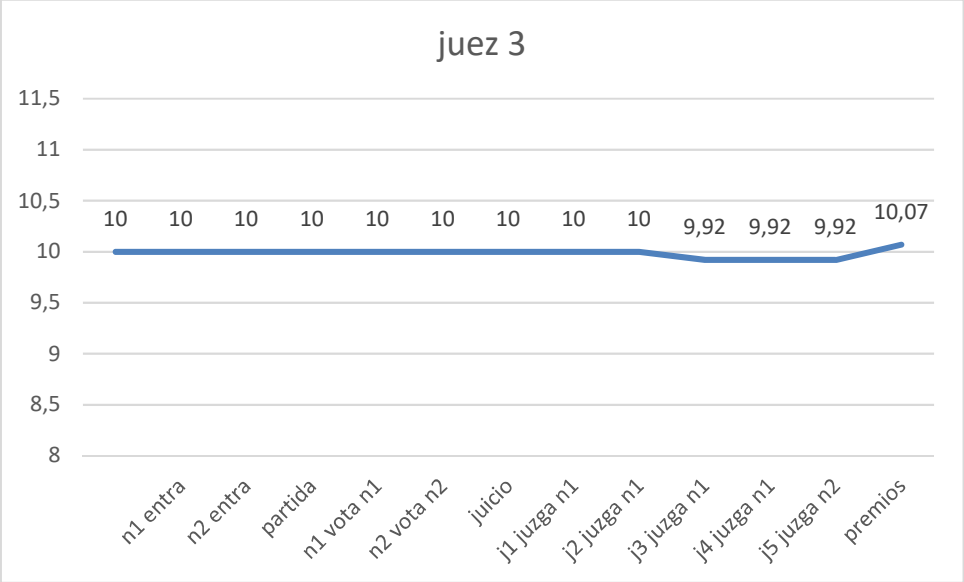


Figure 44 Gráfica del juez 4 en partidas de dos jugadores con juicio y disputa entre jueces sin comisiones

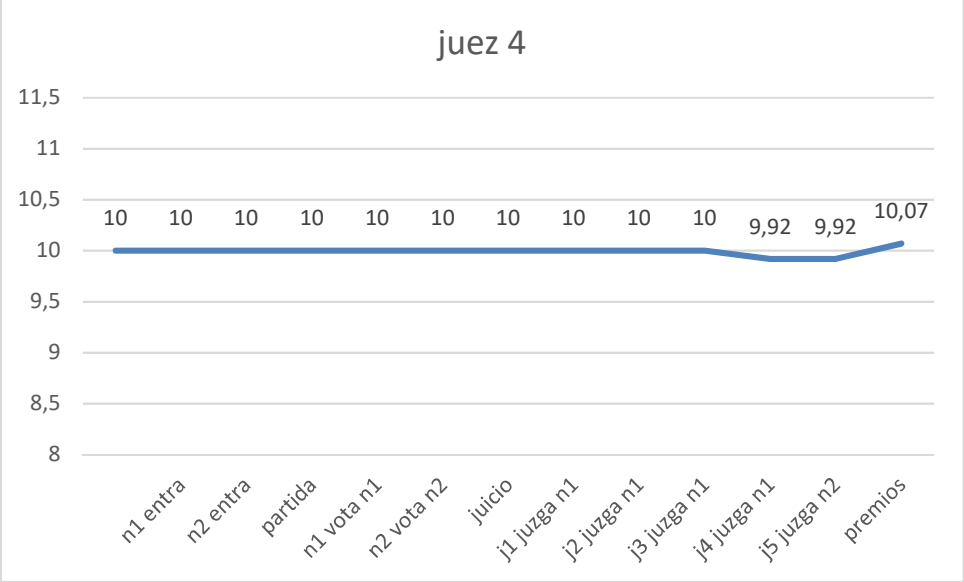


Figure 45 Gráfica del juez 5 en partidas de dos jugadores con juicio y disputa entre jueces sin comisiones

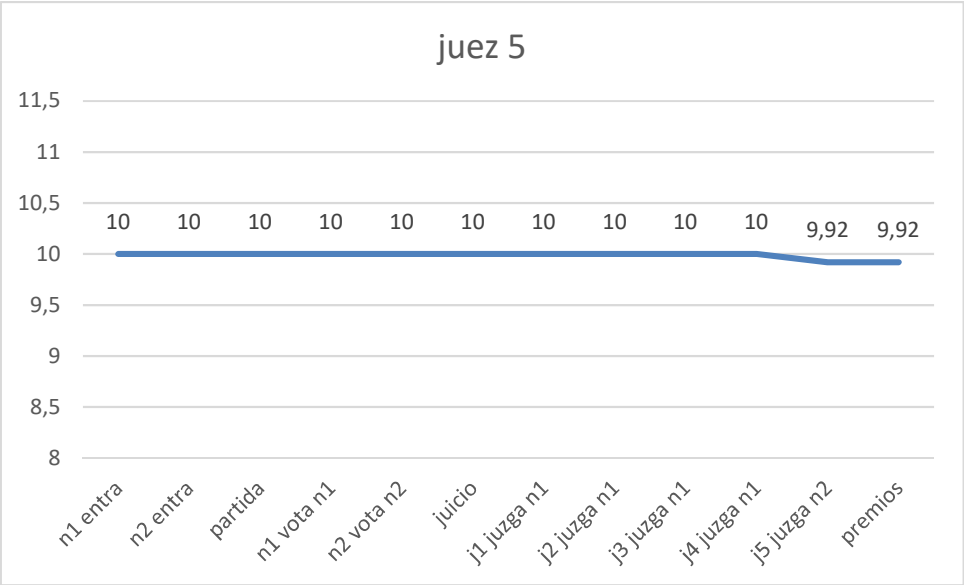


Figure 46 Gráfica del juez 5 en partidas de dos jugadores con juicio y disputa entre jueces sin comisiones

Con comisiones:

		n1 entra	n2 entra	partida	n1 vota n1	n2 votan2	juicio	j1 juzga n1	j2 juzga n1	j3 juzga n1	j4 juzga n1	j5 juzga n2	premios
jugador 1	10	8,75	8,75	8,75	8,73	8,73	8,73	8,73	8,73	8,73	8,73	8,73	10,93
jugador 2	10	10	8,75	8,75	8,75	8,73	8,73	8,73	8,73	8,73	8,73	8,73	8,73
juez 1	10	10	10	10	10	10	10	9,9	9,9	9,9	9,9	9,9	10,05
juez 2	10	10	10	10	10	10	10	10	9,9	9,9	9,9	9,9	10,05
juez 3	10	10	10	10	10	10	10	10	10	9,9	9,9	9,9	10,05
juez 4	10	10	10	10	10	10	10	10	10	10	9,9	9,9	10,05
juez 5	10	10	10	10	10	10	10	10	10	10	10	9,9	9,9

Tabla 8 Saldos de jugadores 1 y 2 y jueces 1-5 en una partida de 2 jugadores con juicio y disputa entre jueces con comisiones

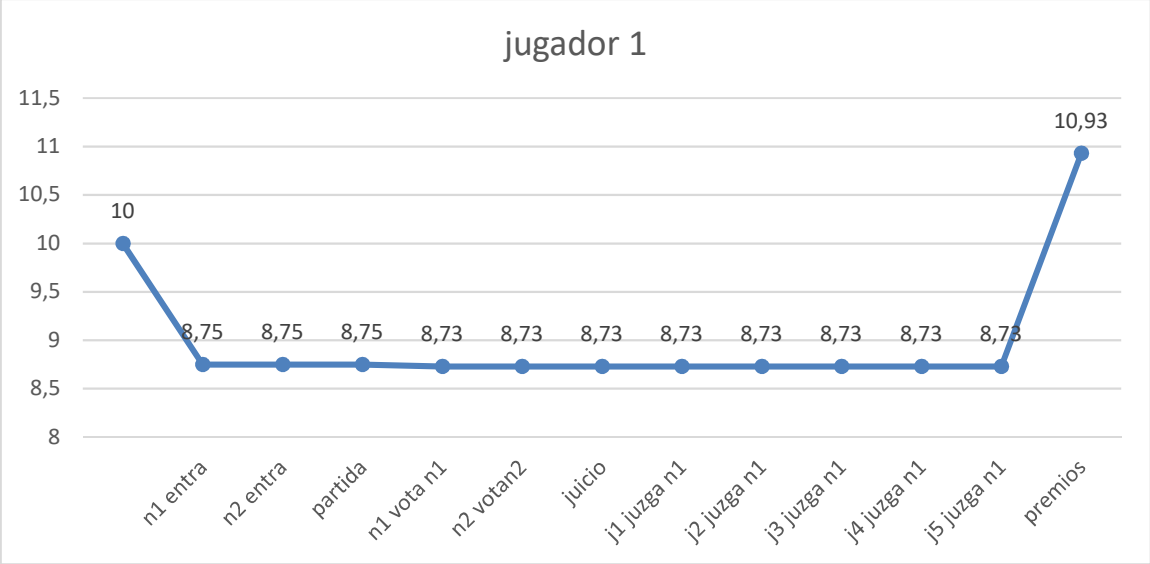


Figure 47 Gráfica del jugador 1 en partidas de dos jugadores con juicio y disputa entre jueces con comisiones

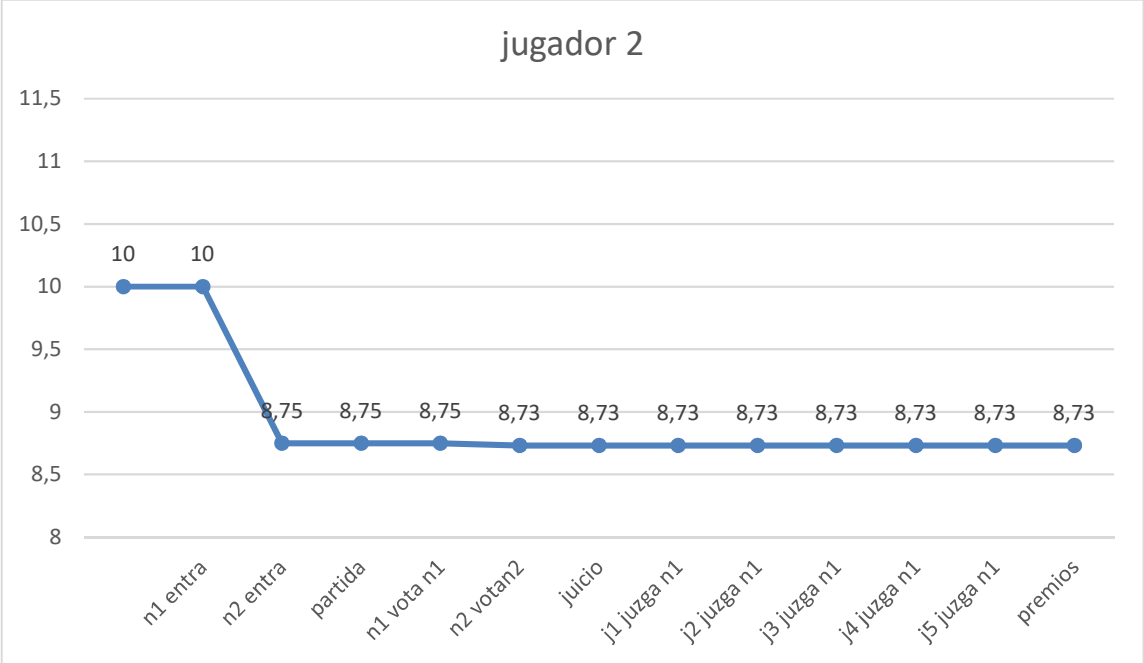


Figure 48 Gráfica del jugador 2 en partidas de dos jugadores con juicio y disputa entre jueces con comisiones

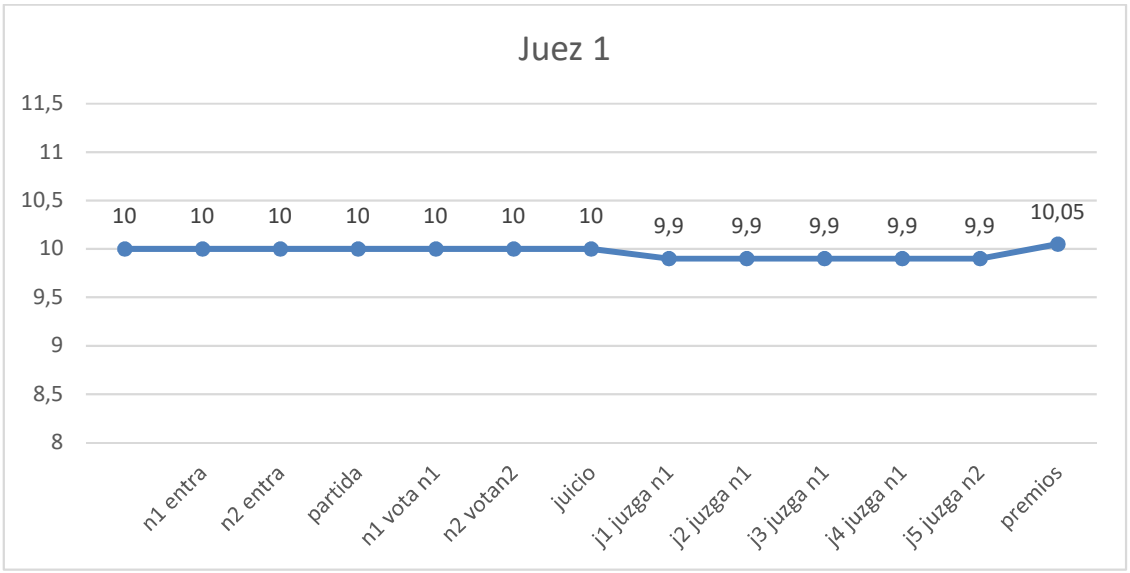


Figure 49 Gráfica del juez 1 en partidas de dos jugadores con juicio y disputa entre jueces con comisiones

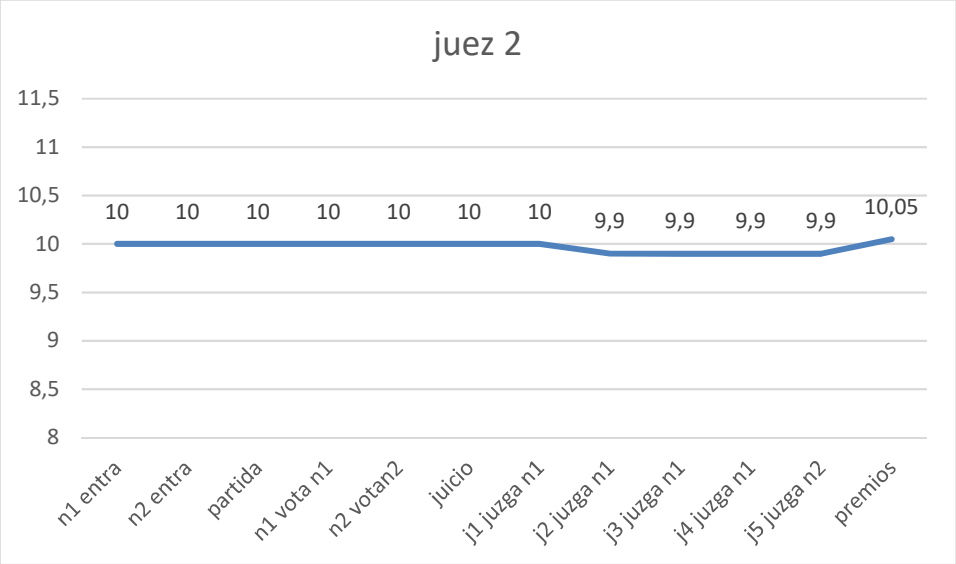


Figure 50 Gráfica del juez 2 en partidas de dos jugadores con juicio y disputa entre jueces con comisiones

