



UNIVERSIDAD DE MÁLAGA



E.T.S. INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADUADO EN INGENIERÍA INFORMÁTICA

DESARROLLO DE VIDEOJUEGOS DEFI Y SU
SOSTENIBILIDAD ECONÓMICA

DEVELOPMENT OF DEFI GAMES AND ITS ECONOMIC
SUSTAINABILITY

Realizado por
SERGIO GONZÁLEZ SICILIA

Tutorizado por
ISAAC AGUDO RUIZ

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE DE 2022



UNIVERSIDAD
DE MÁLAGA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
[GRADUADA/O EN TITULACIÓN]

Desarrollo de videojuegos DeFi y su sostenibilidad económica

Development of DeFi games and its economic sustainability

Realizado por
Sergio González Sicilia

Tutorizado por
Isaac Agudo Ruiz

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2022

Fecha defensa: Septiembre de 2022

Resumen

Hoy en día las finanzas descentralizadas, Bitcoin, Ethereum, NFT, criptomonedas, son cada vez más conocidos y junto a esto la normalización de esta tecnología, dejando de ser para muchas personas algo complejo y abstracto, y convirtiéndose en una oportunidad para muchos usuarios gracias a sus funcionalidades.

Junto al avance de la tecnología blockchain y la aparición de los smart contracts en 2014 por parte de Ethereum, se ha abierto un nuevo mundo de posibilidades, y una de ellas ha sido su utilización en el mundo de los videojuegos, esta moda se ha extendido mucho a lo largo de los últimos años, y uno de sus mayores exponentes ha sido el videojuego Axie Infinity creado en el año 2018.

La motivación de este proyecto es la investigación de esta tecnología aplicada al mundo de los videojuegos, ver las diferentes formas en las que se podría implementar y la viabilidad económica en cada una de ellas, ya que en la actualidad muchas personas han empezado a vivir o a conseguir beneficio de este tipo de videojuegos, ya sea por la especulación de NFTs o simplemente el hecho de jugarlos.

A parte se creará un prototipo donde se pondrán algunas de estas utilidades de la tecnología blockchain en un videojuego simple que mediante el uso de NFTs puedas jugar y ganar recompensas.

Palabras clave:

Blockchain, Ethereum, Contratos Inteligentes, Solidity, Videojuegos

Abstract

Nowadays decentralized finance, Bitcoin, NFT, cryptocurrencies are becoming more and more popular and along that the normalization of this technology, ceasing to be something complex and abstract for many people and becoming an opportunity for many users thanks to its functionalities.

Along with the advancements of the blockchain technology and the appearance of smart contracts in 2014 by Ethereum, a new world of possibilities has opened up, and one of them has been its use in the ambit of video games, this trend has spread a lot over the last few years, and one of its greatest exponents has been the Axie Infinity video game created in 2018.

The motivation of this project is the investigation of this technology applied to the video games, to see the different ways in which it could be implemented and the economic viability in each of them, since nowadays many people have begun to live or make profit out of this type of video games, either by speculating on NFTs or simply by playing them.

In addition, a prototype will be created where some of these blockchain technology utilities will be put into a simple video game that using NFTs you can play and earn rewards.

Keywords:

Blockchain, Ethereum, Smart contracts, Solidity, Video games

Índice

Resumen	1
Abstract	1
Índice	1
Introducción	1
1.1 Motivación	1
1.2 Objetivos	3
1.3 Estructura de la memoria	3
Estudio del arte	5
2.1. Vista general.....	5
2.2. Conceptos básicos.....	6
2.2.1. Algoritmo de consenso	6
2.2.2. Sistema distribuido	6
2.2.3. Funciones hash	6
2.2.4. Criptografía de clave pública	7
2.2.5. Red Peer-to-peer (P2P).....	7
2.3. Blockchain	8
2.4.1 Layers.....	9
2.5. Ethereum.....	10
2.5.1. Funcionamiento de la blockchain.....	10
2.5.2. Aplicaciones.....	13
Blockchain y videojuegos.....	17
3.1. Videojuegos tradicionales	17
3.2. Aplicaciones de la tecnología blockchain a los videojuegos.....	18
3.2.1. Metaverso.....	18
3.2.2. Play to earn.....	19
Prototipo y especificaciones	27
4.1. Ejemplos videojuegos play to earn.....	27
4.1.1. Click to Earn	27
4.1.2. Axie Infinity	28
4.2. Motivación y descripción del prototipo.....	28
4.3. Tecnologías empleadas en el prototipo.....	28
4.3.1. Lenguajes de programación	28
4.3.2. Frameworks, entorno de desarrollo y tecnologías utilizadas.....	29
Diseño de la prueba de concepto	31
5.1. Metodología de espiral.	31
5.2. Fases del proyecto.	32
5.3 Requisitos funcionales.....	33
5.4 Requisitos no funcionales.....	33

5.5. Diseño del contrato	34
5.6. Casos de uso	34
5.7 Funcionamiento conexión con contratos	42
Implementación	43
6.1. Contrato inteligente.....	43
6.1.1. Variables Cryptochickens.....	44
6.1.2. Funciones Cryptochickens	47
6.2. Estructura del código	48
6.3. Cliente.....	51
Testing y resultados	53
Conclusiones y líneas futuras	57
8.1. Conclusiones	57
8.2. Líneas futuras	58
Referencias.....	61
Manual de Instalación.....	63
A.1. Instalación de metamask	63
A.2. Conexión con wallet	64
A.3. Mintear chickens	68
A.4. Visualización de chickens y recompensas	70
A.4.1. Subida de nivel	71
A.4.2. Cambio de nombre.....	72
A.4.3. Transferencia de chicken.....	74
A.5. Combate	75
A.6. Retirar recompensas.....	78
A.7 Reproducción de chickens.....	79

1

Introducción

1.1 Motivación

En los últimos años, la tecnología blockchain ha aumentado muchísimo su popularidad y uso, esto ha sido propiciado en gran parte por las criptomonedas y la oportunidad que brindan a sus usuarios que buscan la descentralización, donde grandes fondos de inversión e incluso países como El Salvador han puesto su confianza con la intención de que sean las finanzas del futuro. Es por ello por lo que el aprendizaje de esta tecnología, sus fundamentos y sus diferentes aplicaciones son algo que cada vez va a ir cobrando más importancia.

Una de las principales aplicaciones de la tecnología blockchain son los *smart contracts* o contratos inteligentes que son programas informáticos que de manera autónoma y descentralizada realizan la función que su creador les haya programado. La diferencia que ofrecen con respecto a la programación común es que estos códigos se encuentran almacenados en la blockchain de manera pública, permanente e inmutable, donde cualquier persona puede acceder a sus códigos y confirmar su legitimidad y al ser inmutables se evita la manipulación de estos.

Otro concepto importante son las DeFi o finanzas descentralizadas, concepto que hace referencia a un sistema financiero construido sobre la tecnología blockchain. Su principal característica radica en que son los propios usuarios quienes intercambian (ofertan y demandan) activos y servicios financieros directamente entre ellos, sin intermediarios, para usarlos como mecanismo de inversión o financiación.

En los últimos años el dinero almacenado y número de usuarios en las tecnologías DeFi ha aumentado en gran cantidad.

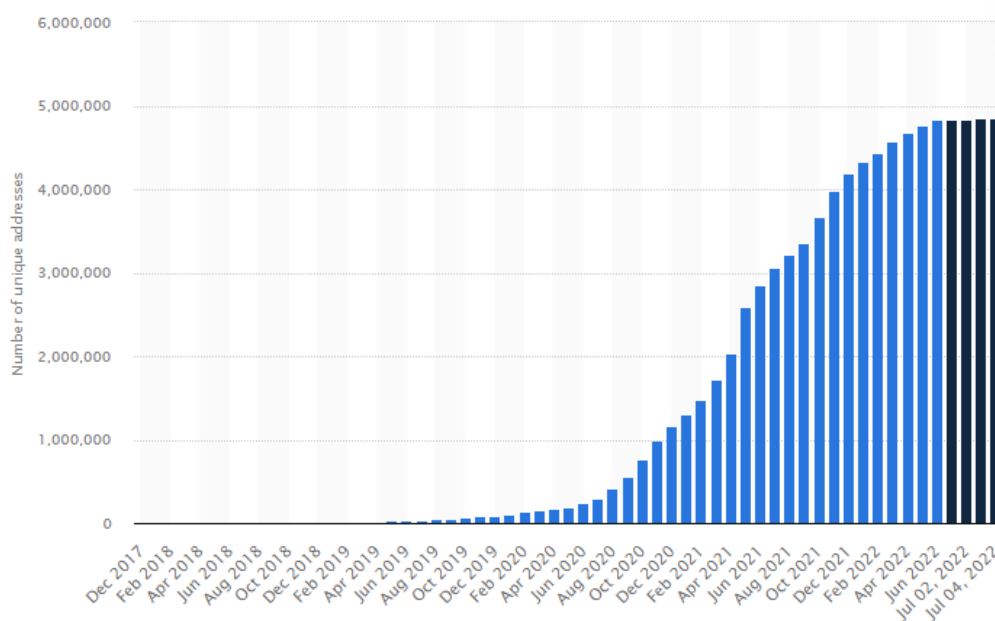


Figura 1.1. Crecimiento del número de usuarios activos en tecnologías DeFi. (extraída de <https://www.statista.com/statistics/1297745/defi-user-number/>)

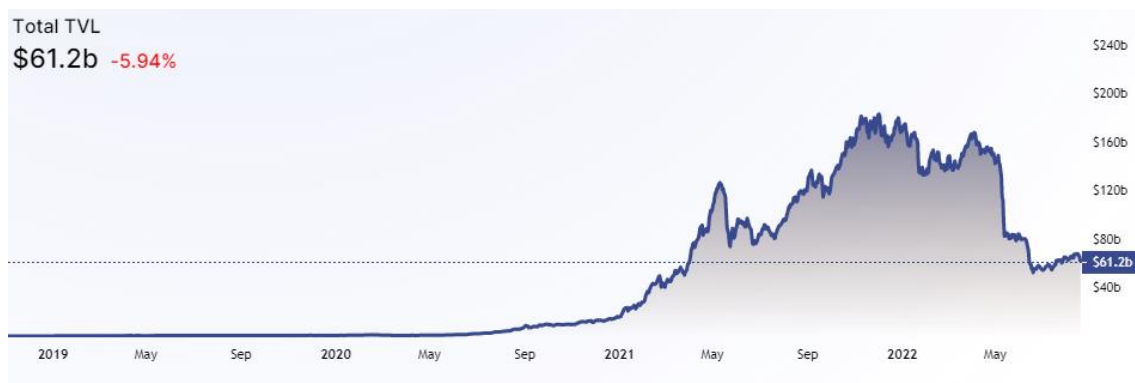


Figura 1.2. Cantidad de dólares almacenados en tecnologías DeFi. (extraído de <https://defillama.com/>)

Junto a todo este crecimiento, y gracias a las tecnologías DeFi en conjunto con la de los NFTs ha surgido una nueva ola con nuevas aplicaciones para la blockchain y una de estas es su aplicación en los videojuegos, los videojuegos DeFi, que crecido exponencialmente en un muy corto periodo de tiempo y junto a su gran crecimiento surge la duda de si estos son realmente el futuro o simplemente una moda pasajera, de si su aplicación en este momento es la forma correcta o si existen otras posibilidades por explorar para su instauración en los videojuegos actuales.

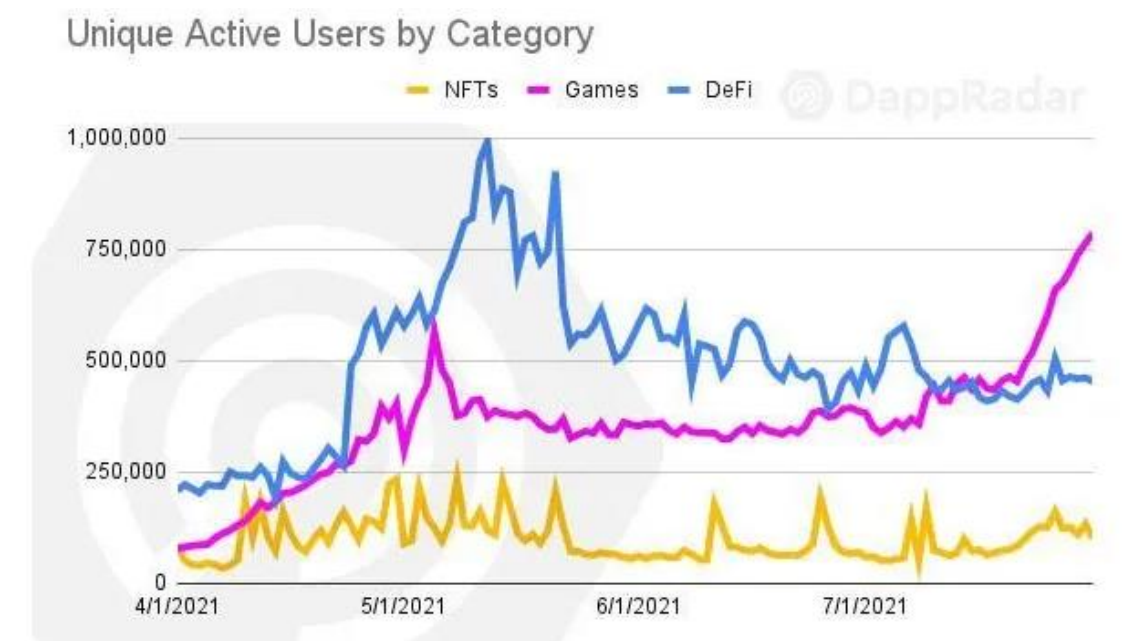


Figura 1.3. Comparativa de usuarios activos entre NFTs, DeFi y juegos DeFi. (extraído de <https://dappradar.com/>)

1.2 Objetivos

El objetivo principal de este proyecto es la investigación y familiarización con las tecnologías blockchain (Bitcoin, Ethereum...), desde sus inicios hasta la actualidad, analizándola en profundidad y buscando las oportunidades que esta puede brindar.

Por otro lado, se analizarán las diferentes maneras que estas herramientas se pueden aplicar a los videojuegos, analizando las aproximaciones que se han realizado hasta la fecha y buscando cuál de ellas es la que *a priori* es la más acertada por sostenibilidad y por utilidad.

Por último, se propone la elaboración de un prototipo de aplicación web, en la que mediante el uso de NFTs, tokens y contratos inteligentes se cree un videojuego sencillo con los conocimientos adquiridos, utilizando solidity. El desarrollo aparte de tener su implementación tecnología blockchain, utiliza tecnologías de desarrollo web actuales, tales como React o NodeJS.

En este videojuego se pondrán en práctica algunas de las utilidades de los NFT y tokens que se explican en la memoria, será un juego simple donde se podrán conseguir NFTs, utilizarlos para atacar, ganar recompensas etc.

1.3 Estructura de la memoria

El proyecto se ha dividido en 8 capítulos. A continuación, se detallará de manera breve de que trata cada uno de estos capítulos:

- Capítulo 1. Introducción.
Explica la motivación, objetivos y estructura del proyecto.

- Capítulo 2. Estudio del arte.
Estudio sobre el funcionamiento de la blockchain, Ethereum y conceptos importantes.
- Capítulo 3. Blockchain y videojuegos.
Estudio sobre los videojuegos tradicionales y los videojuegos DeFi.
- Capítulo 4. Prototipo y especificaciones.
Ejemplos de videojuegos DeFi y resumen del prototipo y tecnologías utilizadas.
- Capítulo 5. Diseño de la prueba de concepto.
Explicación de la metodología de desarrollo, análisis y especificaciones de requisitos y casos de uso dentro del proyecto.
- Capítulo 6. Implementación
Estructura del desarrollo del proyecto, partes del contrato inteligente y partes del cliente.
- Capítulo 7. Testing y resultados
Comprobaciones del buen funcionamiento del contrato mediante herramientas de Testing.
- Capítulo 8. Conclusiones y líneas futuras
Apartado donde se sacan las conclusiones sobre la sostenibilidad de los videojuegos DeFi y mejoras posibles al prototipo creado.

2

Estudio del arte

En este apartado se estudiará como es el funcionamiento original de la tecnología blockchain, el funcionamiento de las redes como Bitcoin y Ethereum.

2.1. Vista general

En el año 2008, Satoshi Nakamoto creador de bitcoin, creó un modelo donde varios ordenadores conectados creaban una base de datos colaborativa y segura, por lo que se puede tomar a la blockchain como simplemente una base de datos distribuida. Esta base de datos consiste en una cadena de bloques donde cada uno almacena datos que han sido encriptados y se les ha otorgado un identificador único llamado **hash**.

Para el funcionamiento de la blockchain existen los **mineros** que se dedican a computar y resolver los bloques que se están produciendo, añadiéndole las transacciones generadas. Una vez estos bloques han sido procesados, se envían a la red para que el resto de los usuarios tengan una copia de la base de datos, a esto se le llama **algoritmo de consenso distribuido** que quiere decir que no existe una base de datos centralizada, sino que para cada cambio producido es necesario un acuerdo entre todos los mineros, esto hace que sea muy segura y difícil de alterar.

Los bloques que se van generando no pueden ser eliminados y cada uno de estos posee un código hash que es dependiente del código hash del anterior bloque minado. Esto significa que si un bloque intenta ser cambiado el resto de los bloques posteriores tendrían que ser modificados para poder encajar con el bloque modificado.

Esta fue la idea original para la blockchain en el momento de su creación, pero hoy en día su significado ha sido expandido a muchos más ámbitos y ha dejado de ser una simple base de datos descentralizada, gracias a los avances realizados por Ethereum como la implementación de contratos inteligentes.

2.2. Conceptos básicos

Para poder tratar más en profundidad toda esta tecnología primero es importante comentar algunos de los conceptos más importantes.

2.2.1. Algoritmo de consenso

Es un mecanismo que permite a usuarios o maquinas coordinarse en un entorno distribuido y en conjunto tomar decisiones, donde entre los diferentes nodos se debe llegar a un acuerdo incluso si algunos de estos poseen una respuesta errónea.

Para poder participar en estas decisiones un usuario debe poseer un stake (participación), esta puede conseguirse de varias formas, por ejemplo, con capacidad de computación, pero ¿Por qué un usuario iba a utilizar sus recursos para poder formar parte de las decisiones?, pues la respuesta es sencilla y es porque se reparten recompensas, que normalmente son en forma de la criptomoneda nativa de la blockchain en concreto. [5]

Los algoritmos de consenso más utilizados son Proof of Work (PoW) y Proof of Stake (PoS) en los cuales se profundizará más adelante.

2.2.2. Sistema distribuido

Es un conjunto de programas o sistemas informáticos que utilizan recursos de computación repartidos en diferentes nodos para lograr un objetivo común. La finalidad de estos es eliminar cuellos de botella, los problemas de errores y de seguridad. [2]

2.2.3. Funciones hash

El “hashing” es el proceso por el cual se genera una salida de datos con una extensión fija, sea cuales sean los datos que vengan de entrada. Aparte son deterministas esto quiere decir que de una entrada siempre se va a obtener una misma salida y viceversa. Normalmente los algoritmos de hashing se diseñan como funciones direccionales, esto quiere decir que no son fáciles de descryptar y para ello se deben emplear una gran cantidad de tiempo y recursos.

Gracias a las funciones hash, las blockchains y otros sistemas son capaces de mantener una alta integridad y seguridad de los datos.

Las características básicas que debe cumplir una función hash son:

- **Resistencia a la colisión:** inviable encontrar dos entradas distintas que produzcan el mismo hash como salida.
- **Resistencia a preimagen:** inviable descifrar una función hash ya procesada.
- **Efecto avalancha:** cambios mínimos en la entrada producen hashes totalmente distintos.



Figura 2.1. Ejemplo efecto avalancha

Como se puede apreciar en la **figura 2.1**, el simple cambio de la “s” de minúscula a mayúscula genera un hash totalmente diferente.

2.2.4. Criptografía de clave pública

También conocida como criptografía asimétrica, utiliza dos claves una privada y una publica para la encriptación de información, no es único de las blockchain, esto se utiliza en muchos otros ámbitos de la informática como por ejemplo para conexiones seguras con sitios web (SSL). [1]

En el caso de las blockchains es importante destacar que no se utiliza de la misma forma que en las conexiones web. En la blockchain esta metodología es necesaria para que no cualquier persona pueda acceder a tus activos, poder firmar transacciones y comprobar la autenticidad e integridad de estas, por esto cuando se crea una wallet al usuario se le otorgan una clave privada y una pública. En el caso de Ethereum y Bitcoin, estos utilizan el algoritmo de firma de curva elíptica (ECDSA).

2.2.5. Red Peer-to-peer (P2P)

En las ciencias de la computación, una red peer to peer consiste en un grupo de dispositivos que en conjunto almacenan y comparten datos. Cada participante se le denomina nodo y actúa como un peer. Normalmente todos los nodos tienen un poder similar y realizan la misma tarea. [10]

En las tecnologías financieras, este término se utiliza normalmente para denominar los intercambios de criptomonedas a través de una red distribuida. Una red P2P permite a sus usuarios realizar compras y ventas sin la necesidad de un intermediario.

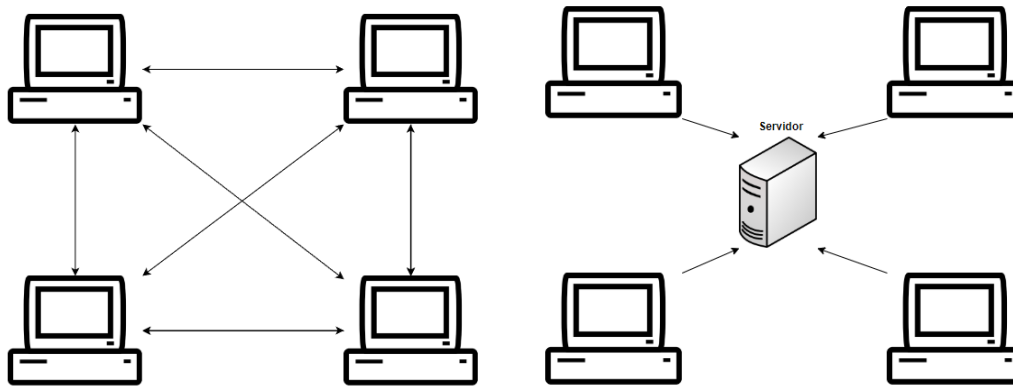


Figura 2.2. Diferencia entre sistemas P2P y centralizados

Como se puede observar en la figura 2.2 en el caso de P2P todos los dispositivos se encuentran conectados entre sí compartiendo información, mientras en el diseño tradicional existe un servidor que regula el paso de información entre los diferentes dispositivos.

2.3. Blockchain

El funcionamiento de la blockchain está basado en la unión de diferentes bloques que contienen información, estos bloques suelen tener unos campos de información determinados, pero los dos principales serían su dirección hash y la dirección hash del bloque anterior, los hashes de cada bloque se obtienen a partir del hash del anterior bloque, por lo que si se intenta modificar de forma fraudulenta uno de estos bloques, habría que cambiar toda la estructura de la cadena.

Un problema que se podría plantear es que hoy en día los ordenadores pueden calcular miles de hashes por segundo por lo que cualquier usuario podría modificar un bloque y a la vez modificar el resto de la cadena en muy poco tiempo, para esto se utiliza una estrategia llamada Proof of work. En el caso de la blockchain de bitcoin este proof of work tiene una duración de 10 minutos, esto quiere decir que para calcular un bloque es necesario computar información por 10 minutos, gracias a esto se hace inviable la posibilidad de modificar de forma fraudulenta cualquier bloque ya que sería imposible recalcularlo.

A esto se le suma una red P2P donde cualquier usuario puede conectarse y formar parte, esta red otorga una copia del historial de bloques completo a cada nodo y en el momento en el que se genera un nuevo bloque, este es enviado a los diferentes nodos para que confirmen que ese bloque es correcto, una vez el bloque ha sido confirmado se añade a la blockchain. A esta metodología se la denomina consenso y gracias a esto se eligen que bloques son correctos y cuales no lo son.

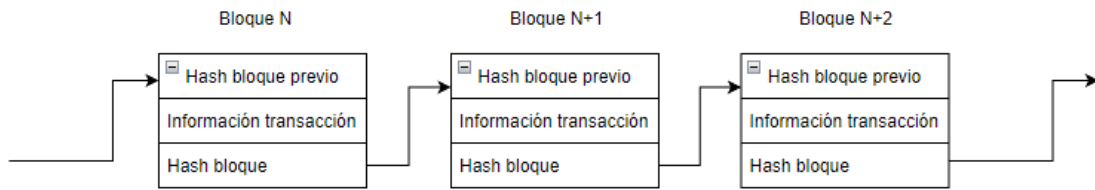


Figura 2.3. Esquema diseño bloques blockchain

En un principio únicamente servía para el almacenamiento de información, pero gracias a Ethereum se añadieron los contratos inteligentes que abrieron un nuevo abanico de posibilidades.

Estos contratos permiten que se puedan crear protocolos en los que no se necesita un intermediario y en los que simplemente mediante la aplicación de reglas, estos pueden realizar transacciones y si no se están de acuerdo con estas reglas simplemente no ejecutar la transacción.

2.4.1 Layers

Los dos tipos de layers más importantes son las layer-1 y las layer-2, es importante saber cómo funcionan y los problemas que plantean para el entendimiento de las blockchains y entender uno sus problemas hoy en día, el cual es la escalabilidad.

- **Layer-1:** Una red de layer-1 es una forma de llamar a una blockchain de base, ejemplos de estas pueden ser Ethereum, Bitcoin y solana. Nos referimos a estas como layer-1 porque son las redes principales de su ecosistema. [3]

En otras palabras, un protocolo de layer-1 es cuando el protocolo en si mismo es el que procesa y finaliza las transacciones de su blockchain. Normalmente estos poseen un token nativo para pagar transacciones.

Un problema muy común de estas redes es su incapacidad de escalar. En el caso de bitcoin y algunas otras grandes blockchains han tenido muchos problemas con el tiempo en el que se tarda en procesar las transacciones en momentos en los que existe mucho tráfico.

El problema de la velocidad viene derivado de la estrategia de proof of work que siguen algunas blockchains y que hacen que al momento de producirse mucha carga de información no se tenga los suficientes recursos como para procesarlas. Algunas soluciones que se han planteado para mejorar este problema son:

- Incrementar el tamaño de los bloques, permitiendo más transacciones por bloque.
- Cambiar el mecanismo de consenso de PoW a PoS, esto lo está realizando Ethereum.
- Implementar sharding. Un tipo de particionamiento de base de datos.

- **Layer-2:** Las blockchain de layer-2 son aquellas que se encuentran alojadas encima de otras blockchains de layer-1. Estas blockchains se crean para mejorar la escalabilidad de las layer-1. [3]

Las soluciones de layer-2 se apoyan en redes secundarias que trabajan en paralelo o independientemente de las redes principales y estas pueden ser de diferentes tipos:

- **Rollups:** Son las más comunes y se basan en juntar fuera de la cadena las transacciones y después enviarlo a la red principal para que se procesen todas juntas.
- **Sidechains:** Son blockchains independientes con sus propios validadores. Donde la sidechain no depende de la red principal pero la red principal debe hacer caso a la sidechain para validar sus transacciones.
- **Nested blockchains:** Esta solución se basa en tener una blockchain “padre” que posee varias blockchains “hijas” que son las que procesan toda la información. La blockchain “padre” en este caso no participa en el procesamiento de transacciones, sino que simplemente se limita a decidir problemas de acuerdo entre los “hijos”

2.5. Ethereum

En 2013, el programador Vitalik Buterin presentó la idea de una blockchain que permitiese desplegar aplicaciones descentralizadas de manera permanente e inmutable dentro de ella. En 2014 se comenzó la programación del proyecto y fue lanzado el 30 de julio del año 2015. Gracias a su aparición se comenzaron a crear nuevos tipos de proyectos descentralizados, tales como las finanzas descentralizadas (DeFi), que son aplicaciones que permiten el intercambio financiero sin intermediarios, los NFTs que son tokens únicos que representan propiedad y las criptomonedas.

2.5.1. Funcionamiento de la blockchain

La blockchain de Ethereum posee muchos elementos interesantes como la validación de sus bloques, su criptomoneda, como funcionan sus cuentas... Todos estos conceptos se explicarán a continuación.

Diseño

Ethereum es una red de ordenadores (nodos) que entre ellos llegan a un consenso en cada bloque que se crea para decidir si añadirlo o no a la blockchain. Cada bloque debe poseer un identificador que hace referencia al bloque anterior de la cadena para poder ser valido. Cuando un nodo añade un bloque nuevo, todas las transacciones dentro de ese bloque son ejecutadas, ya sea alterando el balance de divisas o ejecutando funciones de contratos.

Ether

Ether es la criptomoneda que forma parte del protocolo de Ethereum como recompensas para los mineros de PoW que añaden bloques. El Ether se representa en las wallets de los usuarios como un entero sin signo asociado a cada cuenta y se mide en Wei (10^{18} Wei = 1 Ether), hoy en día por cada bloque que se mina se producen 2 ether.

Cuentas

En Ethereum existen dos tipos de cuentas, las cuentas de usuario y los contratos. Las cuentas de usuario son las únicas que pueden crear transacciones y para que una transacción sea válida tiene que estar firmada con la clave privada de la cuenta. El algoritmo utilizado para realizar la encriptación con la firma es ECDSA (Algoritmo de firma de curva elíptica digital).

Los contratos son las cuentas que están relacionadas con código y almacenamiento. Estos contratos poseen funciones que toman argumentos y pueden devolver valores como en la programación tradicional.

Direcciones de cuenta

Las direcciones de Ethereum se componen de "0x" (identificador común del sistema hexadecimal) concatenado con los 20 bytes más a la derecha del keccak-256 de la clave publica generada con el algoritmo ECDSA.

Virtual machine (VM)

La máquina virtual de Ethereum (EVM) es el entorno en el que se ejecutan las transacciones en Ethereum. Esta incluye una pila, un almacenamiento en memoria, la cantidad de gas, un puntero para las operaciones y un almacenamiento permanente para todas las cuentas (incluidos los contratos. Cuando una transacción llama a una función de un contrato, los argumentos en la llamada se añaden en la pila y la máquina virtual (EVM) traduce el contrato a bytes y se introduce en la pila de operaciones. Estos datos almacenados en la pila pueden ser después almacenados en memoria o almacenados de forma permanente. El EVM ha sido implementado en varios lenguajes de programación tales como, C++, C#, Go, Haskell, Javascript, Rust etc.

Gas

El gas es una unidad de una cuenta que el EVM usa para el cálculo de la cuota de transacción, que es la cantidad de ETH el emisor de una transacción tiene que pagar para que un minero incluya la transacción en el siguiente bloque. Este valor es calculado dependiendo de la cantidad de recursos que gaste la transacción que se esté realizando.

Bomba de dificultad

La bomba de dificultad es una característica que posee Ethereum que hace que cada cierto número de bloques se aumente la dificultad de minar bloques, esto está hecho para que los mineros no consigan demasiado control sobre la red y para incentivar las mejoras en la red. Esta bomba de dificultad fue colocada originalmente con la finalidad de transformar el PoW a PoS una característica que eliminará a los mineros.

Mensajes y transacciones

Las transacciones en Ethereum se refieren a cualquier acción iniciada por una cuenta de un usuario externo humano (los contratos no pueden), para recibir o enviar información. [4]

Una transacción que cambie un estado dentro de la EVM necesita ser transmitido a toda la red. Cualquier nodo puede transmitir una transacción para que se ejecute en la EVM; tras esto un minero ejecutará la transacción y se propagará al resto de la red. Las transacciones requieren el pago de cuotas y requieren ser minadas para ser validas.

Una transacción incluye la siguiente información:

Nombre	Descripción
recipient	La dirección que recibe la transacción
signature	Es la firma del emisor. Se crea cuando la clave privada del emisor firma la transacción y confirma que el emisor ha autorizado a la transacción.
nonce	Un numero incremental que indica el numero de la transacción de las que ha hecho la cuenta hasta el momento.
value	Es la cantidad de ETH que se envía desde el emisor al receptor.
data	campo opcional con datos arbitrarios (este campo lo utilizan las transacciones a contratos).
gasLimit	Cantidad máxima de gas que puede ser consumido por la transacción. Cuanta mayor cantidad de gas, mayor probabilidad de que la transacción sea procesada.
maxPriorityFeePerGas	La máxima cantidad de gas incluida como propina para el minero que procese la transacción.
maxFeePerGas	Cantidad máxima de gas que quiere ser pagada por la transacción.

Tabla 2.4.1. Campos principales de una transacción en Ethereum.

Validación de bloques y minado

En apartados anteriores comentamos el uso de los mecanismos de consenso de Proof of Work (PoW) y Proof of Stake (PoS). Estos mecanismos se utilizan para que sistemas distribuidos puedan trabajar en conjunto y mantenerse seguros.

- **1. Proof of Work (PoW):** Es un mecanismo que permite a la red descentralizada de Ethereum llegar a un consenso o al menos estar de acuerdo en cosas como el balance de cada cuenta o el orden de las transacciones. Esto evita que los usuarios realicen “dobles gastos” de sus monedas y asegura la red de Ethereum.

El PoW está basado en un algoritmo que establece las reglas y el trabajo que los mineros tienen que hacer. Minar es el “trabajo” en sí, es el acto de añadir

bloques validos en una cadena. A mayor cantidad de “trabajo” realizado, mayor será la cadena.

Este “trabajo” requiere que los mineros vayan realizando una búsqueda del código hash del siguiente bloque mediante fuerza bruta, probando la solución una a una.

Los mineros que consiguen crear un bloque satisfactoriamente reciben 2 ETH, por esto se ha formado una gran comunidad de mineros en el mundo y esto ha influido a la subida de precio de muchos productos electrónicos aparte de incentivar el gasto eléctrico.

- **2. Proof of Stake (PoS):** Este mecanismo aún no ha sido implementado de forma oficial en la red de Ethereum, pero este lo será a lo largo del año 2022 y 2023 produciendo la eliminación de PoW y de los mineros.

A diferencia del PoW donde todas las transacciones son decididas por los mineros, en PoS los usuarios depositan sus ETH en un validador que utiliza los activos de sus usuarios para formar partes de las decisiones de la red de Ethereum. Esto es un sistema parecido a lo que serían las acciones de una empresa donde un validador poseería determinada cantidad de ETH y podría participar en las decisiones que se debe tomar para validar los bloques.

Las ventajas con respecto al PoW son amplias:

- Mayor eficiencia energética.
- La barrera de entrada para poder participar es mucho mejor porque no hace falta un ordenador o gráficas para minar.
- Reduce el riesgo de centralización, ya que se incentiva la creación de nuevos y más nodos que validen los bloques.
- Al no haber tanto gasto eléctrico se pueden reducir las recompensas en ETH para los validadores.

2.5.2. Aplicaciones

Usos popularizados en Ethereum son la creación de tokens fungibles (ERC20) y no-fungibles (ERC721), con una gran variedad de propiedades, tales como crowdfunding, finanzas descentralizadas, organizaciones autónomas descentralizadas (DAOs), juegos y apuestas.

Código de los contratos inteligentes

Los contratos de Ethereum están escritos en lenguajes de programación de alto nivel y son compilados por la EVM y desplegados en la blockchain de Ethereum y estos son escritos en su mayoría en el lenguaje de Solidity (lenguaje con similitudes a javascript). Uno de los problemas de los contratos inteligentes son los bugs, tales como agujeros de seguridad que no pueden ser reparados debido a la inmutabilidad de los contratos.
[11]

Tokens ERC-20

ERC-20 (Ethereum Request Comment 20) es un estándar para tokens fungibles utilizados en la blockchain de Ethereum. Un token fungible es aquel que es intercambiable por otro token. Los tokens no fungibles (NFTs) no pueden ser intercambiados. [8]

Este estándar permite a los smart contracts una manera de ser intercambiados. Los tokens pueden ser la representación de una característica, un derecho, una propiedad, un acceso, una criptomoneda o simplemente algo que no es único y que puede ser transferido. El estándar permite a los tokens que representan a uno de estos factores junto a su smart contract puedan ser intercambiado por otro que también lo permita.

El ERC-20 fue propuesto en 2015 como una forma de estandarizar la manera en las que se programaban los tokens en los smart contracts, esta propuesta fue aceptada siguiendo el procedimiento habitual de la comunidad de desarrolladores de Ethereum en 2017, y desde ese momento todos los tokens creados deben tener esa estructura.

Las funciones que se estandarizaron fueron:

- TotalSupply: Número total de tokens máximos que se van a poder crear.
- BalanceOf: La cantidad de tokens que posee cada dueño de los tokens.
- Transfer: Permite realizar transacciones de los tokens especificando una cantidad y un destinatario.
- TransferFrom: Ejecuta automáticamente una transferencia de tokens desde una cuenta específica.
- Approve: Permite a cuentas autorizadas sacar un número determinado de tokens.
- Allowance: Devuelve la cantidad de tokens que tiene una cuenta al dueño del contrato.
- Transfer: Un evento lanzado cuando se confirma una transacción exitosa.
- Approval: Registro con la información de un evento aprobado.

Y todas estas reglas hacen que todos los tokens creados dentro de la red de Ethereum estén estandarizados y funcionen de manera uniforme dentro del ecosistema de Ethereum.

Este tipo de estandarizaciones se pueden encontrar en diferentes blockchains, tales como en la binance smart chain con su estándar BEP-2 o la blockchain de Solana con su estándar SPL, cada uno con sus diferencias, pero con la misma finalidad.

Tokens ERC-721 (NFTs)

Al igual que el ERC-20 es un tipo de estándar de programación para los smart contracts, este fue aceptado en enero de 2018 y se utiliza para la implementación tokens no fungibles en la red de Ethereum, el hecho de no ser fungibles hace que estos no puedan ser reemplazados una vez creados y son completamente únicos [9].

Fungibilidad significa que un activo puede ser intercambiable y que entre ese activo todos son indistinguibles unos de otros.

Un NFT puede ser utilizado para representar un objeto del mundo real en la blockchain, pero también puede utilizarse como un método de coleccionables digitales.

Los NFTs se han popularizado en la cultura actual como la nueva forma de arte. Aunque tiene potenciales usos en otros sectores tales como los videojuegos, crear certificados, licencias o incluso permitir propiedad fraccionada de algunos activos.

Los principales métodos del estándar de programación ERC721 son:

- `BalanceOf (owner)`: Devuelve la cantidad de activos que posee un usuario.
- `OwnerOf (tokenId)`: Devuelve la dirección del dueño de un NFT.
- `TransferFrom(from,to,tokenId)`: Permite enviar un NFT de una cuenta a otra.
- `Approve(to, tokenId)` : Permite dar permisos a un usuario a poder intercambiar nuestro NFT.

Finanzas descentralizadas (DeFi)

Las DeFi ofrecen instrumentos tradicionales de las finanzas en una arquitectura descentralizada, fuera del control de las compañías y de los gobiernos, algunas de las opciones que ofrecen las DeFi son la posibilidad de generar un interés por nuestros activos y la posibilidad de intercambio de criptomonedas de manera descentralizada.
[13]

A raíz de las DeFi surgieron las **organizaciones autónomas descentralizadas (DAO)** que invierten activos como un grupo. Cualquier persona que sea propietaria de una inversión en un token de gobernanza de una DAO puede participar en las decisiones de la organización. Cuanta mayor cantidad de tokens se posean mayor será tu capacidad de voto. Normalmente los fondos de estas organizaciones son invertidos en tokens, NFTs y en generar intereses.

3

Blockchain y videojuegos

En este apartado se obtendrá una idea general de la situación en la que se encuentran los videojuegos hoy día y las posibilidades que ofrece la blockchain en el campo de los videojuegos.

3.1. Videojuegos tradicionales

La industria de los videojuegos es uno de los sectores de la tecnología que más ha crecido en los últimos años, donde se estiman que existen más de 2300 empresas de desarrollo con una cantidad de 66000 empleados directos. Si se incluyen los empleados indirectos tales como los desarrolladores que vienen de otras marcas, el total aumenta a 220000 empleados. Con esto nos podemos dar cuenta de que es uno de los mercados más grandes en la actualidad. [6]



Figura 3.1. Grafica de valor de mercado de la industria de los videojuegos (extraído de <https://www.wepc.com/news/video-game-statistics/>)

Existen varias formas en las que los videojuegos tradicionales consiguen obtener beneficio en la actualidad y estas son:

Free to play

Son los videojuegos en los que se dan a los jugadores el acceso a una parte significativa del contenido total del juego sin la necesidad de realizar ningún tipo de pago. La forma en la que las empresas obtienen beneficio de este tipo de juegos es gracias a las facilidades que te pueden ofrecer en el progreso del juego gracias a realizar pagos. [7]

Modelo tradicional

Este modelo se basa en que, para poder disfrutar del juego, antes debes pagar, esto por lo general permite una experiencia de juego completa una vez has realizado el pago. [7]

Loot-boxes

El modelo de loot-boxes se refiere a videojuegos ya sean free to play o de modelo tradicional, los cuales ofrecen la posibilidad de abrir consumibles virtuales con la posibilidad de obtener de forma aleatoria objetos para ser utilizados en el videojuego en cuestión. [7]

Modelo de suscripción

Este modelo está basado en pagar de forma recurrente ya sea mensual o anualmente una cuota para poder tener la posibilidad de jugar al juego en cuestión. [7]

3.2. Aplicaciones de la tecnología blockchain a los videojuegos

En los últimos 2 años se ha popularizado una nueva rama de los videojuegos aparte de las anteriores mostradas, y es una en la que se han implementados tecnologías blockchain a los videojuegos y es llamada **play to earn**

3.2.1. Metaverso

El metaverso es un concepto referido a un universo en línea que combina diferentes espacios virtuales. Este tipo de mundo permitirá a los usuarios trabajar, conocer gente, socializar y jugar juntos.

Hoy en día el metaverso aun no es una realidad en su totalidad, pero muchas compañías están buscando su implementación. Los videojuegos hoy en día son lo más cercano a la existencia de un metaverso.

La finalidad del metaverso es la conexión entre el mundo físico, las finanzas y el mundo virtual. Toda esta visión se encuentra muy relacionada con la blockchain, las criptomonedas y los NFTs ya que permiten otorgar una propiedad de posesión a los activos virtuales.

3.2.2. Play to earn

Es una aproximación en la que en el mismo juego se añade alguna metodología en la que cada usuario puede monetizar sus horas de juego utilizando la tecnología blockchain y sacar un beneficio económico, este beneficio dependerá de la habilidad del usuario o de la cantidad de tiempo que le dedique. [12]

Esta estrategia de play to earn se ha implementado de diferentes maneras que se detallaran a continuación:

NFTs

A la tecnología de los NFTs explicados con anterioridad se les da una de las aplicaciones probablemente más útiles dentro del paradigma de los NFTs y es la capacidad de darle utilidad, en la que el NFT puede ser un personaje de un videojuego, una carta, una mejora, un aspecto etc. y que no simplemente sea una imagen o un video simple que a priori no poseen utilidad.

Estos NFTs en videojuegos aparte de poder utilizarse dentro de ellos, permiten en muchos casos sacarles un beneficio ya sea con bonificaciones económicas o algún tipo de ventaja con respecto al resto, esto hace que sea muy interesante y que en algunos casos se vuelvan muy cotizados debidos a su exclusividad.

Ejemplos de NFTs en videojuegos:

- **Metaverso:** Los NFTs son una parte vital del metaverso donde se utilizan para comprar las tierras, que son parcelas de espacio en el metaverso en las que los usuarios pueden construir y funcionan de manera similar a lo que sería un terreno en la vida real.
- **Axie:** En el videojuego de Axie cada uno de los personajes con los que se pueden jugar son NFTs que aparte de poder utilizarse para jugar, permiten la posibilidad de reproducirlos y tener hijos con los que poder jugar o comerciarlos para que otras personas lo utilicen.
- **Videojuegos de cartas y coleccionables:** Al igual que en la vida real existen cartas que se pueden coleccionar o jugar, han nacido videojuegos donde la tecnología blockchain permite que todas estas cartas puedan ser NFTs coleccionables, útiles y con capacidad de propiedad.

Ventajas e inconvenientes de los NFTs en videojuegos

Como hemos visto los NFTs a priori nos ofrecen la posibilidad de tener propiedad de activos digitales, esto es algo bastante muy interesante dentro de los videojuegos ya que una de las formas en la que se monetiza hoy en día los videojuegos en la venta de aspectos, personajes, habilidades etc.

Todos estos activos que normalmente se encuentran dentro de las bases de datos de los videojuegos podrían transformarse en NFTs que permitan su intercambio, propiedad y todas las ventajas que ofrecen los NFTs, incluso podría llegar el momento en el que estos NFTs se pudiesen utilizar en diferentes videojuegos al mismo tiempo.

Por lo general los NFTs a simple vista serían muy beneficiosos dentro de los videojuegos, el problema surge cuando un videojuego depende en su totalidad de los NFTs. Esto hace que el videojuego pase a un segundo plano y entre en acción la especulación que es uno de los grandes problemas que tienen los NFTs hoy en día. En ese caso la especulación haría que muchos de estos activos se extinguieran y los usuarios que los posean los venderían por cantidades desorbitadas haciendo que muchos usuarios no pudiesen acceder a determinadas partes del videojuego en cuestión.

Tokens/Criptomonedas ERC-20

En los videojuegos tradicionales normalmente te ofrecen recompensas en puntos para poder progresar mientras juegas a ellos, pero esos puntos normalmente son solo útiles en el videojuego en cuestión, al ver esto se planteó la idea de implementar tokens blockchain a los videojuegos, estos al igual que los tradicionales permiten avanzar en el juego, pero a su vez dan la posibilidad de poder intercambiarlos por dinero real.

Estos tokens son propios de cada videojuego, existen su supply (cantidad de tokens disponibles) puede ser limitado o infinito, y mediante su intercambio permiten a los usuarios y a la desarrolladora del videojuego conseguir un beneficio de manera muy sencilla.

Existen varios tipos de tokens y tecnologías utilizadas en los videojuegos blockchain:

- **Oráculos:** Es una tecnología que permite que la cantidad de recompensas sea fija, esto quiere decir que, si el token original sube demasiado de precio, no haga que las personas no puedan entrar al videojuego debido a su alto precio y que las personas que ya estén jugando ganen cantidades desmesuradas de dinero, por lo que el oráculo regula estas recompensas de manera que nunca se disparen los precios, uno de los oráculos más utilizados es chainlink.
- **Tokens de economía:** son tokens directamente enfocados a la monetización de los usuarios dentro del videojuego y que estos mismos puedan sacar beneficios. Estos tokens suelen ser generados jugando al videojuego en cuestión y suelen tener una cantidad de tokens infinita.
- **Tokens de gobernanza:** Este tipo de tokens representan al igual que harían las acciones dentro de una empresa, la capacidad de participar en decisiones dentro del videojuego ya sea la inclusión de nuevas funcionalidades, cambios en las funcionalidades ya existentes etc. Este tipo de tokens no son únicos de los videojuegos blockchain sino de una gran cantidad de proyectos donde se busca la descentralización y que los usuarios que poseen estos tokens decidan o formen parte de todas las decisiones dentro de la empresa.

Tipos de implementaciones de tokens en videojuegos:

- **Token único:** Este tipo de videojuegos suelen poseer el token de economía y de gobernanza dentro del mismo token, esto hace que la barrera de entrada (dinero necesario para poder acceder al juego) y las recompensas sean mayores.
- **Token de gobernanza y token de economía:** Separan la economía del juego en dos tokens diferentes, esto suele ser bastante beneficioso ya que permite que no se descapitalice el juego con tanta facilidad.
- **Token único con oráculo:** Este tipo de videojuegos se basan en la capacidad de los oráculos de poder utilizar elementos externos a la blockchain para poder calcular de forma absoluta el valor de las recompensas, por ejemplo si el token tiene un valor de 1\$ y las recompensa que se dan son 20\$, el videojuego te otorgará 20 tokens, pero si ese token tiene un valor de 10\$, el videojuego únicamente te otorgará 2 tokens, con esto se elimina la posibilidad en la que un token sube tanto su precio que los usuarios no se pueden permitir acceder al videojuego.

Ventajas e inconvenientes de las criptomonedas en videojuegos

Como se ha comentado, la mayor ventaja que tienen la implementación de una criptomoneda en la economía de un juego es la posibilidad de obtener un beneficio económico de manera sencilla.