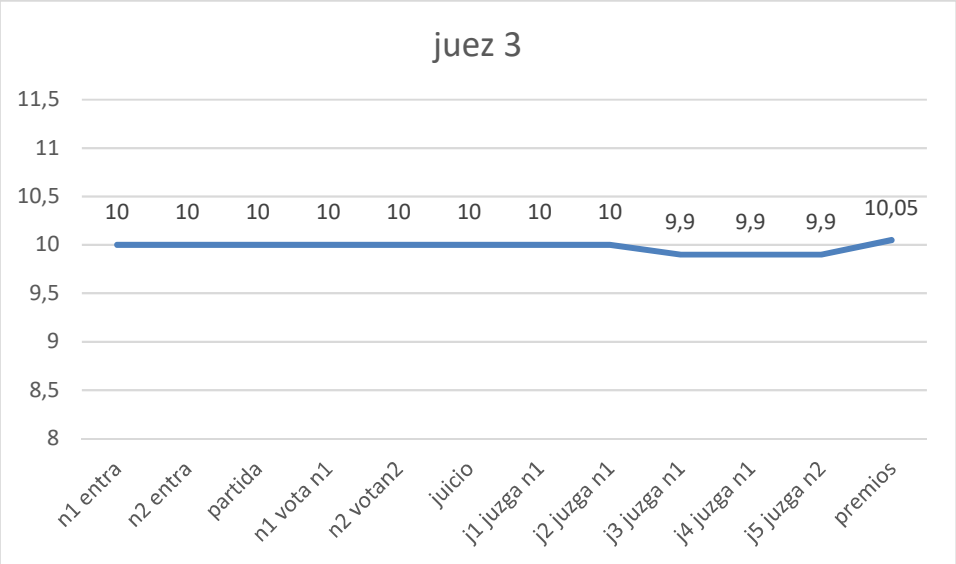


Figure 51 Gráfica del juez 3 en partidas de dos jugadores con juicio y disputa entre jueces con comisiones

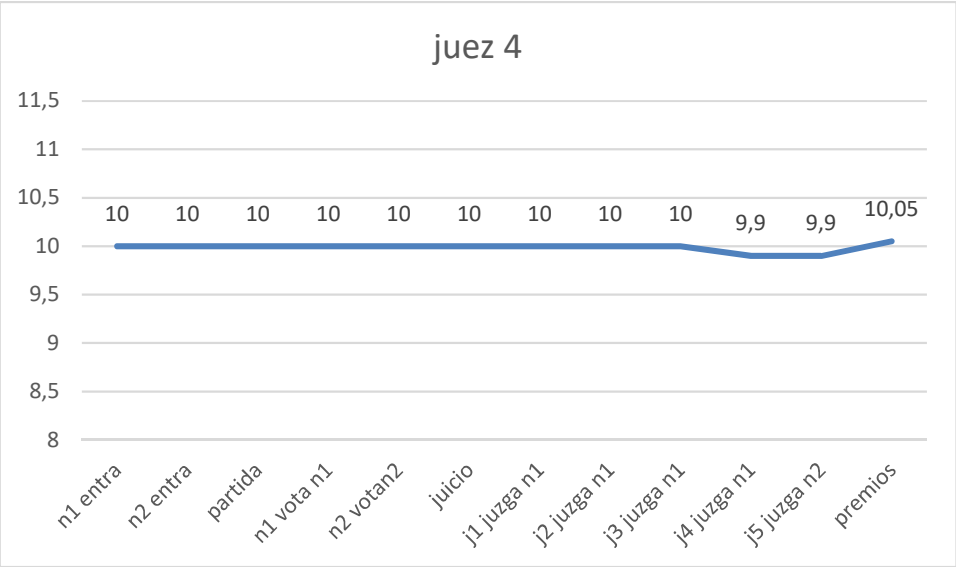


Figure 52 Gráfica del juez 4 en partidas de dos jugadores con juicio y disputa entre jueces con comisiones

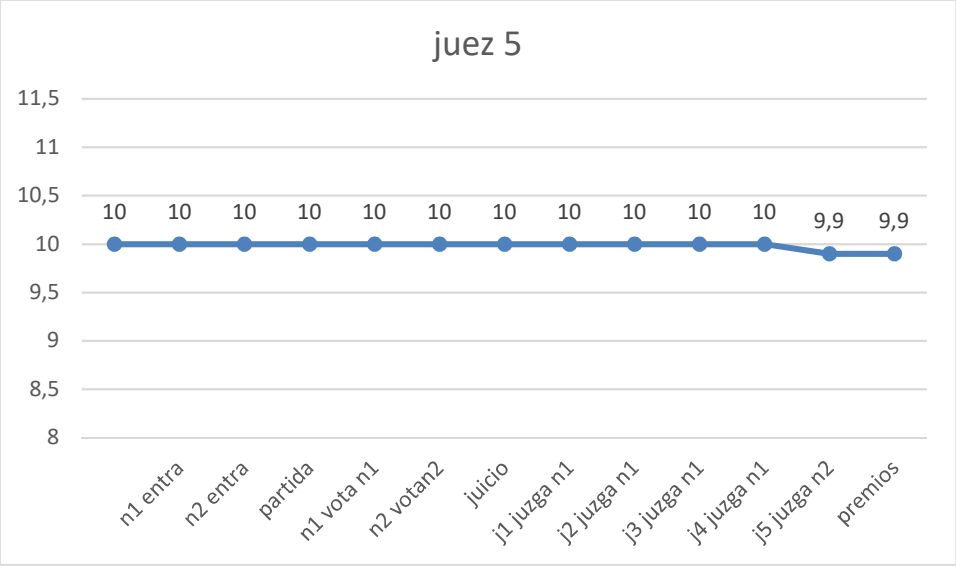


Figure 53 Gráfica del juez 5 en partidas de dos jugadores con juicio y disputa entre jueces con comisiones

6.1.2 CASOS DE USO CON MUCHOS JUGADORES (>4)

En este tipo de partidas, la fianza que aporta cada jugador en concepto de validación (con el objetivo de pagar un juicio si esté llegase a realizarse) no paga completamente el juicio. El coste del juicio, al ser el número de jugadores muy grande y en consecuencia el número de jueces muy grande, es muy elevado. Para poder costear el juicio es necesario utilizar un pequeño porcentaje del bote total de la apuesta. Este porcentaje debe ser suficientemente pequeño como para que el ganador apenas note el cambio. Un porcentaje aceptable debería estar en el rango de 0-10%.

Un juego que recientemente ha ganado mucho tráfico y popularidad es Fornite. En éste, 100 jugadores juegan todos contra todos. El último que quede con vida gana la partida.

6.1.2.1 Cien jugadores sin juicio.

100 jugadores desconocidos entre sí desean jugar una partida de Fornite en la que cada jugador apueste 1 euro. El ganador se lleva 100 euros. Gainatgame les permite encontrar una partida que cumpla estos requisitos, es decir, pone en contacto a estos 100 jugadores desconocidos para que jueguen la partida. Una vez los 100 jugadores se encuentran en partida, está comienza. Cuando la partida ha finalizado cada jugador vota al ganador. Se necesita un 80% de mayoría en los votos de los jugadores para determinar un ganador y no ir a juicio.

- Juicio: no
- Apuesta: 1€
- Fondos iniciales de jugadores: 10€
- Fianza por jugador para validación: 0,20€

Tras finalizar la partida los jugadores votan: Un 80% de los jugadores votan al jugador 1, un 5% votan al jugador 2, un 5% votan al jugador 100 y un 10% no vota. El porcentaje de votos es finalmente del 88% del total de votos emitidos. Como se ha alcanzado una mayoría del 88%

no es necesario un juicio. El jugador 1 es el ganador de la apuesta. Este recibe 100 euros (el premio por ganar) y su fianza para validación (0,20€). Como no ha sido necesario un juicio todos los jugadores reciben de vuelta su fianza para la validación. Cabe remarcar que, aunque los jugadores hayan sido deshonesto (votando a un jugador distinto del jugador 1) o no hayan votado, estas acciones no han supuesto ningún perjuicio a ningún otro jugador ya que el jugador 1 recibe su premio íntegramente al no haberse requerido un juicio.

6.1.2.2 Cien jugadores con juicio unánime

Idéntico al caso anterior, sin embargo, en éste, en las votaciones se obtiene un 70% de votos al jugador 1, un 5% al jugador 2, un 5% al jugador 3, un 5% al jugador 4, un 5% al jugador 5, un 5 % al jugador 100 y un 5% no vota. El porcentaje de votos para el jugador 1 es del 73% de los votos emitidos. Como no se ha obtenido una mayoría igual o superior al 80% se requiere un juicio. Para determinar el número de jueces necesarios hay que tener en cuenta que, aunque haya 100 jugadores en la partida, solo existe disputa entre 6 de ellos (todos los votos van dirigidos a los jugadores 1, 2, 3, 4, 5, 100) Siguiendo con lo especificado en el capítulo 5, con 29 jueces será suficiente. [jueces = (jugadores votados * 6) -7].

- Apuesta: 1€
- Juicio: sí
- Número de jueces: 29
- Fondos iniciales de jugadores y jueces: 10€
- Fianza por jugador para validación: 0,20€
- Necesario para validación (5% de 100€): 5€
- Cantidad máxima utilizada del bote (5 - 80%*100*0,2): 1€
- Recompensa mínima para cada juez: 5€ / 29 = 0,17€
- Fianza por juez para validación (el doble de la recompensa mínima): 0,34€

Una vez el juicio comienza, todos los jueces votan de forma unánime al jugador 1. El jugador es declarado el ganador de la apuesta. Recibiendo 100€ (bote del premio) y 0,20€ (su fianza para

validación). El 70% de jugadores que votaron al jugador 1 reciben de vuelta su fianza para validación. El 30% de jugadores que votaron diferente del jugador 1 o no votaron no reciben de vuelta su fianza puesto que es su responsabilidad que se haya necesitado un juicio. En este caso un 30% de los jugadores no han sido honesto lo que supone que 6€ vayan destinados a validación. Cada juez recibe (6€ /29) 0,20€ siendo esto mayor a lo esperado ya que el porcentaje ha sido del 70% y no del 80%. No ha sido necesario utilizar nada del bote ya que el número de jugadores deshonestos (30 jugadores) ha sido mayor al mínimo (20 % del total de jugadores).

6.1.2.3 Cien jugadores con juicio y conflicto de jueces

Idéntico al caso anterior, sin embargo, en éste, una vez el juicio se celebra no existe unanimidad en las votaciones de los jueces. Un 80% de los jueces ha votado al jugador 1 y un 20% al jugador 2. Hay 29 jueces. 23 han votado al jugador 1 y 6 al jugador 2. En este caso, los jueces que no pertenece a la mayoría no recuperan su fianza. Estas fianzas se reparten entre los jueces honestos. Es decir, 6€ de las fianzas de los jugadores + 0,34€ * 6. Un total de 8,06 €. Lo que supone un total de 0,35€ por juez (8,06/23). Los jueces honestos también reciben de vuelta su fianza por lo que reciben un total de 0,69€ (0,35€ + 0,34€). El jugador 1 recibe 100,20€ (bote + fianza). Y los jugadores pertenecientes al 70% que ha votado al jugador 1 recibe de vuelta 0,20€ (fianza).

En el siguiente esquema se aprecian tanto los pagos de jugadores y jueces como los balances finales de los mismo tras completar la partida:

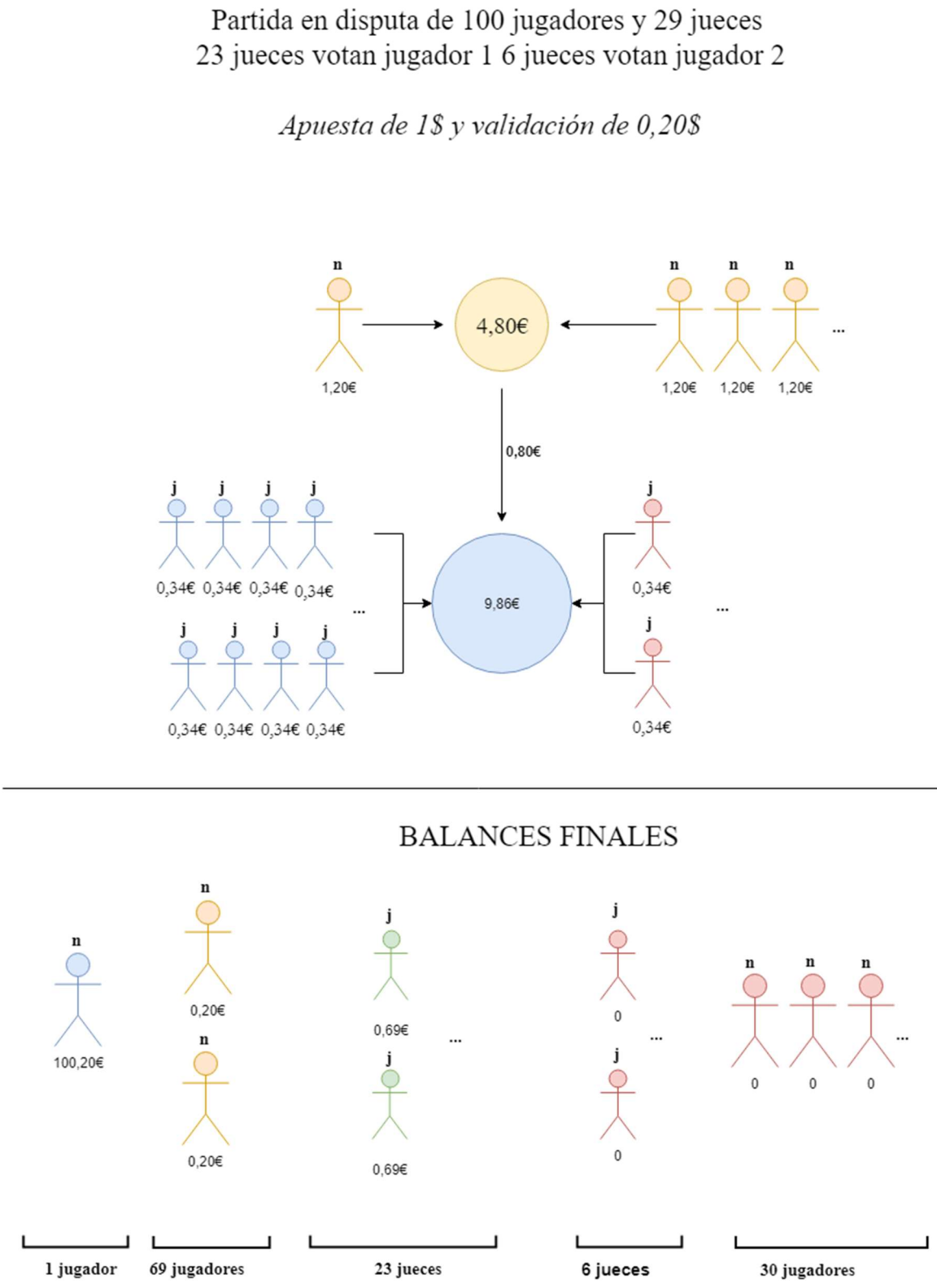


Figure 54 Diagrama y balances finales de una partida con 100 jugadores, juicio y conflicto entre jueces

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

El objetivo de este proyecto ha sido el de analizar, diseñar y desarrollar un sistema innovador que aporte valor real a los usuarios permitiéndoles realizar apuestas de forma fiable y segura incluso con desconocidos. Este objetivo se ha cumplido en tanto que se ha conseguido diseñar, desarrollar e implementar una plataforma que funciona, tiene una experiencia de usuario competitiva, permite su uso en tiempo real (gracias a que sus comisiones son bajas), posee una interfaz de usuario responsive y fácil de utilizar y, sobre todo, posee una lógica blindada en la que se han tenido en cuenta todos los posibles casos y acciones por parte de los usuarios para asegurar que las apuestas se realizan correctamente y que éstas sean justas para todas las partes.