Pero, por el contrario, ya sea por la mentalidad de las personas o porque aún no se ha encontrado una solución eficiente, estos juegos dejan de limitarse a ser jugados, sino que se transforman en una especie de trabajo. Todo esto hace que el comportamiento de una economía dentro de estos juegos se asemeje a la que podría ser la economía de un país, donde aparecen crisis, momentos de expansión económica etc.

Al asemejarse a una economía real, esto hace que el mismo videojuego dependa en su totalidad de esta economía y en el momento en el que esta economía colapsa el juego cae con ella.

Esto es debido a que para acceder a estos juegos normalmente hay que realizar una inversión de capital, y las mismas personas que lo juegan por el hecho de haberse introducido al videojuego de esta manera, lo dejan de tomar como un divertimento y se transforma en un trabajo.

En el momento en el que la economía de un videojuego de este tipo quiebra, todas las personas que estuviesen jugando al videojuego, perderán el dinero de su inversión y con esto la popularidad del videojuego, aparte de causar muchos problemas con los inversores.

Esto lo vamos a comprobar en las siguientes graficas referidas a cada uno de los tipos de economías que hemos descrito con anterioridad.

Economía con Oráculo:



Figura 3.2. Grafica de valor de la criptomoneda ETERNAL (extraído de <https://poocoin.app/tokens/0xd44fd09d74cd13838f137b590497595d6b3feea4>)

En esta gráfica se pueden distinguir tres zonas bien diferenciadas:

1. **Fase de expansión** en la que muchos usuarios empiezan a ingresar al videojuego esto produce que la criptomoneda aumente su valor ya que todo el dinero de los usuarios que ingresan se mantiene dentro del contrato de la criptomoneda. Gracias a esta subida constante, los usuarios tienen confianza en el videojuego y no venden sus criptomonedas y las reinvierten en el juego para ya sea obtener más recompensas o simplemente mejorar.
2. **Fase de estancamiento** en la que la cantidad de usuarios que ingresan al videojuego empieza a no ser suficiente como para mantener las recompensas que se están otorgando a diario a los usuarios que se encuentran jugando. En esta etapa empieza a generarse la incertidumbre y muchos usuarios comienzan a vender sus criptomonedas.
3. **Fase de caída o desplome** en la que los usuarios pierden la confianza por el proyecto y con ello se disminuye la cantidad de usuarios que ingresan y los que ya están dentro jugando quieren dejar de jugar, por lo que se realiza una venta masiva de criptomonedas y con ello un desplome en el precio.

En el caso que se muestra en la **figura 3.2** es un videojuego que poseía un oráculo esto hizo que la subida y la bajada fuesen mucho más abruptas. Esto es debido a que las recompensas son siempre del mismo valor y la barrera de entrada es siempre la misma.

Por ejemplo, imaginemos el siguiente escenario:

Una criptomoneda vale 1\$ y las recompensas que se otorgan son 5\$, es decir por cada recompensa otorgaremos 5 criptomonedas que equivaldrían a 5\$.

Si para ingresar necesitamos 20\$ habría que pagar 20 criptomonedas, esto en una gráfica ascendente donde el precio nunca para de subir es genial ya que la cantidad de criptomonedas que entran son menores de las que salen ya que si un usuario entró

con 20 criptomonedas cuando valían 1\$ y la moneda ha subido a 2\$ ahora las recompensas serán solo de 2.5 criptomonedas para pagar las recompensas.

En un mundo idílico donde esta subida es infinita, esta podría ser la estrategia del éxito, pero en el momento en el que la gráfica da la vuelta y empieza a bajar ocurre totalmente lo contrario y el contrato que contenía las criptomonedas de gente que pagó 1 criptomoneda a valor de 20\$ para entrar, ahora tiene que dar 5 criptomonedas a valor de 1\$ para pagar las recompensas, debido a esto el contrato se queda sin liquidez y la economía quiebra.

Economía de token único:



Figura 3.3. Gráfica de valor de la criptomoneda CPAN (extraído de <https://poocoin.app/tokens/0x04260673729c5f2b9894a467736f3d85f8d34fc8>)

En la **figura 3.3** se muestra la gráfica del videojuego **cryptoplanes** que tenía una economía de token único en el que se pueden diferenciar las mismas fases que en el anterior pero como podemos ver se encuentran mucho más suavizadas esto es debido a que no se tiene el efecto del oráculo.

En este caso las recompensas siempre son una misma cantidad de criptomonedas, si la recompensa es 10 criptomonedas y la moneda vale 10 céntimos, pues la recompensa total será un euro, si la moneda vale 10 euros pues la recompensa serán 100 euros, lo mismo para la barrera de entrada, si comprarte un NFT son 100 criptomonedas a 10 céntimos pues costará 10 euros, y si la moneda vale 10 euros pues costarán 1000 euros.

Al igual que con el oráculo en el momento en el que los usuarios pierden la confianza la economía empieza a morir, en este caso es algo mucho más controlado, pero se puede apreciar como poco a poco el valor va disminuyendo y con esto la muerte de la economía y del videojuego.

Economía de token de gobernanza y token de recompensa



Figura 3.4. Grafica de valor de la criptomoneda AXS (extraído de <https://coinmarketcap.com/currencies/axie-infinity/>)



Figura 3.5. Grafica de valor de la criptomoneda SLP (extraído de <https://coinmarketcap.com/currencies/smooth-love-potion/>)

En la **figura 3.4** se muestra la gráfica del precio del token AXS, el token de gobernanza del videojuego Axie, y en la **figura 3.5** se muestra la del SLP, el token de recompensa. En este caso, aunque se puede ver una subida y una bajada bastante importante en ambos tokens, la gran diferencia es que en ningún momento llegan a bajar a 0 al igual que los anteriores, ya que el hecho de poseer dos tokens hace que la economía sea mucho más robusta y en épocas de descenso no colapse.

Aparte del hecho de poseer 2 tokens existe una diferencia primordial con los anteriores, y es la capacidad de quema de tokens mediante algunas mecánicas dentro del juego, en los anteriores el único uso que tenía el token era o para sacar beneficio o para mejorar alguna cosa, pero sin ningún cambio significativo en la experiencia de

juego. En cambio, en Axie la reinversión de las recompensas que se consiguen es una parte primordial para poder realmente mejorar en el juego.

Gracias a esto se incentiva la reinversión y no únicamente la obtención de beneficio, esto hace que la economía sea más robusta y no caiga tan abruptamente como en los ejemplos anteriores.

Junto a esto, el hecho de tener un token de gobernanza incentiva aún más la confianza de las personas ya que mediante la obtención de este token los mismos usuarios pueden decidir sobre el futuro del videojuego.

Conclusión

Tras haber analizado las 3 economías podemos ver que es complicado crear un sistema que sea robusto en su totalidad y no se vea afectado por una inflación excesiva.

Las principales herramientas necesarias para poder tener una economía robusta serían:

1. Mecanismos de quema de tokens que mantengan la economía.
2. Mecanismos de reinversión que incentiven al usuario a utilizar sus recompensas para mejorar en el juego.
3. Transparencia y lógica, donde cualquier movimiento que se realice por parte del videojuego sea contrastado con sus usuarios para evitar la incertidumbre.

Como conclusión podemos sacar que la introducción de una economía de criptomonedas dentro de un videojuego puede ser perjudicial ya que puede matar al juego completo, por lo que realmente lo que debería buscarse es que esta economía sea únicamente un complemento y no el videojuego en sí y que el juego se venda como un divertimento y no como un trabajo. Este es el problema de muchos de los videojuegos de criptomonedas, el hecho de publicitarse como una manera de hacer dinero y no como un método de diversión, esto atrae los problemas de la especulación y con ello los problemas económicos que se pueden ver en el mundo real.

4

Prototipo y especificaciones

El prototipo se ha realizado con la finalidad de poner en práctica estas tecnologías blockchain dentro de un videojuego y ver como funcionarían y de qué forma serían compatibles.

4.1. Ejemplos videojuegos play to earn

4.1.1. Click to Earn

En los dos últimos años se han puesto de moda y se han creado una gran cantidad de juegos *click to earn* que son juegos muy simples que mediante el uso de NFTs y economía, permiten mediante probabilidades la obtención de recompensas.

Estos juegos son muy sencillos y no requieren ningún tipo de habilidad, simplemente hay que realizar clicks y ver si se tiene suerte para ganar las recompensas. Ejemplos muy conocidos de este tipo de juegos han sido juegos como **CryptoPlanes** o **CryptoMines** ambos muy similares donde en uno tenías NFTs de aviones y otro de mineros y con ellos realizabas carreras o minabas planetas respectivamente y tenías la posibilidad de obtener recompensas en forma de criptomonedas que podías o utilizar en el juego o venderlas por dinero real.

En este tipo de videojuegos, el beneficio de los creadores del videojuego es la posesión de cantidades de tokens nativos del juego que se iban desbloqueando a lo largo del tiempo a los integrantes del grupo y venderlos para sacar el beneficio. De este tipo de videojuegos han existido una gran cantidad de clónicos con más o menos éxito.

4.1.2. Axie Infinity

Es un juego online basado en NFTs con una economía basada en Ethereum que fue creado en el año 2018 y se puede jugar en todas las plataformas.

Los jugadores en este juego pueden coleccionar y conseguir NFTs similares a la forma de un ajolote. Con estas criaturas se puede combatir contra los de otros usuarios, puedes reproducirlos para obtener más etc. La manera en la que generan beneficio en el juego es mediante una cuota de 4.25% cada vez que se realiza una compra o una venta de estos personajes.

Este juego fue el primero en popularizarse de forma masiva e incluso se generó un negocio en el que se contrataban a personas para que jugaran para ganar recompensas, a esto se le llamó el *colonialismo digital*.

4.2. Motivación y descripción del prototipo

La motivación de la realización del proyecto es poner en práctica varias de las mecánicas de los juegos y comprobar que se pueda crear un videojuego útil, para ello se va a crear un juego del estilo click to earn y se le van a añadir mecánicas de Axie como la reproducción de NFTs.

El prototipo se llamará CryptoChickens y estará basado en el uso de NFTs con forma de pollos que mediante un ADN generado a partir del nombre que se les otorga tendrán colores diferentes, se podrán reproducir los pollos entre ellos para obtener huevos de los que nacerán hijos con el ADN de los padres combinados, se podrá combatir con los pollos para obtener recompensas, habrá un sistema de niveles con algunos beneficios para pollos con más nivel y un historial de las victorias y derrotas de cada pollo.

4.3. Tecnologías empleadas en el prototipo

Para el desarrollo del videojuego se han seleccionado las siguientes tecnologías.

4.3.1. Lenguajes de programación

- **HTML:** HyperText Markup Language, es un lenguaje de enmarcado para mostrar documentos diseñados para ser mostrados en los navegadores web. Mediante etiquetas permite elegir la distribución en la que se muestran todos los elementos que forman parte de una web. [18]
- **CSS:** Cascading Style Sheets, es un lenguaje utilizado para describir la presentación de un documento escrito con el lenguaje HTML o XML. Permite cambiar la forma y estilo en la que se muestra cualquier elemento dentro de una web. [17]
- **JavaScript:** Es un lenguaje de programación utilizado en el desarrollo web, que permite crear contenido web dinámico e interactivo. [27]
- **Solidity:** Es el lenguaje creado por el equipo de Ethereum, con una sintaxis parecida a la de javascript y que permite la creación de contratos inteligentes en blockchains como Ethereum, Polygon y sus derivados. [14] [15]

4.3.2. Frameworks, entorno de desarrollo y tecnologías utilizadas

- **Remix:** Es un entorno de desarrollo de código abierto de Ethereum que permite escribir, compilar y hacer debug de contratos inteligentes y códigos de solidity. [21]
- **Truffle:** Es un entorno de desarrollo que permite el desarrollo, la compilación, el testing y el despliegue de contratos inteligentes a cualquier blockchain. [19]
- **Ganache:** Es una blockchain personal diseñada para Ethereum para la creación de aplicaciones, esta red local pertenece al ecosistema de truffle y puedes desplegar contratos en ella utilizando truffle. [20]
- **React:** Es una librería de código abierto utilizada para el desarrollo de front-end de javascript y creación de interfaces de usuario. La ventaja que ofrece react es la creación de componentes de código reutilizable evitando la repetición de código. Aparte de poseer la sintaxis JSX que permite la inclusión de un lenguaje similar a HTML dentro de javascript. [22]
- **Moralis:** Es una librería que facilita conectarse, obtener datos e interactuar con tokens y NFTs dentro de las blockchains. [23]
- **Metamask:** Es un software que implementa una cartera de criptomonedas y que permite interactuar con la blockchain de Ethereum. Permite interactuar con aplicaciones descentralizadas mediante su extensión en navegadores o mediante su aplicación móvil. [24]
- **Visual Studio Code:** Es el editor de código en el que se ha programado la aplicación completa, tanto los contratos inteligentes utilizando truffle como la interfaz de usuario con react. Visual studio permite la instalación de plugins que facilitan el desarrollo de diferentes maneras.

5

Diseño de la prueba de concepto

En este capítulo se analizarán la metodología utilizada para el desarrollo, los actores y todos los requisitos y funcionalidades necesarias para el prototipo de la aplicación, estos requisitos se dividen en funcionales, referidos a las diferentes funcionalidades que debe presentar la aplicación, y los no funcionales, referidos a las características del funcionamiento de la aplicación. Aparte se desarrollarán los casos de uso más importantes dentro de la aplicación.

5.1. Metodología de espiral.

Para la realización del trabajo se ha seleccionado el modelo en espiral. Este modelo se basa en la realización de iteraciones en las que antes de cada una de ellas se definen unos objetivos a cumplir y una vez terminada esa iteración se repite de nuevo añadiendo la siguiente etapa planteada, esto se repite continuamente hasta que se completa el desarrollo completo.

Esta metodología es de bastante utilidad ya que el hecho de necesitar unos contratos inteligentes bien definidos es algo esencial, por lo que, al utilizar el desarrollo en espiral, estos serán revisados en cada iteración por lo tanto habrá menos posibilidades de que en la versión final se encuentren fallos inesperados. Esto es vital ya que una vez desplegado un contrato este no puede volver a ser modificado.

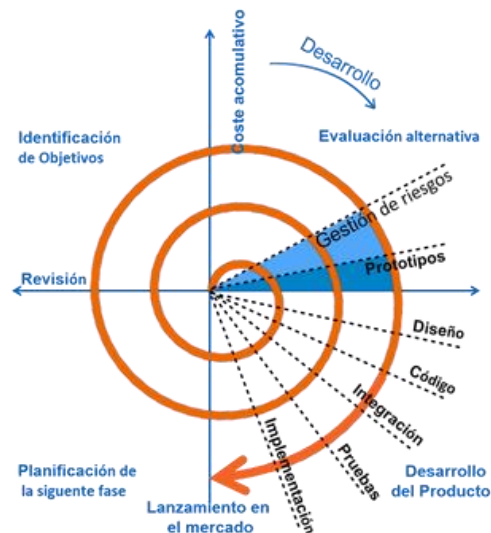


Figura 5.1. Fases del desarrollo en espiral (extraída de <https://prezi.com/k5slpplsud6/modelo-en-espiral-desarrollo-de-software/>)

En la **figura 5.1** se muestra de forma resumida la metodología empleada, la cual nos permite reducir los riesgos de fallos de las primeras iteraciones, en este caso centrado en el desarrollo del contrato inteligente, donde en cada iteración se añadirán nuevas funcionalidades.

5.2. Fases del proyecto.

- ❖ 1ª Iteración. Investigación de las tecnologías blockchain. Análisis de las tecnologías utilizadas en videojuegos descentralizados.
 - Investigación sobre Bitcoin y Ethereum: 25 horas.
 - Investigación videojuegos DeFi: 15 horas.
 - Investigación tecnologías para proyectos descentralizados: 20 horas.
- ❖ 2ª Iteración. Análisis de requisitos, diseño e implementación de las funcionalidades de creación de chickens.
 - Análisis de requisitos: 8 horas.
 - Diseño de contrato inteligente con funcionalidades de creación y edición de chickens: 8 horas.
 - Implementación de las funcionalidades: 10 horas.
 - Testing de las funcionalidades: 5 horas.
- ❖ 3ª Iteración. Análisis de requisitos, diseño e implementación de combates de chickens y reproducción de chickens.
 - Análisis de requisitos: 5 horas.
 - Diseño funcionalidad Breeding: 10 horas.
 - Diseño funcionalidad Battle: 10 horas.
 - Implementación de funcionalidades: 20 horas.
 - Testing de las funcionalidades: 15 horas.

- ❖ 4ª Iteración. Investigación sobre las tecnologías utilizadas en el desarrollo de la interfaz de usuario. Análisis de requisitos, diseño e implementación de transferencia, sistema de niveles y manejo de fondos.
 - Investigación sobre React y Moralis: 25 horas.
 - Análisis de requisitos: 5 horas
 - Diseño funcionalidad de transferencia: 7.5 horas.
 - Diseño de la funcionalidad de sistema de niveles: 7.5 horas.
 - Diseño de la lógica de manejo de fondos: 5 horas
 - Implementación de las funcionalidades 15 horas.
 - Testing de las funcionalidades: 5 horas.
- ❖ 5ª Iteración. Diseño del videojuego web descentralizado. Desarrollo de la interfaz de usuario e implementación de la conexión con el contrato.
 - Diseño interfaz de usuario: 5 horas.
 - Desarrollo interfaz de usuario: 20 horas.
 - Conexión interfaz – contrato: 20 horas.
- ❖ 6ª Iteración. Memoria del proyecto.
 - Crear documentación del TFG: 30 horas.

5.3 Requisitos funcionales

- **R.F.1 Conexión con wallet web3:** El sistema debe permitir la conexión con cualquier wallet a la aplicación y que esta detecte si se posee crédito.
- **R.F.2 CRU de NFTs:** El sistema debe permitir crear NFTs, leerlos una vez creados y poder modificar algunos de sus atributos.
- **R.F.3 Administración de fondos:** El sistema debe regular los fondos, almacenándolos en caso de compra de NFTs y distribuyéndolos en caso de que el usuario los obtenga y permitiendo su retiro.
- **R.F.4 Funcionalidad NFT:** El sistema debe permitir que una vez se obtengan los NFTs estos puedan utilizarse en diferentes actividades, tales como reproducción de sus NFTs o utilizarlos en batalla para obtener recompensas.

5.4 Requisitos no funcionales

- **R.N.F.1 Internet:** Para el acceso a la aplicación es necesaria una conexión a internet
- **R.N.F.2 Wallet:** Para la utilización de la aplicación es necesario tener instalada alguna wallet de web3.
- **R.N.F.3 Seguridad:** El sistema no debe permitir el acceso de agentes externos a los datos almacenados y puedan modificarlos.
- **R.N.F.4 Transparencia:** La aplicación debe ser desplegada en la blockchain, teniendo código abierto y cualquier persona poder acceder al código.

- **R.N.F.5 Blockchain:** La blockchain utilizada será Ethereum o cualquiera de sus derivadas, Polygon, BSC etc.
- **R.N.F.6 Interfaz de usuario:** La interfaz implementada, será simple y fácil de utilizar.

5.5. Diseño del contrato

En el desarrollo de una aplicación de web3 a diferencia de en el desarrollo tradicional, no hace falta tener una base de datos centralizada en la que almacenar los datos, sino que directamente estos datos se almacenan en la blockchain, a continuación, se muestra un diagrama de lo que sería la “base de datos” del proyecto:

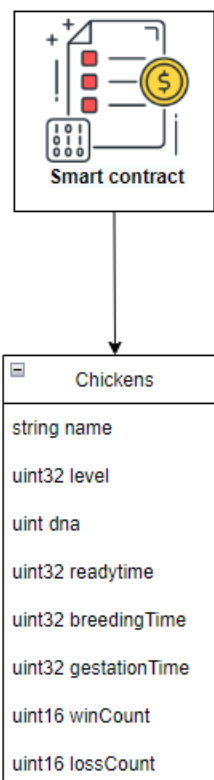


Figura 5.2. Diseño del almacenamiento del contrato

En la **figura 5.2**, se puede apreciar que únicamente es una tabla, que en solidity se representa con un array de tipo Chickens, Chickens es un struct con todos los campos mostrados en la imagen, utilizando esta información el contrato utiliza las diferentes funciones implementadas para el funcionamiento del proyecto.

5.6. Casos de uso

Los casos de uso son una forma sencilla de describir el comportamiento del sistema y de todas las maneras en las que los actores podrían trabajar con el proyecto. En este caso vamos a utilizar de actores al usuario con wallet sin conectar, usuario con wallet conectada, usuario que es dueño de un chicken y usuario dueño de dos chickens.

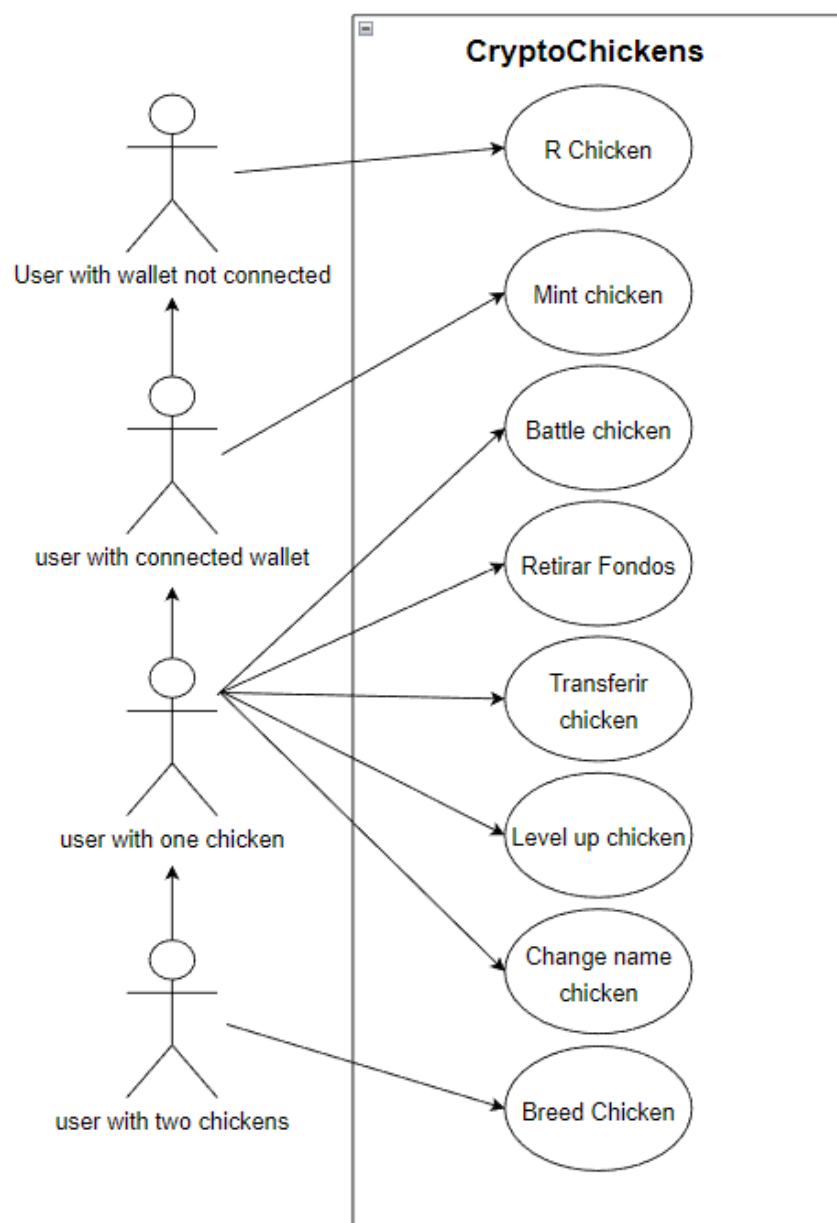


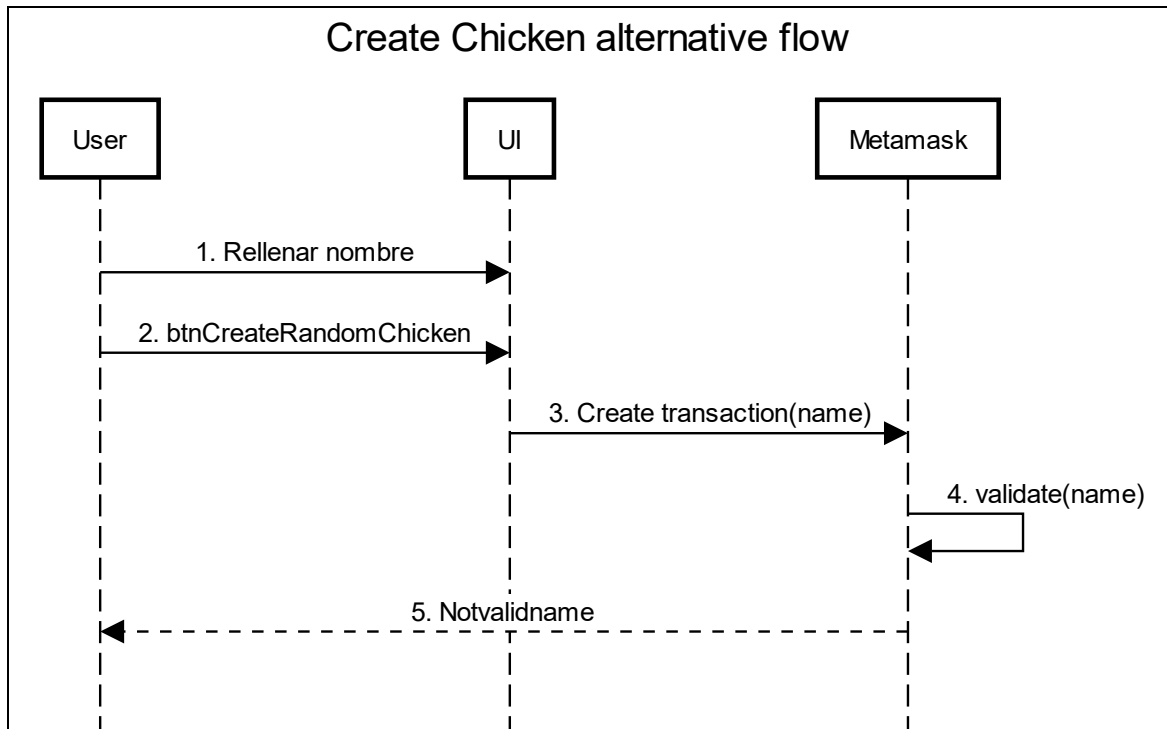
Figura 5.3. Diseño de casos de uso

Aquí se pueden observar las diferentes acciones que pueden realizar cada uno de los usuarios dependiendo de la cantidad de chickens o de si están logueados o no con su wallet.

A continuación, se describirán cada uno de los diferentes casos de uso mediante sus diagramas de secuencia.

Titulo	CU1- C Chickens R.F.2
Descripción	Creación de un chicken.

Precondición	El usuario debe encontrarse en la página de inicio donde se crean los NFTs, con su wallet conectada
Postcondición	Se crea el chicken y se le otorga propiedad al usuario
Escenario principal	
1.El usuario escribe un nombre para su chicken. 2.El usuario clic en el botón “créate random chicken”. 3.La UI crea una llamada al contrato con los datos introducidos. 4.El usuario acepta en su wallet la transacción. 5.El contrato acepta la llamada y crea el chicken y le otorga propiedad al usuario. 6.Se muestra el chicken en la UI.	
Escenario alternativo	
4.A.El usuario cancela la transacción. 5.A Salta una notificación con el error. 4.B El usuario no introduce el nombre. 5.B Salta una notificación con el error.	
Diagrama de secuencia normal	
<p style="text-align: center;">Create Chicken flow</p> <pre> sequenceDiagram participant User participant UI participant Metamask participant Ethereum Network participant Contract participant contract User->>UI: 1. Rellenar nombre User->>UI: 2. btnCreateRandomChicken UI->>Metamask: 3. Create transaction(name) activate Metamask Metamask->>Metamask: 4. validate(name) deactivate Metamask Metamask-->>User: 5. Pop-Up confirm transaction User->>Metamask: 6. Confirm Transaction activate Metamask Metamask->>Ethereum Network: 7. sendTransaction(name) deactivate Metamask activate Ethereum Network Ethereum Network->>Ethereum Network: 8. ValidateTransaction(name) deactivate Ethereum Network Ethereum Network->>Contract: 9. executeTransaction(name) activate Contract Contract->>contract: 10. createChicken(name) activate contract deactivate contract Contract-->>Ethereum Network: 11. confirmTransaction deactivate Contract deactivate Ethereum Network Ethereum Network->>Metamask: 13. Transaction confirmed deactivate Ethereum Network activate Metamask deactivate Metamask Metamask-->>UI: 14. Transaction confirmed deactivate Metamask activate UI UI->>UI: 15. Show chicken deactivate UI UI->>User: 16. Show chicken deactivate UI </pre>	
Diagrama de secuencia alternativo	



Titulo	CU2- Realizar una batalla R.F.4
Descripción	El chicken en propiedad se utiliza para pelear por una recompensa
Precondición	El usuario debe encontrarse en la página de battle y con un chicken en su propiedad
Postcondición	Dependiendo de si gana o pierde se le otorgaran o no las recompensas y su chicken quedará en cooldown
Escenario principal	
1.El usuario selecciona uno de sus chickens. 2.El usuario selecciona uno de los enemigos al que atacar. 3.El usuario clicla en el botón "Fight". 4.La UI crea una llamada al contrato con los datos introducidos. 5.El usuario acepta en su wallet la transacción. 6.El contrato acepta la llamada y ejecuta la batalla con una victoria. 7.Se le añade la recompensa al usuario. 8.Se muestra en la UI.	
Escenario alternativo	
6.A. El contrato acepta la llamada y ejecuta la batalla con una derrota. 7.A. No se le añade la recompensa.	

- 8.A. Se muestra en la UI.
- 5.B el usuario cancela la transacción.
- 6.B. Se muestra un error de transacción en pantalla.
- 5.C. el usuario no posee suficiente saldo.
- 6.C. se muestra un error de la transacción en pantalla.

Diagrama de secuencia normal

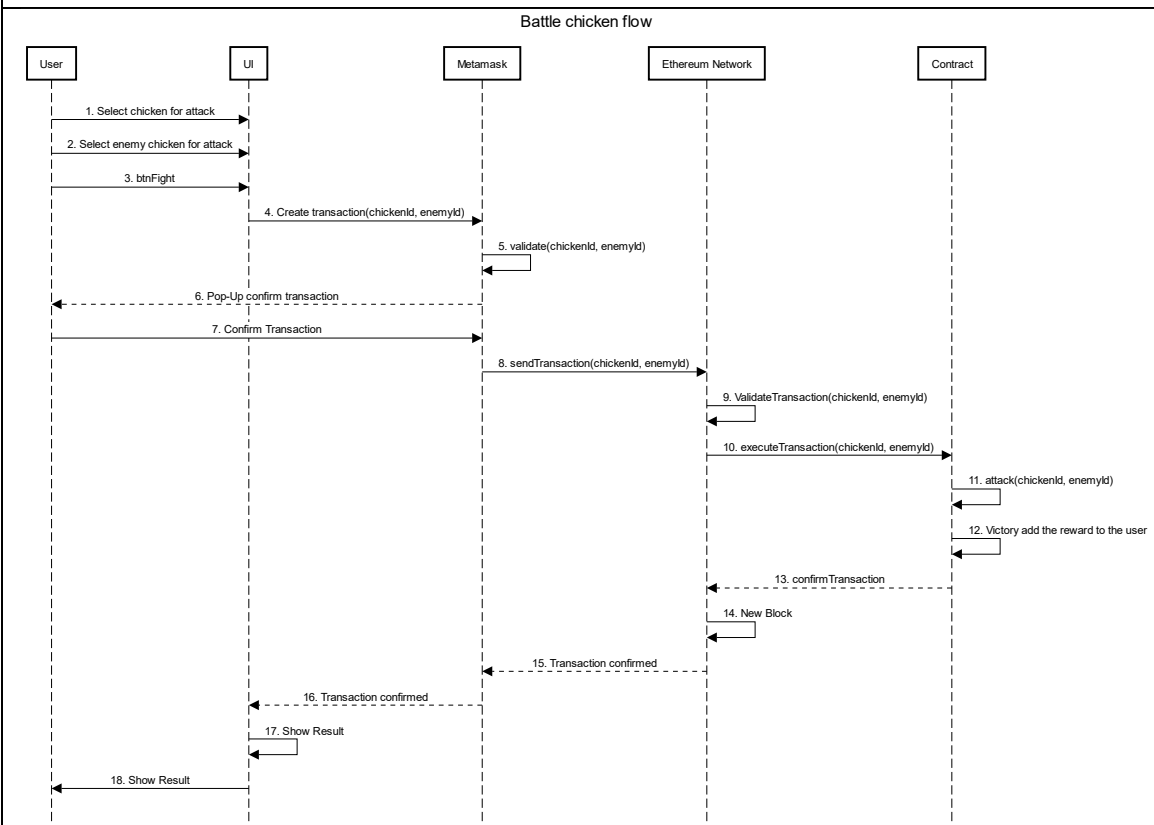
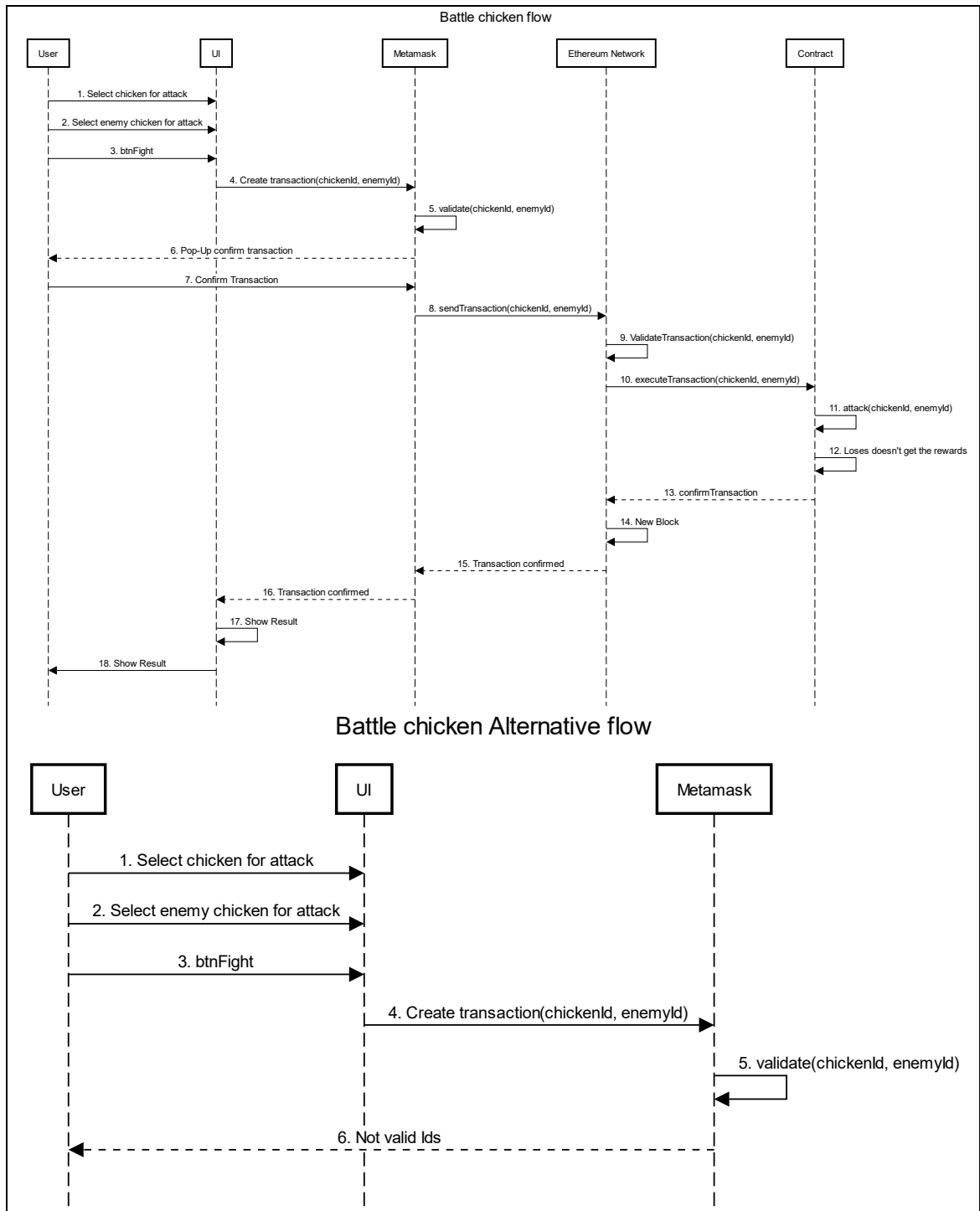


Diagrama de secuencia alternativo



Titulo	CU3- Transfer chicken R.F.4
Descripción	Un usuario regala un chicken a otro usuario
Precondición	El usuario debe encontrarse en la página de se chicken e introduce la dirección del usuario a la que quiere transferírselo
Postcondición	El usuario destinatario recibe la propiedad del chicken

Escenario principal

- 1.El usuario escribe una dirección a la que enviar la propiedad de su chicken.
- 2.El usuario clic en el botón “transfer”.
- 3.La UI crea una llamada al contrato con los datos introducidos.
- 4.El usuario acepta en su wallet la transacción.
- 5.El contrato acepta la llamada y se cambia la propiedad del chicken.
- 6.Se muestra el chicken en la UI.

Escenario alternativo

- 4.B el usuario cancela la transacción.
- 5.B. Se muestra un error de transacción en pantalla.
- 4.C Los datos introducidos no son válidos.
- 5.C se muestra un error por pantalla.

Diagrama de secuencia normal

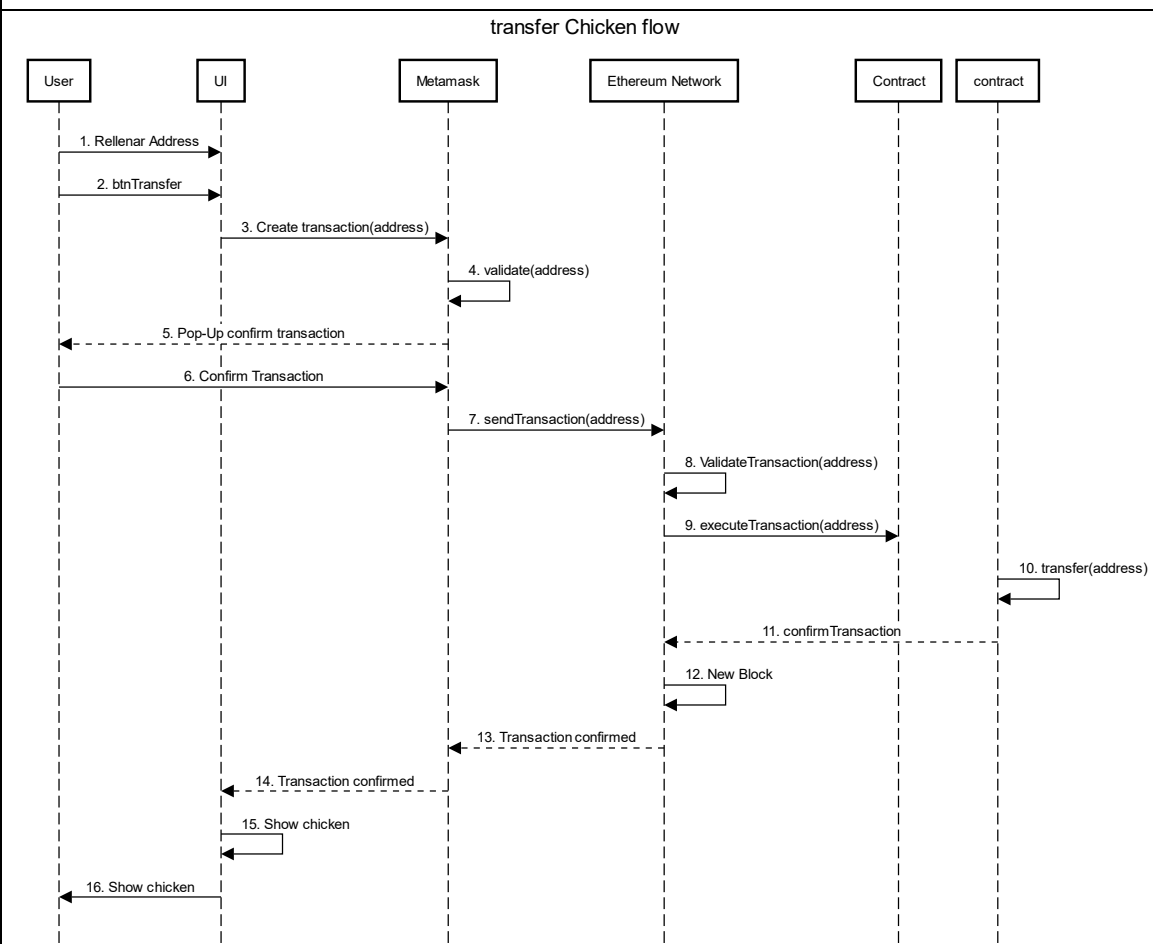
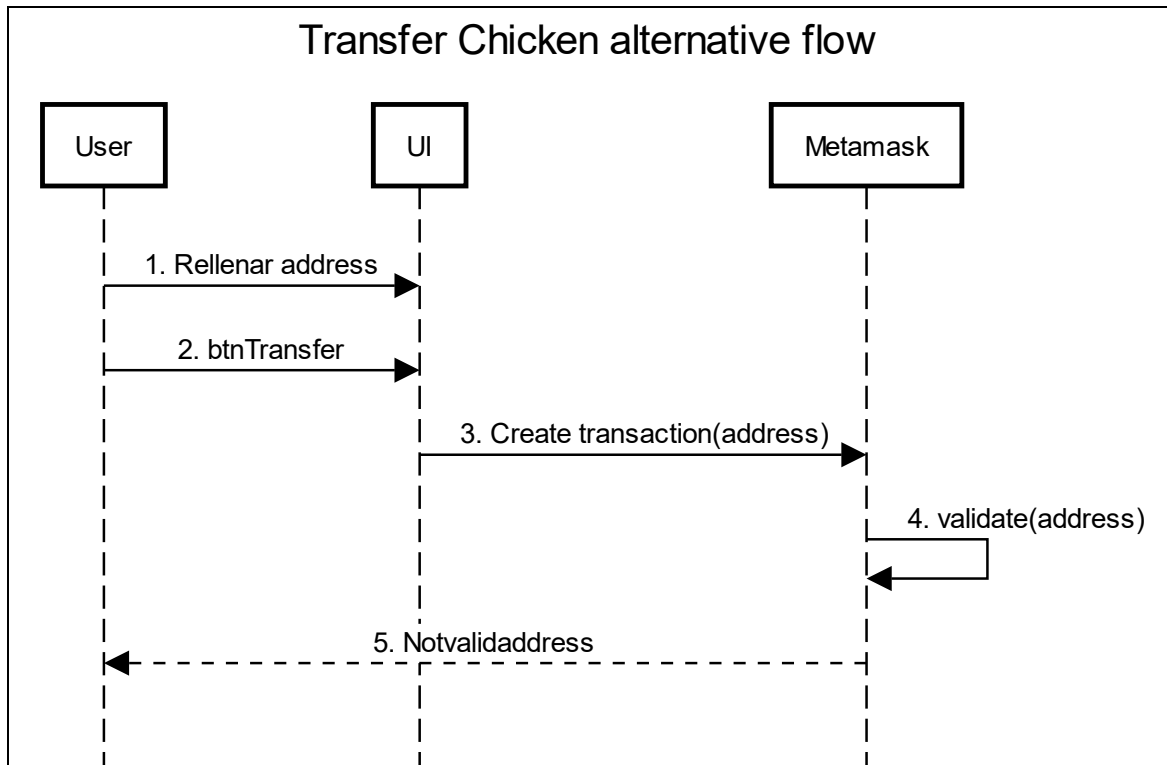


Diagrama de secuencia alternativo



Titulo	CU4- Breeding chicken R.F.4
Descripción	Un usuario reproduce dos chickens y obtiene un huevo
Precondición	El usuario debe encontrarse en la página de breeding y estar en posesión de al menos dos chickens
Postcondición	El usuario obtiene un huevo y sus chickens quedan en cooldown por 3 días.
Escenario principal	
1.El usuario selecciona uno de sus chickens. 2.El usuario selecciona a su segundo chicken. 3.El usuario clic en el botón “Breed”. 4.La UI crea una llamada al contrato con los datos introducidos. 5.El usuario acepta en su wallet la transacción. 6.El contrato acepta la llamada. 7.La transacción se ejecuta. 8.El usuario obtiene un huevo. 9.Se muestra en la UI.	
Escenario alternativo	
5.B el usuario cancela la transacción. 6.B. Se muestra un error de transacción en pantalla.	
Diagrama de secuencia normal	

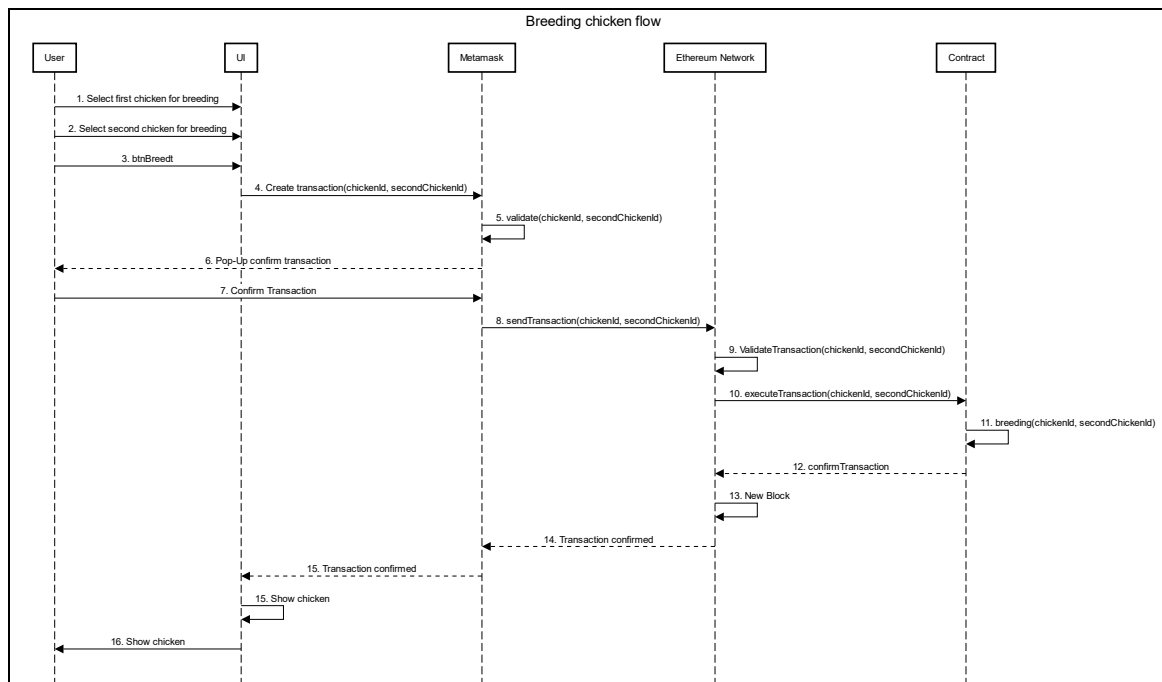
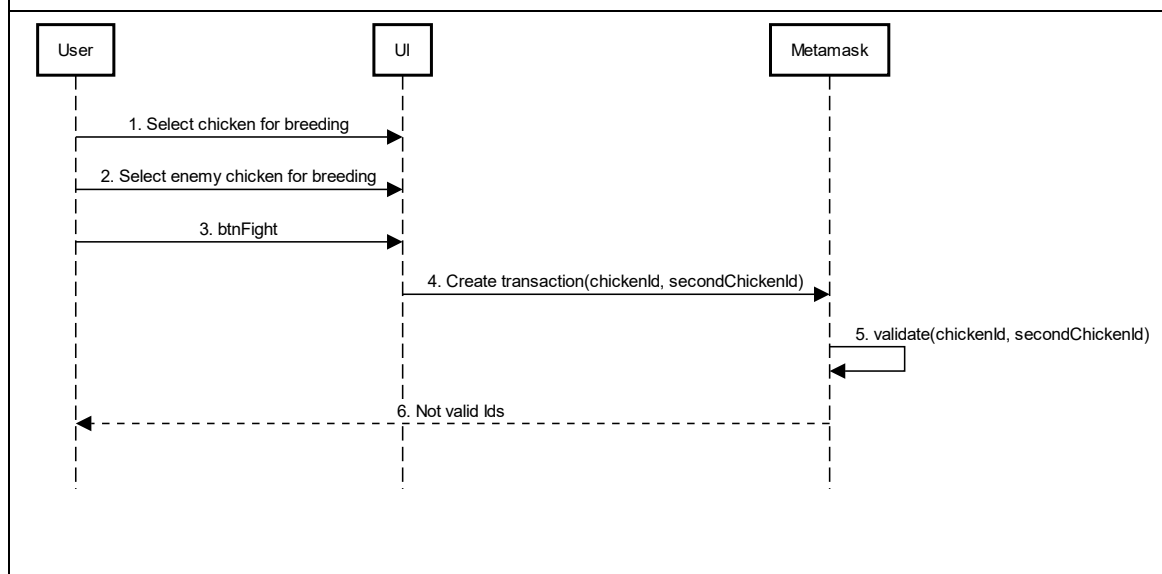


Diagrama de secuencia alternativo



5.7 Funcionamiento conexión con contratos

En un desarrollo web tradicional el frontend y el backend se encuentran conectados mediante peticiones http, en el caso de web3 es similar, pero en vez de conectarte a un endpoint propio, se utiliza la wallet de metamask de intermediario. Metamask se encuentra conectada al endpoint de una blockchain y utiliza peticiones JSON-RPC para hacer llamadas a los contratos de esa blockchain y mediante esto obtener y enviar información de ellos.