Se ha realizado una investigación y análisis extensivo sobre Blockchain con el fin de comprender qué supone la integración de esta tecnología en mercados tradicionales como el de las apuestas y el de los videojuegos. Además, gracias a un estudio del mercado de las apuestas, analizando los competidores existentes y el gran potencial que éste muestra, se ha conseguido demostrar por qué Gainatgame es innovador y se encuentra en un nicho casi sin explotar.

Algunos trabajos y mejoras a realizar en el futuro son:

- En la plataforma que se ha desarrollado es posible realizar apuestas tanto en criptomonedas como en tokens. El siguiente paso a seguir en Gainatgame es la realización de la ITO con el fin de que los usuarios puedan adquirir estos tokens. Las ITOs son procesos complejos, costosos y burocráticos. Se estima que una ITO efectiva puede costar en torno a $50 mil.
- Actualmente la plataforma solo cubre partidas públicas entre desconocidos. Sería deseable que los usuarios puedan crear y configurar sus propias partidas privadas. Así como restringir el acceso a ellas a quien deseen. Esta nueva funcionalidad ya la suministra el Smart Contract sin embargo no existe front-end que lo respalde.
- Desarrollar un sistema de rating tanto de usuarios como de jueces donde se valore su honestidad en partidas anteriores. De esta forma, si un juez ha no ha actuado

honestamente en juicios anteriores, se le podrán ofrecer menos juicios en el futuro en incluso prohibirle volver a ser juez.

- Desarrollar un sistema de incentivos, similar al diseñado para el host, en el que se premie a los jugadores que suban pruebas visuales de buena calidad. Cuando un juez este juzgando las pruebas este podrá seleccionar de qué prueba ha sacado la información en la que se ha basado para el veredicto. La compensación sería económica y saldría del bote total de la apuesta en partidas de muchos jugadores. En partidas de pocos jugadores no sería necesario.
- Desarrollar un sistema de Elo para medir la habilidad de los usuarios en los distintos juegos. El objetivo es que los emparejamientos entre usuarios sean más justos. Sin embargo, si se empareja a los mejores con los mejores y los peores con los peores, los peores tendrían una ventaja frente a los mejores. Esto se soluciona ofreciendo otras ventajas auxiliares a los mejores que compenses esta división por nivel. Se podría reducir las comisiones cobradas por la plataforma cuanto mayor nivel se tenga y mejor sea el usuario. Otra opción es limitar el tamaño de apuesta según el nivel, de forma que a medida que un usuario suba de nivel y de habilidad vaya desbloqueado nuevos tamaños de apuesta.
- De forma paralela al punto anterior, se podría diseñar un sistema VIP. Un sistema VIP es un sistema de recompensas que premia a un usuario de forma proporcional a su actividad en la plataforma. Por ejemplo, en casas de apuestas, a medida que se va apostando y utilizando su plataforma, la casa regala premios, descuentos, ofertas, etc.… De esta forma, el usuario recibe de vuelta una parte de los beneficios que la plataforma ha ingresado gracias a las acciones de ese mismo usuario.
- Desarrollar un sistema que permita a los usuarios consultar su histórico de partidas y conocer sus estadísticas en la plataforma.
- Incluir Fortmatic como alternativa a MetaMask. Fortmatic está diseñado para que no requiera ningún esfuerzo para el desarrollador y sea totalmente compatible con una Dapp que trabaje con MetaMask. Sin embargo, en este proyecto se ha determinado que para simplificar lo máximo posible el proceso de comunicación con Blockchain para el usuario, solo se utilice MetaMask.

- Gainatgame se centra en juegos de suma cero con un único ganador. Sin embargo, es habitual que en las apuestas y en las competiciones, no solo gane un premio el primero. Es posible otorgar premios, por ejemplo, a los tres primeros. Para conseguir esta nueva funcionalidad habría que rediseñar la función del Smart Contract encargada de repartir los premios. Además, si se desea que los premios sean específicos de cada juego es necesario incorporar estos nuevos valores al Smart Contract por lo que sería necesario desplegarlos de nuevo.
- Funcionalidad de Challenge propuesta y explicada en el Capítulo 5.

Capítulo 8. BIBLIOGRAFÍA

1. Proof of toss white paper. (2019, April 23). Retrieved June 25, 2019, from https://neironix.io/documents/whitepaper/4238/White_Paper_PROOF-OF-TOSS-WHITEPAPER.pdf
2. Juego Online. (n.d.). Retrieved June/July, 2019, from https://www.epdata.es/datos/juego-online-espana-datos-estadisticas/161
3. Fenech, G. (2019, January 30). Blockchain In Gambling And Betting: Are There Real Advantages? Retrieved July 10, 2019, from https://www.forbes.com/sites/geraldfenech/2019/01/30/blockchain-in-gambling-and-betting-are-there-real-advantages/#17b92be27c63
4. Dirección General de Ordenación del Juego. (n.d.). Retrieved June/July, 2019, from https://www.ordenacionjuego.es/es/estudios-informes
5. Dirección General de Ordenación del Juego. (n.d.). Retrieved July 10, 2019, from https://www.ordenacionjuego.es/es/estudio-prevalencia
6. Dirección General de Ordenación del Juego. (n.d.). Retrieved July 10, 2019, from https://www.ordenacionjuego.es/es/descarga-datos-mercado-juego-online
7. Tiempo de NegociosMedio de comunicación dedicado a los negocios. "Sistemas De Pago Electrónico Seguros En La Actualidad." Tiempo De Negocios, 18 Mar. 2019, tiempodenegocios.com/sistemas-de-pago-electronico-segur
8. "PayPal: Volumen Anual De Pagos Móviles Mundiales 2018." Statista, es.statista.com/estadisticas/635953/evolucion-del-volumen-de-pagos-moviles-mundiales-realizados-con-paypal/.
9. "Previsión Del Volumen De Pagos Móviles 2016-2019." Statista, es.statista.com/estadisticas/681592/prevision-del-volumen-de-pagos-moviles-en-el-mundo/.
10. "Fraude Online: Porcentaje De Empresas Por Tasa De Fraude 2016." Statista, es.statista.com/estadisticas/589080/empresas-que-fueron-victimas-de-fraude-online-por-tasa-de-fraude-en-espana/.

11. "59% Del Total De Fraudes En 2018 Fueron Por Medios Cibernéticos: Condusef: Marketing 4 Ecommerce - Tu Revista De Marketing Online Para e." Commerce, 18 Oct. 2018, marketing4ecommerce.mx/59-del-total-fraudes-en-2018-fueron-medios-ciberneticos-condusef/.

12. Swan. Melanie Editorial: O'Reilly Media (2015). Blockchain, Blueprint for a new economy

13. "Centralizado, Descentralizado y Distribuido (p2p) ¿Cuáles Son Las Diferencias?" Desarrolloactivo.com, desarrolloactivo.com/articulos/centralized-decentralized-distributed-p2p/.

14. /@VitalikButerin, Vitalik Buterin. "The Meaning of Decentralization - Vitalik Buterin." Medium, Medium, 6 Feb. 2017, medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274.

15. D, Safelayer Marketing. "Beneficios De La Tecnología De Clave Pública." Safelayer, Safelayer, 26 Apr. 2016, www.safelayer.com/es/recursos/66-articulos/infrastructuras-de-clave-publica/454-beneficios-de-la-tecnologia-de-clave-publica.

16. Bitcoin. bitcoin.org/bitcoin.pdf. Accessed 2019.

17. https://ethereum.stackexchange.com/questions/550/which-cryptographic-hash-function-does-ethereum-use

18. Artemine. (n.d.). Retrieved May/June, 2019, from http://www.artemine.org/

19. Vitalik Buterin - Vida del joven genio creador de Ethereum. (n.d.). Retrieved March/April, 2019, from https://www.miethereum.com/vitalik-buterin/

20. Proof of Work - What it Is and How Does it Work? - Async Labs: Software development and Marketing agency. (2018, July 31). Retrieved April/May, 2019, from https://www.asynclabs.co/blog/proof-of-work-what-it-is-and-how-does-it-work/

21. Antonopoulus, Andreas. **Editorial**: O'Reilly Media (2017, 2ª ed.). , Mastering Bitcoin

22. Antonopoulus, Andreas. Wood, Gavin.-**Editorial**: O'Reilly Media (2018)., Mastering Ethereum: Building Smart Contracts and Dapps

23. E. (n.d.). Whitepaper-Ethereum. Retrieved April/May, 2019, from https://github.com/ethereum/wiki/wiki/[Spanish]-White-Paper.md has

24. /@luith. (2019, February 02). The Ethereum Virtual Machine-How does it work? - MyCrypto. Retrieved April/May, 2019, from https://medium.com/mycrypto/the-ethereum-virtual-machine-how-does-it-work-9abac2b7c9e

25. Ethereum and Smart Contracts. (n.d.). Retrieved March/April, 2019, from https://steemit.com/cryptocurrency/@encryptcy/ethereum-denominations

26. ArcBlock Follow. (2018, September 27). Introduction to Ethereum Smart Contracts. Retrieved April/May, 2019, from https://www.slideshare.net/ArcBlock/introduction-to-ethereum-smart-contracts

27. Solidity - El lenguaje de programación de los Smart Contracts y Ethereum. (2019, May 27). Retrieved March/April, 2019, from https://www.miethereum.com/smart-contracts/solidity/

28. El estándar ERC-20 de Ethereum para Tokens. (2019, May 27). Retrieved March/April, 2019, from https://www.miethereum.com/smart-contracts/erc20/

29. ¿Qué es un ERC721? Conoce las características básicas de los tokens coleccionables de Ethereum: CriptoNoticias - Bitcoin, blockchains y criptomonedas. (2019, June 20). Retrieved February/March, 2019, from https://www.criptonoticias.com/redes-protocolos/token-coleccionable-aspectos-basicos-estandar-erc721/

30. /@ico_land. (2017, September 19). ICO or ITO? What's the difference? - IcoLand. Retrieved March 1, 2019, from https://medium.com/@ico_land/ico-or-ito-whats-the-difference-4055ce345bbd

31. Invertir en 'tokens' para financiar 'startups'. (2017, December 19). Retrieved January/February, 2019, from https://startupxplore.com/es/blog/invertir-tokens/

32. ErnestoB. (2019, April 13). ¿Que son las Initial Exchange Offerings (IEO)? las exchanges a la carga. Retrieved May/June, 2019, from https://bitcoin.es/criptomonedas/que-son-las-initial-exchange-offerings-ieo-las-exchanges-a-la-carga/

33. ETH Gas Station. (n.d.). Retrieved May/June 2019, from https://ethgasstation.info/

34. Scalable Blockchain Infrastructure. (n.d.). Retrieved March 9, 2019, from https://infura.io/

35. E. (n.d.). RPC 2.0 Specification. Retrieved March 9, 2019, from https://www.jsonrpc.org/specification

36. Antonopoulus, Andreas. Wood, Gavin.-**Editorial**: O'Reilly Media (2018)., Mastering Ethereum: Building Smart Contracts and Dapps

37. Web3.js - Ethereum JavaScript API. (n.d.). Retrieved March 9, 2019, from https://web3js.readthedocs.io/en/1.0/

38. Wallets. (n.d.). Retrieved March/April, 2019, from https://docs.ethhub.io/using-ethereum/wallets/web/

39. Mist Ethereum: Apps. (n.d.). Retrieved March/April, 2019, from https://electronjs.org/apps/mist

40. "MetaMask." GitHub, github.com/MetaMask.

41. Truffle Suite. (n.d.). Ganache. Retrieved March/April, from https://www.trufflesuite.com/ganache

42. Truffle Suite. (n.d.). Truffle. Retrieved March/April, from https://www.trufflesuite.com/truffle

43. Ethereum Stack Exchange. (n.d.). Retrieved April/May, from https://ethereum.stackexchange.com/

44. Igor Yalovoy. (2018, November 21). How Much Does It Cost to Run DApp in 2018. Retrieved March 9, 2019, from https://ylv.io/how-much-does-it-costs-to-run-dapp-in-2018/

45. F. (n.d.). Get Started. Retrieved March 9, 2019, from https://developers.fortmatic.com/

46. Key Vault. (n.d.). Retrieved March 9, 2019, from https://azure.microsoft.com/es-es/services/key-vault/

47. /@austin_48503. (2018, October 08). Ethereum Meta Transactions - Austin Thomas Griffith. Retrieved March/April, 2019, from https://medium.com/@austin_48503/ethereum-meta-transactions-90ccf0859e84

48. Name. (n.d.). RSK - Smart Contract Platform Secured by the Bitcoin Network |. Retrieved March 9, 2019, from https://www.rsk.co/es/

49. B. (n.d.). Sidechains in Blockchain. Retrieved March 9, 2019, from https://blockstream.com/sidechains.pdf

50. Name. (n.d.). Preguntas frecuentes: RSK - Smart Contract Platform Secured by the Bitcoin Network - Part 3. Retrieved March 4, 2019, from https://www.rsk.co/es/preguntas-frecuentes/page/3/?cat=federacion

51. Prediction Markets: A Decentralized Oracle & Prediction Market Protocol. (n.d.). Retrieved March 23, 2019, from https://www.augur.net/

52. Open-Source Peer-to-Peer Online Gaming Platform. (n.d.). Retrieved March 30, 2019, from https://decent.bet/

53. Tencent. (n.d.). Retrieved May 10, 2019, from http://www.tencent.com/en-us/company.html

54. Tencent. (n.d.). Retrieved May 10, 2019, from http://www.tencent.com/en-us/index.html

55. Cómo realizar apuestas en los eSports. (n.d.). Retrieved July 10, 2019, from https://www.apuestas-esports.com/

56. Esports. (n.d.). Retrieved July 10, 2019, from https://sports.betway.es/es/sports/cat/esports

57. ApuestasOnline.net. (n.d.). Apuestas de eSports: Mejores Casas de Apuestas: 2019. Retrieved July 10, 2019, from https://apuestasonline.net/esports/