6

Implementación

El desarrollo se ha dividido en diferentes etapas, primero se ha comenzado con los contratos, han sido testeados para confirmar su buen funcionamiento y por último se ha creado la interfaz.

6.1. Contrato inteligente

Para comenzar con el desarrollo nos hemos basado en el proyecto de **CryptoZombies** [16] el cual nos ha ayudado a entender el funcionamiento de los contratos en videojuegos DeFi. Tras esto se ha comenzado el proyecto con la herramienta web de remix, la cual permite desarrollar, desplegar y probar de forma simple los contratos, en un inicio ha sido muy útil para la familiarización con la tecnología y poder desarrollar las primeras funcionalidades.

El contrato está dividido en diferentes ficheros para una legibilidad mayor, estos van heredando unos de otros y al final el único que se despliega es el contrato de `cryptochickens`, el cual hereda de todos los anteriores.

La estructura es la siguiente:

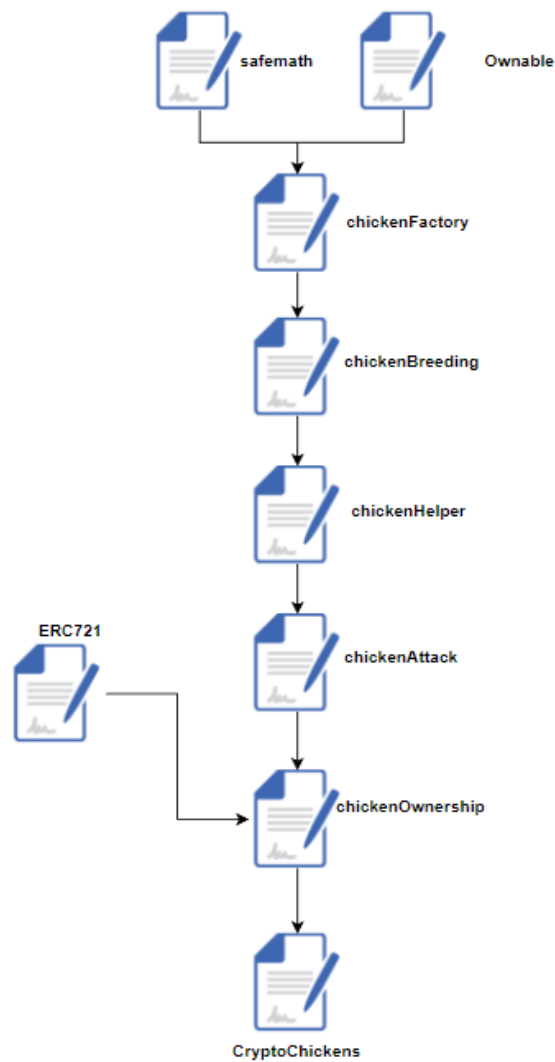


Figura 6.1. Estructura de contratos

Como se puede distinguir en la **Figura 6.1** existen varios contratos que poseen interfaces de implementación tales como **ERC721** con su estándar de NFTs que posee las funciones necesarias para la creación de NFTs y **ownable** el cual posee funciones para poder verificar la seguridad de algunas funciones de los contratos. Por otro lado, está el contrato **safemath** el cual permite que no se produzca overflow en las variables de tipo entero, esto quiere decir que cuando una variable alcanza su límite esta vuelve al valor 0 por ejemplo uint8 con un valor de 255 al sumarle 1 esta volvería a 0 y gracias a este contrato nos ayuda a verificar esto. Este problema fue solventado en la versión 0.8.0 de solidity.

6.1.1. Variables Cryptochickens

En nuestro contrato como se comentó con anterioridad se utiliza un struct de **chicken** donde se contiene toda la información de cada uno de los chickens creados:

Tipo de dato	Nombre	Descripción
string	name	Nombre del chicken.
uint	dna	ADN del chicken que determina el tipo de chicken.
uint32	level	Nivel del chicken.
uint32	readyTime	Tiempo de enfriamiento en cada ataque.
uint32	breedingTime	Tiempo de enfriamiento después de haberse realizado un breeding.
uint32	gestationTime	Tiempo en el que un huevo tarda en abrirse.
uint16	winCount	Número de victorias de un chicken.
uint16	lossCount	Número de derrotas de un chicken.

Tabla 6.1. Estructura chicken

Mediante este struct se determinan las características de cada uno de los chickens creados, cada uno de estos datos se almacena en un array llamado Chickens.

Aparte de esto, se tienen algunas variables con las cuales regular algunos aspectos del funcionamiento del contrato:

Tipo de dato	Nombre	Descripción	valor
uint	DnaDigits	Cantidad de dígitos que posee el adn.	16
uint	dnaModulus	Es una potencia de 10 elevado a DnaDigits y gracias a estos se consiguen los dígitos del adn.	$10^{**}dnaModulus$
uint	cooldownTime	Tiempo de enfriamiento después de haberse realizado un breeding.	1 days
uint	GestationCooldownTime	Tiempo en el que un huevo tarda en abrirse.	2 days
uint	breedingCooldownTime	Tiempo de enfriamiento al breedear.	3 days
uint	rewards	Cantidad de recompensa en	0.001 ether

		Ethereum al ganar una batalla.	
uint	createChickenFee	Cantidad de ether a pagar para poder crear un chicken.	0.01 ether

Tabla 6.2. Variables de contrato

Todas estas variables permiten ajustar el videojuego una vez desplegado y cada uno posee funciones get para obtener sus valores y set con las cuales el dueño del contrato puede ajustar cada una de las variables si fuese necesario.

Algunas de estas variables tienen valores que como days o ether los cuales su valor se obtiene mediante los siguientes conceptos:

El tiempo se mide en segundos por lo que cada unidad en uint significa un segundo y, por lo tanto

$$nDays = \frac{Uint}{60*60*24}$$

donde uint es el número de segundos por lo tanto 1 days son 86.400sg y el valor de la variable tendría un valor de 86400.

De la misma forma se obtienen los **ether** la cual se mida en wei y equivale a 10^{18} por lo tanto una variable como rewards de valor 0.001 ether equivaldrá a 10.000.000.000.000.000 uint.

$$1 \text{ ether} = 10^{18} \text{ wei}$$

Aparte se han utilizado varios mappings que nos permiten relacionar cada uno de los usuarios con diferentes aspectos del proyecto:

Nombre	Relación	Descripción
chickenToOwner	uint=>address	Permite almacenar y obtener el dueño de un chicken de manera sencilla.
ownerChickenCount	address=>uint	Permite almacenar y obtener la cantidad de chickens de cada usuario.
pendingWithdraws	address=> uint	Permite almacenar y obtener la cantidad de recompensas obtenidas por cada usuario.

Tabla 6.3. Mappings cryptochickens

Con esto se termina la estructura de las principales variables lo último que nos quedan son las funciones.

Se explicarán las más representativas y se obviarán getters y setters.

6.1.2. Funciones Cryptochickens

Nombre	Entrada	Salida	Descripción
createRandomChicken	- string _name		Crea un chicken con adn aleatorio y todos los cooldown a 0.
_generateRandomDna	- string _str	uint	Genera un numero aleatorio a partir de su nombre.
Breeding	- uint _firstChickenId - uint _secondChickenId		Permite breedear dos chickens y obtener un huevo.
_createChicken	- uint _name - uint dna		Función privada que permite crear un chicken.
_createEgg	- uint _name - uint dna		Función privada que crea un huevo con dos días de cooldown hasta abrirse.
attack	- uint _chickenId - uint _difficulty		Se realiza un ataque que mediante probabilidad se puede ganar.
withdrawUser	- address user		Permite al usuario recoger sus recompensas.
levelUp	- uint _chickenId		Permite al usuario que mediante pago se pueda subir de nivel a su chicken.
changeName	- uint _chickenId - string _newName		Permite al usuario cambiar el nombre de su chicken.
TransferFrom	- address _from - address _to - uint256 _tokenId		Permite transferir un chicken a otro usuario.

Tabla 6.4. Funciones más reconocidas del contrato

De aquí podemos comentar el funcionamiento de **_generateRandomDna**, que genera un adn random a partir del nombre proporcionado, utilizando una función hash que posee Ethereum llamada **keccak256**, la cual es una versión derivada de SHA3. Funciona introduciéndole una variable de bytes por lo que se utiliza otra función llamada **abi.encodePacked** la cual transforma un string a una variable de bytes. Esta variable se transforma a uint y se divide por la variable **dnaModulus** mostrada anteriormente para obtener los últimos 18 dígitos de ese string y así formar el ADN.

Otro aspecto interesante es el cálculo de los cooldowns (enfriamientos), para esto se utiliza una función de solidity llamada **now**, esta nos devuelve un numero uint con los segundos que han pasado desde el 1 de enero de 1970 hasta el día de hoy. Para

calcular los cooldowns, se le suma a este número cualquiera de las variables establecidas de cooldown y se le atribuyen al chicken en cuestión.

La función **_createEgg**, es similar a **_createChicken** pero posee un tiempo de gestación de 2 días, al cabo de esos dos días el huevo se abre y el chicken se podrá utilizar.

Las funciones de **breeding**, **attack** y **transferFrom**, poseen modificadores que permiten verificar que es el dueño del chicken es el que está realizando la transacción.

Nombre	Entrada	Descripción
onlyOwnerOf	- Uint_chickenId	Permite comprobar si el que realiza la transacción es el dueño del chicken.
aboveLevel	- Uint level - uint _chickenId	Permite comprobar si un chicken está por encima de un nivel determinado.
onlyOwner		Permite comprobar si el usuario que está realizando la transacción es el dueño del contrato.

Tabla 6.5. Modificadores

Estos modificadores ayudan a realizar tareas de comprobación recurrentes en varias funciones del contrato y lanzan un error en la transacción si no se cumplen sus requisitos.

6.2. Estructura del código

El proyecto se ha realizado con truffle, como se indicó anteriormente, para crear proyectos de truffle, primero se debe instalar truffle mediante la consola de comandos utilizando el comando:

```
yarn global add truffle
```

Tras esto utilizamos el comando unbox que nos proporciona truffle, en mi caso hice unbox del ejemplo que nos proporciona truffle en su documentación:

```
truffle unbox metacoin
```

que nos crea una estructura de proyecto tal que así:

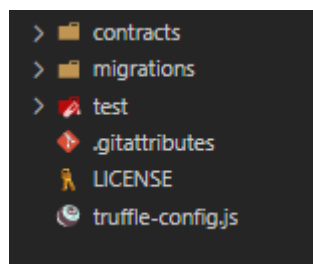


Figura 6.2. Estructura ficheros truffle

Tras esto se creó una carpeta llamada client y se generó la aplicación de react mediante:

```
mkdir client  
cd client  
npx create-react-app .
```

Tras esto se genera la aplicación de react básica y ya se tiene toda la estructura necesaria para empezar a desarrollar la aplicación.

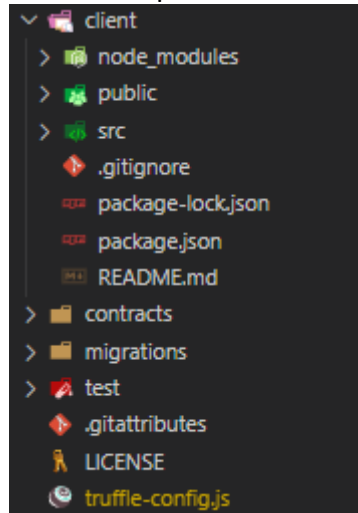


Figura 6.3. Estructura de ficheros completa

Retomando truffle, esta nos permite utilizar sus diferentes funcionalidades mediante algunos comandos que nos indican, tales como:

```
truffle compile (compilar)  
truffle migrate (desplegar contratos a la red)  
truffle test (ejecutar tests)
```

Uno de los ficheros más importantes que podemos ver en un proyecto de truffle es el fichero truffle-config.js en el que se indica la información sobre las networks a las que se va a desplegar el proyecto y las versiones de compiladores que vamos a necesitar para compilar nuestros contratos, en este caso el fichero se ve de la siguiente forma:

```
const path = require("path");

module.exports = {
  contracts_build_directory: path.join(__dirname, "client/src/contracts"),

  networks: {
    development: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "5777",
    },
  },
  compilers: {
    solc: {
      version: "0.5.16",
    },
  },
};
```

Figura 6.4. Contenido truffle-config.js

Se puede apreciar que el compilador que se está utilizando es la versión 0.5.16 y la red en la que se está lanzando el contrato es en el localhost con puerto 7545 y network id = 5777.

Por otro lado, dentro de la carpeta contracts se almacenarán nuestros contratos, en la carpeta test será donde desarrollaremos las pruebas para poder comprobar el buen funcionamiento de los contratos y por último en la carpeta de migrations será donde se configurará el lanzamiento de nuestros contratos.

Aparte de esto, para poder crear el entorno de desarrollo es necesario utilizar alguna herramienta que nos permita crear una blockchain local, en este caso se utilizó la aplicación de ganache en su versión con interfaz gráfica (existe una versión por consola), aunque también existe otro método utilizando el comando **truffle development**, pero la más sencilla es utilizando ganache.

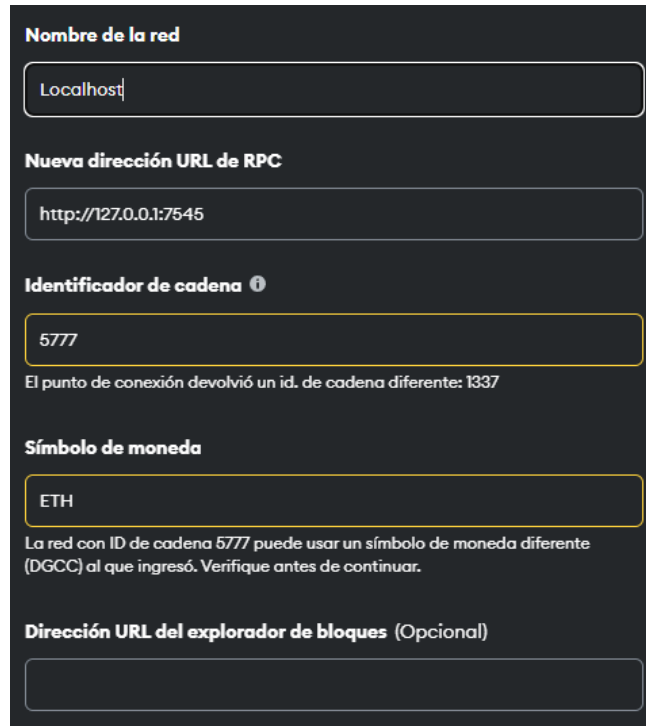
La configuración de nuestra red será la siguiente:

SERVER	WORKSPACE
<p>HOSTNAME</p> <p>127.0.0.1 - Loopback Pseudo-Interface 1 ▼</p>	<p>WORKSPACE NAME</p> <p>CryptoChickens</p>
<p>PORT NUMBER</p> <p>7545</p>	<p>TRUFFLE PROJECTS</p> <p>C:\Users\sgs98\Desktop\ei\CryptoChickens\truffle-config.js</p>
<p>NETWORK ID</p> <p>5777</p>	<p>ADD PROJECT REMOVE PROJECT</p>
<p>AUTOMINE</p> <p><input checked="" type="checkbox"/></p>	
<p>ERROR ON TRANSACTION FAILURE</p> <p><input checked="" type="checkbox"/></p>	

Figura 6.5. Configuración ganache

Como se puede ver los datos de conexión son los mismos que en el fichero truffle-config y se le añade su dirección en el workspace.

Tras esto es necesario configurar nuestra metamask con las mismas credenciales para poder acceder mediante está a nuestra blockchain local.



The image shows the Metamask configuration interface with the following fields and values:

- Nombre de la red:** Localhost
- Nueva dirección URL de RPC:** http://127.0.0.1:7545
- Identificador de cadena:** 5777
El punto de conexión devolvió un id. de cadena diferente: 1337
- Símbolo de moneda:** ETH
La red con ID de cadena 5777 puede usar un símbolo de moneda diferente (DGCC) al que ingresó. Verifique antes de continuar.
- Dirección URL del explorador de bloques (Opcional):** (Empty field)

Figura 6.6. Configuración Metamask

6.3. Cliente

Aunque no es lo principal en el proyecto, ya que se centra más en la tecnología blockchain, es necesaria una forma de poder interactuar con los contratos de manera sencilla, por lo que se ha intentado crear una interfaz accesible y sencilla de utilizar.

Para ello se ha utilizado el framework de javascript, react que permite una forma sencilla de desarrollo ya que permite escribir un lenguaje parecido a html llamado JSX (javascript XML), en conjunto con javascript, aparte de permitir la creación de componentes reutilizables de maneras muy sencillas.

En conjunto con react se han utilizado algunas librerías útiles para poder facilitar el desarrollo, conexión e interacción con los contratos de blockchain aparte de algunas librerías para facilitar el diseño y otras para enrutamientos.

- **Moralis:** Contiene toda la librería de moralis que permite la conexión e interacción con contratos de web3 de manera sencilla.
- **React:** Contiene la librería completa de React.
- **React-dom:** Es un paquete que permite herramientas para el uso del DOM (Document object model) de una manera eficiente.

- **React-moralis:** Junto con moralis proporciona componentes y hooks muy útiles para la conexión con contratos.
- **React-router-dom:** Es una librería que permite poder crear los diferentes enrutamientos en la web de manera sencilla, mediante el uso de componentes tales como Route, Link y Routes o hooks como useParams.
- **Web3uikit:** Es una librería con un conjunto de componentes muy útiles para el desarrollo de web3 tales como cartas, notificaciones, botones etc.

Páginas y rutas

En el proyecto se han desarrollado diferentes rutas, en las cuales se podrá interactuar con la aplicación:

Ruta	Componente	Descripción
<code>"/</code>	<code><Minting/></code>	En esta vista se pueden crear los chickens y se visualizarán los últimos creados.
<code>"/chickenbattle"</code>	<code><Battle/></code>	En esta vista puedes utilizar tus chickens para combatir por recompensas.
<code>"/chicken/:id"</code>	<code><Chicken/></code>	En esta vista puedes ver todos los atributos de tus chickens.
<code>"/chickens"</code>	<code><MyChickens/></code>	En esta vista se pueden ver tus chickens en propiedad y un apartado con las recompensas acumuladas y un botón para retirarlas.
<code>"/breeding"</code>	<code><Breeding/></code>	En este apartado puedes aparear a dos chickens y obtener huevos.
<code>"/about"</code>	<code><AboutUs></code>	Esta vista muestra información general sobre la página y los contribuyentes a su desarrollo
<code>"*"</code>	<code><NotPage/></code>	Esta vista muestra error cuando se accede a una vista no contemplada entre las anteriores.

Tabla 6.6. Rutas de la aplicación

Testing y resultados

El testing es una de las partes fundamentales del desarrollo de software, y nos permiten asegurarnos de que un producto funciona de la manera en la que debería. En el caso del desarrollo de contratos inteligentes, la fase de testing es una parte crítica, ya que una vez desplegado no puede ser modificado, por lo que se debe tener un contrato con un funcionamiento correcto.

En este caso los tests están enfocados a confirmar que todas las funcionalidades del contrato funcionan correctamente y que los permisos de acceso a determinadas funciones son correctos ya que si un usuario pudiese acceder por ejemplo al saldo del contrato esto podría ser fatal ya que podría ser robado. Los tests están realizados en JavaScript Mocha que es propio del framework de Truffle [25], aparte se ha añadido chai para poder confirmar las funciones en las que se obtienen errores.

A la hora de realizar los tests es necesario primero especificar el contrato al que se le van a realizar las pruebas, para esto se utiliza el método `artifacts` que nos permite importar el contrato.

```
const CryptoChickens = artifacts.require("CryptoChickens");
```

Estos tests se encuentran dentro de la carpeta `test` del proyecto en el fichero `chickenFactory.js`.

Los tests se inician con el bloque de `contract` el cual habilita las funcionalidades de Truffle. Antes de realizarse los tests truffle primero compila el contrato y después lo despliega en la red que tengamos habilitada, ganache en este caso, tras esto es cuando los tests comienzan y se puede entrar en el bloque de `contract`. En este caso se le han añadido algunas direcciones dentro de los argumentos para poder utilizarlos en los diferentes tests.

```
contract("CryptoChickens", ([deployer, user1, user2]) => {
```

Antes de realizar cada test se comprueba que el contrato ha sido lanzado mediante la función `deployed()`.

```
    const chickenContract = await CryptoChickens.deployed();
```

Los tests se han dividido en diferentes partes, cada uno englobando una de las funcionalidades principales que se mostraron en la **figura 6.1**, a continuación, se mostraran algunos de estos tests:

```
describe("Chicken Factory testing", async () => {
  it("should create a chicken", async () => {
    const cryptoChickens = await CryptoChickens.deployed();
    await cryptoChickens.createRandomChicken("Chicken 1", {
      from: deployer,
      value: web3.utils.toWei("0.01", "ether"),
    });

    //can't create a chicken if it isn't enough ether
    await await cryptoChickens.createRandomChicken("Chicken 2", {
      value: web3.utils.toWei("0.001", "ether"),
      from: user1,
    }).should.be.rejected;

    const totalChickens = await cryptoChickens.getChickenLength();
    assert.equal(totalChickens, 1);
  });

  it("get chicken by id", async () => {
    const cryptoChickens = await CryptoChickens.deployed();
    const chicken = await cryptoChickens.getChickenFromId(0);
    assert.equal(chicken.name, "Chicken 1");
  });

  it("set and get reward chicken", async () => {
    const cryptoChickens = await CryptoChickens.deployed();
    await cryptoChickens.setRewards(web3.utils.toWei("0.01", "ether"));

    //only the deployer can set rewards
    await await cryptoChickens.setRewards(web3.utils.toWei("0.01",
"ether")), {
      from: user1,
    }).should.be.rejected;
    const rewards = await cryptoChickens.getRewards();
    assert.equal(rewards, web3.utils.toWei("0.01", "ether"));
  });
});
```

```

it("set and get chicken fee", async () => {
  const cryptoChickens = await CryptoChickens.deployed();
  await cryptoChickens.setCreateChickenFee(
    web3.utils.toWei("0.01", "ether")
  );
  //only the deployer can set the fee
  await await cryptoChickens.setCreateChickenFee(
    web3.utils.toWei("0.01", "ether"),
    {
      from: user1,
    }
  ).should.be.rejected;
  const chickenFee = await cryptoChickens.getCreateChickenFee();
  assert.equal(chickenFee, web3.utils.toWei("0.01", "ether"));
});
});

```

En este caso se comprueban las diferentes funciones presentes en el fichero *chicken factory* donde mediante el uso de chai [26] nos aseguramos mediante la función de *should.be.rejected* que en determinados casos la respuesta de los métodos será errónea ya sea porque no se posea acceso a la función dependiendo del usuario o que no se pueda realizar la función porque no se han añadido suficientes fondos o por que se han introducido datos erróneos. Aparte mediante la función *assert.equal* confirmamos que la salida de nuestras funciones es la correcta.

De la misma forma que se han realizado los tests en el caso de *chicken factory*, estos se han realizado para el resto de los contratos que han sido programados, comprobando que ningún usuario no autorizado pueda ejecutarlas como que si no se cumplen determinados requisitos no se puedan ejecutar y que se obtienen los resultados esperados después de ejecutar las funciones de *chickenattack*, *chickenbreeding*, *chickenhelper* y *chickenownership*.

En total se han realizado 18 tests unitarios, y estos pueden ser ejecutados mediante la consola de comandos introduciendo *truffle test*, y que en este caso se obtiene el siguiente resultado:

```
Contract: CryptoChickens
Deployment testing
  ✓ Deployment successful (49ms)
Chicken Factory testing
  ✓ should create a chicken (3377ms)
  ✓ get chicken by id (213ms)
  ✓ set and get reward chicken (1942ms)
  ✓ set and get chicken fee (5208ms)
Chicken Breeding testing
  ✓ breed two chickens (10187ms)
  ✓ breeding cooldown (444ms)
  ✓ gestation time (266ms)
chicken helper testing
  ✓ get chicken by owner (176ms)
  ✓ set and get levelup fee (797ms)
  ✓ level up chicken (830ms)
  ✓ change chicken name (6506ms)
  ✓ withdraw to user (1268ms)
  ✓ withdraw to deployer (1743ms)
chicken ownership testing
  ✓ transfer chicken (949ms)
  ✓ chicken approval (562ms)
Chicken attack testing
  ✓ fight two chickens (3595ms)
  ✓ set and get probabilities (1673ms)

18 passing (41s)
```

Figura 9.1. Resultados tests

Como se puede apreciar todos los tests realizados han sido pasados con éxito por lo que se asegura un buen funcionamiento de los contratos.

8

Conclusiones y líneas futuras

8.1. Conclusiones

Algo es claro y es que el ecosistema Blockchain es una tecnología increíble, un mundo nuevo por explorar y con muchas oportunidades, todo esto en gran parte gracias a Ethereum y los smart contracts que han revolucionado el mundo de las blockchain haciéndola pasar de algo más parecido a una base de datos descentralizada a la posibilidad de poder crear aplicaciones descentralizadas de cualquier ámbito.

Para este trabajo se ha realizado una investigación exhaustiva de las tecnologías dentro del ecosistema blockchain, y se han puesto en práctica estos conocimientos para la creación de un prototipo en el que se intentan demostrar las capacidades de la blockchain dentro de un videojuego.

De esta forma se ha creado en prototipo de CryptoChickens, el cual ha sido desarrollado con los estándares de aplicación descentralizada. Es de código abierto, los datos se almacenan en el contrato inteligente y se puede desplegar en redes como Ethereum y sus derivados.

El contrato inteligente desarrollado gestiona la lógica del juego y al ser público desplegado en la blockchain cualquier usuario puede comprobar su funcionamiento, esto permite la transparencia y se evita la posibilidad de timos. El contrato permite la obtención de NFTs, combatir con ellos, reproducirlos etc.

También se ha desarrollado una interfaz de usuario con la que interactuar con el contrato de manera sencilla e intuitiva. Con esto se ha puesto a prueba el funcionamiento de una aplicación descentralizada completa. La interfaz permite

conectarse con nuestra wallet, realizar transacción e interactuar con las diferentes funciones de la aplicación.

Respecto a la finalidad del trabajo de buscar si es posible la sostenibilidad de los videojuegos DeFi tal y como se plantean hoy en día, como se comentó durante el proyecto, de la forma que se encuentran hoy en día en la que los videojuegos se basan en su economía y obtener grandes cantidades de dinero, es algo insostenible y a largo plazo todos los proyectos hasta la fecha que han procedido con esa premisa han quebrado en el corto plazo.

Un videojuego como tal está hecho para divertirse y en el momento en el que se transforma en un trabajo el videojuego pierde su verdadera finalidad, aparte el tema de la economía es una carga adicional que se le suma al desarrollo del juego, esto ha ocurrido con muchos proyectos donde en el momento que veían que se aproximaba una crisis de su economía, al igual que en una economía real, realizaban recortes de recompensas o cambios de mecánicas. Estos recortes al igual que en la economía real, producen una ralentización de la economía por lo que se estanca y no mejora, las personas dejan de interesarse por el juego y quiebra. También se ha intentado de la forma contraria en la que, en vez de realizar recortes, se aumentaban las recompensas para que más usuarios se interesasen por él y de igual manera la economía acababa quebrando.

Lo que podemos sacar de conclusión de todo esto, es que la blockchain y los NFTs deben ser un complemento de los videojuegos, algo que haga más sencillo el uso del videojuego en sí, ya sea utilizando NFTs para diferentes aspectos del videojuego como aspectos, personajes etc. o una economía basada en criptomonedas que funcione como complemento y que permita a los usuarios sacar un pequeño beneficio pero en ningún caso que se otorguen recompensas excesivas ya que a largo plazo se vuelve insostenible.

8.2. Líneas futuras

Existen varios aspectos en los que se podría ampliar el prototipo desarrollado, y que no han sido incluidos en el proyecto debido a sus dimensiones. Se enuncian a continuación algunas líneas que se podrían mejorar.

Creación de un Marketplace: En este proyecto los chickens pueden ser transferidos a otros usuarios, pero no se pueden vender, por lo que se propone que se pudiera implementar en el contrato una posibilidad de comerciar los chickens con otros usuarios, poder comprar y vender los NFTs.

Implementación de un token ERC-20: En el proyecto se utiliza ETH para comprar los NFTs y repartir las recompensas, se propone la implementación de un token ERC-20 para la compra y las recompensas, esto permitiría un mayor control sobre la economía y no depender del valor que tenga ETH en cada momento.

Implementación de mecánicas de combate: En el proyecto los combates se realizan únicamente por probabilidad y no dependen de la habilidad del jugador, por lo que se propone la implementación de algún tipo de modelo de combate por turnos.

Creación de NFTs de forma automática: En el proyecto los diferentes NFTs dependen del ADN de cada uno, pero al final solo existen 10 diseños diferentes, se propone la implementación de un mecanismo de generación de NFTs mediante el uso de Opensea y la creación de una API de una base de datos para almacena

Referencias

- [1] Criptografía de clave pública, IBM.
<https://www.ibm.com/docs/es/integration-bus/10.0?topic=overview-public-key-cryptography>
- [2] Sistema distribuido, Atlassian.
<https://www.atlassian.com/es/microservices/microservices-architecture/distributed-architecture>
- [3] Binance Academy Layers 1-2, Binance.
<https://academy.binance.com/en/articles/blockchain-layer-1-vs-layer-2-scaling-solutions>
<https://academy.binance.com/en/articles/what-is-layer-1-in-blockchain>
- [4] Transacciones en Ethereum, Ethereum.
<https://ethereum.org/en/developers/docs/transactions/>
- [5] Mecanismos de consenso, Ethereum.
<https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/>
- [6] Industria de videojuegos, Wikipedia.
https://en.wikipedia.org/wiki/Video_game_industry
- [7] Monetización videojuegos, Wikipedia.
https://en.wikipedia.org/wiki/Loot_box
<https://en.wikipedia.org/wiki/Free-to-play>
- [8] Tokens ERC-20, Investopedia.
<https://www.investopedia.com/news/what-erc20-and-what-does-it-mean-ethereum/>
- [9] Tokens ERC-721, Decrypt.
<https://decrypt.co/resources/erc-721-ethereum-nft-token-standard>
- [10] Peer-to-Peer Networks, Binance.
<https://academy.binance.com/en/articles/peer-to-peer-networks-explained>
- [11] ¿Que son los contratos inteligentes?, Binance.
<https://academy.binance.com/es/articles/what-are-smart-contracts>
- [12] ¿Qué son los juegos NFT y cómo funcionan?, Binance.
<https://academy.binance.com/en/articles/what-are-nft-games-and-how-do-they-work>
- [13] Guía para principiantes de finanzas descentralizadas (DeFi), Binance.
<https://academy.binance.com/en/articles/the-complete-beginners-guide-to-decentralized-finance-defi>
- [14] Solidity documentation, Ethereum.
<https://docs.soliditylang.org/en/v0.5.16/>
- [15] Learn Solidity, Youtube.
<https://www.youtube.com/watch?v=gyMwXuJrbJQ&t=3727s>
- [16] DeFi games course. CryptoZombies. <https://cryptozombies.io/en/course>
- [17] CSS. Learn CSS. <https://web.dev/learn/css/>
- [18] HTML, W3Schools. <https://www.w3schools.com/html/>

- [19] Truffle Suite. <https://www.trufflesuite.com/>
- [20] Ganache. <https://www.trufflesuite.com/ganache>
- [21] Remix. <https://remix.run/docs/en/v1/tutorials/blog>
- [22] React. <https://reactjs.org/docs/getting-started.html>
- [23] React Moralis. <https://docs.moralis.io/>
- [24] MetaMask. <https://metamask.io/>
- [25] Testing your contracts, Truffle Suite.
<https://trufflesuite.com/docs/truffle/testing/testing-your-contracts/>
- [26] Chai Library. <https://www.chaijs.com/>
- [27] JavaScript. <https://developer.mozilla.org/en-US/docs/Learn/JavaScript>
- [28] Repositorio CryptoChickens. <https://github.com/SergioCltn/CryptoChickens>

Apéndice A

Manual de Instalación

En el siguiente manual se pretende enumerar las diferentes posibilidades que brinda la aplicación desarrollada y como utilizarlas. Como punto adicional se ha creado un repositorio en GitHub donde en el fichero **README** se explica la forma de instalación del entorno local. [28]

A.1. Instalación de metamask

Para poder interactuar con la aplicación es necesario la instalación de metamask que es una extensión de navegador, la podemos instalar mediante su página oficial.

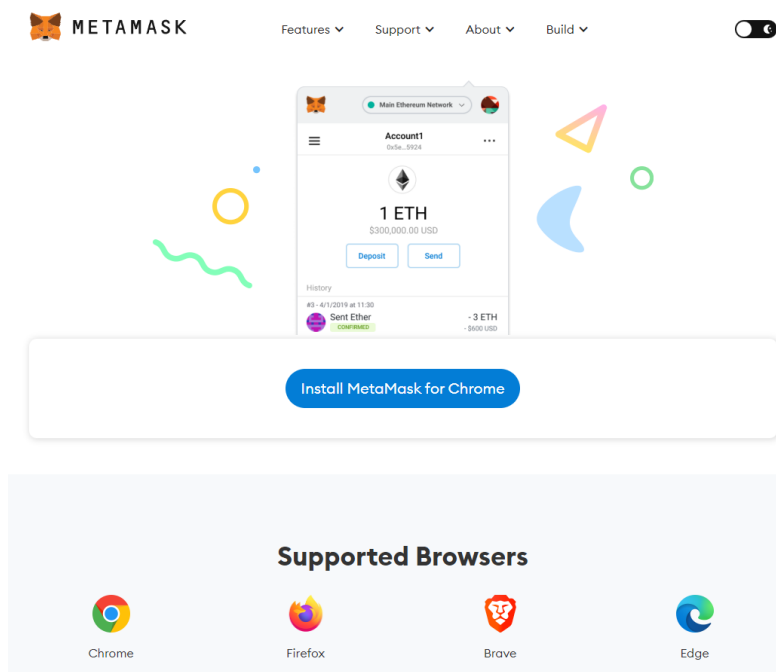


Figura A.1. Descargar Metamask (extraída de <https://metamask.io/download/>)

Una vez descargada e instalada la extensión, podremos crear o importar cuentas, y tendremos que añadir una contraseña para poder desbloquear la wallet en nuestro navegador. A partir de este momento cada vez que vayamos a realizar una acción en la aplicación metamask nos pedirá una confirmación.

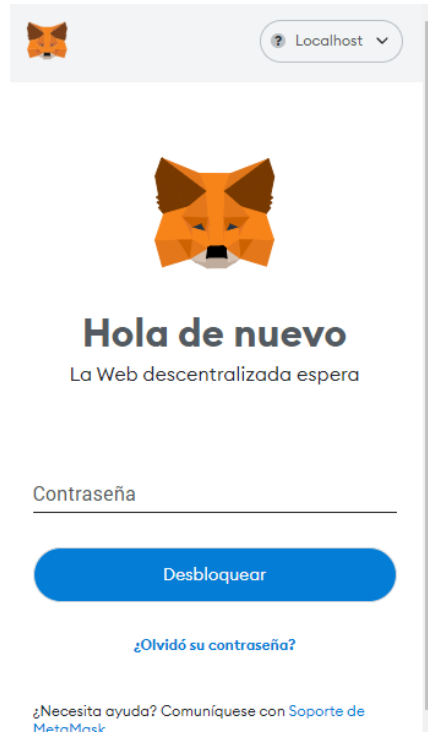


Figura A.2. Interfaz de inicio de sesión Metamask.

Y una vez logueados nos aparecerá la siguiente pestaña:



Figura A.3. Interfaz Metamask.

A.2. Conexión con wallet

Para poder utilizar todas las funcionalidades es necesario conectar la wallet.

Para poder conectar la wallet a nuestra blockchain local en ganache, es necesario importar una cuenta desde este, ya que estas cuentas vienen con fondos que son necesarios para pagar el gas y el precio de algunas funcionalidades.

Para ello debemos importarla utilizando la clave privada dentro de nuestro metamask de la siguiente forma:

1. En ganache clicamos en la llave de la derecha

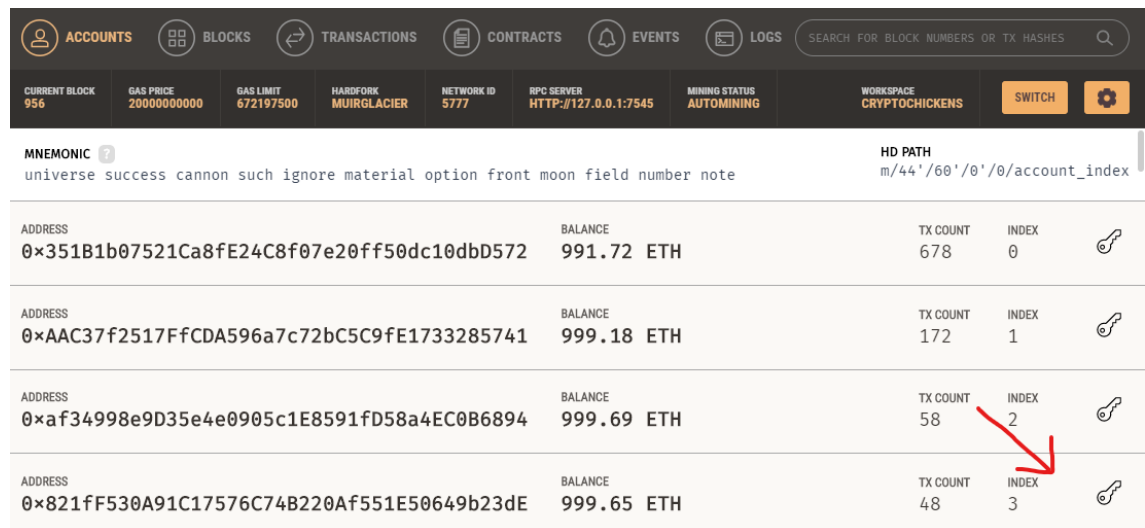


Figura A.4. Interfaz Ganache.

2. Copiamos la clave privada de la cuenta que hayamos seleccionado

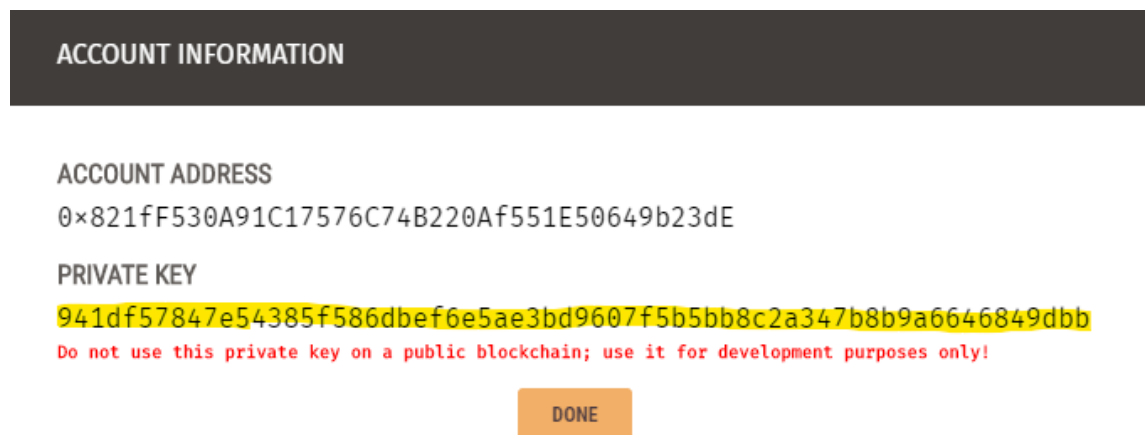


Figura A.5. Interfaz para obtener private key.

3. Accedemos al metamask y le clicamos a importar cuenta y pegamos la clave privada que hemos copiado

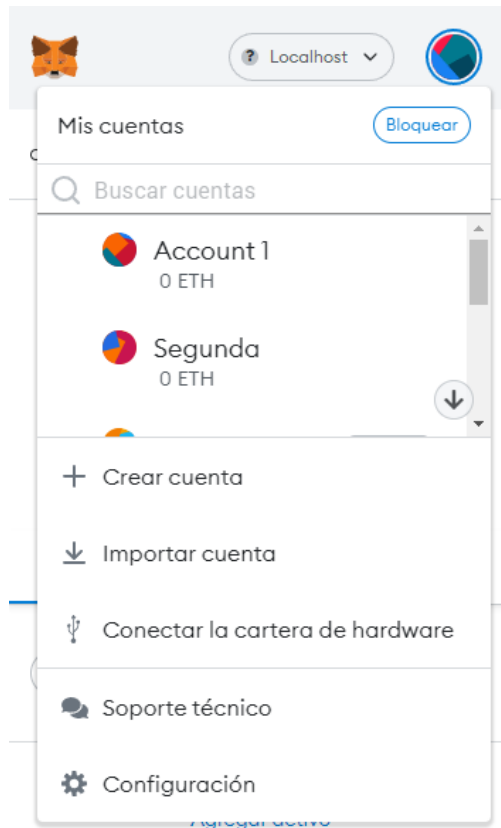


Figura A.6. Interfaz Metamask para importar cuenta.

Una vez importada la cuenta ya podemos conectarnos a la aplicación clicando en el botón “Connect Wallet”, hasta que la cuenta no está conectada no se nos permitirá interactuar con ninguna funcionalidad.

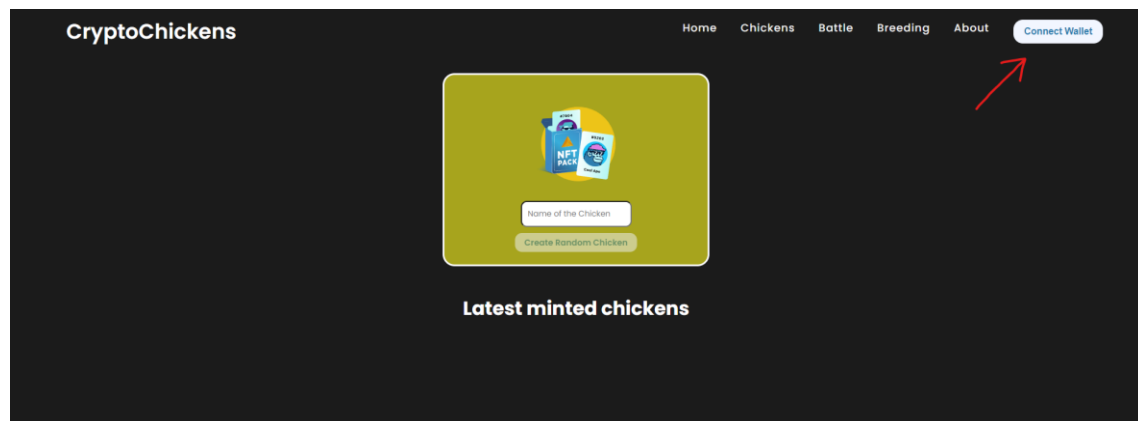


Figura A.7. Vista Home “Connect Wallet”

Seleccionamos la opción de metamask.

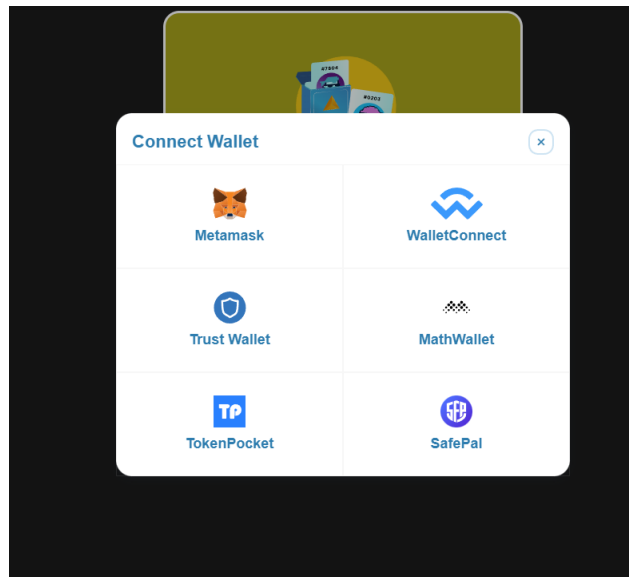


Figura A.8. Vista selección de Wallet.

Metamask abrirá una pestaña donde debemos clicar en siguiente y acto seguido clicamos en conectar.

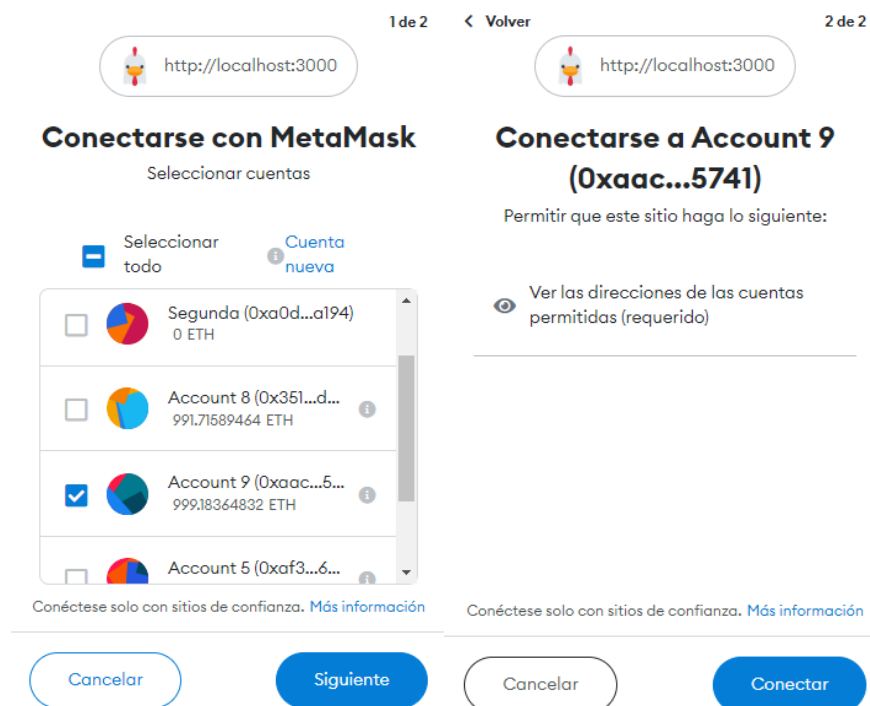


Figura A.9. Metamask conexión a aplicación.

Una vez conectados aparecerá nuestra cantidad de ETH y nuestra dirección de metamask en la esquina derecha y ya se nos permitirá utilizar las funciones de la web.