58. Los eSports reportar. (2019, May 14). Retrieved July 10, 2019, from https://www.marca.com/esports/2019/05/14/5cdad70822601d3c758b4581.html

59. P. (n.d.). Manual Verification. Retrieved from https://playerslounge.co/

60. Mobile eSports Platform & Software. (n.d.). Retrieved July 10, 2019, from https://www.skillz.com/

61. J. (2017). Https://biblio.juridicas.unam.mx/bjv.

62. Ethereum Dapps. (n.d.). Retrieved July 10, 2019, from https://www.researchgate.net/figure/A-common-architecture-for-an-Ethereum-Dapp_fig3_326693413

ANEXO A: CÓDIGO SMART CONTRACT

```
pragma solidity >=0.4.22 <0.6.0;

contract Gag {

  struct Match {
    address[] playersAddresses;
    mapping(address => Player) players; //JUGADORES DE ESTE PARTIDO
    mapping(address => uint) votes;
    bool distributedReward;
    address winner;
    bool isOnTrial;
    mapping(address => Judge) judges; //JUGADORES DE ESTE PARTIDO
    bool hasTrialFinished;
  }


  struct Player {
    bool isInMatch;
    uint64 timestamp;
    bool hasVoted;
    uint8 voteCount;
    address votedTo;
    uint8 judgeCount;
  }

  struct Judge {
    bool hasJudged;
    address judgeTo;
  }


  //VARIABLES
  mapping(string => Match) matches; //KING OF THE HILL MATCHES
  address owner;

  //BETTIN SIZES
  uint bet = 2 ether;
  uint judgeBet = 1 ether;
  uint feeRatio = 1000;

  //NUMBER OF PLAYERS
  uint maxNumberOfPlayers = 2;
  uint minNumberOfPlayers = 2;

  //DURATION
  uint minDuration = 2;
  uint minTimeToTrial = 2;

  //PROOF OF GOOD WILL
  uint minVotesRatio = 6000; //60% de los votos mínimo para poder
  uint significantVotesRatio = 2000;  // SI alguien tiene el 25% de los votos aunque haya perdido ya es
sospechoso
```

```
 //PROOF OF EXTERNAL VALIDATION
 uint minJudgesRatio = 2500; //25% minimo de jueces. Si hay 100 jugadores 25 jueces.
 uint maxJudgesRatio = 5000; //50% maximo de jueces. Si hay 100 jugadores 50 jueces.
 uint minJudges = 5; //Da igual cuantos jugadores haya minimo 5 judges. Es decir 3 votos.

 string[] trials;

 constructor(uint _bet, uint _fee, uint _minNumberOfPlayers,
   uint _maxNumberOfPlayers,
   uint _minDuration,
   uint _minTimeToTrial,
   uint _minVotesRatio,
   uint _significantVotesRatio,
   uint _minJudgesRatio,
   uint _maxJudgesRatio,
   uint _minJudges) public payable {
   owner = msg.sender;
   bet = _bet;
   feeRatio = 1000;
   minNumberOfPlayers = _minNumberOfPlayers;
   maxNumberOfPlayers = _maxNumberOfPlayers;
   minDuration = _minDuration;
   minTimeToTrial = _minTimeToTrial;
   minVotesRatio = _minVotesRatio;
   significantVotesRatio = _significantVotesRatio;
   minJudgesRatio = _minJudgesRatio;
   maxJudgesRatio = _maxJudgesRatio;
   minJudges = _minJudges;
 }

 modifier ifOwner {
   require(msg.sender == owner);
   _;
 }


 ///MATCH
 //FUNCTIONS


 function joinMatch(string memory id) public payable {
   //QUE NO ESTE TERMINADA LA PARTIDA
   require(!matches[id].distributedReward);
   //QUE NO ESTE LLENA LA PARTIDA
   require(matches[id].playersAddresses.length < maxNumberOfPlayers);
   //SI SE HA SUPERADO EL MINIMO DE JUGADORES, QUE NO HAYAN PASADO MAS DE 15
MIN DESDE QUE EL ULTIMO JUGADOR SE UNIO A LA PARTIDA.
   //ESTO PERMITE QUE JUGADORES SE UNAN CADA 5 MINUTOS ETERNAMENTE
   if (matches[id].playersAddresses.length > minNumberOfPlayers) {
     require(matches[id].players[matches[id].playersAddresses[matches[id].playersAddresses.length - 1]].timestamp
+ 15 * 60 > now);
   }

   //QUE EL JUGADOR NO ESTE YA EN LA PARTIDA
```

```
  require(!matches[id].players[msg.sender].isInMatch);
  //QUE LA APUESTA SEA SUFICIENTE
  require(msg.value >= bet);

  Match storage _match = matches[id];
  //30k gas +
  _match.playersAddresses.push(msg.sender);
  //GUARDAMOS LA ADDRESS DEL NUEVO JUGADOR
  //20k gas +
  _match.players[msg.sender].isInMatch = true;
  //CREAMOS EL NUEVO JUGADOR
  _match.players[msg.sender].timestamp = uint64(now);
  //GUARDAMOS LA FECHA EN LA QUE SE HA UNIDO EL NUEVO JUGADOR
  msg.sender.transfer(msg.value - bet);
  //DEVOLVEMOS EL DINERO RESTANTE AL JUGADOR

 }

 function changeIsInMatch(string memory id) public {

  matches[id].players[msg.sender].isInMatch = true;
 }

 function exitMatch(string memory id) public {
  require(matches[id].players[msg.sender].isInMatch);
  require(matches[id].playersAddresses.length < minNumberOfPlayers);
  matches[id].players[msg.sender].isInMatch = false;
  msg.sender.transfer(bet);
  bool index = false;
  for (uint i = 0; i < matches[id].playersAddresses.length - 1; i++) {
   if (matches[id].playersAddresses[i] == msg.sender) {
    index = true;
   }

   if (index) {
    matches[id].playersAddresses[i] = matches[id].playersAddresses[i + 1];
   }
  }
  matches[id].playersAddresses.length--;
 }


 function vote(string memory id, address votedTo) public {

  require(matches[id].players[msg.sender].isInMatch);
  //COMPROBAMOS QUE ESTE JUGADOR PERTENEZCA A LA PARTIDA
  require(!matches[id].players[msg.sender].hasVoted);
  //COMPROBAMOS QUE NO HAYA VOTADO YA
  require(matches[id].players[votedTo].isInMatch);
  //COMPROBAMOS QUE EXISTA EL JUGADOR A QUIEN QUIERE VOTAR
  Player storage player = matches[id].players[msg.sender];
  player.hasVoted = true;
  player.votedTo = votedTo;
  matches[id].players[votedTo].voteCount ++;
```

```
}

function getWinner(string memory id) public view returns (address, uint) {
  uint8 winningVoteCount = 0;
  uint8 winnerPosition;
  for (uint8 prop = 0; prop < matches[id].playersAddresses.length; prop++) {
    if (matches[id].players[matches[id].playersAddresses[prop]].voteCount > winningVoteCount) {
      winningVoteCount = matches[id].players[matches[id].playersAddresses[prop]].voteCount;
      winnerPosition = prop;
    }
  }

  return (matches[id].playersAddresses[winnerPosition], winningVoteCount);
}

function distributeReward(string memory id) public {

  require(matches[id].playersAddresses.length > 0);
  require(now > matches[id].players[matches[id].playersAddresses[0]].timestamp + minDuration);
  require(!matches[id].distributedReward);


  (address winnerNotPayable, uint winningVoteCount) = getWinner(id);

  require(winningVoteCount > minVotesRatio / 10000 * matches[id].playersAddresses.length);
  //60% of votes

  matches[id].distributedReward = true;
  matches[id].winner = winnerNotPayable;
  address payable winner = address(uint160(winnerNotPayable));
  require(matches[id].players[winnerNotPayable].isInMatch);
  //QUE EL WINER ESTE EN LA PARTIDA
  uint reward = (matches[id].playersAddresses.length * bet);
  uint fee = reward * feeRatio /10000;
  winner.transfer(reward - fee);

}

//VIEWS


function getHost(string memory id) public view returns (address) {

  return matches[id].playersAddresses[0];

}

function getVotes(string memory id, address pAddress) public view returns (uint) {

  return matches[id].players[pAddress].voteCount;

}

function isAddressOnMatch(string memory id, address playerAddress) public view returns (bool){
```

```
  for (uint i = 0; i < matches[id].playersAddresses.length; i++) {
   if (playerAddress == matches[id].playersAddresses[i]) {
    return true;
   }
  }

  return false;
}

function getTimeLeft(string memory id) public view returns (uint) {

  require(matches[id].playersAddresses.length > 0);
  if (now >= matches[id].players[matches[id].playersAddresses[0]].timestamp + minDuration) {
   return 0;
  } else {
   return matches[id].players[matches[id].playersAddresses[0]].timestamp + minDuration - now;
  }

}

function getTimeStamp(string memory id, address pAddress) public view returns (uint) {

  return matches[id].players[pAddress].timestamp;

}

function getBalance() public view returns (uint) {

  return address(this).balance;

}

function getMatchSize(string memory id) public view returns (uint) {

  return matches[id].playersAddresses.length;

}

function getPlayerIsInMatch(string memory id, address pAddress) public view returns (bool) {

  return matches[id].players[pAddress].isInMatch;


}

//TRIALS
//FUNCTIONS
function challengeMatch(string memory id) public {
  require(!matches[id].isOnTrial);
  matches[id].isOnTrial = true;
  trials.push(id);
}

function judgeMatch(string memory id, address judgeTo) public {
```

```
    require(matches[id].isOnTrial);
    //QUE LA PARTIDA ESTE EN TRIAL
    require(!isAddressOnMatch(id, msg.sender));
    //QUE EL JUDGE NO SEA UN JUGADOR
    require(!matches[id].judges[msg.sender].hasJudged);
    //QUE ESTE JUDGE NO HAYA VOTADO YA
    require(matches[id].players[judgeTo].isInMatch);
    //QUE EL JUGADOR A QUIEN QUIERE VOTAR EXISTA

    Judge storage judge = matches[id].judges[msg.sender];
    judge.hasJudged = true;
    judge.judgeTo = judgeTo;
    matches[id].players[judgeTo].judgeCount ++;

  }

  //VIEWS
  function getJudgeCount(string memory id, address _address) public view returns (uint) {

    return matches[id].players[_address].judgeCount;

  }

  function isMatchOnTrial(string memory id) public view returns (bool) {

    return matches[id].isOnTrial;

  }


}
```

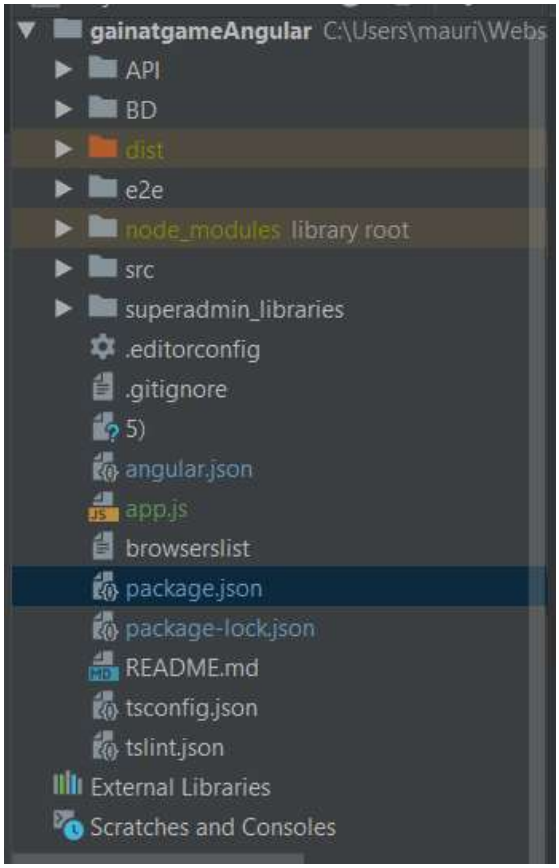
ANEXO B: ESTRUCTURA Y CÓDIGO DE CLIENTE

El cliente de Gainatgame se ha desarrollado con Angular 8. Para la funcionalidad se han usado diversos módulos auxiliares de Node los cuales se pueden apreciar en el archivo package.json:

```
{
  "name": "my-app",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^8.0.1",
    "@angular/cdk": "^8.0.1",
    "@angular/common": "^8.0.1",
    "@angular/compiler": "^8.0.1",
    "@angular/core": "^8.0.1",
    "@angular/flex-layout": "^8.0.0-beta.26",
    "@angular/forms": "^8.0.1",
    "@angular/http": "^7.2.15",
    "@angular/material": "^8.0.1",
    "@angular/platform-browser": "^8.0.1",
    "@angular/platform-browser-dynamic": "^8.0.1",
    "@angular/router": "^8.0.1",
    "core-js": "^3.1.4",
    "express": "^4.17.1",
    "fs": "0.0.1-security",
    "hammerjs": "^2.0.8",
    "jquery": "^3.4.1",
    "karma-cli": "^2.0.0",
    "mysql": "^2.17.1",
    "ngx-socket-io": "^2.1.1",
    "ngx-spinner": "^7.1.4",
    "rxjs": "^6.5.2",
    "rxjs-tslint": "^0.1.7",
    "socket.io": "^2.2.0",
    "tslib": "^1.10.0",
    "web3": "^1.0.0-beta.55",
    "web3-eth": "^1.0.0-beta.55",
    "webpack": "^4.34.0",
    "zone.js": "^0.9.1"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "^0.800.3",
    "@angular/cli": "~8.0.3",
    "@angular/compiler-cli": "^8.0.1",
    "@angular/language-service": "^8.0.1",
    "@types/jasmine": "^3.3.13",
    "@types/jasminewd2": "~2.0.3",
    "@types/node": "^12.0.8",
    "codelyzer": "^5.1.0",
    "jasmine-core": "^3.4.0",
    "jasmine-spec-reporter": "^4.2.1",
    "karma": "^4.1.0",
    "karma-chrome-launcher": "^2.2.0",
    "karma-coverage-istanbul-reporter": "~2.0.1",
```

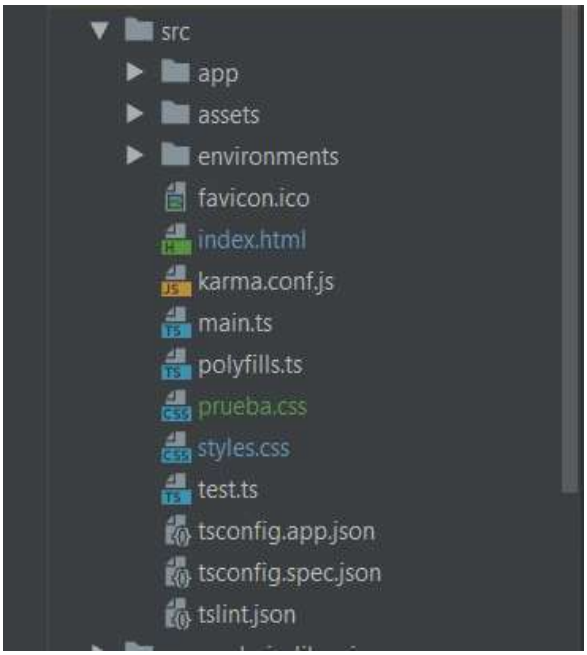
```
    "karma-jasmine": "^2.0.1",
    "karma-jasmine-html-reporter": "^1.4.2",
    "protractor": "^5.4.2",
    "ts-node": "~8.3.0",
    "tslint": "^5.17.0",
    "typescript": "^3.4.5"
  },
  "browser": {
    "http": false,
    "https": false,
    "net": false,
    "path": false,
    "stream": false,
    "tls": false
  }
}
```

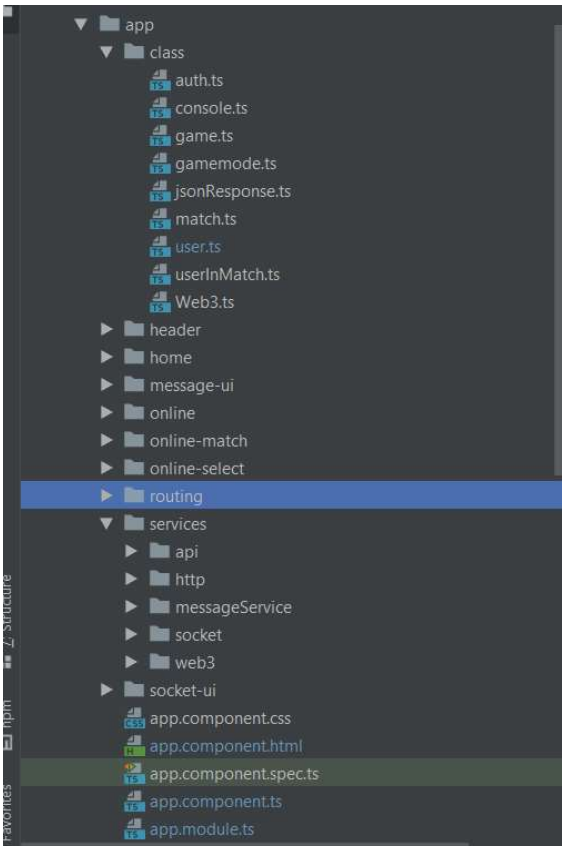
La estructura de componentes es la siguiente:



Donde app.js es el servidor de Node (anexo C), BD contiene funcionalidad para conectarse con la base de datos Mysqli, API contiene funcionalidad para conectarse mediante http a la Api de php, superadmin_libraries es la template utilizada para la interfaz y src contiene todos los componentes de la aplicación.

La estructura del src es la siguiente:





Debido a la gran cantidad de código que tiene el cliente de Angular únicamente se adjuntará el más importante:

1. Web3 Service:

```
import {Injectable} from '@angular/core';
import Web3 from 'web3';
import {MessageService} from '../messageService/message.service';
import {SocketService} from '../socket/socket.service';
import {HttpClient, HttpClientModule} from '@angular/common/http';
import * as $ from 'jquery';
import * as http from 'http';
```

```
/* const contractJson =
require('../../../../../ethereum/gag/build/contracts/Gag');
const jsonFile = '../../../../../ethereum/gag/build/contracts/Gag';
const jsonAbi = JSON.parse(fs.readFileSync(jsonFile, 'utf8'));
const abi = jsonAbi.abi;*/

declare let window: any;

@Injectable({
  providedIn: 'root'
})
export class Web3Service {
  private gagAddress = '0xbF36b6d27C019AAe9645eE798f66EA08677dd4B0';
  private gagAbi = null;
  private web3Provider = null;
  public web3: Web3;
  public defaultAddres: string;
  public rivalAddress: string;
  public jsonFileContent: string;


  constructor(private http: HttpClient, private messageService:
MessageService, private socketService: SocketService) {
    this.getAbi();
    if (typeof window.web3 !== 'undefined') {
      this.web3Provider = window.web3.currentProvider;
      // this.web3Provider = new
Web3.providers.HttpProvider('http://localhost:7545');
    } else {
      this.web3Provider = new
Web3.providers.HttpProvider('http://localhost:7545');
    }
    this.web3 = new Web3(this.web3Provider);
    this.getAccounts();
  }

  isMetamask() {
    console.log(window.web3);
  }

  getAbi() {
    const jsonFile = 'assets/Gag.json';
    this.http.get('assets/Gag.json', {responseType:
'text'}).subscribe(data => {
      const that = this;
      that.gagAbi = data.toString();
      that.gagAbi = JSON.parse(that.gagAbi).abi;
      console.log(that.gagAbi.abi);
    });
  }

  vote1(matchId, address) {
    const that = this;
    this.getContract().methods.vote(matchId, address).send({from:
this.defaultAddres, gas: 3000000}, (error, transactionHash) => {
```

```
      console.log('vote SUCCESS');
      that.messageService.add('WEB3: ' + 'vote SUCCESS');
      console.log(error);
    });
  }

  vote2(matchId, address) {
    const that = this;
    this.getContract().methods.vote(matchId, address).send({from:
this.rivalAddress, gas: 3000000}, (error, transactionHash) => {
      console.log('vote SUCCESS');
      that.messageService.add('WEB3: ' + 'vote SUCCESS');
      console.log(error);
    });
  }

  getContract() {
    return new this.web3.eth.Contract(this.gagAbi, this.gagAddress);
  }

  getAccounts() {
    const that = this;
    this.web3.eth.getAccounts((error, accounts) => {
        that.defaultAddres = accounts[0];
        that.rivalAddress = accounts[1];
        console.log(accounts[0]);
      }
    );
  }

  /*getAccountBalance() {
    const that = this;
    this.web3.eth.getBalance(this.defaultAddres, function (error,
balance) {
      console.log(that.toEther(balance) + ' ether');
      that.messageService.add('WEB3: ' + that.toEther(balance) + '
ether');
    });
  }*/

  joinMatch(matchId: string) {
    const that = this;
    // AMOUNT BET
    const amountToSend = this.web3.utils.toWei('2', 'ether');
    console.log(amountToSend);
    this.getContract().methods.joinMatch(matchId).send({
      from: that.defaultAddres,
      gas: 3000000,
      value: amountToSend
    }, (error, transactionHash) => {
      console.log('addPlayer SUCCESS');
      if (!error) {
        that.socketService.emitEvent('acceptMatch', 1);
      }
      that.messageService.add('WEB3: ' + 'addPlayer SUCCESS error-> '
```

```
+ error);
      console.log(error);

    });
  }

  exitMatch(matchId: string) {
    const that = this;
    this.getContract().methods.exitMatch(matchId).send({
      from: that.defaultAddres,
      gas: 3000000,
      value: 0
    }, (error, transactionHash) => {
      console.log('addPlayer SUCCESS');
      if (!error) {
        that.socketService.emitEvent('cancelMatch', 1);
      }
      that.messageService.add('WEB3: ' + 'addPlayer SUCCESS error-> '
+ error);
      console.log(error);

    });
  }

  joinMatch2(matchId: string) {
    const that = this;
    // AMOUNT BET
    const amountToSend = this.web3.utils.toWei('2', 'ether');
    console.log(amountToSend);
    this.getContract().methods.joinMatch(matchId).send({
      from: that.rivalAddress,
      gas: 3000000,
      value: amountToSend
    }, (error, transactionHash) => {
      console.log('addPlayer SUCCESS');
      if (!error) {
        that.socketService.emitEvent('acceptMatch', 2);
      }
      that.messageService.add('WEB3: ' + 'addPlayer SUCCESS error-> '
+ error);
      console.log(error);

    });
  }

  getSize() {
    const that = this;
    this.getContract().methods.getMatchSize().call({from:
this.defaultAddres}, (error, result) => {
      console.log('size: ' + result);
      that.messageService.add('WEB3: ' + 'size: ' + result);
    });
  }

  addPlayer(matchId: string, pAddress: string) {
```

```
    const that = this;
    const amountToSend = this.web3.utils.toWei('2', 'ether');
    console.log(amountToSend);
    this.getContract().methods.addPlayerToMatch(matchId,
this.defaultAddres).send({
      from: that.defaultAddres,
      gas: 3000000,
      value: amountToSend
    }, (error, transactionHash) => {
      console.log('addPlayer SUCCESS');
      that.messageService.add('WEB3: ' + 'addPlayer SUCCESS');
      console.log(error);
    });
  }

  toEther(wei: string) {
    return this.web3.utils.fromWei(wei, 'ether');
  }
}
```

2. Socket Service:

```
import {Injectable} from '@angular/core';
import {Socket} from 'ngx-socket-io';
import {MessageService} from '../messageService/message.service';
import {RestApiService} from '../http/restApi/rest-api.service';
import {Router} from '@angular/router';
import {NgxSpinnerService} from 'ngx-spinner';

@Injectable({
  providedIn: 'root'
})
export class SocketService {
  timeLeftToStart = 60;
  timerToStartBool = false;
  intervalToStart;

  constructor(public restApiService: RestApiService,
              public messageService: MessageService,
              private socket: Socket,
              private router: Router,
              private spinner: NgxSpinnerService) {
    this.initializeSocket();
  }

  // socket functions:
  initializeSocket() {
    this.socket.on('newUser', (user) => {
      // this.messageService.add('NEW USER');
      this.restApiService.user = user;
      if (user.isInQueue) {
        this.spinner.show();
      } else {
```

```
        this.spinner.hide();
      }
      this.actualizarPageLocation();
    });
    this.socket.on('matchInfo', (match) => {
      this.restApiService.match = match;
      // this.messageService.add('NEW MATCH');
      // console.log('this.match : ' + JSON.stringify(this.match));
      // this.actualizarTimers();
      // this.emitEvent('matchInfo', 1);
      if (this.restApiService.match.users[0].accepted === 1) {
        this.startTimerToStart();
      } else {
        this.pauseTimerToStart();
      }
      this.actualizarPageLocation();
    });
  }

  startTimerToStart() {
    if (this.timerToStartBool === false) {
      this.messageService.add('startTimer');
      this.timerToStartBool = true;
      this.intervalToStart = setInterval(() => {
        if (this.timeLeftToStart > 0) {
          this.timeLeftToStart--;
        } else {
          this.emitEvent('startMatch', 1);
        }
      }, 1000);
    }
  }

  pauseTimerToStart() {
    if (this.timerToStartBool === true) {
      this.timerToStartBool = false;
      this.messageService.add('pauseTimer');
      clearInterval(this.intervalToStart);
      this.timeLeftToStart = 60;
    }
  }

  actualizarPageLocation() {
    if (this.restApiService.user && this.restApiService.match) {
      if (this.restApiService.user.isInMatch === 1 && this.router.url !==
'/online-match') {
        console.log('Navigating to match');
        this.router.navigate(['online-match']);
      }
    }
  }

  emitEvent(name: string, data: number) {
    this.messageService.add('SOCKET: emitEvent name-> ' + name);
    this.socket.emit(name, data);
```

```
    }

}
```

3. Online-match component

```
import {Component, OnInit} from '@angular/core';
import {RestApiService} from '../services/http/restApi/rest-api.service';
import {SocketService} from '../services/socket/socket.service';
import {MessageService} from '../services/messageService/message.service';
import {Web3Service} from '../services/web3/web3.service';
import {ApiService} from '../services/api/api.service';

declare var $: any;


@Component({
  selector: 'app-online-match',
  templateUrl: './online-match.component.html',
  styleUrls: ['./online-match.component.css']
})
export class OnlineMatchComponent implements OnInit {

  public winnerSelectNickname;

  constructor(public messageService: MessageService, public socketService: SocketService
    , public restApiService: RestApiService
    , public apiService: ApiService
    , public web3Service: Web3Service) {
  }

  ngOnInit() {
  }

  onWinnerChange(event: any) {
    this.winnerSelectNickname = event.target.value;
  }

  joinMatch() {
    if (this.restApiService.match.users[0].accepted === 0) {
      this.web3Service.joinMatch(this.restApiService.match.id);
    } else {
    }
  }

  exitMatch() {
    if (this.restApiService.match.users[0].accepted === 1) {
      this.web3Service.exitMatch(this.restApiService.match.id);
    }
  }


}
```

4. . Online-match HTML:

```
<button style="display:none;"
(click)="this.web3Service.isMetamask()">get Abi</button>
<div *ngIf="restApiService.match">

  <div style="display: none;" class="row quick-stats">
    <div class="col-sm-6 col-md-3">
      <div class="quick-stats__item">
        <div class="quick-stats__info">
          <h2>987,459</h2>
          <small>Total Leads Recieved</small>
        </div>

        <div class="quick-stats__chart peity-bar" style="display:
none;">6,4,8,6,5,6,7,8,3,5,9</div>
        <svg class="peity" height="36" width="65">
          <rect fill="rgba(255,255,255,0.85)"
x="0.8981818181818183" y="12" width="2.694545454545455"
                height="24"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="5.389090909090909"
y="20" width="2.6945454545454552"
                height="16"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="9.88" y="4"
width="2.6945454545454535" height="32"></rect>
          <rect fill="rgba(255,255,255,0.85)"
x="14.370909090909093" y="12" width="2.6945454545454535"
                height="24"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="18.86181818181818"
y="16" width="2.6945454545454552"
                height="20"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="23.35272727272727"
y="12" width="2.6945454545454552"
                height="24"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="27.84363636363636"
y="8" width="2.6945454545454552" height="28"></rect>
          <rect fill="rgba(255,255,255,0.85)"
x="32.334545454545456" y="4" width="2.6945454545454552"
                height="32"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="36.82545454545454"
y="24" width="2.6945454545454623"
                height="12"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="41.31636363636363"
y="16" width="2.6945454545454552"
                height="20"></rect>
          <rect fill="rgba(255,255,255,0.85)"
x="45.807272727272725" y="0" width="2.6945454545454552"
                height="36"></rect>
        </svg>
```

```
      </div>
    </div>

    <div class="col-sm-6 col-md-3">
      <div class="quick-stats__item">
        <div class="quick-stats__info">
          <h2>356,785K</h2>
          <small>Total Website Clicks</small>
        </div>

        <div class="quick-stats__chart peity-bar" style="display:
none;">4,7,6,2,5,3,8,6,6,4,8</div>
        <svg class="peity" height="36" width="65">
          <rect fill="rgba(255,255,255,0.85)"
x="0.8981818181818183" y="18" width="2.694545454545455"
                height="18"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="5.389090909090909"
y="4.5" width="2.6945454545454552"
                height="31.5"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="9.88" y="9"
width="2.6945454545454535" height="27"></rect>
          <rect fill="rgba(255,255,255,0.85)"
x="14.370909090909093" y="27" width="2.6945454545454535"
                height="9"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="18.86181818181818"
y="13.5" width="2.6945454545454552"
                height="22.5"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="23.35272727272727"
y="22.5" width="2.6945454545454552"
                height="13.5"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="27.84363636363636"
y="0" width="2.6945454545454552" height="36"></rect>
          <rect fill="rgba(255,255,255,0.85)"
x="32.334545454545456" y="9" width="2.6945454545454552"
                height="27"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="36.82545454545454"
y="9" width="2.6945454545454623" height="27"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="41.31636363636363"
y="18" width="2.6945454545454552"
                height="18"></rect>
          <rect fill="rgba(255,255,255,0.85)"
x="45.807272727272725" y="0" width="2.6945454545454552"
                height="36"></rect>
        </svg>
      </div>
    </div>

    <div class="col-sm-6 col-md-3">
      <div class="quick-stats__item">
        <div class="quick-stats__info">
          <h2>$58,778</h2>
          <small>Total Sales Orders</small>
        </div>

        <div class="quick-stats__chart peity-bar" style="display:
```

```
none;">9,4,6,5,6,4,5,7,9,3,6</div>
        <svg class="peity" height="36" width="65">
          <rect fill="rgba(255,255,255,0.85)"
x="0.8981818181818183" y="0" width="2.694545454545455"
height="36"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="5.389090909090909"
y="20" width="2.6945454545454552"
                height="16"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="9.88" y="12"
width="2.6945454545454535" height="24"></rect>
          <rect fill="rgba(255,255,255,0.85)"
x="14.370909090909093" y="16" width="2.6945454545454535"
                height="20"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="18.86181818181818"
y="12" width="2.6945454545454552"
                height="24"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="23.35272727272727"
y="20" width="2.6945454545454552"
                height="16"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="27.84363636363636"
y="16" width="2.6945454545454552"
                height="20"></rect>
          <rect fill="rgba(255,255,255,0.85)"
x="32.334545454545456" y="8" width="2.6945454545454552"
                height="28"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="36.82545454545454"
y="0" width="2.6945454545454623" height="36"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="41.31636363636363"
y="24" width="2.6945454545454552"
                height="12"></rect>
          <rect fill="rgba(255,255,255,0.85)"
x="45.807272727272725" y="12" width="2.6945454545454552"
                height="24"></rect>
        </svg>
      </div>
    </div>

    <div class="col-sm-6 col-md-3">
      <div class="quick-stats__item">
        <div class="quick-stats__info">
          <h2>214</h2>
          <small>Total Support Tickets</small>
        </div>

        <div class="quick-stats__chart peity-bar" style="display:
none;">5,6,3,9,7,5,4,6,5,6,4</div>
        <svg class="peity" height="36" width="65">
          <rect fill="rgba(255,255,255,0.85)"
x="0.8981818181818183" y="16" width="2.694545454545455"
                height="20"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="5.389090909090909"
y="12" width="2.6945454545454552"
                height="24"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="9.88" y="24"
width="2.6945454545454535" height="12"></rect>
```

```
          <rect fill="rgba(255,255,255,0.85)"
x="14.370909090909093" y="0" width="2.6945454545454535"
                height="36"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="18.86181818181818"
y="8" width="2.6945454545454552" height="28"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="23.35272727272727"
y="16" width="2.6945454545454552"
                height="20"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="27.84363636363636"
y="20" width="2.6945454545454552"
                height="16"></rect>
          <rect fill="rgba(255,255,255,0.85)"
x="32.334545454545456" y="12" width="2.6945454545454552"
                height="24"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="36.82545454545454"
y="16" width="2.6945454545454623"
                height="20"></rect>
          <rect fill="rgba(255,255,255,0.85)" x="41.31636363636363"
y="12" width="2.6945454545454552"
                height="24"></rect>
          <rect fill="rgba(255,255,255,0.85)"
x="45.807272727272725" y="20" width="2.6945454545454552"
                height="16"></rect>
        </svg>
      </div>
    </div>
  </div>

  <div class="card">
    <div class="card-body">

      <h1 style="font-size: 22px;" class="card-title text-
center">Online Match</h1>
      <h4 class="card-subtitle text-center">BET:
{{restApiService.match.moneyBet *
      restApiService.match.users.length}} GGs /
        GAME: {{restApiService.match.gamemode.game.name}} /
GAMEMODE: {{restApiService.match.gamemode.toString}}

      </h4>

      <div style="display:none;" class="row">
        <div class="col-lg-12">
          <div class="card widget-time">
            <div class="time">
              <span class="time__hours">05</span>
              <span class="time__min">33</span>
              <span class="time__sec">13</span>
            </div>
          </div>
        </div>
      </div>

      <div class="text-center">
        <button style="font-size: 15px"
```

```
                *ngIf="restApiService.match.users[0].accepted !== 1
&& restApiService.match.users[0].accepted !== '1'"
                type="button" (click)="joinMatch()"
                class="btn btn-outline-success">ENTER MATCH
        </button>

        <button style="font-size: 15px"
                *ngIf="restApiService.match.users[0].accepted !== 0
&& restApiService.match.users[0].accepted !== '0'"
                type="button" (click)="exitMatch()"
                class="btn btn-outline-danger">EXIT MATCH
        </button>
      </div>


      <br>

      <div class="tab-container">
        <ul class="nav nav-tabs nav-fill" role="tablist">
          <li class="nav-item">
            <a class="nav-link active" data-toggle="tab"
href="#home-2" role="tab">Players</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" data-toggle="tab" href="#profile-2"
role="tab">Chat</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" data-toggle="tab" href="#messages-
2" role="tab">Votes</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" data-toggle="tab" href="#settings-
2" role="tab">Files</a>
          </li>
        </ul>

        <div class="tab-content">
          <div class="tab-pane active fade show" id="home-2"
role="tabpanel">
            <table class="table mb-3">
              <thead class="thead-inverse">
              <tr>
                <th>#</th>
                <th>Nickname</th>
                <th>Account</th>
                <th>Accepted</th>
              </tr>
              </thead>
              <tbody>
              <tr *ngFor="let user of restApiService.match.users;
let i = index">
                <th scope="row">{{i + 1}}</th>
                <td>{{user.user}}</td>
                <td>{{user.email}}</td>
```

```
                <td class="text-red" [ngClass]="{'text-green' :
apiService.getBoolFromAccepted(user.accepted)}">
                  {{apiService.getTextFromAccepted(user.accepted)}}
                </td>
              </tr>
              </tbody>
            </table>
          </div>
          <div class="tab-pane fade" id="profile-2"
role="tabpanel">
            chat
          </div>
          <div class="tab-pane fade" id="messages-2"
role="tabpanel">
            <div class="card widget-ratings">
              <div class="card-body text-center">
                <h4 class="card-title">Mauri is the winner!</h4>

                <div class="widget-ratings__item">
                  <div class="float-left">Mauri <i class="zmdi
zmdi-star"></i></div>
                  <div class="float-right">100% (2 votes)</div>

                  <div class="widget-ratings__progress">
                    <div class="progress">
                      <div class="progress-bar bg-light"
role="progressbar" style="width: 100%;" aria-valuenow="100"
                           aria-valuemin="0" aria-
valuemax="100"></div>
                    </div>
                  </div>
                </div>

                <div class="widget-ratings__item">
                  <div class="float-left">Iñigo <i class="zmdi
zmdi-star"></i></div>
                  <div class="float-right">0% (0 votes)</div>

                  <div class="widget-ratings__progress">
                    <div class="progress">
                      <div class="progress-bar bg-light"
role="progressbar" style="width: 0%;" aria-valuenow="0"
                           aria-valuemin="0" aria-
valuemax="100"></div>
                    </div>
                  </div>
                </div>


              </div>
            </div>
          </div>
          <div class="tab-pane fade" id="settings-2"
role="tabpanel">
            files
```

```
          </div>
        </div>
      </div>
    </div>
  </div>


</div>


<div style="display: none;" *ngIf="restApiService.match"
class="content__inner text-center">
  <div class="contacts row">
    <div class="col-lg-5">
      <div id="card1" style="background-color: red" class="card
team__item">
        <img src="../../../images/contacts/8.jpg" class="team__img"
alt="">

        <div class="card-body">
          <h4 class="card-title">{{
restApiService.match.users[0].nickname}} </h4>
          <h6 class="card-subtitle">{{
restApiService.match.users[0].zone}} </h6>

          ADDRESS: {{web3Service.defaultAddres}}
          <button (click)="acceptMatch1()" lstyle="margin: 5px"
type="button"
                  class="btn btn-light btn-block-" title="" data-
toggle="tooltip"
                  data-placement="bottom"
                  data-original-title="Accept Match">
            ACCEPT MATCH
          </button>
          <div style="display: none" class="team__social">

            accept: {{restApiService.match.users[0].accepted}}<br>
            leaved: {{restApiService.match.users[0].leaved}}<br>
            declared:
{{restApiService.match.users[0].declared}}<br>
            reported:
{{restApiService.match.users[0].reported}}<br>


            <button
(click)="socketService.emitEvent('cancelMatch')" style="margin:
5px" type="button"
                    class="btn btn-light btn--icon " title="" data-
toggle="tooltip"
                    data-placement="bottom"
                    data-original-title="Cancel Match">
              <i class="zmdi zmdi-close"></i>
            </button>
          </div>
        </div>
```

```
      </div>
    </div>

    <div class="col-lg-2 text-center">
      <br><br>
      <h1>VS</h1>

    </div>

    <div class="col-lg-5">
      <div class="card team__item">
        <img src="../../../images/contacts/1.jpg" class="team__img"
alt="">

        <div class="card-body">
          <h4 class="card-title">{{
restApiService.match.users[1].nickname}} </h4>
          <h6 class="card-subtitle">{{
restApiService.match.users[1].zone}} </h6>
          ADDRESS: {{web3Service.rivalAddress}}
          <button (click)="acceptMatch2()" lstyle="margin: 5px"
type="button"
                  class="btn btn-light btn-block-" title="" data-
toggle="tooltip"
                  data-placement="bottom"
                  data-original-title="Accept Match">
            ACCEPT MATCH
          </button>
        </div>
      </div>

    </div>

  </div>
</div>
<div style="display:none;" *ngIf="!restApiService.match"
class="content__inner text-center">
  <div class="card">
    <div class="card-body">
      <header class="content__title">
        <h1>Online Match</h1>
        <div class="actions">
          <a href="" class="actions__item zmdi zmdi-trending-
up"></a>
          <a href="" class="actions__item zmdi zmdi-check-all"></a>

          <div class="dropdown actions__item">
            <i data-toggle="dropdown" class="zmdi zmdi-more-
vert"></i>
            <div class="dropdown-menu dropdown-menu-right">
              <a href="" class="dropdown-item">Refresh</a>
              <a href="" class="dropdown-item">Manage Widgets</a>
              <a href="" class="dropdown-item">Settings</a>
            </div>
          </div>
```

```
        </div>
        <small> You are not in any match</small>

      </header>
    </div>
  </div>

</div>
```

ANEXO C: CÓDIGO SERVIDOR NODE.JS

```
// IMPORTS
const app = require('express')();
const http = require('http').Server(app);
const io = require('socket.io')(http);
const fs = require('fs');
const no = require('util');
const EventEmitter = require('events');
let bd = require('./BD/conexion').localConnect();
let API = require('./API/getUser');
let Web3 = require('web3');
let web3;

// CONNECT TO BD
bd.connect(function (err) {
  if (err) throw err;

}); //Successful

// ETHEREUM
let contractAddress = '0xaaC002f19B162C1c028C1B44019Ad1E07fCD877E';
const contractJson = JSON.parse(fs.readFileSync('../ethereum/gag/build/contracts/Gag.json', 'utf8'));

if (typeof web3 !== 'undefined') {
  web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:7545"));
} else {
  web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:7545"));
}

let gagContract;
//Default account
web3.eth.getAccounts(function (error, accounts) {
  web3.eth.defaultAccount = accounts[0];
  gagContract = new web3.eth.Contract(contractJson.abi, contractAddress);
  console.log(accounts[0]);
});

io.on("connection", socket => {
  // Log whenever a user connects
  console.log('socket connected');
  socket.on('disconnect', function () {
    console.log('user disconnected');
    clearInterval(intervalFunc());
  });
  socket.on("getBalance", id => {
    console.log("getBalance id: " + id);
    //Calling a method
    gagContract.methods.getBalance().call({from: web3.eth.defaultAccount}, (error, result) => {
      console.log('balance: ' + result);
    });
  });
  socket.on('queue', (queue) => {
    console.log("queue changed to: " + queue);
    //socket.emit('queue', 1);
    queueClock = +queue;
  });
  socket.on('matchAction', (action) => {
    console.log("matchAction EVENT " + action);
```

```
  if (action === "acceptMatch") {
    API.acceptMatch();
  } else if (action === "cancelMatch") {
    API.cancelAccept();
  } else if (action === "leaveMatch") {
    API.leaveMatch();
  } else if (action === "reportMatch") {
    API.reportMatch();
  } else if (action === "matchInfo") {
    socket.emit('matchInfo', API.getMatchInfo());
  }

});
socket.on('acceptMatch', (action) => {
  API.acceptMatch();
});
socket.on('cancelMatch', (action) => {
  API.cancelAccept();
});
socket.on('leaveMatch', (action) => {
  API.leaveMatch();
});
socket.on('reportMatch', (action) => {
  API.reportMatch();
});
socket.on('matchInfo', (action) => {
  //socket.emit('matchInfo', API.getMatchInfo());
});

/*socket.on("createMatch", id => {
  console.log("createMatch id: " + id);
  //Calling a method
  gagContract.methods.createMatch('1').send({from: web3.eth.defaultAccount, gas: 3000000}, (error, transactionHash) => {
    console.log("createMatch SUCCESS");
  });
});*/

// QUEUE
let queueClock = 0;
let matchClock = 0;
count = 0;

//le damos valores al principio para que no sean undefined la primer ciclo de reloj
userJson = API.getUser();
userJsonClock = userJson;

setInterval(intervalFunc, 5000);

// ACTUALIZAR CLOCKS
function actualizarClocks(userJsonClock) {
  queueClock = userJsonClock.isInQueue;
  matchClock = userJsonClock.id;
  console.log("queue clock: " + userJsonClock.isInQueue);
  console.log("match clock: " + userJsonClock.isInMatch);
}

// HOW IS QUEUUE TIMER
function intervalFunc() {

  console.log("timer ticks");
  let userJson = API.getUser();
```

```
  if (userJson.isInQueue === 1) {
    let howIsQueue = API.howIsQueue();
    if (howIsQueue.result === 1) {
      console.log("EN COLA (RIVAL ENCONTRADO)");
    } else {
      console.log("EN COLA (BUSCANDO RIVAL..)");
    }
  } else { //QUEUE
    //NO ESTA EN COLA
  }

  if (userJson.isInMatch === 1) {
    console.log("EN PARTIDA");
    let matchJson = API.getMatchInfo();
    if (matchJson) {
      socket.emit('matchInfo', matchJson);
      console.log('MATCH INFO EVENT');
    }
    //vamos a ver si la partida ya se ha aceptado por ambos para empezarla
    if (matchJson.users) {
      if (matchJson.users[0].accepted === 1 && matchJson.users[1].accepted === 1) {
        API.startMatch();
      }
    }
  } else {//MATCH
    //NO ESTA EN PARTIDA
  }

  if (userJson) {
    socket.emit('newUser', userJson); //Pasamos al cliente el nuevo user
    console.log('NEW USER EVENT ' + count);
    count++;
  }

 }

});

// SERVER STARTED
http.listen(4444);
```

ANEXO D: CÓDIGO SERVIDOR PHP

El servidor php actúa como Api. Existen 3 endpoints con los que tanto el cliente como servidor de Node interactúa:

1. **Rest Api.php:** Este es el primer endpoint. Contiene funcionalidad genérica para obtener información.

```php
<?php

include "conexionAngular.php";
include("classes/Api.php");

if (!isset($_GET['auth'])) {

    echo -1;
    exit();

} else {

    $auth = $_GET['auth'];
}

$function = isset($_GET['function']) ? $_GET['function'] : '';
$user = new User($mysqli, $auth);
$api = new Api($mysqli, $user);
$source = "restApi";
$error = "";
$result = -1;

if ($function == "getOnlineGames") {

    $consoleId = isset($_GET['id_console']) ? $_GET['id_console'] : '';
    echo json_encode($api->getOnlineGames($consoleId));

} elseif ($function == "getOnlineGamemodes") {

    $consoleId = isset($_GET['id_console']) ? $_GET['id_console'] : '';
    $gameId = isset($_GET['id_game']) ? $_GET['id_game'] : '';
    $gamemodes = $api->getOnlineGamemodes($consoleId, $gameId);
    echo json_encode($api->getOnlineGamemodes($consoleId, $gameId));

} elseif ($function == "getUser") {

    echo json_encode($user);

} elseif ($function == "getUsers") {

    $users = array();
    $users[0] = $user;
    echo json_encode($users);

} elseif ($function == "getConsoles") {

    echo json_encode($api->getConsoles());

} else {
```

```
    echo -1;
}
```

2. **QueueApi.php:** Este es el segundo endpoint. Contiene funcionalidad relacionada con la búsqueda de partida y la creación de la misma.

```
<?php

include "conexionAngular.php";
include("classes/Api.php");
include("classes/JsonResponse.php");
include("classes/Server.php");

if (!isset($_GET['auth'])) {

    echo -1;
    exit();

} else {

    $auth = $_GET['auth'];
}

$function = isset($_GET['function']) ? $_GET['function'] : '';
$user = new User($mysqli, $auth);
$server = new Server($mysqli, $auth);
$api = new Api($mysqli, $user);
$source = "queueApi";
$error = "";
$result = -1;


if ($function == "enterQueue") {

    $consoleId = isset($_GET['id_console']) ? $_GET['id_console'] : '';
    $gamemodeId = isset($_GET['id_gamemode']) ? $_GET['id_gamemode'] :
'';
    $gameId = isset($_GET['id_game']) ? $_GET['id_game'] : '';
    $moneyBet = isset($_GET['money_bet']) ? $_GET['money_bet'] : '';

    $user->enterQueue($gameId, $consoleId, $gamemodeId, $moneyBet);
    // echo json_encode($user);
    echo json_encode(new JsonResponse($auth, $source, $function, $user-
>result, $user->error));
    //echo $user->result;

} elseif ($function == "howIsQueue") {

    $resultQueue = $server->howIsQueue();
    if ($resultQueue == 1) {

        echo json_encode(new JsonResponse($auth, $source, $function, 1,
```

```
$server->error));

    } elseif ($resultQueue == -1) {

        //TODO gestionar errores
        //$user->exitQueue();
        echo json_encode(new JsonResponse($auth, $source, $function, -1,
$server->error));

    } else {

        echo json_encode(new JsonResponse($auth, $source, $function, 0,
$server->error));

    }

} elseif ($function == "exitQueue") {

    $result = $user->exitQueue();
    echo json_encode(new JsonResponse($auth, $source, $function, $user-
>result, $user->error));

} else {

    $result = 0;
    $error = "Function doent exists";
    echo json_encode(new JsonResponse($auth, $source, $function, $result,
$error));

}
```

3. **MatchApi.php:** Este es el tercer endpoint. Contiene toda la funcionalidad necesaria una vez que se ha encontrado partida y que el usuario se encuentra en la misma.

```
<?php

include "conexionAngular.php";
include("classes/Api.php");
include("classes/JsonResponse.php");
include("classes/Server.php");

if (!isset($_GET['auth'])) {

    $error = "no auth";
    exit();

} else {

    $auth = $_GET['auth'];
```

```php
}

$function = isset($_GET['function']) ? $_GET['function'] : '';
$user = new User($mysqli, $auth);
$server = new Server($mysqli, $auth);
$api = new Api($mysqli, $user);
$source = "matchApi";
$error = "";
$result = -1;


if ($user->isInMatch() == 0) { //solo se puede usar la matchApi si esta
en match

    $error = "not in match";
    echo json_encode(new JsonResponse($auth, $source, $function, -1,
$error));
    exit();

} else {

    $userInMatch = new UserInMatch($mysqli, $auth);
    $match = new Match($mysqli, $auth);
}


if ($function == "getTimeLeftToAcceptMatch") {

    echo $match->getTimeToAcceptMatch();

} elseif ($function == "getMatchTime") {

    echo $match->getMatchTime();

} elseif ($function == "getMatchInfo") {

    echo $match->getMatchInfo();

} elseif ($function == "hasMatchStarted") {

    echo $match->started;

} elseif ($function == "hasMatchFinish") {

    echo $match->finished;


} elseif ($function == "hasMatchReported") {

    echo $match->reported;


} elseif ($function == "acceptMatch") {

    $userInMatch->acceptMatch();
```

```
    echo json_encode(new JsonResponse($auth, $source, $function,
$userInMatch->result, $userInMatch->error));


} elseif ($function == "leaveMatch") {

    $userInMatch->leaveMatch();
    echo json_encode(new JsonResponse($auth, $source, $function,
$userInMatch->result, $userInMatch->error));

} elseif ($function == "reportMatch") {

    $userInMatch->reportMatch();
    echo json_encode(new JsonResponse($auth, $source, $function,
$userInMatch->result, $userInMatch->error));

} elseif ($function == "cancelAccept") {

    $userInMatch->cancelAccept();
    echo json_encode(new JsonResponse($auth, $source, $function,
$userInMatch->result, $userInMatch->error));

} else {

    $error = "function doesnt exist";
    echo -1;
}

//echo $error;
```

Para realizar estos tres endpoints ha sido necesario el uso de clases auxiliares:

1. API.CLASS

```
<?php
include("User.php");
include("UserInMatch.php");
include("Game.php");
include("Gamemode.php");
include("Console.php");
include("Match.php"); //Match-> incluye UserUserMatch
class Api
{
   private $mysqli;
   public $user;
   //contructor

   function __construct(mysqli $mysqli, $user)
   {
```

```
        $this->user = $user;
        $this->mysqli=$mysqli;
    }

    public function getOnlineGames($consoleId)
    {

        $sql = "SELECT * from games where id IN(Select id_game from games_consoles where online = 1 and id_console =
$consoleId)";
        $result = $this->mysqli->query($sql);
        $games = array();
        $i = 0;
        while ($row = mysqli_fetch_assoc($result)) {
            $games[$i] = new Game($this->mysqli, $row['id']);
            $i++;
        }
        if($result){
            return $games;
        }else{
            return 0;
        }

    }

    public function getGames(){

        $sql = "SELECT id from games where id = $this->id";
        $result = $this->mysqli->query($sql);
        $games = array();
        $i = 0;
        while ($row = mysqli_fetch_assoc($result)) {
            $games[$i] = new Game($this->mysqli, $row['id']);
            $i++;
        }
        return json_encode($games);
    }

    public function getConsoles()
    {

        $sql = "SELECT * FROM consoles";
        $result = $this->mysqli->query($sql);
        $consoles = array();
        $i = 0;
        while ($row = mysqli_fetch_assoc($result)) {
            $consoles[$i] = new Console($this->mysqli, $row['id']);
            $i++;

        }
        return $consoles;
    }

    public function getOnlineGamemodes($consoleId,$gameId){

        $sql= "SELECT id from gamemodes where id_console = $consoleId and id_game = $gameId";
        $result = $this->mysqli->query($sql);

        $gamemodes = array();
        $i = 0;
        while ($row = mysqli_fetch_assoc($result)) {
            $gamemodes[$i] = new Gamemode($this->mysqli, $row['id']);
```

```
        $i++;
    }

    if($result){
        return  $gamemodes;
    }else{
        return 0;
    }


}
}

?>
```

1. USER.CLASS:

```
<?php

class User
{

    public $id;
    private $mysqli;
    public $user;
    public $country;
    public $mastery;
    public $lastLogin;
    public $email;
    public $defaultIcon;
    public $isInQueue;
    public $isInMatch;
    public $timeCreated;
    public $error;
    public $result;

    //contructor

    function __construct(mysqli $mysqli, $id)
    {

        $this->id = $id;
        $this->mysqli = $mysqli;

        $sql = "SELECT * from users where id = $this->id";
        $result = $this->mysqli->query($sql);
        $row = mysqli_fetch_assoc($result);

        $this->user = $row['user'];
        $this->country = $row['country'];
        $this->mastery = $row['mastery'];
        $this->lastLogin = $row['last_login'];
        $this->email = $row['email'];
        $this->defaultIcon = $row['default_icon'];
        $this->isInQueue = $this->isInQueue();
        $this->isInMatch = $this->isInMatch();
        date_default_timezone_set("Europe/Madrid");
```

```
        $timestamp = date_default_timezone_get();
        $mydate = date('Y-m-d H:i:s', strtotime($timestamp));
        $this->timeCreated = $mydate;
    }


    public function isInQueue()
    {


        $sql = "SELECT * from queues where finished = 0 and id_user = '$this->id'";
        $this->mysqli->query($sql);

        if ($this->mysqli->affected_rows == 1) {

            return 1;

        } else {


            return 0;
        }

    }

    public function isInMatch()
    {
        $sql = "SELECT * from matches where ((online = 1 and private = 0) or (online = 0 and private = 1)) and ((started = 0
and NOW() < ADDTIME(time_created, '10000') or started=1) and finished = 0 and id IN (Select id_match from
users_matches where leaved = 0 and id_user = $this->id))";
        $this->mysqli->query($sql);

        if ($this->mysqli->affected_rows == 1) {

            return 1;

        } else {

            return 0;
        }


    }

    public function isValidQueue($gamemodeId, $consoleId, $gameId)
    {

        //cambiar estado a "in queue"
        $sql = "SELECT * from gamemodes where online = 1 and id = $gamemodeId and id_console = $consoleId and
id_game=$gameId";
        $this->mysqli->query($sql);
        //existe un gamemode con esas caracteresticas? id_game,id_console?

        if ($this->mysqli->affected_rows > 0) {

            return 1;

        } else {

            return 0;
        }
```

```
    }

    public function isValidPrivateMatch($gamemodeId, $consoleId, $gameId)
    {


        //cambiar estado a "in queue"
        $sql = "SELECT * from gamemodes where private = 1 and id = $gamemodeId and id_console = $consoleId and
id_game=$gameId";
        $this->mysqli->query($sql);
        //existe un gamemode con esas caracteresticas? id_game,id_console?

        if ($this->mysqli->affected_rows > 0) {

            return 1;

        } else {

            return 0;
        }

    }

    public function enterQueue($gameId, $consoleId, $gamemodeId, $moneyBet)
    {

        $isValidQueue = $this->isValidQueue($gamemodeId, $gameId, $consoleId);
        $isInQueue = $this->isInQueue();
        $isInMatch = $this->isInMatch();

        if ($isValidQueue == 1 and $isInQueue == 0 and $isInMatch == 0) {

            date_default_timezone_set("Europe/Madrid");
            $timestamp = date_default_timezone_get();
            $mydate = date('Y-m-d H:i:s', strtotime($timestamp));


            //inserto la nueva queue

            $sql = "INSERT INTO queues (id_user,id_gamemode,money_bet,time_entered,finished) VALUES ($this-
>id,$gamemodeId,$moneyBet,'$mydate',0)";
            $this->result = $this->mysqli->query($sql);

            if ($this->result) {

                $this->result = 1;
                $this->error = "enter_queue success";

            } else {
                $this->result = 0;
                $this->error = "Imposible to join queue";
            }


        } else {

            $this->result = 0;
            $this->error = "Not valid queue, Already on queue or Already on match";

        }
```

```
    }

    public function getQueueInfo() //queue es siempre Online=1 Private = 0;
    {

        $sql = "SELECT * from queues where finished = 0 and id_user = '$this->id'";
        $result = $this->mysqli->query($sql);
        $row = mysqli_fetch_assoc($result);
        $gamemodeId = $row['id_gamemode'];


        if ($this->mysqli->affected_rows == 1) {

            $sql1 = "SELECT * from gamemodes where id = $gamemodeId ";
            $result1 = $this->mysqli->query($sql1);
            $row1 = mysqli_fetch_assoc($result1);

            $row['game_name'] = $row1['game_name'];
            $row['id_game'] = $row1['id_game'];
            $row['console_name'] = $row1['console_name'];
            $row['id_console'] = $row1['id_console'];
            $row['gamemode_name'] = $row1['to_string'];
            $row['online'] = 1;
            $row['private'] = 0;


            return $row;

        } else {


            return "0";
        }
    }

    public function exitQueue()
    {

        if ($this->isInQueue() == 1) {

            $sql = "UPDATE queues SET finished= 1 where id_user = '$this->id'";
            $this->result = $this->mysqli->query($sql);

            if ($this->result == 1) {

                return 1;

            } else {

                $this->error = "error while exiting queue";
                return 0;
            }
        } else {

            $this->error = "not in queue";
            return 0;
```

```
    }


}

public function createMatch($queue, $opponentId) //1= match created; 0 = error while creating
{
    $gamemodeId = $queue['id_gamemode'];
    $moneyBet = $queue['money_bet'];
    $online = $queue['online'];
    $private = $queue['private'];

    if ($this->isInMatch() == 0) { // no estoy ya en una partida

        date_default_timezone_set("Europe/Madrid");
        $timestamp = date_default_timezone_get();
        $mydate = date('Y-m-d H:i:s', strtotime($timestamp));

        $sql = "INSERT INTO matches(time_created,id_host,id_gamemode,money_bet,online,private,finished) VALUES
('$mydate',$this->id,$gamemodeId,$moneyBet,$online,$private,0)";
        $result = $this->mysqli->query($sql);

        $idMatch = $this->mysqli->insert_id;


        if ($result == 1) {

            //Validar si ya esta el usuario ni el oponente estan metidos en la partida. (users_matches duplicated)
            $sql3 = "SELECT * from users_matches where id_match =  $idMatch and id_user = $this->id and id_user <>
$opponentId";
            $result3 = $this->mysqli->query($sql3);


            if ($this->mysqli->affected_rows == 0) { //si no esta ya en la partida

                $sql4 = "INSERT INTO users_matches (id_user, id_match, leaved, declared, reported) VALUES ($this-
>id,$idMatch,0,0,0)";
                $result4 = $this->mysqli->query($sql4);

                //metemos al opponente tmb

                $sql5 = "INSERT INTO users_matches (id_user, id_match, leaved, declared, reported) VALUES
($opponentId,$idMatch,0,0,0)";
                $result5 = $this->mysqli->query($sql5);

                if ($result4 == 1 and $result5 == 1) {

                    return 1;

                }

            }

        }


    }

    return 0; //si llega aqui esque algo ha ido mal

}
```

```
  public function createPrivateMatch($gamemodeId, $moneyBet) //1= match created; 0 = error while creating
  {

    if ($this->isInMatch() == 0) { // no estoy ya en una partida

        date_default_timezone_set("Europe/Madrid");
        $timestamp = date_default_timezone_get();
        $mydate = date('Y-m-d H:i:s', strtotime($timestamp));

        $sql = "INSERT INTO matches(time_created,id_host,id_gamemode,money_bet,online,private,finished) VALUES
('$mydate',$this->id,$gamemodeId,$moneyBet,0,1,0)";
        $result = $this->mysqli->query($sql);

        $idMatch = $this->mysqli->insert_id;


        if ($result == 1) {

            //Validar si ya esta el usuario ni el oponente estan metidos en la partida. (users_matches duplicated)
            $sql3 = "SELECT * from users_matches where id_match =  $idMatch and id_user = $this->id";
            $result3 = $this->mysqli->query($sql3);


            if ($this->mysqli->affected_rows == 0) { //si no esta ya en la partida

                $sql4 = "INSERT INTO users_matches (id_user, id_match, leaved, declared, reported) VALUES ($this-
>id,$idMatch,0,0,0)";
                $result4 = $this->mysqli->query($sql4);

                if ($result4 == 1) {

                    return 1;

                }

            }

        }


    }

    return 0; //si llega aqui esque algo ha ido mal

  }

  public function getUserGameInfo($gameId)
  {

    $sql = "SELECT * from users_games where id_user = $this->id";
    $result = $this->mysqli->query($sql);
    $row = mysqli_fetch_assoc($result);

    return $row;
  }


}
?>
```

2. USERINMATCH.CLASS:

```php
<?php

class UserInMatch extends User
{

   public $id;
   private $mysqli;
   var $matchId;
   var $creationResult;
   var $accepted;
   var $leaved;
   var $declared;
   var $reported;
   var $teamId;
   var $nickname;
   var $level;
   var $zone;
   var $division;
   var $error;
   public $result;

   //private $numberOfTeams;


   //contructor

   function __construct(mysqli $mysqli, $id)
   {
      $this->id = $id;
      $this->mysqli = $mysqli;

      $sql = "SELECT * from users where id = $this->id";
      $result = $this->mysqli->query($sql);
      $row = mysqli_fetch_assoc($result);

      $this->user = $row['user'];
      $this->country = $row['country'];
      $this->mastery = $row['mastery'];
      $this->lastLogin = $row['last_login'];
      $this->email = $row['email'];
      $this->defaultIcon = $row['default_icon'];

      $sql = "SELECT * from users_matches where leaved = 0 and id_user = $this->id and id_match IN
(SELECT id from matches where finished =0)";
      $result = $this->mysqli->query($sql);


      if ($this->mysqli->affected_rows == 1) {

         $row = mysqli_fetch_assoc($result);
```

```
            $this->creationResult = 1;
            $this->matchId = $row['id_match'] + 0;
            $this->leaved = $row['leaved'] + 0;
            $this->declared = $row['declared'] + 0;
            $this->reported = $row['reported'] + 0;
            $this->teamId = $row['id_team'] + 0;
            $this->accepted = $row['accepted'] + 0;

        } else {

            $this->error = "affected rows = 0; Este user no esta en la partida ID: " . $this->id;

        }

        $sql1 = "SELECT * from users_games where id_user = $this->id and id_game = (Select id_game from
gamemodes where id = (Select id_gamemode from matches where id = $this->matchId))";
        $result1 = $this->mysqli->query($sql1);
        $row1 = mysqli_fetch_assoc($result1);

        $this->nickname = $row1['nickname'];
        $this->level = $row1['level'];
        $this->zone = $row1['zone'];
        $this->division = $row1['division'];


    }


    public function getCreationResult()
    {

        return $this->creationResult;

    }

    public function getError()
    {
        return $this->error;

    }

    public function acceptMatch()
    {

        if ($this->accepted == 1) {

            $this->error = "ya tiene accepted = 1";
            $this->result = 0;
            return 0; //ya ha aceptado la partida

        } else {
            $sql = "UPDATE users_matches SET accepted = 1 where id_user = $this->id and id_match = $this-
>matchId";
            $result = $this->mysqli->query($sql);
            if ($result == 1) {
```

```
            $this->result = 1;
            return 1;

        } else {
            $this->error = "fallo al update users_matches(accepted = 1)";
            $this->result = 0;
            return 0;
        }
    }
  }

  public function leaveMatch()
  {

    if ($this->leaved == 1) {
        $this->result = 0;
        $this->error = "ya tiene leaved = 1";
        return 0; //ya ha abandonado la partida

    } else {

        $sql = "UPDATE users_matches SET leaved = 1 where id_user = $this->id and id_match = $this-
>matchId";
        $result = $this->mysqli->query($sql);

        if ($result == 1) {
            $this->result = 1;
            return 1;

        } else {
            $this->error = "fallo al update users_matches(leaved = 1)";
            $this->result = 0;
            return 0;
        }

    }


  }

  public function reportMatch()
  {

    if ($this->reported == 1) {

        $this->error = "ya tiene reported = 1";
        $this->result = 0;
        return 0; //ya ha abandonado la partida

    } else {
        $sql = "UPDATE users_matches SET reported = 1 where id_user = $this->id and id_match = $this-
>matchId";
        $result = $this->mysqli->query($sql);
        if ($result == 1) {
            $this->result = 1;
```

```
            return 1;
        } else {
            $this->error = "fallo al update users_matches(reported = 1)";
            $this->result = 0;
            return 0;
        }
    }


}

public function cancelAccept()
{
    if ($this->accepted == 0) {

        $this->error = "ya tiene accepted = 0";
        $this->result = 0;
        return 0; //ya ha cancelado el accept

    } else {
        $sql = "UPDATE users_matches SET accepted = 0 where id_user = $this->id and id_match = $this-
>matchId";
        $result = $this->mysqli->query($sql);
        if ($result == 1) {
            $this->result = 1;
            return 1;

        } else {
            $this->error = "fallo al update users_matches(accepted = 0)";
            $this->result = 0;
            return 0;
        }
    }

}

public function hasUserAccept()
{

    return $this->accepted;
}

public function hasUserLeaved()
{

    return $this->leaved;
}

public function hasUserReported()
{

    return $this->reported;
}

}
```

```
?>
```

3. SERVER.CLASS:

```
<?php



class Server
{
    private $id;
    private $mysqli;
    private $user;
    var $result;
    var $error;

    //contructor

    function __construct(mysqli $mysqli, $id)
    {

        $this->id = $id;
        $this->user = new User($mysqli, $id);
        $this->mysqli = $mysqli;
        $this->result = "1";
        $this->msg = ""; //mensaje cuando no da error
        $this->error = "null"; //mensaje cuando da error

    }



    public function analyzeTimeToAcceptInMatches(){

        $sql = "SELECT * from matches where finished = 0 and started = 0";
        $result = $this->mysqli->query($sql);
        $row = mysqli_fetch_assoc($result);
    }
    //User
    public function obtainUserGameInfoOfGameIsPlaying($userId)
    {
        $queue = $this->user->getQueueInfo();
        $gamemodeId = $queue['id_gamemode'];

        $sql = "SELECT * from users_games where id_user ='$userId' and id_game = (Select id_game from gamemodes where
id = '$gamemodeId')";
        $result = $this->mysqli->query($sql);
        $row = mysqli_fetch_assoc($result);

        return $row;


    }

    public function obtainUserGameInfo($userId,$gameId)
```

```
    {

        $sql = "SELECT * from users_games where id_user ='$userId' and id_game = $gameId";
        $result = $this->mysqli->query($sql);
        $row = mysqli_fetch_assoc($result);

        return $row;


    }

    //Queue
    public function howIsQueue()// 1 = FoundOpponent; 0 = NotFoundOpponent; -1 = error
    {

        $isInQueue = $this->user->isInQueue();
        $isInMatch = $this->user->isInMatch();

        /*echo "<br>";
         echo "in queue: " . $isInQueue;
         echo "in match: " . $isInMatch;
         echo "<br>";*/

        if ($isInQueue == 1 and $isInMatch == 0) {

            $queue = $this->user->getQueueInfo();
            $opponentId = $this->findOpponentId();

            if ($opponentId == -1) {

                $this->error = "error while finding opponent";
                return -1;

            } elseif ($opponentId > 0) { // si me toca a mi y ha encontrado opponent:

                $opponent = new user($this->mysqli, $opponentId);

                $exitQueueOpponentResult = $opponent->exitQueue();
                $exitQueueResult = $this->user->exitQueue();

                if ($exitQueueOpponentResult != 1 or $exitQueueResult != 1) {

                    $this->error = "Error while exiting queue and exiting queue to opponent";
                    return -1;
                }

                if ($this->user->createMatch($queue, $opponentId) == 1) {

                    return 1;

                } else {

                    $this->error = "Error while creating match";
                    return 1;
                }

            } elseif ($opponentId == 0) {

                return 0;

            } else {
```

```
            $this->error = "unknown error";
            return -1;
        }

    } elseif ($isInQueue == 0 and $isInMatch == 1) {


    } elseif ($isInQueue == 1 and $isInMatch == 1) {

        $this->error = "IN queue and IN match";
        return -1;

    } elseif ($isInQueue == 0 and $isInMatch == 0) {

        $this->error = "NOT in queue and NOT in match";
        return -1;

    } else {

        $this->error = "unknown error";
        return -1;

    }

  }

  public function findOpponentId() // return 0 of opponent ID
  {

    $queue = $this->user->getQueueInfo();
    $gamemodeId = $queue['id_gamemode'];
    $moneyBet = $queue['money_bet'];

    $sql = "SELECT id_user from queues where id = (SELECT MIN(id) from queues where id_gamemode =
$gamemodeId and money_bet = $moneyBet and finished = 0)";
    $result = $this->mysqli->query($sql);
    $row = mysqli_fetch_assoc($result);


    if ($this->mysqli->affected_rows == 1) {


        if ($row['id_user'] == $this->id) { //Si me toca a mi en la cola yo = MIN queue


            //busco la info de mi rival que es el MIN queue + 1 O MIN queue que no sea yo
            $sql2 = "SELECT id_user from queues where id = (SELECT MIN(id) from queues where id_user <> $this->id
and id_gamemode = $gamemodeId and money_bet = $moneyBet and finished = 0)";
            $result2 = $this->mysqli->query($sql2);
            $row2 = mysqli_fetch_assoc($result2);

            if ($this->mysqli->affected_rows == 1) {

                return $row2['id_user'];

            } else {

                $this->msg = "Is my turn But there is no opponent yet";
                return 0;
            }
```

```
        } else {

            //no me toca a mi en la cola por lo que no hago nada. Muestro cuanto me queda para llegar al min
            $this->msg = "Position in queue: " . $this->id - $row['min_id'];
            return 0;
        }


            } else {


                $this->error = "There is no MIN queue and it has to be as you
are in queue. ERROR";
                return -1;
            }


        }

        //error
        public function getError()
        {
            return $this->error;

        }


}

?>
```

4. MATCH.CLASS:

```
<?php

class Match
{

    private $userId;
    private $mysqli;

    public $id;
    public $hostId;
    public $timeStarted;
    public $timeCreated;
    public $timeToAccept;
    public $gamemode;
    public $moneyBet;
    public $numberOfAccepts;
    public $online;
    public $private;
    public $started;
    public $finished;
```

```
    public $reported;
    public $numberOfUsers;
    public $users;

    var $state;

    var $creationResult;
    var $error;


    //private $numberOfTeams;


    //contructor

    function __construct(mysqli $mysqli, $userId)
    {

        $this->userId = $userId;
        $this->mysqli = $mysqli;

        $sql = "SELECT * from matches where  finished = 0 and id = (Select id_match from users_matches where leaved
= 0 and id_user =$this->userId limit 1)";
        $result = $this->mysqli->query($sql);
        $row = mysqli_fetch_assoc($result);

        if ($this->mysqli->affected_rows == 1) {

            $this->id = $row['id'];//id Match
            $this->creationResult = 1; //correcto
            $this->hostId = $row['id_host'];
            $this->timeStarted = $row['time_started'];
            $this->timeCreated = $row['time_created'];
            $this->gamemode = new Gamemode($mysqli,$row['id_gamemode']);
            $this->moneyBet = $row['money_bet'];
            $this->online = $row['online'];
            $this->private = $row['private'];
            //$this->started = $row['started'];
            $this->finished = $row['finished'];
            $this->reported = $row['reported'];

            $sql1 = " SELECT IF(0=(SELECT SUM(accepted)-number_of_players FROM matches AS a,users_matches
AS b,gamemodes AS c WHERE a.id = b.id_match AND a.id_gamemode = c.id),1,0) AS started";
            $result1 = $this->mysqli->query($sql1);
            $row1 = mysqli_fetch_assoc($result1);
            $this->started = $row1['started'];

            $this->timeToAccept = $this->getTimeToAcceptMatch();
            $this->users = array();
            $this->users[0] = new UserInMatch($this->mysqli, $this->userId);

            $sql2 = "SELECT * from users_matches where id_user <> $this->userId and id_match=$this->id";
            $result2 = $this->mysqli->query($sql2);

            if ($this->mysqli->affected_rows > 0) {

                $i = 1;
                while ($row2 = mysqli_fetch_assoc($result2)) {

                    $this->users[$i] = new UserInMatch($this->mysqli, $row2['id_user']);
                    if ($this->users[$i]->getCreationResult() != 1) $this->creationResult = 0; //ha habido error creando rivales
```

```
            $i++;
        }

    }

} else {

    $this->creationResult = 0;
    $this->error = "affected rows = 0, no existe una partida en la que este. Imposible ya que InMatch = 1";

}


}

public function getMatchTime()
{

if ($this->started == 1) {

    date_default_timezone_set("Europe/Madrid");
    $timestamp = date_default_timezone_get();
    $mydate = date('Y-m-d H:i:s', strtotime($timestamp));

    $timeStarted = date('Y-m-d H:i:s', strtotime($this->timeStarted));

    $datetime1 = new DateTime();
    $datetime2 = new DateTime($timeStarted);
    $interval = $datetime1->diff($datetime2);


    $timeTo = $interval->format("%h h %i min y %s seg");
    return $timeTo;


} else {

    $error = "match not started yet";
    return 0;
}
}

public function getTimeToAcceptMatch()
{

date_default_timezone_set("Europe/Madrid");
$timestamp = date_default_timezone_get();
$mydate = date('Y-m-d H:i:s', strtotime($timestamp));

$timeCreated = date('Y-m-d H:i:s', strtotime($this->timeCreated));

$datetime1 = new DateTime();
$datetime2 = new DateTime($timeCreated);
$interval = $datetime1->diff($datetime2);

if ($interval->format("%i") > 0 or $interval->format("%h") > 0) {

    $error = "lleva mas de 5h minuto";

    $timeTo = 60 * 60 - ($interval->format("%i") * 60 + $interval->format("%s") * 1);
    return $timeTo;
```

```
    } else {

        $timeTo = 60 * 60 - $interval->format("%i") * 60;
        return $timeTo;
    }

  }

  public function hasMatchStarted()// 1 = started,0 not started, -1 error;
  {

    if ($this->started == 0) {

        return 0;

    } elseif ($this->started == 1) {

        return 1;
    }

  }

  public function getMatchObject()
  {
    return $this;
  }

  public function getMatchInfo()
  {
    return json_encode($this);
  }

  public function getError()
      {
          return $this->error;

      }
}

?>
```