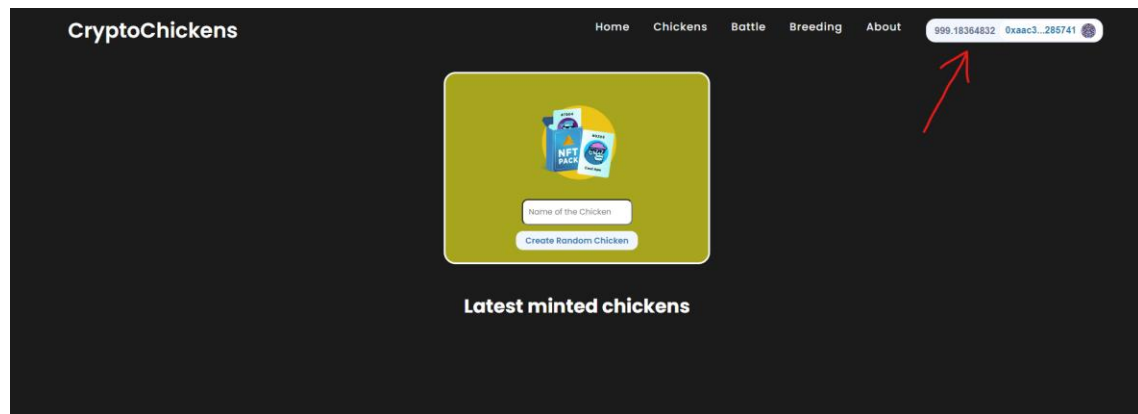


Figura A.10. Vista Home con metamask conectado.

A.3. Mintear chickens

Un usuario que ya ha sido logueado con su metamask ya se le permite mintear, pero aún no puede utilizar ninguna de las funciones extras ya que para esto es necesario la obtención de algún NFT.

La interfaz de **Home** es sencilla se tiene una barra superior donde podemos seleccionar las diferentes pestañas de la aplicación, también se tiene un apartado en un recuadro amarillo donde podemos introducir el nombre de nuestro NFT y crearlo y en la parte inferior una sección donde se muestran los últimos chickens minteados en la aplicación ya sean nuestros o de otros usuarios.

Para mintear nuestro chicken rellenamos el nombre y pulsamos el botón “Create random Chicken”.



Figura A.11. Apartado “Create Chicken”.

Una vez hemos pulsado el botón nos aparecerá una pestaña de metamask para confirmar la transacción, esta transacción tendrá un valor de 0.01 ETH que es el valor por defecto para poder obtener un chicken, junto a una cantidad de gas para pagar la transacción.

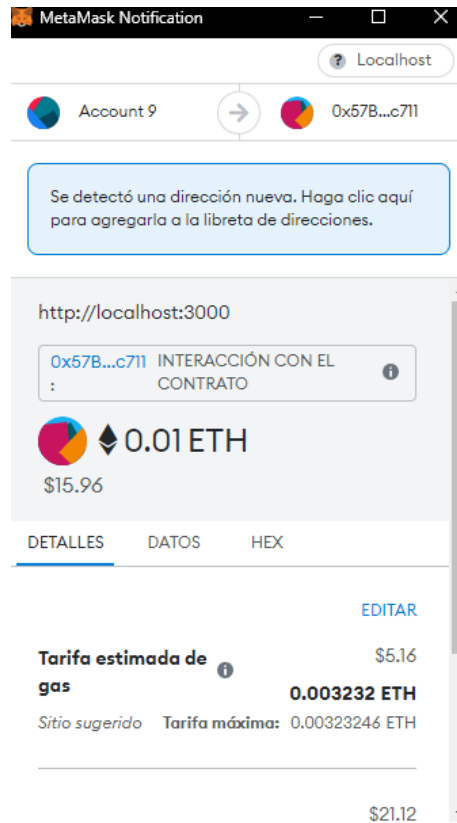


Figura A.12. Interfaz Metamask aceptar “Create random Chicken”.

Aceptamos la transacción y una vez tiene éxito nos aparecerá una notificación de que se ha completado la transacción y obtendremos nuestro chicken que podremos visualizar en la pestaña de chickens.

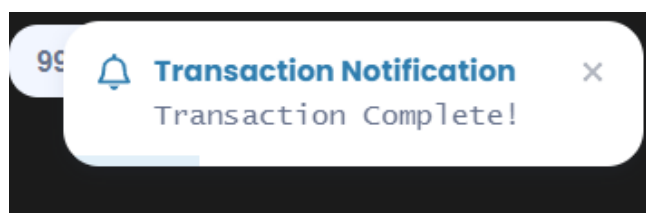


Figura A.13. Notificación transacción completada.

Aparte aparecerá en la sección “Latest minted chickens” ya que es el último que ha sido mintado.

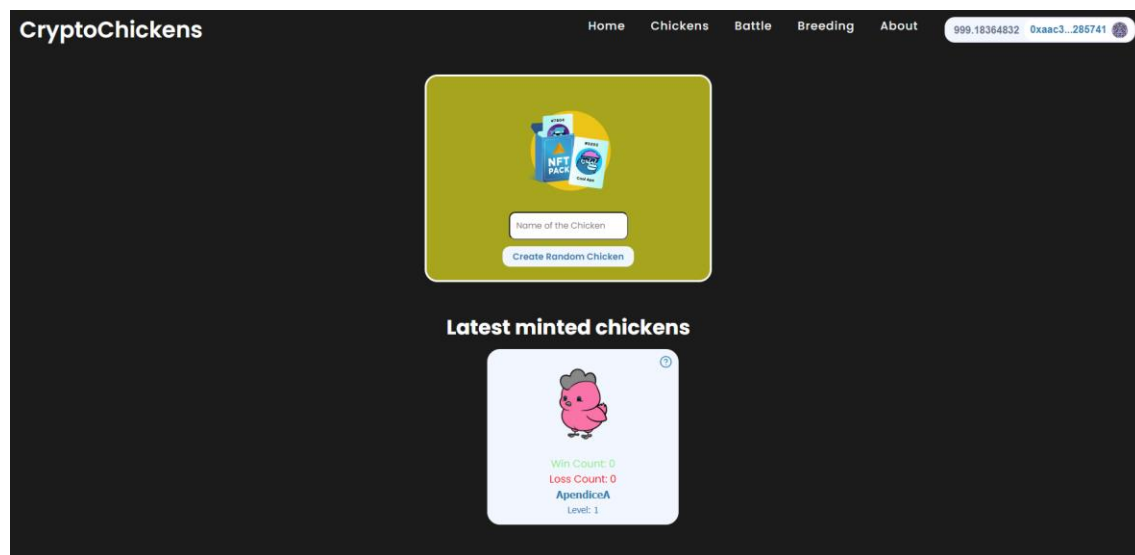


Figura A.14. Vista Home tras crear chicken.

En la **figura A.14.** podríamos clicar en cualquiera de los “Latest chickens” y podríamos acceder a sus datos al igual que veremos en el apartado **A.4.**

A.4. Visualización de chickens y recompensas

Accediendo al apartado de “**Chickens**” desde la barra superior podremos visualizar una nueva pestaña donde se nos muestran nuestras recompensas acumuladas jugando y nuestros chickens en propiedad, en este caso aparecerá nuestro chicken con nombre “ApendiceA”.

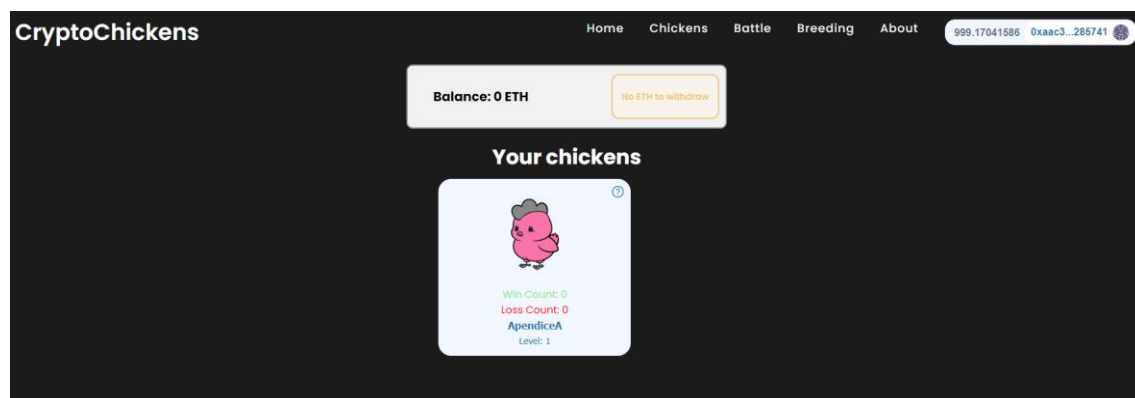


Figura A.15. Vista Chickens.

Si clicamos en nuestro chicken nos llevará a una pestaña con los datos más en profundidad de nuestro NFT.

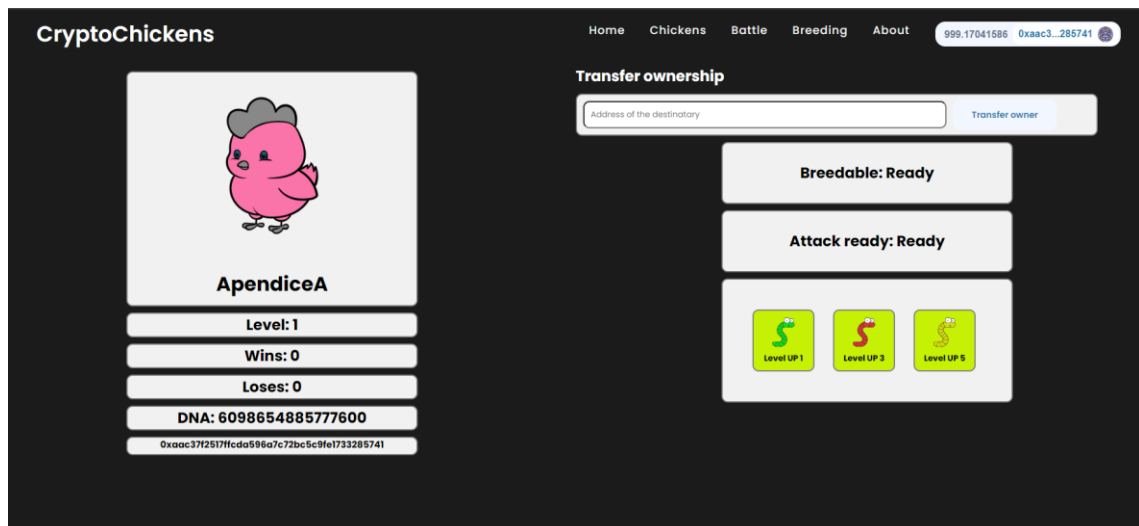


Figura A.16. Vista estadísticas chicken.

En esta pestaña de la **figura A.16.** podemos ver todas las estadísticas que posee nuestro chicken aparte de un apartado donde podemos subir de nivel a nuestro chicken pagando y otro para enviar nuestro chicken a otro usuario si se quisiera.

A.4.1. Subida de nivel

Una de las funciones dentro de chickens es la subida de nivel, para poder utilizarla se seguirán los siguientes pasos.

Clicamos en uno de los 3 gusanos, el verde sube un nivel con un coste de 0.001, el rojo sube 3 niveles con un coste de 0.003 y el rojo sube 5 niveles con el coste de 0.005. Clicamos en el amarillo y nos aparece una pestaña de metamask para aceptar la transacción.

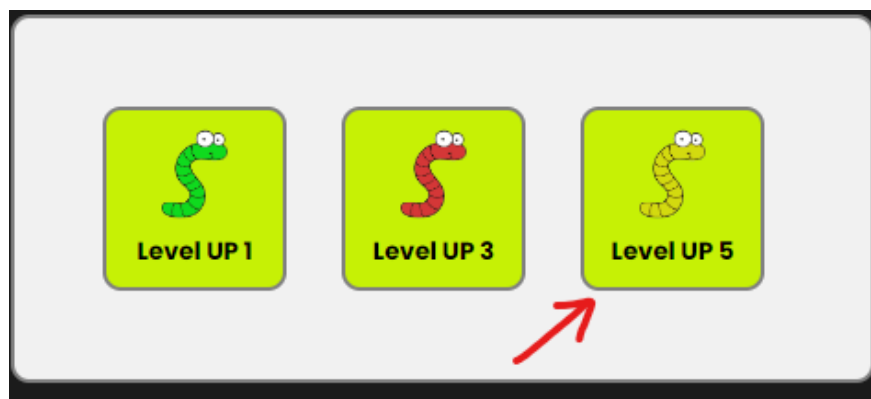


Figura A.17. Opciones de subida de nivel.

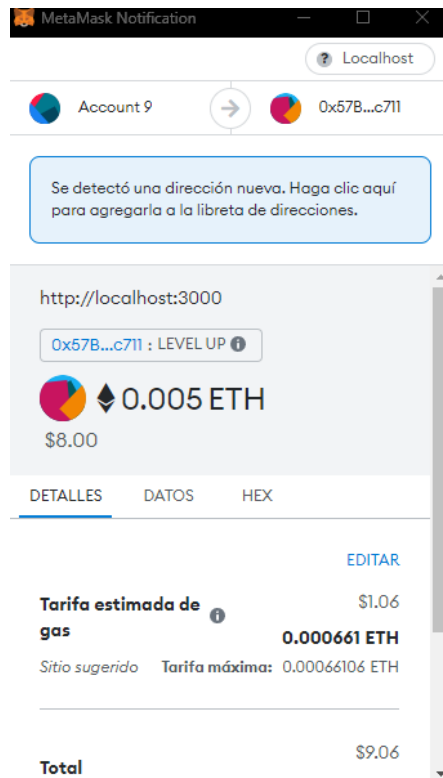


Figura A.18. Confirmación transacción Metamask.

Aceptamos la transacción, se mostrará una notificación de que la transacción se ha completado y nuestro chicken subirá 5 niveles.

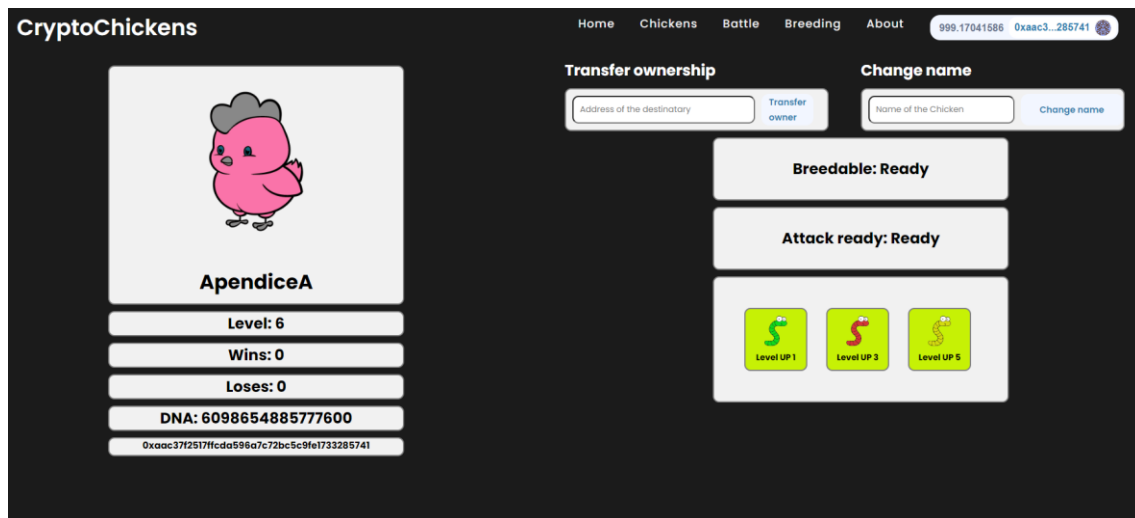


Figura A.19. Vista estadísticas chicken después de subir de nivel.

Un chicken cuando se encuentra por encima de nivel 2 se le permite cambiar de nombre, por lo que al subirle 5 niveles y estar al nivel 6 se nos desbloquea esa opción para cambiarle el nombre.

A.4.2. Cambio de nombre.

Otra función que podemos utilizar en la vista **Chickens** es la de cambio de nombre una vez que el nivel de nuestro chicken es superior a 2.

Para utilizarla debemos introducir un nombre dentro del campo de texto de cambio de nombre y clicar el botón “Change name”.



Figura A.20. Change name.

Se nos abrirá una pestaña de metamask para confirmar la transacción y la confirmamos.

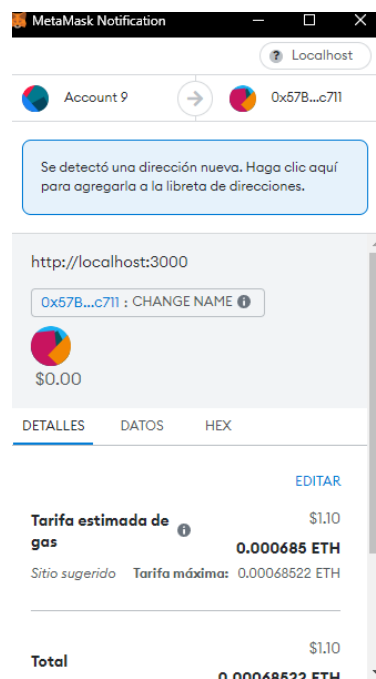


Figura A.21. Confirmación transacción Metamask.

Aparecerá una notificación de que la transacción se ha completado con éxito y nuestro chicken habrá cambiado de nombre.



Figura A.22. Estadísticas del chicken después del cambio de nombre.

A.4.3. Transferencia de chicken.

Otra de las funciones que nos ofrece esta vista es la capacidad de transferir nuestro chicken a otra dirección.

Para poder utilizar esta función rellenamos el campo de texto de transfer ownership con una dirección válida y pulsamos el botón "Transfer Ownership".

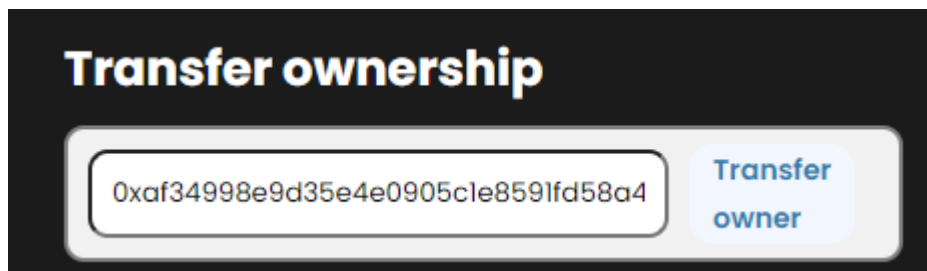


Figura A.23. Transfer ownership.

Se nos abrirá una pestaña de metamask para confirmar la transacción y la confirmamos.

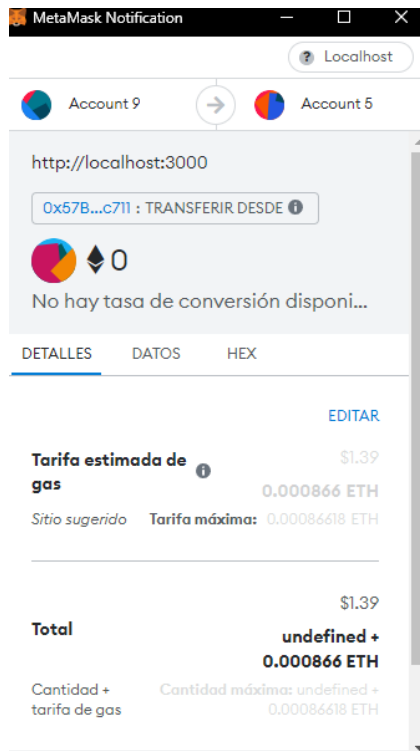


Figura A.24. Confirmación transacción Metamask.

Aparecerá una notificación de que la transacción se ha completado con éxito y nuestro chicken habrá sido transferido a la dirección que hemos proporcionado y dejará de ser de nuestra propiedad.

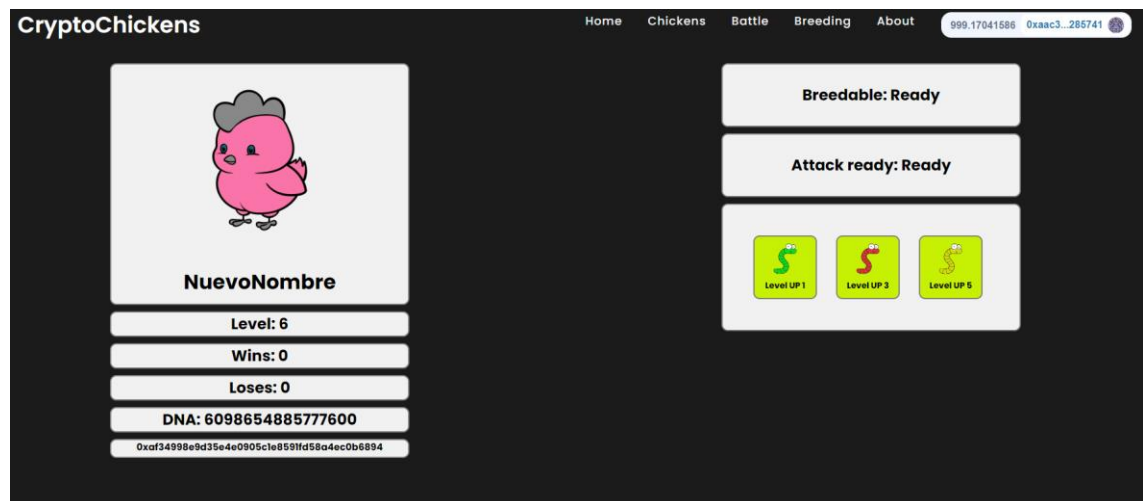


Figura A.25. Vista estadísticas chicken si no es de nuestra propiedad.

Como se puede observar en la **figura A.25.** cuando un chicken no es de nuestra propiedad las funciones de cambio de nombre y de transferencia desaparecen.

A.5. Combate

Cuando tenemos en propiedad algún chicken que está disponible para atacar podemos acceder a la pestaña de **Battle** para combatir.

Para acceder a ella clicamos en opción de Battle en la barra superior y nos aparecerá la siguiente vista.

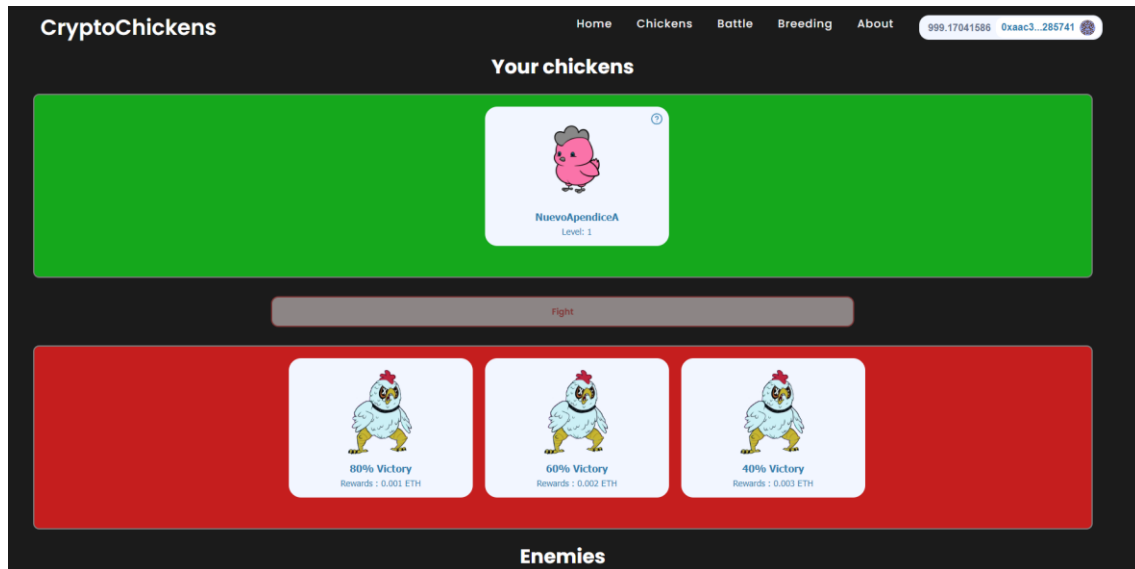


Figura A.26. Vista Battle.

En esta vista se nos muestra en la zona verde superior todos nuestros chickens disponibles para combatir y en la zona roja inferior se nos muestran los diferentes enemigos, cada uno con su probabilidad de victoria y recompensas que ofrecen. Cuanta mayor probabilidad más fácil es ganar el combate, pero nos darán menos recompensas.

Para utilizar esta vista debemos seleccionar cual de nuestros chickens utilizar y a cuál de los enemigos atacar, para ello clicamos sobre ellos y se nos marcaran con un tick verde. Tras esto se nos activará el botón “Fight” para poder clicarle.

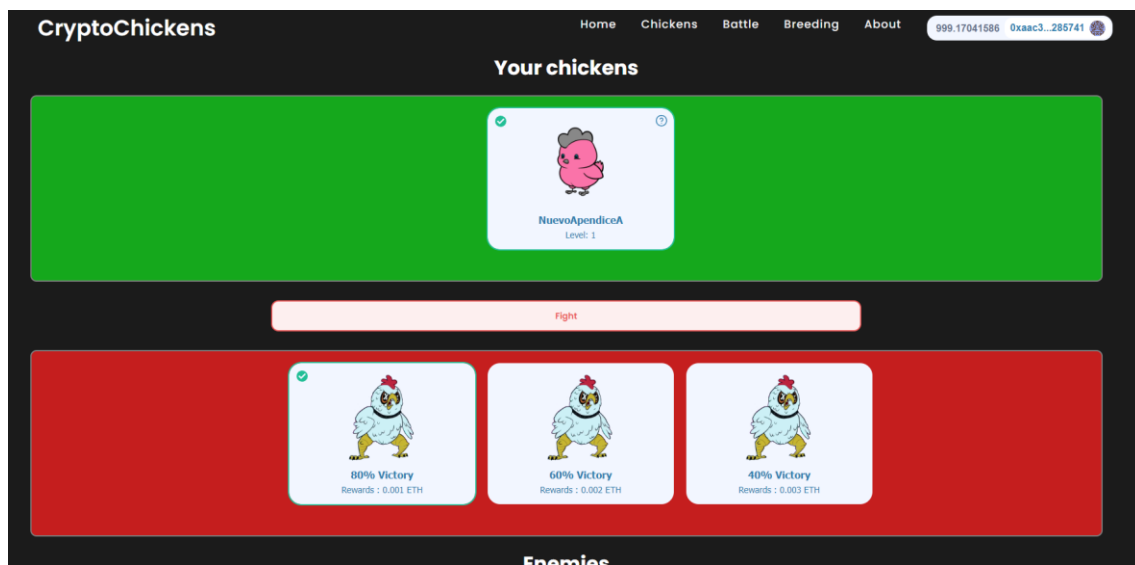


Figura A.27. Vista Battle después de seleccionar.

Clicamos el botón “Fight” y nos abrirá una pestaña en metamask para la confirmación.

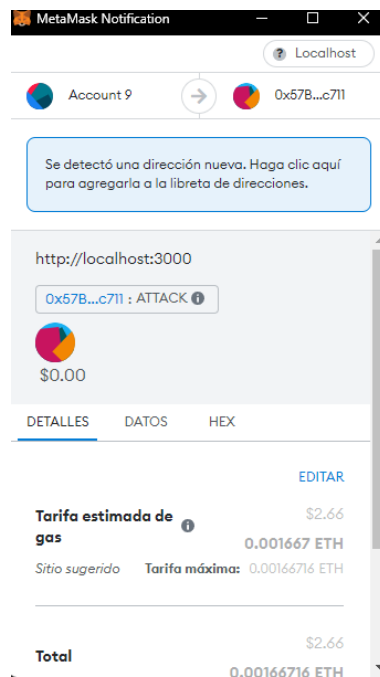


Figura A.28. Confirmación transacción Metamask.

Confirmamos la transacción y aparecerá una notificación que nos mostrará si hemos ganado o perdido la batalla. Si perdemos no nos darán ninguna recompensa y si ganamos nos darán la recompensa del enemigo que hayamos atacado.

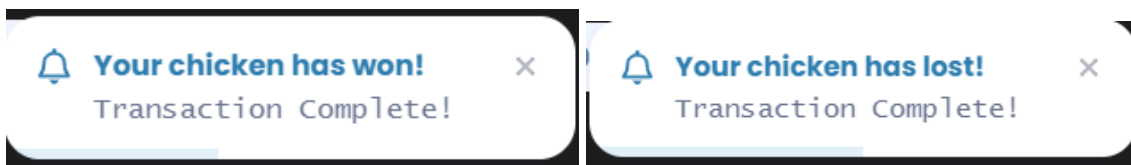


Figura A.29. Notificaciones tras combate.

En este caso ha sido victorioso por lo que obtenemos una recompensa. Una vez realizado el ataque el chicken entra en cooldown durante un día, tras este tiempo podrá atacar de nuevo.

Como no tenemos ningún chicken más disponible para ataque nos aparece el siguiente mensaje en la vista de battle.

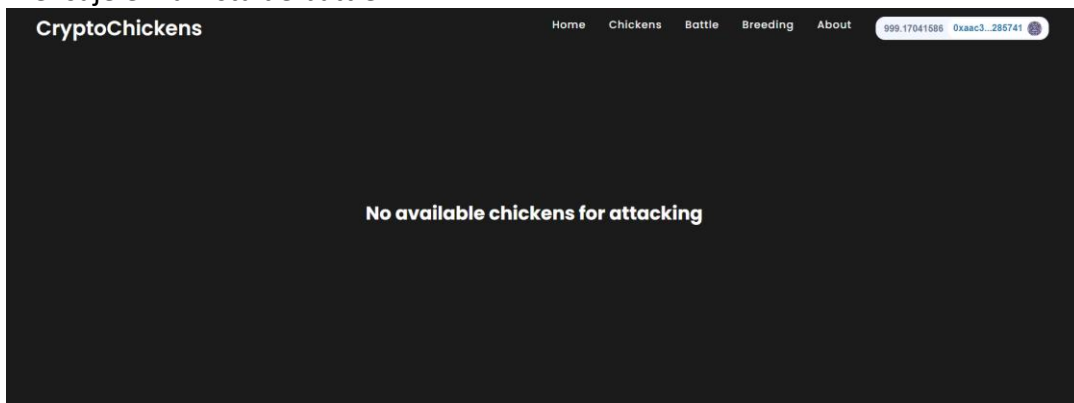


Figura A.30. Vista Battle sin chickens disponibles.

Para poder ver nuestras recompensas podemos entrar en la pestaña de chickens en la barra superior y ver cuantas recompensas hemos acumulado.

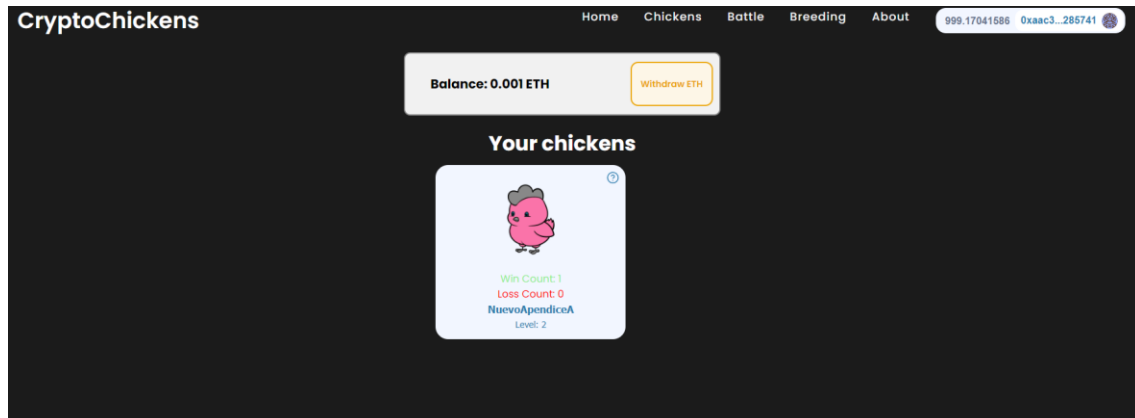


Figura A.31. Vista Chickens con recompensas acumuladas.

Como se puede observar en la **figura A.31.** se tienen acumuladas 0.001 ETH después de la victoria al chicken seleccionado anteriormente. Ahora nuestro chicken no puede atacar hasta el día siguiente y en sus estadísticas se ha sumado una victoria y ha subido un nivel.

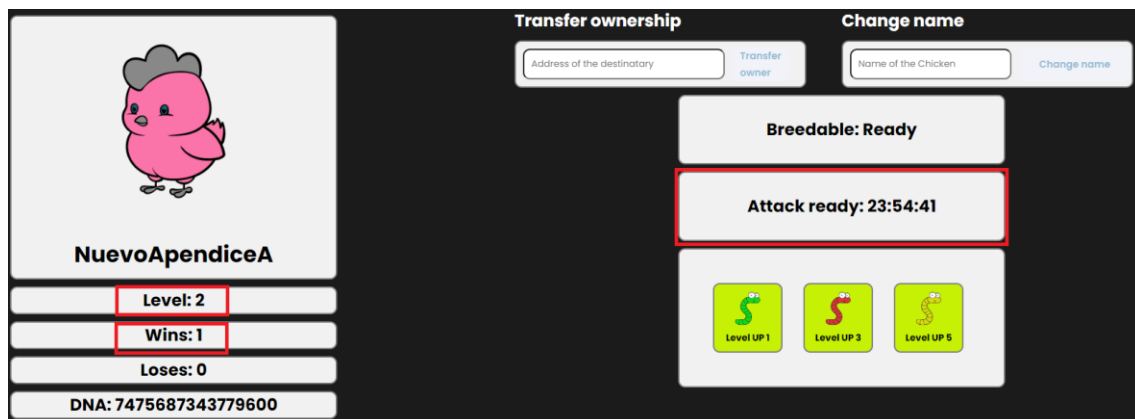


Figura A.32. Vista de estadísticas actualizadas tras el combate.

A.6. Retirar recompensas

Si nos encontramos en la vista de **Chickens** con recompensas acumuladas se nos activará el botón de “Withdraw ETH”

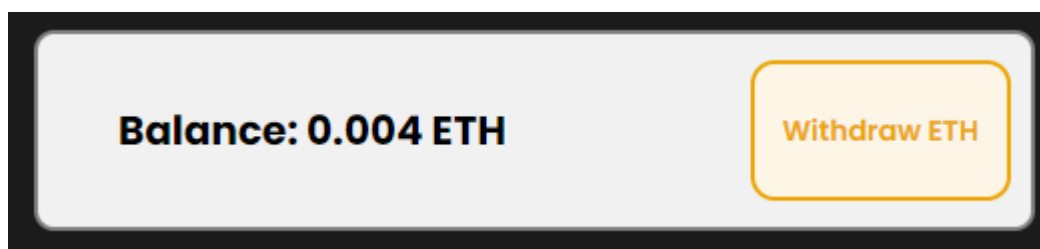


Figura A.33. Apartado de balance con recompensas acumuladas.

Si clicamos en este botón nos aparecerá una pestaña de Metamask para confirmar la transacción.

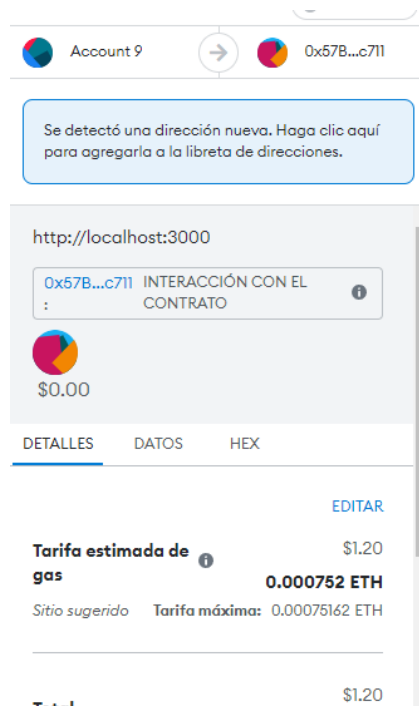


Figura A.34. Confirmación transacción Metamask.

La aceptamos, aparecerá una notificación de que se ha realizado la transacción y ese saldo acumulado se nos transferirá a nuestra wallet de metamask y el saldo acumulado dentro del contrato se volverá 0.

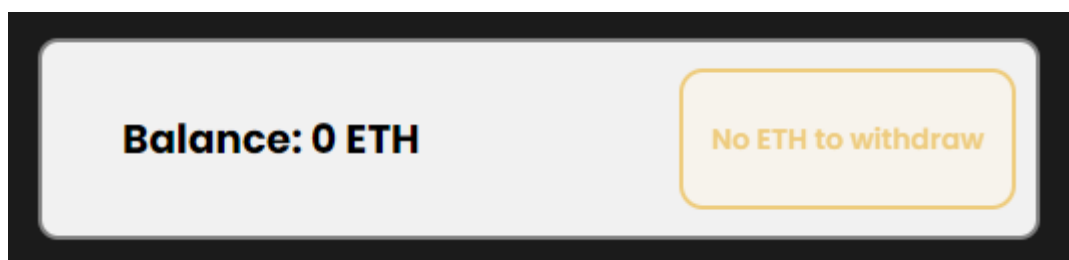


Figura A.35. Apartado de balance sin recompensas acumuladas.

A.7 Reproducción de chickens

Para acceder a esta función es necesario tener 2 chickens disponibles para reproducir y acceder a la pestaña de **Breeding** en la barra superior.

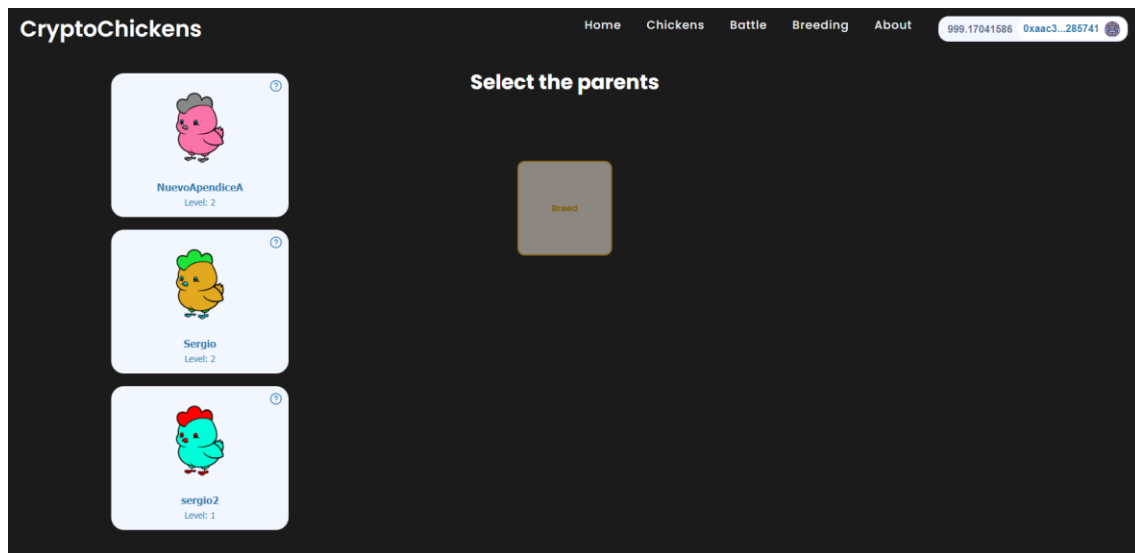


Figura A.36. Vista Breeding.

Como podemos observar en la **figura A.36.** a la izquierda se nos muestran nuestros chickens disponibles para reproducción, al igual que en la pestaña de Battle, seleccionamos uno de ellos para reproducirlo.

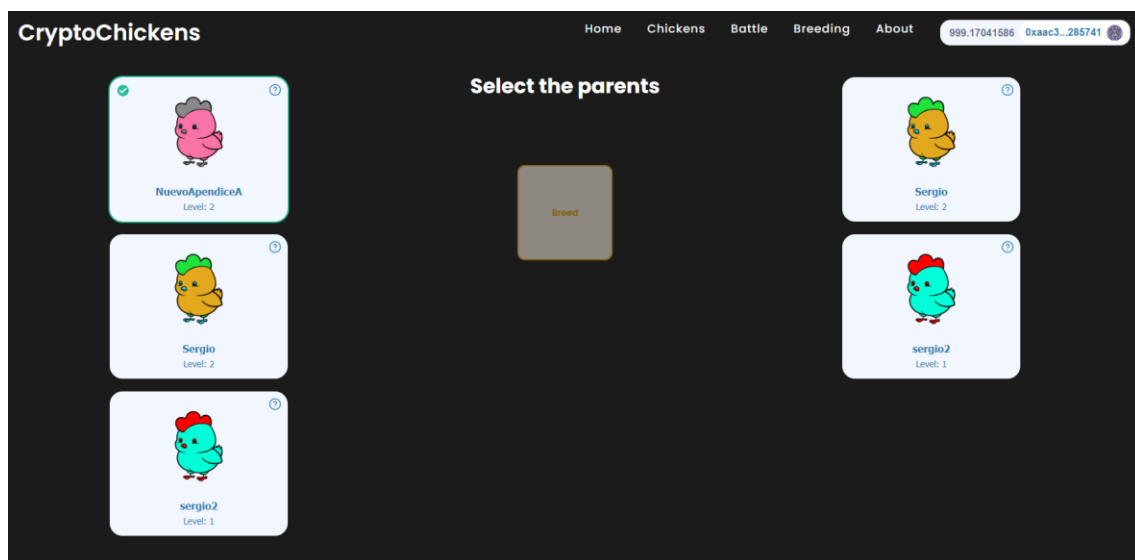


Figura A.37. Vista Breeding tras seleccionar el primer chicken.

Como podemos ver en la **figura A.37.** una vez seleccionamos un chicken, nos aparecen en el otro lado los chickens restantes para reproducir, si seleccionamos uno se nos activará el botón de “Breed”.

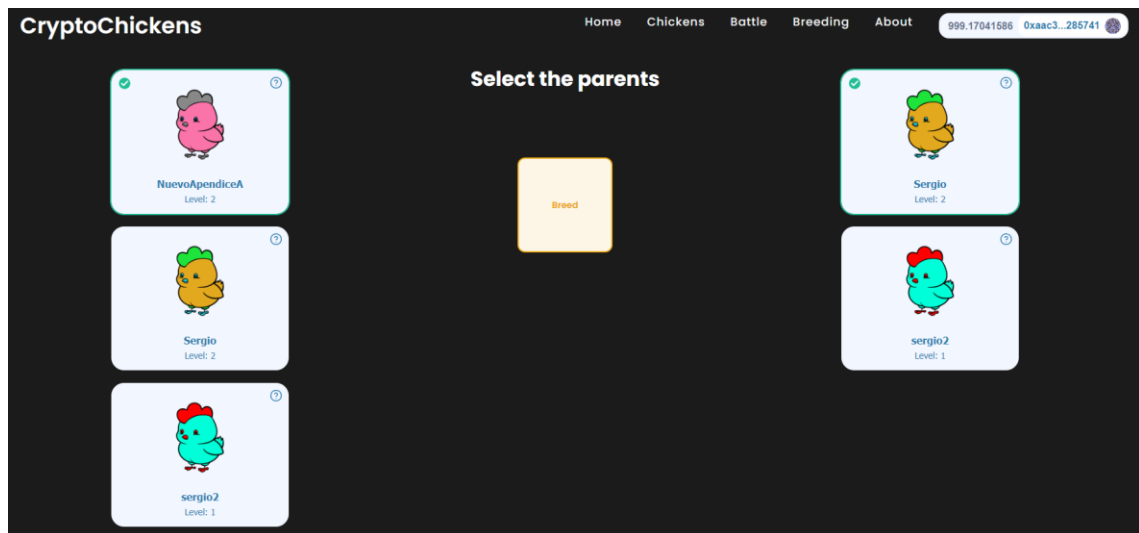


Figura A.38. Vista Breeding tras seleccionar los dos chickens.

Una vez hemos seleccionado ambos chickens, clicamos el botón “Breed” y se nos abrirá una pestaña de metamask para confirmar la transacción.

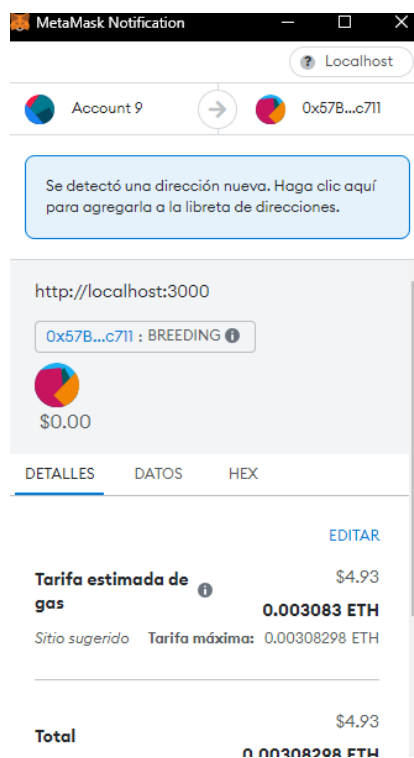


Figura A.39. Confirmación transacción Metamask.

Confirmamos la transacción, nos aparecerá una notificación de que se ha completado con éxito y se generará un huevo que se abrirá en 2 días y los chickens que se han reproducido no podrán reproducirse en los próximos 3 días. Para ver nuestro huevo accedemos a la pestaña de **Chickens**.

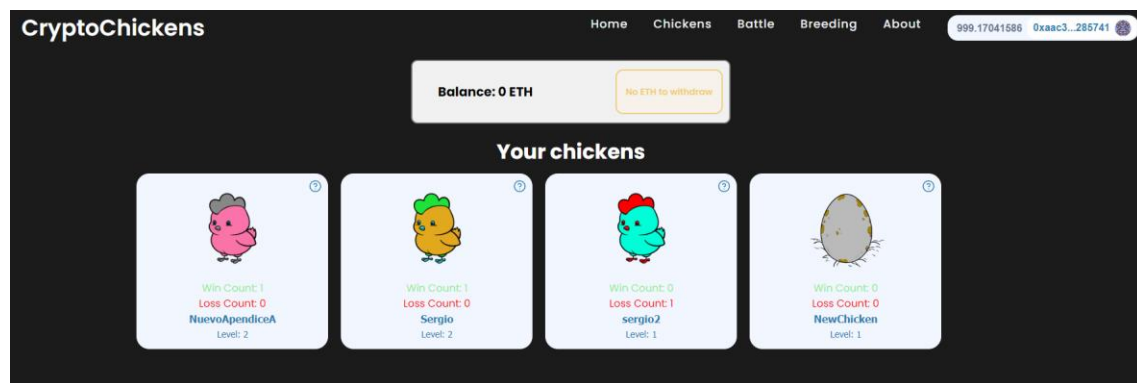


Figura A.40. Vista chickens tras haber reproducido los chickens.

Y si clicamos en el huevo podemos ver sus estadísticas en profundidad y el tiempo que le falta para abrirse.

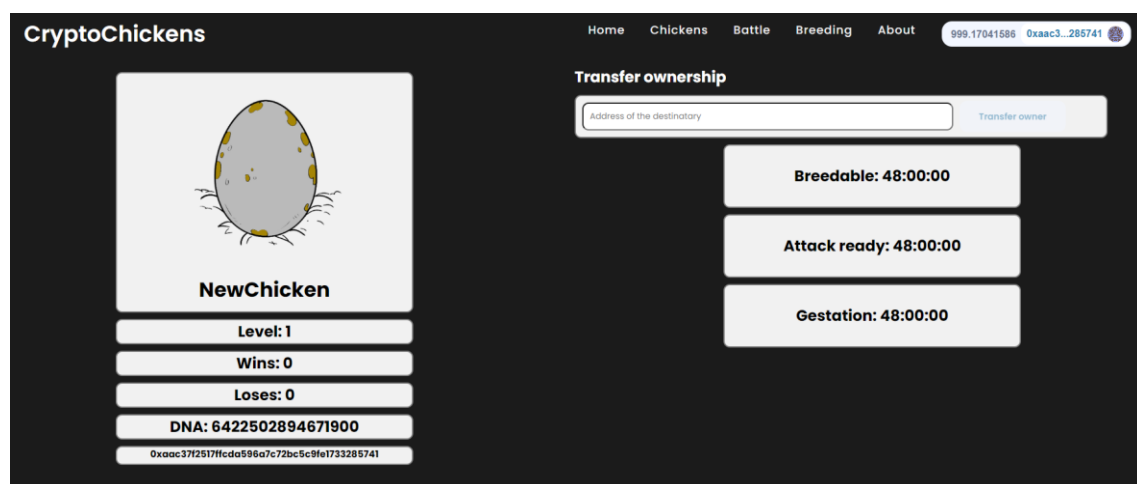


Figura A.41. Vista estadísticas del huevo.



UNIVERSIDAD
DE MÁLAGA

| **uma.es**

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga