Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

ÁREA DE ARQUITECTURA Y TECNOLOGÍA DE LOS COMPUTADORES

DISEÑO E IMPLEMENTACIÓN DE VIDEOJUEGO DE TIPO BLOCKCHAIN PARA MÓVILES

D. González Álvarez, Miguel

TUTOR: D. José Antonio Sánchez Sánchez

FECHA: Junio de 2022

Índice

Índice de figuras

Resumen

Desde hace ya varios años las aplicaciones que la tecnología blockchain tiene han cobrado mucha relevancia en diferentes ámbitos como pueden ser la seguridad o el mundo financiero.

Un campo en el que esta tecnología ha impactado con fuerza recientemente, es en el mundo de los videojuegos donde su implementación para la creación de Tokens No Fungibles (NFT) ha supuesto la aparición de un nuevo paradigma que ha influenciado la aparición de un nuevo tipo de videojuegos que se encuentran en un momento de expansión y muchas empresas ya han empezado a sacar sus propios juegos que utilizan esta tecnología.

Concretamente en este Trabajo de Fin de Grado se ha llevado a cabo una investigación sobre la literatura referente a la tecnología blockchain, el funcionamiento de la misma y las diferentes implementaciones que se han llevado a cabo de esta.

Por otro lado, también se han estudiado las ventajas y desventajas que la utilización de esta tecnología en videojuegos puede traer consigo, se muestran como ejemplos algunos de los juegos más importantes en la actualidad y se lleva a cabo un análisis de la literatura existente sobre el uso que esta tecnología ha tenido en el ámbito de los videojuegos.

Finalmente se plantea la creación de un videojuego que utiliza NFTs para representar a sus personajes y que permite llevar a cabo combates simples para obtener una recompensa final.

También se analiza de manera teórica como se podría realizar un posible despliegue en un entorno real y se desarrolla una blockchain propia de ejemplo para su incorporación en el videojuego.

Las conclusiones extraídas de esta investigación es que la tecnología blockchain está en expansión y puede llegar a ser muy relevante dentro del mundo de los videojuegos pero también hay características intrínsecas de la misma como la falta de escalabilidad de muchas blockchains que podría llegar a dinamitar la estandarización de esta tecnología y convertir su utilización en un producto residual.

Palabras clave: Cadena de bloques | Videojuegos | Contratos inteligentes | Bitcoin | Criptoactivos | Token No Fungible

Acrónimos

ADN: Ácido Desoxirribonucleico

AXS: Axie Infinity Shard

BasS: Blockchain as a Service

BSC: Binance Smart Chain

CSS: Cascading Style Sheets

DAO: Organización Autónoma Descentralizada

DOM: Modelo de Objeto de Documento

ERC: Ethereum Requests for Comments

ETH: Ethers

EVM: Máquina Virtual de Ethereum

HTML: Hyper Text Markup Language

HTTP: Hyper Text Transfer Protocol

IDE: Integrated Development Environment

JSON: JavaScript Object Notation

NFT: Non Fungible Token

PaaS: Platform as a Service

SaaS: Software as a Service

SHA-256: Secure Hash Algorithm 256

SLP: Small Love Potion

TFG: Trabajo de Fin de Grado

URL: Localizador de Recursos Uniforme

1. Introducción

En este capítulo se van a explicar las razones que han llevado a la creación de este Trabajo de Fin de Grado así como los objetivos y metodología que se han seguido durante el desarrollo del mismo.

1.1.- Objetivo.

El objetivo principal de este Trabajo Fin de Grado es el de conocer la importancia e impacto que la tecnología de cadena de bloques (blockchain) tiene en el presente y puede llegar a alcanzar en el futuro.

Para alcanzar ese objetivo, se plantean unas metas secundarias que deberán ser abordadas y que son las siguientes:

1. Analizar la tecnología blockchain y determinar su importancia e impacto en el mundo de los videojuegos.
2. Profundizar en las características que definen a los juegos blockchain más populares del mercado en la actualidad.
3. Diseñar e implementar un prototipo de un videojuego que cumpla con las características principales y requisitos que se esperarían en juego de su tipo.
4. Planificar un posible despliegue en un entorno real en el que se utilicen criptomonedas.

1.2.- Motivación.

La importancia que el blockchain ha tenido en los últimos años es innegable puesto que propició el desarrollo del ahora mundialmente conocido ***bitcoin***, pero la relevancia de esta tecnología no se limita solo al ámbito financiero, sino que las posibles aplicaciones se expanden a todo tipo de sectores. [1]

Algunos ejemplos serían campos como la educación o la sanidad donde se tienen ejemplos como la posible utilización de contratos inteligentes entre alumnos y profesores para llegar a acuerdos sobre el contenido y los plazos de entrega de las tareas o en el caso de la industria farmacéutica donde permitiría verificar la autenticidad y procedencia de los medicamentos facilitando así un sistema de control que evite que falsificaciones

lleguen al paciente final mejorando indirectamente la salud de los pacientes y reduciendo las pérdidas que las empresas farmacéuticas tienen por este tipo de medicamentos fraudulentos. [2]

La industria de los videojuegos no es una excepción a lo explicado anteriormente y la aparición de blockchain puede tener un papel muy importante para revolucionar y evolucionar la forma en que no solo se utilizan los videojuegos, sino también el enfoque que se les da a la hora de desarrollarlos.

Los videojuegos casi desde su origen han ido evolucionando al mismo tiempo que iban apareciendo innovaciones tecnológicas que abrían nuevas posibilidades. Se tienen numerosos ejemplos como el salto que se dio al pasar de utilizar 2D al 3D, la posibilidad de ejecutar juegos en la nube o los videojuegos como servicio.

Estos conceptos ya han sido abordados por grandes empresas del sector tecnológico como lo son Google con el desarrollo de Google Stadia, Nvidia con el desarrollo de Geforce Now para el caso del streaming de videojuegos en la nube o Microsoft con su Xbox Game Pass como ejemplo del concepto de videojuegos como un servicio que imita la estrategia seguida por Netflix para el mundo de las series y películas.

La pregunta que da origen a esta investigación es "¿Se extenderá la tecnología blockchain dentro del mundo de los videojuegos?" y para poder darle respuesta es que se ha realizado este trabajo de investigación.

1.3.- Metodología.

Este trabajo se ha llevado a cabo siguiendo una metodología de investigación en la cual primero se realizó un estudio de la tecnología blockchain en general para conocer las bases teóricas e históricas que llevaron a la adopción y crecimiento que actualmente tiene blockchain.

En segundo lugar, se realizó un estudio de como repercutió esta innovación sobre el mundo de los videojuegos y como están diseñados los principales juegos que emplean esta tecnología.

El tercer paso fue la creación de un videojuego que emplease estas técnicas para ejemplificar mejor como se desarrollan este tipo de videojuegos. Al tratarse de un trabajo eminentemente de investigación y no de desarrollo de software, no se propuso un diseño y desarrollo formal desde el punto de vista de la ingeniería de software.

El último paso consistió en analizar cómo realizar un posible despliegue de dicho videojuego y vislumbrar las posibles vías de trabajo futuro en el campo de los videojuegos tipo blockchain.

1.4.- Planificación del trabajo.

Este trabajo se ha planificado para completarse en un periodo de 8 meses a lo largo de los cuales se han abordado tareas que permiten alcanzar los objetivos propuestos en los puntos anteriores de este capítulo.

El diagrama de Gantt resultante de la planificación de estas tareas se puede ver en la Figura *1.1*.

Figura 1.1.- Diagrama de Gantt

2. Introducción a la tecnología blockchain

Las bases de la cadena de bloques (blockchain) nacen entre finales de los años 80 y principios de los 90 pero esto se produjo solamente de forma teórica y no fue hasta el año 2008 que fue publicado el documento titulado "Bitcoin: A peer to peer Electronic Cash System" donde una persona o grupo de personas bajo el seudónimo de Satoshi Nakamoto establecían los pilares para la creación de una red de intercambio de dinero entre personas sin necesidad de ninguna autoridad central y sustentando las bases del sistema sobre la tecnología blockchain [1].

2.1.- ¿Qué es blockchain?

Una forma rápida de describirlo sería que es como un registro de información distribuido en el que todos los participantes son "iguales" entre sí por lo que en este tipo de arquitectura no se tendrían diferentes roles tipo servidor/cliente, sino que todos los miembros que conforman la red son al mismo tiempo servidores y clientes al mismo tiempo [3].

Para entenderlo más fácilmente estos conceptos se tiene el ejemplo de U Torrent en el que todas las personas que utilizan la aplicación se descargan archivos, pero al mismo tiempo los suben y transmiten ya que no existe una entidad centralizada que contenga todos los archivos, sino que cada miembro posee "trozos" de los diferentes archivos y los comparte con todos los demás miembros. [4]

En el caso de blockchain esos trozos de información se agrupan y se les conoce como bloques los cuales están ordenados cronológicamente y además poseen un código alfanumérico conocido como hash y están firmados digitalmente por la persona que encontró el código.

Otra característica muy importante es la "inmutabilidad de la cadena de bloques" lo que significa que en blockchain es imposible editar o borrar la información ya existente, esto se consigue sin la intervención de ningún servidor central que verifique que nadie está intentando modificar la información existente, sino que son los propios usuarios los que se encargan de monitorear y verificar cada bloque haciendo imposible la alteración de la información almacenada.

Además de las características antes nombradas existe también un protocolo de consenso entre los miembros de la red conocido como *Proof of Work* (prueba de trabajo), en el que se profundizará más adelante en este trabajo, el cual determina que código será el usado como hash para el bloque que se quiere crear.

Volviendo al terreno financiero que popularizó esta tecnología se puede resumir lo antes explicado en que este sistema funciona como una contabilidad pública en la que todos tienen el historial de intercambios que se han ido realizando y en caso de que alguien intente justificar que una transacción no existió o que esta fue de un importe diferente no podría hacerlo porque todos los demás miembros tienen una copia del historial completo y pueden comprobar cuál fue la transacción real.

2.2.- Tipos de blockchain.

Actualmente las redes blockchain pueden clasificarse de forma general en públicas, privadas o federadas teniendo cada tipo una arquitectura y propiedades características de cada agrupación [3], [5].

2.2.1.- Blockchain públicas.

Las redes públicas se caracterizan por ofrecer una máxima libertad ya que cualquiera podría unirse a este tipo de red y pasar a ser un nodo totalmente valido y funcional. Utilizando un consenso público se consigue que la toma decisiones no se vuelva caótica (este tipo sería el más conocido por el público general y por ejemplo Ethereum o Bitcoin pertenecerían a este grupo) y con esto se puede maximizar tanto la transparencia como la descentralización de la red, aunque a costa de una gran ineficiencia en términos de eficiencia eléctrica y de utilización de capacidad de cómputo debido a las *Proof of Work*. [6]

Para motivar a los usuarios a que realicen la validación de los bloques existe el concepto de recompensa por realizar esta tarea entregándole al nodo que realiza la validación una cantidad de la propia criptomoneda.

En resumen, este tipo de blockchain apuesta por la transparencia, la descentralización y por permitir un gran número de usuarios los cuales están en igualdad de condiciones dentro de la red.

2.2.2.- Blockchain privadas.

Las redes privadas serian aquellas que tienen como requisito de entrada el recibir una invitación de algún miembro fundador de la red y su funcionamiento está gobernado de acuerdo a un grupo de normas determinadas por las personas que crean la red.

La diferencia fundamental con las redes públicas es que en las privadas existe la figura del administrador que es un usuario o grupo de usuarios que tiene el privilegio de determinar las normas que se aplican.

Normalmente estas redes se utilizan para que solo pequeños grupos de usuarios de todos los que conforman la red comercien entre sí. En caso de permitir que todos los miembros realicen transacciones seguiría siendo un pequeño grupo el encargado de establecer las reglas a seguir o de validar las transacciones que se realizan dentro de la red.

El ejemplo más conocido de este tipo de blockchain seria *Hyperledger* que es un proyecto de la fundación Linux de código abierto que sirve como base para la creación de muchos proyectos que utilizan la tecnología de blockchains privadas.

En resumen, este tipo de blockchain sacrifica la transparencia y la descentralización a cambio de obtener un mayor control sobre la propia cadena de bloques y sobre el uso que los usuarios pueden hacer de esta.

2.2.3.- Blockchain federadas.

Las redes federadas comparten algunas características de los dos tipos anteriores teniendo en común con las privadas que solamente se pueden unir usuarios mediante una invitación, pero en este caso todas las personas tendrán derecho a tomar decisiones por consenso como si fuese una red pública. Además, son varias organizaciones las que rigen las normas y el mantenimiento de la cadena mientras que en la privada solamente hay una única organización.

El ejemplo más conocido de este tipo de redes sería *Alastria* la cual utiliza dos redes operativas conocidas como red T y red B estando la primera basada en *Ethereum* y la segunda en *Hyperledger*.

En resumen, este tipo de blockchain obtiene la descentralización de las blockchain públicas, pero con una transparencia y uso de recursos intermedio entre una pública y una privada.

2.3.- BaaS (Blockchain as a Service).

Además de los tres tipos anteriores existe un concepto conocido como BaaS que se basa en que sea una compañía la que, utilizando la nube, ofrezca el almacenamiento de la información de la blockchain y evitando la necesidad de utilizar hardware potente ya que serían los servidores de esa compañía los encargados de tareas como el mantenimiento de la cadena [7].

Al igual que ocurre con el SaaS (*Software as a Service*), el BaaS permite a las empresas poner en funcionamiento aplicaciones de manera rápida y sencilla permitiendo una adopción rápida adopción de la tecnología de bloques.

Actualmente la aplicación de blockchain es algo muy demandado por todo tipo de empresas pero aquellas de menor tamaño probablemente no tienen ni los medios físicos ni los conocimientos necesarios para la configuración de las aplicaciones que utilizan esta tecnología y es aquí donde el BasS gana gran relevancia ya que además de ofrecer la potencia y almacenamiento de la nube también simplifica mucho la creación de aplicaciones y algunos proveedores incluso ofrecen asistencia profesional a aquellos que contraten sus servicios.

2.4.- Como se realiza una transacción.

Para ejemplificar como se realizaría una transacción utilizando tecnología blockchain se ha tomado el caso de bitcoin porque fue pionera en la creación de este tipo de transferencias y contiene todos los elementos básicos necesarios [7].

2.4.1.- Wallets and addresses.

El primer paso a tener en cuenta sería la creación, en caso de no tenerla previamente, de las carteras (*wallets*) y las direcciones (*addresses*).

Las primeras funcionan como ficheros que proporcionan acceso a múltiples direcciones de Bitcoin, es decir, una *wallet* no está asociado a una única *address*. Las direcciones o addresses son cada una de las cadenas de letras y números que identifica su propio balance de bitcoin [8].

Esto se puede entender como que las direcciones serian cuentas bancarias cada una con su propio balance de ingresos y gastos mientras que una *wallet* sería la propia entidad bancaria, pero hay que matizar una diferencia importante y es que, aunque el propio sistema no obliga a que cada transacción acabe resultando en la creación de una nueva address, esto acaba siendo algo muy común para incrementar la privacidad.

2.4.2.- Creación de una dirección.

El segundo paso es la creación de una nueva dirección de bitcoin lo cual acarrea la formación de dos llaves, una publica y una privada. La privada quedará almacenada en el *wallet* del usuario que creo la dirección mientras que la publica será conocida por todos los usuarios de la red.

Siguiendo los conceptos básicos de la criptografía lo que se hace es que la persona que quiere realizar la transacción la firmara con la clave privada que se generó previamente por lo que se garantiza que esa transacción es de esa persona.

Cuando alguien quiera enviar información a este usuario tendrá que encriptar con la clave publica del usuario la cual todo el mundo conoce, pero solo el usuario podrá descifrar la información.

Por otra parte, si el usuario es quien quiere enviar la información al resto de la red entonces el encriptará con su clave privada lo que permitirá que cualquiera pueda descifrarlo usando la clave publica, pero garantizando que la información proviene del usuario en cuestión.

2.4.3.- Petición de transacción.

El tercer paso, sería la realización del pago para lo cual se tomará un ejemplo llamando “usuario A” al usuario que quiere recibir el pago mientras que el usuario que va a enviar la transacción será llamado “usuario B”.

Como ya se pudo comprobar en el paso anterior, los usuarios pueden crear direcciones y en este caso es el usuario A quien creará una nueva dirección de bitcoin la cual quedará firmada con la clave privada A [5].

Después de esto el usuario B formará una petición de transacción que firmará con la clave privada de la dirección (de las varias que puede tener en su *wallet*) de la cual quiere extraer los bitcoins para pasarlos a la dirección que el usuario A creo.

Las claves públicas de las dos direcciones implicadas son accesibles por cualquier usuario y como la clave privada solo la tienen los usuarios esto garantiza que cualquier miembro de la red pueda verificar que fue el usuario B quien hizo la petición (descifrando con la clave pública de esa dirección de B).

2.4.4.- Verificación de la transacción y minado.

El cuarto paso sería que el resto de usuarios de la red, normalmente los conocidos como mineros validen que la transacción es correcta. Esto se realiza mediante un protocolo de consenso conocido como *Proof of work* y cada 10 minutos se genera un nuevo bloque en la cadena [8].

Los mineros lo que tendrán que hacer es generar un hash criptográfico que permita transformar una información (*data*) que se sumista como entrada (*input*) en una cadena de información de una longitud determinada.

Para la creación de este nuevo hash se tiene que usar los anteriores hashes de la cadena combinado con el nuevo bloque de la transacción y con un *nonce*.

El minero tendrá que encontrar el *nonce* para el cual se genere un *hash* que empiece por un número determinado de ceros y como es imposible predecir que hash se obtendrá a partir de una data concreta pues lo que se hace es probar todas las posibles combinaciones hasta encontrar aquel nonce que genere el hash que empieza por el número adecuado de ceros.

Gracias a esta forma de crear los nuevos hashes es que la cadena mantiene su inmutabilidad puesto que si alguien altera un valor de una transacción anterior estaría alterando ese *hash* y con ello todos los *hashes* posteriores serian cambiados en cadena por lo que serían radicalmente distintos.

Cuando un minero al que se ha nombrado como “usuario C” encuentra el *hash* adecuado este adquiere el derecho de añadir la primera transacción del bloque lo que se conoce como “coinbase transaction”.

Entonces se crea una nueva dirección en la *wallet* del usuario C y se le transfieren una cantidad variable de bitcoin la cual originalmente era de 50 bitcoin pero que se reduce a la mitad cada vez que se crean 210.000 bloques aproximadamente, causando que a fecha de la creación de este documento la recompensa sea de 6.25 bitcoin por bloque lo cual a

precio actual del bitcoin (ya que este también es muy variable) son aproximadamente unos 400.000 dólares.

2.4.5.- Inmutabilidad de la transacción.

El último paso sería que el usuario C notifique al resto de nodos que ha encontrado la combinación adecuada para que estos validen si es correcto el *hash* que ha encontrado antes de añadirlo finalmente a la cadena.

Para poder ser añadido deberá ser validado por al menos la mitad más uno de todos los nodos que están participando.

De esta forma se garantiza la autenticidad de la cadena ya que si algún atacante quisiera modificar el valor de una transacción no solo tendría que rehacer todo el trabajo que el usuario C llevó a cabo para encontrar el "*nounce* ganador" (número aleatorio que se añade para que a partir de unos mismos datos se generen diferentes hashes) sino que además tendría que repetir todo el trabajo de todos los mineros que han venido tras el usuario C puesto que los bloques de todos ellos estaban utilizando para sus nuevos *hashes* el *hash* que el usuario C había creado, lo cual es casi imposible.

2.5.- Características más importantes.

En este punto se van a estudiar las características que diferencian a blockchain y le han permitido convertirse en una tecnología cuyo uso se ha extendido mundialmente durante los últimos años.

2.5.1.- Confianza.

En las bases de datos que tradicionalmente se utilizan se asume que ninguno de los nodos va a introducir información falsa pero la idea revolucionaria es el protocolo de consenso que hace que los nodos puedan compartir información sin necesidad de confiar en otros nodos de tal forma que si algún nodo intenta introducir datos falsos sean el resto de nodos los que rechacen ese intento [9].

Los mineros que como ya se explicó son quienes tienen el rol de crear nuevos bloques y de verificar los nuevos que se añaden a la cadena por lo que no es necesario confiar en un nodo ya que, aunque este intente introducir información falsa además de eso tendría que ser capaz de que la mitad más uno de todos los nodos valide su bloque no veraz.

2.5.2.- Inmutabilidad.

La cadena de bloques no puede ser alterada de forma deshonesta por ningún nodo gracias al protocolo de consenso que hace que se necesite que la menos la mitad más uno de los nodos de la red valide un bloque para que este pueda ser añadido.

2.5.3.- Privacidad.

La privacidad en blockchain se garantiza la privacidad gracia a utilizar criptografía de clave asimétrica en lugar de simétrica.

En la criptografía simétrica se producía el problema de que no se puede conseguir el "no repudio en origen" que consiste en que no se puede identificar de donde viene la información de forma única y veraz cosa que no pasa en criptografía asimétrica por poseer dos claves ligadas matemáticamente entre si en vez tener una única clave.

Ahora ejemplificare como gracias a estas dos claves se garantiza la seguridad y para eso se volverá a ejemplificar tomando a un "usuario A" y un "usuario B".

El primer caso que se puede tener es que el usuario A le quiere mandar un mensaje al usuario B que solo quiere que el usuario B pueda leer entonces lo que hará será cifrar el mensaje con la clave publica de B garantizando que solo será el usuario B quien pueda leerlo pues nadie más posee la clave privada B.

El otro caso posible es que A quiera enviar un mensaje que se garantice que fue escrito por el por lo que lo cifrara con su clave privada y aunque cualquiera podrá leerlo descifrando con la clave publica de A se garantiza a todo el que lo lea que el origen del mensaje es el usuario A.

2.5.4.- Transparencia.

Utilizando blockchain se puede crear un almacenamiento de datos descentralizado y altamente transparente es decir que cualquier transacción entre dos usuarios es visible para el resto de usuarios.

La cadena de bloques se actualiza de manera constante y todos los participantes de la red tiene acceso a la cadena valida. Esto ofrece la posibilidad de tener una alta trazabilidad de todas las transferencias que se llevan a cabo.

De esta forma si un usuario A le pasa a un usuario B una cantidad de bitcoin esto puede ser visto por todos los miembros de la red lo cual proporciona una transparencia casi absoluta.

2.5.5.- Anonimato.

El anonimato en blockchain está garantizado porque toda la información de un miembro de la red como su nombre o su correo electrónico se guardan en forma de *hash* criptográfico.

Como es casi imposible descifrar este *hash* sin conocer el valor de la clave privada la cual solo está en conocimiento del participante que se unió a la red se puede asegurar la privacidad de la información de los usuarios.

Este valor *hash* es también publico para el resto de miembros de la cadena, aunque sí identifica a un usuario lo hace manteniendo ocultos los datos personales del mismo y la única forma posible en que la información puede quedar expuesta es si el propio usuario entrega la clave privada que utilizaste para crear el *hash* o se la roban por otros medios ajenos al sistema blockchain.

2.5.6.- Alta disponibilidad.

Al contrario que en un modelo cliente/servidor en blockchain se tiene una red de nodos distribuida por lo que todos son a la vez cliente y servidor.

Esto permite que si en algún momento uno de los nodos dejase de estar operativo esto no desencadenaría en la caída de toda la red ya que el resto de nodos podrían seguir funcionando entre si sin ningún problema evitando así no solo la perdida de la cadena sino también haciendo que esta siga operativa.

Por ejemplo, en un modelo cliente/servidor si alguien borrase la cadena del servidor esta se podría perder y dejar de ser accesible mientras con una red *peer to peer* (red entre pares) como blockchain la cadena seguiría intacta en el resto de nodos.

2.6.- Protocolos de consenso.

En este punto se van a estudiar los posibles protocolos que se pueden aplicar cuando se desarrolla una blockchain así como las características que diferencian a cada uno de ellos.

2.6.1.- Proof of work.

Existen un total de 16^{64} valores de hash criptográficos posibles para la función SHA-256 (*Secure Hash Algorithm 256*), pero no todos ellos son válidos ya que cada dos semanas la red de bitcoin define un objetivo mínimo para el hash el cual es el número de ceros por el cual debe iniciarse el *hash*. Para explicar la proporción de valores validos se ha tomado como ejemplo el caso de tener un objetivo de conseguir que el *hash* se inicie con 10 ceros a la izquierda.[10]

Como se está trabajando en hexadecimal el tener el número de ceros que se han marcado a la izquierda supone que de los 16^{64} posibles valores solamente nos quedarían libres para modificar 16^{54} ya que cada cero reduce la magnitud con un factor de 16. El número de *hashes* validos sería de 16^{54} lo que supone una probabilidad de encontrar un hash valido de 16^{-10} como resultado de aplicar la siguiente formula:

$$Nº\ hash\ validos / Nº\ total\ de\ hash = 16^{64 - Objetivo} / 16^{64} \qquad (2.1)$$

Sin embargo, hay que tener en cuenta que podría darse el caso de que agotando todos los posibles valores aun así no se consiga obtener un *hash* válido y es por eso que se introdujo otro campo más a la estructura del bloque, el *timestamp* o marca de tiempo que representa el número de segundos transcurridos desde el 1 de enero de 1960.

Como la marca de tiempo tiene un valor que varía de manera constante con el tiempo por lo que si un minero agotase todas las entradas posibles lo que tendrá que hacer es esperar a que cambie la marca de tiempo para tener otros 16^{54} posibles valores.

Aun con todo esto se sigue planteando otro problema y es que, aunque para un minero promedio el cual es incapaz de agotar todos los posibles valores para un *timestamp* en menos de un segundo, esto no sucede cuando se tienen grandes grupos que trabajan conjuntamente (las conocidas como “granjas de minado”).

2.6.2.- Mempool.

Como ya se comentó anteriormente solo se añade un bloque nuevo a la cadena cada 10 minutos, pero no todas las transacciones que se quieren realizar pueden entrar en el mismo bloque.

Los mineros pueden elegir que transacciones irán al siguiente bloque tomándolas del *mempool* dándoles la posibilidad de alterar de nuevo los valores de *hash* ya que estarían cambiando datos de entrada de la función sha-256.

En caso de que un minero se quedase sin combinaciones para un *timestamp* concreto y unas transacciones pues lo que puede hacer es cambiar las transacciones que quiere añadir al bloque evitando así tener tiempos de inactividad.

2.6.3.- Proof of stake.

En este algoritmo de consenso la decisión sobre que nodo debe minar un bloque se hace de forma aleatoria, pero con diferente proporción en función a factores como la tenencia de monedas. Este protocolo de consenso nació para intentar solucionar varios problemas que afectan a *proof of work*.

El primero de estos problemas era la falta de velocidad y de escalabilidad ya que durante el minado se añade latencia para la aprobación de transacciones y poder así crear nuevos bloques, pero *en proof of stake* las verificaciones son realizadas por nodos que sean grandes tenedores de la criptomoneda, de esta manera se logra que las verificaciones se hagan de forma mucho más rápida.

El segundo problema sería el gran consumo energético que conlleva la forma en que se realiza el minado de bloques, esto en *proof of stake* se sustituye por un proceso de participación reflejada en la tenencia de monedas.

El tercer problema sería la centralización del poder de cómputo de la red en unos pocos miembros el cual es bastante común en *proof of work* como se puede apreciar con los grandes grupos mineros existentes [10].

El cuarto problema sería los “ataques del 51%”, en proof of stake lo que se busca es rebajar el interés financiero que alguien podría tener en realizar este tipo de ataques. En el caso de *proof of work* un grupo minero podría tener el 51% de todo el poder de cómputo y podría modificar la cadena (a pesar de que la probabilidad de que algo así ocurra es remotamente pequeña).

En *proof of stake* esto solo sería posible si el grupo de atacantes tiene el 51% de todas las monedas lo cual no solo requiere un esfuerzo económico muy elevado, sino que en caso de llevar a cabo el ataque el valor de la moneda tendera a caer ya que pasaría a ser muy insegura.

Cabe destacar que en este protocolo la cantidad de moneda no es el único parámetro posible para decidir que probabilidad se tiene de ser elegido nodo validador, sino que se pueden usar otros como el tiempo desde que se crearon esas monedas o el tiempo que lleva el minero siendo parte de la red.

2.6.4.- Proof of burn.

Este protocolo de consenso se basa en que los usuarios deben "quemar" una determinada cantidad de criptomoneda para poder minar bloques. Lo que se hace es crear aleatoriamente direcciones publicas llamadas direcciones devoradoras (*eater addresses*) para las cuales nadie tiene una clave privada. Cuando algún miembro quiere obtener la posibilidad de minar un bloque deberá primero demostrar su compromiso con la red enviando una cantidad de criptomoneda (ya sea la de la propia cadena o de otras criptomonedas) y cuanto mayor sea la cantidad que envía mayor será también el espacio que se le deje minar aumentando así la cantidad que sería posible obtener.

Con este protocolo se consigue que los mineros estén realmente comprometidos con la moneda ya que esto puede ser entendido como que se le exige una "inversión inicial" y al igual que en cualquier inversión se corre el riesgo de que las acciones pierdan valor.

Los principales puntos débiles de este modelo serian por un lado la posibilidad de utilizar otras criptomonedas para alimentar a las eater addresses haciendo que al final se consuma mucha energía puesto que si por ejemplo se paga en bitcoins para haberlos generado utilizando *proof of work* (el protocolo usando en bitcoin) y por otro la necesidad de tener confianza en la criptomoneda que use este protocolo haciendo que el riesgo sea más alto que con otros protocolos.

2.6.5.- Proof of space.

Este protocolo sería similar al *proof of work*, pero cambiando la capacidad de cómputo por la de almacenamiento. En lugar de exigírsele al minero que calcule una función de *hash* se le exige que almacene una cantidad de información.

Al minero se le envía una colección de *hashes* criptográficos que deberá almacenar en su ordenador. Cuando la red envía el reto para conseguir el bloque el minero revisa los hashes que tiene almacenados para ver si alguno coincide con el del reto

haciendo que cuanto mayor sea la cantidad almacenada por el usuario mayor posibilidades tiene este de ganar el reto.

Un ejemplo de redes que implementan este tipo de protocolo de consenso seria *Chía Network* la cual fue creada en agosto de 2017. [11], [12]

2.6.6.- Delegated proof of stake.

Es una variante del *proof of stake* en la que en lugar de ser el nodo elegido el que valide lo que se hace es darle la posibilidad de elegir un delegado o grupo de ellos que serán los encargados de llegar a un consenso entre ellos.

Estos delegados son elegidos en base a su buena reputación y pueden ser sustituidos en cualquier momento mediante el voto, aunque la importancia del voto de un usuario estará siempre ligada a la cantidad de moneda que posea.

La principal ventaja es que la elección de los delegados suele producirse de forma muy rápida lo que permite una mayor escalabilidad de la cadena de bloques, pero al mismo tiempo genera una debilidad importante y es que aquellos usuarios que poseen una gran cantidad de moneda podrían ser los que siempre tomen la decisión de que delegados actúen.

A pesar de esto, la mayor parte de las criptomonedas que utilizan este protocolo como por ejemplo *BitShare* el cual tiene sus propios métodos internos para evitar la centralización del voto en pocas manos.

2.7.- Contratos Inteligentes.

Los contratos inteligentes (*Smart Contracts*) son una serie de scripts que se ejecutan cuando se cumplen unas determinadas condiciones. Este código es público y podrá ser visualizado por todas las partes que intervengan en ese contrato.

Un contrato inteligente puede ejecutarse automáticamente cuando se cumplan las condiciones previamente establecidas ganando así validez sin necesidad de autoridades o intermediarios. Además, una vez creados no pueden modificarse.

Esto tiene muchas utilidades como por ejemplo en caso de herencias se puede hacer un *Smart contract* para que cuando la persona sea inscrita en el registro civil como fallecida automáticamente reparta entre sus herederos la parte que se haya decidido

previamente facilitando así los tramites y sin la necesidad de ningún notario o administración.

Para poder realizar tareas como la antes descrita los *Smart contract* se apoyan en la utilización de "oráculos" que serían programas recopiladores de información externa a la propia cadena pero que puede ser requerida por el contrato inteligente para que este pueda comprobar que la condición de su ejecución se ha cumplido. Por ejemplo, en el caso antes nombrado sería el oráculo el encargado de conectarse a la base de datos del registro civil para comprobar si la persona ha fallecido o no.

No siempre un *Smart contract* tiene un único oráculo ya que existen redes como *ChainLink* que utiliza sus propios *Smart contracts* para poner en contacto a otros contratos inteligentes con los oráculos que necesitan para llevar a cabo su función.

2.8.- Token No Fungible.

Los Token No Fungibles o más conocidos como NFT (*Non Fungible Tokens*) están íntimamente relacionados con las criptomonedas, aunque no deben ser confundidos ya que existen notables diferencias tanto a nivel conceptual como en el uso real que se les da.

Una criptomoneda al igual que el dinero por decreto es un bien fungible lo que quiere decir que puede ser reemplazado por otro igual sin que se produzca una pérdida de valor [13]–[15].

Por otra parte, un bien no fungible clásico sería el arte ya que es muy difícil encontrar dos obras exactamente iguales y cuyo valor se pueda considerar igual ya que siempre existe un cierto componente de subjetividad.

Las diferencias principales entre un NFT y una criptomoneda vienen a ser las mismas que entre el dinero y una obra de arte, el token no fungible es único y no existe ningún otro igual a él mientras que una unidad de bitcoin podría ser reemplazada por otra unidad de la misma moneda.

Los NFT son un derivado de los *Smart Contracts* de *Ethereum* y permiten identificar al creador de un contenido o al propietario del mismo abriendo la posibilidad de que cualquier persona sea poseedora de activos digitales de cualquier tipo identificándolos de forma única.

Lo que hace único a un NFT es que este siempre está respaldado por la cadena de bloques y tiene una serie de metadatos que lo hacen único y al estar dentro de la cadena de bloques estos no pueden ser modificado.

Gracias a estos metadatos se puede saber cuántos propietarios ha tenido un NFT y cuál era su valor por el que fue adquirido la primera vez que se vendió permitiendo rastrearlo.

La creación de un token no fungible se basa en subir un archivo a una plataforma de subasta como pueden ser *Rarible*, *OpenSea* o *KnowOrigin* para que se registre dentro de una cadena de bloques para proceder a su posterior compra-venta.[1] [2] [3]

Existen muchos ejemplos de NFT como pueden ser el "*Everydays: The first 5000 days*" que representa los primeros 5000 días de trabajo del artista Beeple o el de Tim Berners-Lee que está basado en un video que muestra cómo se escribe el código del primer navegador que se creó.[4]

Sin embargo, es importante destacar que la propiedad de un NFT no da a su dueño los derechos de autor sobre la misma y por lo tanto pueden existir diferentes NFT que estén respaldados por un mismo archivo. Esto genera un problema claro, es posible que cualquier persona cree un NFT de un archivo, aunque este no tenga los derechos del mismo dando así lugar a posibles fraudes relacionados con esta tecnología [14], [16], [17].

Esto puede resultar contradictorio cuando se entiende la naturaleza de los tokens no fungibles ya que, ¿cómo es posible subir un archivo del que no se sea autor sin que todo quedase registrado en la cadena de bloques?

El punto clave es, sí que se puede rastrear, pero las tecnologías, así como la forma de hacerlo no son muy intuitivas y muchas personas no están familiarizadas con su uso. Además, muchas de las plataformas para comprar y vender este tipo de activos no se hacen cargo de verificar que el usuario sea el autor real y evitan cualquier posible responsabilidad que el usuario final pueda tener.

[1] https://opensea.io/explore-collections
[2] https://rarible.com/explore/all
[3] https://knownorigin.io/marketplace
[4] https://www.niusdiario.es/vida/visto-oido/video-viral-mas-visto-youtube-se-vende-nft-760-mil-dolares_18_3142920145.html

Es por problemas como este que los NFT al igual que la tecnología blockchain, así como las criptomonedas generan siempre mucha polémica y posiciones enfrentadas entre sus férreos detractores y defensores.

3. Introducción a los videojuegos tipo blockchain

Al tratarse blockchain de una plataforma revolucionaria para muchísimos sectores, el mundo de los videojuegos no iba a ser la excepción y en la aplicación de esta tecnología se pueden generar nuevas posibilidades que antes eran inconcebibles, así como funcionalidades novedosas para el mundo de los videojuegos.

Por una parte, esta tecnología proporciona la habilidad de intercambiar activos digitales entre juegos y por otro lado ofrece la capacidad de intercambiar esos mismos activos sin que ningún desarrollador deba estar implicado directamente.

Con estas dos características se pueden constituir nuevas experiencias y mercados internos dentro de los videojuegos dejando que tanto desarrolladores como jugadores experimenten una mayor conexión con la realidad ya que realmente sus acciones pueden generar un resultado en la realidad.

La esencia de los videojuegos siempre ha sido el entretenimiento el cual se logra a través de diferentes medios como una jugabilidad (*gameplay*) novedoso, una historia atrapante, un aspecto grafico que maraville al jugador o una combinación de las anteriores, esto se aplica también a los juegos blockchain.

Aquellos que tendrán éxito serán los que puedan mantener un buen equilibrio entre el nuevo modelo de negocio que plantea la utilización de esta tecnología y una calidad adecuada en uno o varios de los apartados antes mencionados. [5]

Sin duda se trata de una revolución que puede llegar marcar un antes y un después en el mercado de los videojuegos digitales, aportando descentralización y transparencia sobre lo que sucede en estos juegos a los jugadores y a los propios creadores por igual.[6]

No obstante, primero se deberá poner solución a una serie de problemas, como las limitaciones en la escalabilidad de las redes de bloques o la velocidad a la que se realizan las transacciones.

[5] https://telos.fundaciontelefonica.com/la-cofa/blockchain-y-videojuegos-una-relacion-prometedora/
[6] https://www.adlatina.com/marketing/qu%C3%A9-ventajas-y-qu%C3%A9-desventajas-tiene-el-blockchain-gaming

3.1.- Definición de juego blockchain.

Se puede definir a un videojuego blockchain como aquel videojuego que se ejecuta de manera parcial o total sobre alguna cadena de bloques y que se beneficia del uso de esta tecnología para ofrecer nuevas funcionalidades como por ejemplo su utilización como posible método de pago o como recompensa para los jugadores [18].

En este tipo de juegos el usuario queda convertido en un nodo más de la red blockchain y su actividad dentro del videojuego termina traducida a transacciones que se realizan con otros nodos de la red.

Por ejemplo, la forma en que Axie Infinity[7], el cual es uno de los juegos más populares de este tipo y en el que se profundizara más adelante, maneja su jugabilidad ya que en este juego en función de los *axies* que el usuario haya usado en combate y dependiendo del resultado de la batalla esto termina convirtiéndose en transacciones positivas o negativas dirigidas hacia direcciones de la red *Ethereum*[8].

Por una parte, como se ha visto en apartados anteriores, blockchain es una herramienta para garantizar la propiedad de los activos digitales de cada jugador, así como validar y verificar su transacción.

Esto también puede ser empleado para evitar que los jugadores hagan modificaciones ilegales para obtener ventajas dentro del juego y para que el mismo funcione sin cambios no previstos, impidiendo así la manipulación del juego gracias a la transparencia que proporciona blockchain.

Por último, diferentes videojuegos creados sobre una misma blockchain pueden establecer conexiones entre sí que no serían posibles de otra manera y da solución a un importante problema que se da en algunos juegos.

Por ejemplo, cuando sale la secuela de un videojuego que incorpora obtención de todo tipo de cosméticos actualmente el usuario tiene que plantearse si realmente le merece la pena perder todo lo que ha obtenido, ya sea pagando o mediante horas de juego, cuando se quiere pasar a la siguiente entrega de la saga. Esto dejaría de ser problemático cuando se habla de juegos blockchain ya que los activos digitales que se han ganados en el primero se mantendrían siempre que se continue la cadena de bloques que da soporte al juego.

[7] https://axieinfinity.com/
[8] https://ethereum.org/es/

Es importante no confundir un juego blockchain con un videojuego que utilice criptoactivos o criptomonedas. Todos los videojuegos implementados sobre una cadena de bloques utilizan criptomonedas lo cual no implica que cualquier juego que utilice criptodivisas sea un videojuego blockchain.

Esto no quiere decir que la utilización de criptomonedas dentro de juegos desarrollados de forma convencional no sea interesante ya que también aporta funcionalidades, pero para aprovechar realmente todas las capacidades que aporta la cadena de bloques el juego debe ser desarrollado sobre esta.

3.2.- Ventajas de los juegos blockchain.

En este punto se estudiarán las características positivas que el empleo de la tecnología blockchain trae para aquellos juegos que decidan implementarla así como los beneficios para desarrolladores y jugadores del mismo.

3.2.1.- Economía interna y derecho de propiedad.

Se permite a los usuarios ser propietarios de manera real de sus activos los cuales quedan representados en la cadena de bloques en forma de token más conocido como NFT el cual ya ha sido explicado en secciones anteriores.[9]

Esto previene que los jugadores puedan perder sus skins, personajes o en general cualquier objeto obtenible dentro del juego ya que estos pasarían a encontrarse dentro de la cadena de forma distribuida y ni siquiera los desarrolladores podrían alterar esta propiedad.

Otro aspecto relativo a la propiedad de los activos es que a pesar de que el jugador pase a la secuela de un videojuego este podrá conservar los activos que ya tenía de antes pues siguen formando parte de la cadena de bloques.

La utilización de blockchain elimina también gran parte del control que tienen los desarrolladores en videojuegos convencionales para modificar las ratios de aparición de determinados objetos haciendo que estos valores sean totalmente transparentes para la comunidad y que no puedan ser modificados.

Esto permite que los jugadores puedan llevar a cabo toda clase de intercambios de manera totalmente libre y descentralizada. Además, en aquellos casos en que se

[9] https://www.techspring.mx/usos-de-blockchain-en-gaming/#Cuales_son_los_usos_de_la_blockchain_en_el_gaming

implemente un modelo de jugar para ganar (*play to earn*) los jugadores también pueden obtener criptoactivos como recompensa por el tiempo invertido en el juego haciendo posible que en ciertos casos se llegue a ganar dinero.

En resumen, la utilización de blockchain proporciona muchos derechos a los jugadores que actualmente no tienen o tienen de forma reducida ampliando la transparencia y la seguridad sobre el valor de los objetos obtenidos por los jugadores.

3.2.2.- Descentralización y gobierno de los participantes.

En los juegos actuales se produce una centralización cada vez mayor en servidores llegando al punto en el cual ni comprando en físico un videojuego este funciona por sí solo ya que en la mayoría de los casos vendrá acompañado de un parche de lanzamiento el cual debe ser descargado vía internet desde los servidores de la desarrolladora.[10]

Contrariamente a esto, la tecnología blockchain permite la creación de servidores descentralizados haciendo que el juego siga vivo siempre que su comunidad no muera evitando así problemas como la desaparición de juegos debido a que su base de jugadores se redujo tanto que no era rentable para la empresa el mantenimiento de los servidores como por ejemplo el caso de *Electronics Arts* que cerro los servidores del videojuego *Battlefield Bad Company 2* debido a la antigüedad del mismo y a que sus secuelas absorbieron la base de jugadores que en su momento tuvo.

Como añadido a esto, los jugadores también pasan a ser quienes deciden las reglas que se aplican sobre el propio juego y son ellos los encargados de determinar si se cambian mecánicas o propiedades de elementos del videojuego. Esto permite al juego seguir, aunque los desarrolladores originales abandonen el proyecto.

Con todo esto los videojuegos creados sobre la tecnología blockchain resultan ser mucho más transparentes, así como resistentes a fallos ya que no existe servidor central como punto único de fallo.

[10] https://www.cultture.com/95051-beneficios-de-blockchain-para-la-industria-del-juego

3.2.3.- Seguridad.

Como ya se ha visto blockchain permite la realización de transacciones de manera segura y permanente sin necesidad de un intermediario de confianza que les de validez haciendo muy confiables los intercambios producidos entre usuarios.

Se eliminan de la escena a los tramposos ya que no les será posible hacer modificaciones del código para intentar beneficiarse ni tampoco podrán usar programas externos al juego para internar atacarlo por la propia forma en que se distribuye la red sobra la que se sustentaría.

3.2.4.- Competencia.

Actualmente el mundo de los videojuegos se encuentra compuesto principalmente por dos grupos los estudios triples A y los indies.

Por un lado, se tiene a los primeros los cuales son los estudios de desarrolladores de las compañías más poderosas actualmente como pueden ser *Sony*, *Microsoft*, *Nintendo* o *Blizzard* entre otras.

Por el otro estarían los desarrolladores indies los cuales son grupos compuestos normalmente por una persona o grupo reducido de estas y con presupuestos muy bajos en comparación con los del primer grupo.

Para los pequeños creadores independientes la aparición de blockchain puede suponer una forma de facilitar la no solo la distribución de sus juegos sino incluso el acceso a la colaboración con usuarios y otros desarrolladores.

3.3.- Desventajas de los juegos blockchain.

En este apartado se van a tratar los puntos negativos que pueden traer la tecnología blockchain cuando es aplicada en el ámbito concreto de los videojuegos. Algunas de estas características ya eran intrínsecas de la propia tecnología pero alcanzan nuevas dimensiones al aplicarlas al mundo de los videojuegos.

3.3.1.- Falta de escalabilidad.

La falta de escalabilidad es un problema general que ya se estudio es apartados anteriores, pero cobra aún más importancia cuando se aplica esta tecnología a los videojuegos.

Esto está limitando el posible crecimiento en número de jugadores de los juegos blockchain al mismo tiempo que limita las opciones de los desarrolladores ya que este tipo de juegos tienden a ser sencillos por culpa de la lentitud de la cadena de bloques.

Es por esto que la mayoría de juegos optan por utilizar *Ethereum* ya que por el momento es de las redes más escalables en comparación con por ejemplo *Bitcoin* en donde la creación de un bloque toma al menos 10 minutos mientras que en *Ethereum* este tiempo se reduce a aproximadamente 16 segundos.

3.3.2.- Falta de accesibilidad.

La accesibilidad a este tipo de videojuegos se ve muy limitada por la necesidad de tener un wallet y tus propias addresses para tan siquiera poder empezar a jugar lo cual se convierte en una barrera de entrada para ciertas personas que no estén familiarizadas con el mundo de las blockchains.

Además de esto el propio coste para poder empezar a jugar a veces se vuelve muy elevado por la propia forma en que las criptomonedas, las cuales se utilizan normalmente para adquirir activos en este tipo de juegos, suelen ser deflacionarias por lo que su valor aumenta con el paso de los años.

Por ejemplificar esto, si se toma uno de los juegos blockchain más famosos como es *Axie Infinity* el coste de adquirir un *Axie* ronda actualmente los 300 dólares lo cual es una cantidad excesiva si se compara con los 70 dólares que suele ser el precio actual en la industria de un juego triple A.

Otro problema que perjudica a la accesibilidad es que actualmente hay varias estafas relacionadas con juegos como por ejemplo el caso del juego NFT *Battle Fish* cuya plataforma ha desaparecido llevándose consigo el equivalente a 250.000 dólares.

Este tipo de casos dan mala fama y extienden la desconfianza entre el público general hacia este tipo de juegos causando que la cantidad de potenciales jugadores se reduzca a un nicho bastante especifico.

3.3.3.- Control y modificación.

La imposibilidad de modificar la cadena de bloques es un aspecto que normalmente es considerado una ventaja para la mayoría de los ámbitos en que es utilizada, pero para su aplicación al gaming plantea un problema importante.

Al ser tan difícil modificar los datos que ya se han introducido a la cadena también lo es para los desarrolladores realizar cualquier tipo de cambio en el código ya sea para introducir parches que arreglen errores o para introducir nuevas actualizaciones que intenten balancear ciertos personajes u objetos.

Es cierto que, si se pueden llegar a introducir cambios en el código, pero es un proceso bastante complicado que normalmente termina implicando la creación de una nueva cadena a la que seguir y abandonar la anterior.

Además, los desarrolladores pierden mucho control sobre el producto que han desarrollado ya que la toma de decisiones final acaba quedando en manos de la comunidad.

Si se combina este problema con la falta de escalabilidad se obtiene que al final los juegos de blockchain acaban siendo sencillos lo cual muchas veces lleva a la aparición de juegos clónicos.

3.3.4.- Pagar para ganar.

Uno de los mayores peligros que amenazan a los videojuegos blockchain es la facilidad con la que estos puede corromperse hacia modelos pagar para ganar (*pay to win*) en los cuales aquellos jugadores que realicen grandes ingresos de dinero real o de criptomonedas en el juego tienen una ventaja insalvable sobre los jugadores normales.

Sumado a esto, la brecha entre jugadores que lleven más tiempo en los juegos y los más novatos será aún mayor ya que la inversión que tendrían que hacer los segundos en un corto periodo de tiempo para alcanzar a los primeros sería muy elevada.

Además, por el carácter normalmente deflacionario de la mayoría de criptomonedas el coste de iniciarse a estos juegos aumenta constantemente a medida que pasa el tiempo lo cual es lo que también sustenta el valor de aquellos objetos de las personas que ya se han introducido al juego.

También es importante matizar que en muchos casos deben ser los propios desarrolladores los que se enfoquen a un modelo de jugar para ganar (*play to earn*) y

traten de limitar este tipo de prácticas, pero debido a la propia naturaleza distribuida de estos juegos y a la falta de control que puede existir esto no es fácil de llevar a cabo.

3.4.- Ejemplos de juegos blockchain.

A lo largo de este punto se van a estudiar varios juegos que emplean la tecnología blockchain los cuales tienen una gran popularidad y relevancia en el mercado actual. También se incluirán juegos considerados pioneros a pesar de que actualmente no tengan la importancia que obtuvieron en su momento.

3.4.1.- Cryptokitties.

Este juego fue creado el 28 de noviembre de 2017 por la empresa *Axiom Zen*[11] y se convirtió rápidamente en uno de los videojuegos basados en blockchain más conocidos, aunque igual de rápido que fue su ascenso lo fue su caída principalmente causada por la aparición de múltiples juegos similares como *Axie Infinity* los cuales perfeccionaron muchas de las mecánicas que se inventaron con este juego. [19]

La cadena de bloques elegida para soportar este juego fue *Ethereum* siendo el ETH la única moneda aceptada dentro del juego para realizar compras o ventas. La popularidad que este juego tuvo causo que se sobrecargase la red de *Ethereum* haciendo que la compañía creadora tuviera que aumentar el coste de crear nuevos gatos para así intentar reducir el número de transacciones que se llevaban a cabo [20], [21].

Esto evidencio una de las grandes debilidades que ya se ha comentado acerca de los videojuegos basados en la cadena de bloques como lo es la baja escalabilidad de los mismos.[12]

La existencia de cada gato se registra en la cadena de bloques en forma de NFT y las compras y ventas de estos se producen en forma de contratos inteligentes dentro de *Ethereum*. La expansión que han experimentado los tokens no fungibles se debe en parte también al éxito de este juego que popularizo este tipo de activos.[13]

Aunque *Cryptokitties* había perdido mucha popularidad respecto al momento de su máximo pico, recientemente en el año 2021 acaba de experimentar un aumento de

[11] https://www.frontiersin.org/articles/10.3389/fphy.2021.631665/full
[12] https://www.bbva.com/es/cripto-gatitos-gente-compra-masivamente/
[13] https://es.beincrypto.com/aprende/todo-que-necesitas-saber-acerca-cryptokitties/

jugadores debido al interés generado por la venta de "génesis" un gato de los primeros 100 que se crearon en la cadena y el cual es el cryptogato que se ha vendido por más dinero dentro del juego y el cual tiene un valor estimado de 100 ETH (casi 400.000 dólares actualmente).[14]

En resumen, se puede considerar que *Cryptokitties* fue un juego pionero el cual sentó muchas de las bases que los videojuegos que emplean blockchain utilizan a día de hoy como puede ser el uso de NFT para que el usuario sea dueño de objetos dentro del juego, siendo en este caso gatos, así como la utilización de criptomonedas como único método de intercambio dentro del juego.[15]

3.4.2.- Hunter Coin.

En este juego open source los jugadores actúan como hunters cuya tarea es viajar a través de un mapa 2D para encontrar monedas para después depositarlas en lugares concretos del mapa generados de forma aleatoria conocidos como bancos, traduciéndose esto en una transferencia a su wallet en forma de HUC la cual es la moneda propia del juego.[16]

Una vez que la moneda se encuentra en posesión del jugador esta podrá ser intercambiada por otras criptomonedas como *Bitcoin*, *Ethereum* o incluso por dinero por decreto a través de exchanges como *Binance* o *Poloniex*.

El coste de crear un hunter es de 200 HUC y cada uno de ellos tiene un ataque capaz de matar de un solo golpe a cualquier otro cazador que se encuentre en un bloque cercano costando esto 20 HUC. La clave del éxito es que el jugador sea capaz de predecir el movimiento de sus enemigos y elegir el momento adecuado para atacar.[17]

Además de las mecánicas de combate y búsqueda de monedas por el mapa existe otro item llamado "*Crown of Fortune*" que imposibilita la utilización de su habilidad de ataque al hunter que la lleve, pero a cambio este generara constantemente HUC.

Con estas mecánicas se plantean dos posibles formas de enfocar el juego, por un lado, estarán aquellos que prefieran simplemente recoger monedas de forma pacífica mientras esperar encontrar la *Crown of Fortune*, otros preferirán jugar de forma más

[14] https://ip.digital/nota/10046-cryptokitties-el-primer-videojuego-con-criptomonedas
[15] https://decrypt.co/es/80206/te-acuerdas-de-cryptokitties-los-nft-de-gatitos-estan-explotando-en-precio
[16] https://www.prnewswire.com/news-releases/huntercoin-the-online-game-that-earns-you-money-246237231.html
[17] https://xaya.io/huntercoin-legacy/

agresiva y dedicarse a cazar a todos aquellos jugadores que se encuentren prestándole especial atención a aquel que posea la corona ya que será el que probablemente haya acumulado la mayor cantidad de monedas.

Este juego tuvo que afrontar tres problemas relacionados con el planteamiento original para los cuales se han aportado soluciones parciales por el momento.[18]

El primero de los problemas que tiene este videojuego es el mismo que ya ha sido comentado varias veces a lo largo de este trabajo, la escalabilidad de los juegos blockchain. Teniendo en cuenta lo critico que puede llegar a ser en este juego el predecir o realizar un movimiento ya que hay que recordar que los ataques en este juego son totalmente letales y tienen un coste en moneda del juego lo cual se acaba traduciendo en dinero real y a medida que crece el número de jugadores también lo hacen las ralentizaciones y los problemas de procesamiento en los movimientos de los hunters.

Este problema espera solucionarse en un futuro cuando se consiga reducir el tamaño del juego, aunque es cierto que a medida que el número de jugadores siga creciendo este problema común a todos los juegos blockchain seguirá manifestándose.

El segundo problema fueron los bots ya que un jugador puede crearse más de un hunter para jugar y los hunters que no esté controlando directamente serán manejados por una IA la cual originalmente atacaba constantemente, pero esto fue parcheado añadiendo el coste por ataque ya comentado al inicio.[19]

El tercer problema fue la aparición de jugadores que se dedicaban únicamente a crear una gran cantidad de personajes para obtener el control exclusivo de zonas enteras del mapa y ser los únicos en obtener las recompensas que allí se generasen. Esto fue parcialmente solucionado al insertar los desastres aleatorios que mata a todos los jugadores y obliga a que se reinicie la partida.

3.4.3.- Axie Infinity.

Axie Infinity es un videojuego de tipo multijugador online que fue creado en 2018 por Sky Mavis. Este juego se puede considerar al mismo tiempo *Pay to Win* y *Pay to Earn* puesto que se requiere de una considerable inversión inicial de dinero para poder

[18] https://www.newsbtc.com/all/blockchain-gaming-huntercoin/
[19] https://bitcointalk.org/index.php?topic=435170.5540

conseguir empezar a jugar, pero al mismo tiempo premia mucho a aquellos jugadores que emplean mucho tiempo dentro del videojuego.[20]

La idea principal del juego consiste en coleccionar *Axies* los cuales son unas criaturas representadas por un token NFT dentro de la cadena de bloques de *Ethereum*. Cada uno de estos personajes tiene sus propias características las cuales pueden ir aumentándose con el tiempo permitiendo así dar lugar a batallas contra oponentes cada vez más fuertes.[21]

También existe todo un sistema de genética y crianza de axies para dar lugar a criaturas cada vez más poderosas y únicas, aunque cada *Axie* solo puede aparearse un máximo de 7 veces y además esto implica un coste en moneda del juego para el jugador.

Dentro del juego existen dos tipos de tokens que el usuario puede adquirir para comprar axies, criarlos o combatir los cuales son el AXS y el SLP.

El AXS (*Axie Infinity Shard*) es un tipo de token ERC-20 de *Ethereum* creado en noviembre de 2020 que permite a cada jugador ser premiado por su actividad dentro del juego. Esto no solo sirve para obtener ganancias, sino que una vez el jugador haya acumulado una gran cantidad de esta moneda puede utilizarla para participar en el gobierno del juego.

El SLP (*Small Love Potion*) también es un token ERC-20 pero este solo se consigue cuando se ganan batallas dentro del juego y se requieren como mínimo 100 SLP para poder empezar con la crianza de *Axies* lo cual es una de las claves del juego para obtener cada vez criaturas más fuertes.

Para poder iniciarse en este juego es necesario adquirir un mínimo de 3 *Axies* teniendo el *Axie* más barato que se puede obtener un valor aproximado de 0,1 ETH lo que supone aproximadamente 200 dólares en el momento de la realización de esta investigación, aunque se debe tener en cuenta que en momento de máximos en el precio del *Ethereum* han llegado a costar más de 300 dólares.[22]

Esto supone una barrera para muchas personas ya que dependiendo del valor fluctuante del *Ethereum* el coste de empezar a jugar, así como el valor real de las recompensas obtenidas ira cambiando con el tiempo.

[20] https://www.xataka.com/videojuegos/axie-infinity-juegos-mayor-crecimiento-ingresos-historia-pokemon-nft-que-vende-que-podemos-ganar-dinero-jugando
[21] https://axieworld.medium.com/gu%C3%ADa-de-cr%C3%ADa-de-axies-66423774915c
[22] https://booksfinancieros.com/guia-completa-de-axie-infinity-para-ganar-mas-dinero/

Existen alternativas para intentar evitar esta barrera de entrada como la beca *Axie* en la cual un jugador que generalmente posee muchos *Axies* alquila aquellos que les sobran a otros jugadores por un precio menor al de compra para que así puedan empezar a jugar siempre que se cumplan una serie de normas como que el propietario se lleva un porcentaje de las ganancias que generen los axies prestados, así como poder reclamarle al becario un mínimo de horas de juego.[23]

El problema de las becas *Axie* es que el número de propietarios dispuestos a ofrecerlas es mucho más reducido que la gran oferta de personas que quieren participar en el cómo becario.

Antes de poder empezar a jugar es necesario descargarse y registrarse dentro de la red *MetaMask* o dentro del *wallet* propio de *Axie Infinity* el cual se llama *Ronin*. Esto genera un problema en materia de seguridad, los cibercriminales se centrarán en intentar falsificar o atacar esta *wallet* al ser donde operan la mayoría de los jugadores del juego.

Un ejemplo de esto fue cuando en la propia *play store* de Google apareció una copia de la aplicación *Ronin* para intentar engañar a posibles personas que recién se iniciaban al juego.[24]

Actualmente un jugador que emplee unas 6 horas diarias al juego puede llegar a conseguir entre 150 y 200 SLP en un inicio, aunque esta cantidad va creciendo a medida que sus *Axies* van mejorando y participando en combates de mayor nivel. En cualquier momento el jugador puede convertir sus SLP o AXS directamente a Ethereum mediante bolsas de intercambio de criptomonedas como *Uniswap, Coinbase* o *Binance*.[25]

Este juego no solo parece muy lucrativo para los jugadores a pesar de su riesgo, sino que también lo es para Sky Mavis ya que ellos obtienen un 4.5% de todas las transacciones que se hagan con NFT de sus *Axies*.

Es importante recordar que como cualquier juego basado en blockchain *Axie Infinity* es seguro pues emplea tecnología de bloques para el almacenamiento de sus monedas y NFT, pero está sujeto a problemas de especulación lo que ha llevado a que mucha gente lo califique de burbuja debido a su rápido ascenso ocurrido durante el año

[23] https://www.vidaextra.com/guias-y-trucos/axie-infinity-como-conseguir-beca-jugar-gratis
[24] https://playtrucos.com/los-desarrolladores-de-axie-infinity-advierten-a-los-usuarios-sobre-la-billetera-movil-falsa-de-ronin/
[25] https://www.diariodeldinero.com/que-es-axie-infinity-como-funciona-como-se-gana-dinero-y-con-que-cosas-debes-tener-cuidado/

2021 que lo ha posicionado como uno de los videojuegos blockchain más jugados y que le hace generar más de 100 millones de dólares al mes para la compañía creadora.

3.4.4.- Decentraland.

Decentraland es un videojuego 3D que simula un mundo virtual en el cual los usuarios pueden ser propietarios de parcelas que son representadas mediante un NFT por lo que cada una es única.

También se pueden construir edificios y decorar cada parcela con diferentes objetos los cuales también son representados mediante NFT dentro de la cadena de *Ethereum*. Las distintas parcelas del juego se organizan alrededor de 9 plazas colocadas de forma circular y que son propiedad de la comunidad por lo que ningún usuario podrá monopolizarlas.[26] [27]

El juego está planteado en tres capas llamadas consenso, interacción y de interacción en tiempo real. La primera se encarga de registrar todas las transacciones de propiedades que se producen dentro del juego, la segunda almacena la información para creación de los diferentes objetos que se verán dentro del juego y la tercera se encarga de gestionar las interacciones que se produzcan entre diferentes usuarios.

La primera capa siempre registra en el bloque correspondiente de la cadena blockchain la compra de la propiedad realizada mientras que la segunda normalmente esta almacenada en cada usuario ya que no es necesario que todos los jugadores rendericen todo el contenido sino solamente el que van a visualizar que normalmente se corresponde con los objetos que tienen en su propia parcela.[28] [29]

La moneda para la compra de objetos dentro del juego conocida como MANA es un token ERC-20 de *Ethereum* y se decidió hacerlo de producción limitada ya que como máximo se podrán producir 2.194.460.527 de tokens. Esto hace que el valor de esta moneda se dispare a medida que aumenta la demanda de este videojuego.

El otro token que existe dentro del juego es el llamado token LAND que representa cada una de las unidades de tierra y el cual solo puede ser adquirido mediante el intercambio de su valor en MANA por lo que su desarrollo esta obligatoriamente ligado

[26] https://decentraland.org/
[27] https://academy.bit2me.com/que-es-decentraland-mana/
[28] https://www.giztab.com/decentraland-mana-que-es-y-como-funciona/
[29] https://www.diariobitcoin.com/glossary/decentraland/

al de la otra moneda. Existe un tercer token llamado ESTATE el cual se obtiene cuando se poseen dos parcelas vecinas.[30] [31]

El gobierno del juego se produce mediante una DAO (organización autónoma descentralizada), la posibilidad de votar viene determinada por los tokens que cada usuario posea siguiendo la forma de *proof of stake*. Cada usuario que decida votar debe transferir una cantidad de MANA al *Smart Contract* creado por la DAO, aunque mientras esa cantidad se encuentre dentro del *Smart Contract* no podrá ser utilizada para otros fines.

3.4.5.- Splinterlands.

Splinterlands es un juego de cartas multijugador del estilo de *Hearthstone o Magic the Gathering* lanzado en el año 2020 en el cual cada carta es un NFT dentro de la blockchain HIVE lo que hace que el coste de empezar sea mucho más bajo que en otros juegos que normalmente están basados en la red *Ethereum*.

Para facilitar la accesibilidad este juego permite empezar de forma totalmente gratuita ofreciendo cartas de nivel 1 pero estas no pueden ser vendidas por el usuario por lo que quedarán totalmente ligadas a la cuenta del mismo y no le permitirán participar en eventos ni en misiones que para obtener recompensas.[32]

Aunque el coste de obtener un libro de hechizos para poder empezar a realizar misiones diarias que nos permitan obtener recompensas no supera los 12 dólares siendo uno de los pocos juegos blockchain cuya barrera de entrada a nivel económico es muy baja en comparación al resto.

Este videojuego ha experimentado un crecimiento exponencial a lo largo de 2021 pasando a tener más de 90.000 jugadores teniendo en cuenta que el año anterior no había logrado pasar de los 10.000. [33]

Al igual que ya se ha visto en muchos juegos de esta temática el valor de cada carta viene determinado principalmente por su rareza dentro de las categorías que existen lo cual le da valor como NFT al ser única y permite su intercambio o venta dentro del mercado interno del juego.

[30] https://vicox.legal/que-es-decentraland/
[31] https://theobjective.com/economia/2021-12-04/decentraland-terreno-virtual-record/
[32] https://splinterlands.com/
[33] https://es.cointelegraph.com/news/meet-splinterlands-the-most-popular-blockchain-game-of-the-moment

Es importante diferenciar dos tipos de cartas dentro de *Splinterlands*, las cartas de monstruo que serían las criaturas que se van a ver implicadas en las batallas las cuales tienen un tipo y un nivel que vendrá determinado por el de la carta que lo invoco y el segundo tipo de carta seria las de invocación que representarían a un usuario de magia el cual llama a las criaturas siendo el nivel de la carta de invocación utilizada la que dicta el nivel del monstruo.

El juego divide a los jugadores en 6 ligas según su nivel de habilidad, experiencia y el nivel de poder de la colección de un jugador el cual viene establecido por las cartas que este posea. Una vez que el jugador supere el umbral de su liga se le da la opción de avanzar a la siguiente o quedarse en la que esta, aunque en el caso de elegir quedarse recibirá una penalización en la cantidad de puntos que gana porque se considera que está luchando contra jugadores que son inferiores a él en habilidad.[34]

Dentro de *Splinterlands* se utilizan dos monedas, por un lado, los SPS que serían la criptomoneda del propio juego que se utiliza para la gobernanza del mismo mientras que por otro lado está el DEC que es la moneda en la que se otorgan las recompensas.[35]

Al igual que ya se vio en *Descentraland* aquí se utiliza una DAO para la gobernanza del videojuego mediante la publicación de una serie de contratos inteligentes en BSC (*Binance Smart Chain*).

3.4.6.- Alien worlds.

Según datos tomados de la página *Dappradar*[36], *Alien worlds* se posiciona como el segundo juego que más relevancia está teniendo más de 150.000 mineros activos todos los días y un total de más de 300.000 registrados en la página.

Alien worlds es un videojuego accesible a través de un navegador web y que se ejecuta sobre redes blockchain como *Ethereum* o *Binance Smart Chain*. En este juego se ofrece la posibilidad de jugar como un minero el cual viaja por diferentes planetas para obtener un mineral valioso.

La motivación real que los jugadores pueden tener para realizar esto es que mientras juegan están al mismo tiempo obteniendo el token del videojuego conocido como *Trilium*.

[34] https://es.cointelegraph.com/explained/what-is-splinterlands-a-guide-to-the-nft-magic-card-game
[35] https://criptomundo.com/que-es-splinterlands-y-como-se-juega/
[36] https://dappradar.com/wax/games/alien-worlds

Todo dentro del juego se puede traducir en NFT ya que desde las herramientas que se usan para minar hasta los terrenos de los planetas sobre los que se mina. Los jugadores pueden comprar terrenos para después minar en ellos o alquilárselo a otros jugadores a cambio de un porcentaje de las ganancias que obtengan. Normalmente ese porcentaje este fijado en un 20% pero al tratarse de un juego descentralizado cada propietario puede pedir el porcentaje que el considere adecuado.[37]

El jugador puede invertir parte de sus ganancias para mejorar sus herramientas de minado obteniendo así una mayor cantidad de *Trilium* durante el tiempo que se emplea en minar, también está obteniendo un NFT más valioso que el que representaba su herramienta antigua.[38]

3.5.- Estado del Arte.

La utilización de la tecnología blockchain en videojuegos ha sido estudiada a través de diversas investigaciones y estudios desde que empezó a ser empleada hace aproximadamente 5 años.

Desde hace tiempo, la búsqueda de un modelo de producto software que combinase mecánicas de videojuegos con la tecnología blockchain se ha incluido incluso en instituciones de enseñanza superior. [22]

Este artículo concluye que la utilización de tecnología blockchain en conjunto con videojuegos en el ámbito de la educación aumenta la eficiencia en el aprendizaje por parte de los estudiantes y aumenta la fiabilidad de los sistemas de información internos de los centros de enseñanza.

También ha sido estudiado el impacto que ha tenido la economía de tokens (*Token Economy*) en las monedas y objetos de los videojuegos, así como los peligros que trae consigo debido a las estafas que se han producido alrededor de este tipo de tecnología. [23]

Por su parte, la universidad de OULU realizo un estudio sobre los efectos en la retención de la atención y en el compromiso de los jugadores que la aplicación de

[37] https://cryptoticker.io/en/what-is-alien-worlds-crypto-is-this-nft-game-worth-it/
[38] https://coinmarketcap.com/alexandria/article/what-is-alien-worlds-tlm-features-tokenomics-and-price-prediction

blockchain puede causar. Para ello se realizó una mejora de un proyecto propio de la universidad de OULU llamado "*Ikune Racers*" mediante la blockchain. [24]

Otra aplicación que tiene la tecnología blockchain en videojuegos es la posibilidad de solucionar problemas de propiedad intelectual entre los creadores de videojuegos gracias a la utilización de blockchain para facilitar el cobro de regalías que en muchos casos es un proceso bastante engorroso y problemático. [25]

También se ha debatido acerca del modelo de negocio utilizado por *Axie Infinity* y se han analizado los puntos de mejora que el equipo creador debería abordar en el futuro para continuar siendo líderes en el mercado. [26]

Asimismo la influencia de invertir en juegos Crypto también ha sido objeto de estudio y se ha llegado a la conclusión de que el enorme crecimiento que la industria experimento durante la pandemia fue de gran ayuda para la difusión de la misma pero también advierten de los potenciales peligros que este tipo de inversiones pueden tener para las personas con pocos conocimientos acerca de inversión en Criptoactivos. [27]

Un punto importante a tener en cuenta es la importancia que la tecnología blockchain está teniendo para cambiar los modelos de negocio que las compañías del mundo del entretenimiento estaban utilizando hasta ahora. [28]

También ha sido tratado el tema de las ventajas que podría aportar el blockchain en el mundo del entretenimiento no solo en videojuegos sino también para el desarrollo de otras industrias como la cinematográfica. [29]

Otro aspecto que se ha investigado es la estrecha relación que los juegos Crypto o videojuegos NFT tienen con las apuestas debido a algunas de sus características como pueden ser el uso de la probabilidad para generar recompensas. También se estudiaron los aspectos legales y psicológicos que tienen en común este tipo de juegos con las apuestas. [30], [31]

Complementando al artículo anterior, se está intentando mejorar las diferentes formas en las que se podrían programar las cajas de botín (*Loot Boxes*) usando contratos inteligentes para aumentar la transparencia y la confianza en este tipo de sistemas de recompensa. [32]

También existen investigaciones sobre el diseño y desarrollo de un videojuego basado en realizar simulaciones con el objetivo de aumentar el interés que estudiantes y empresarios tienen sobre esta tecnología. Incluso existen varios artículos relacionados con desarrollos de juegos de similar naturaleza. [33], [34]

Gracias a todos estos estudios se puede vislumbrar la gran importancia que ha tenido la utilización de la blockchain en los videojuegos, así como las diferentes implicaciones (en algunos casos positivas y en otros casos negativas) que la implementación de esta tecnología ha supuesto.

4. Diseño e Implementación de Videojuego Tipo Blockchain

En este capítulo se va abordar el desarrollo de un videojuego de tipo NFT desde el diseño del juego hasta la programación del mismo y haciendo especial hincapié en aquellas partes del mismo en las que la tecnología de cadena de bloques ha jugado un papel importante.

4.1.- Introducción al diseño del videojuego

El videojuego de tipo NFT que se ha desarrollado para este trabajo el cual se llama *Skull Game* (Juego de calaveras) se encuentra inspirado en la forma de funcionar de *Cryptokitties* juego del cual ya se ha hablado anteriormente en el punto 3.4.1.

Para el videojuego implementado en este trabajo se ha pensado en crear una colección de calaveras (*Skulls*) generadas de manera aleatoria en base a las propiedades únicas de cada NFT utilizando para ello una serie de imágenes creadas previamente para poder ser combinadas adecuadamente a nivel de diseño.

El *gameplay* del juego se basa en que todos los usuarios que participen en él, tendrán que pagar para realizar alguna de las acciones por lo que se irán generando transacciones hacia el contrato donde se almacenara el dinero y la cantidad acumulada podrá ser consultada mediante el botón de enseñar balance.

Los NFT generados tendrán diferentes niveles de rareza lo cual se verá reflejado no solo en su diseño sino también en los puntos de salud y de ataque que este tiene. Esta propiedad será inmutable para cada NFT y se le asignará de manera aleatoria al momento de su creación.

Cada NFT también tendrá un nivel que, así como la rareza influirá en la salud y ataque que este tenga, pero a diferencia de la rareza sí que podrá ser modificado a cambio de un realizar un pago.

Todos los usuarios que tengan al menos un NFT podrán enfrentarse al jefe para intentar derrotarlo ya que el premio por vencerlo será obtener el botín que se haya ido acumulando hasta ese momento.

El jefe también atacará de vuelta al NFT y en caso de llegar a quitarle todos sus puntos de salud el atacante solo podrá retirarse del combate. Cada ataque que el usuario quiera realizar tendrá un coste variable que aumentara dependiendo de la rareza y nivel del NFT.

Una vez que un NFT pierde todos sus puntos de salud este no podrá ser subido de nivel ni utilizado para pelear, pero sí que seguirá existiendo como tal, aunque si el usuario quiere seguir utilizándolo también se le da la opción de revivirlo. La posibilidad de revivir al NFT no solo tendrá un coste variable en forma de transacción, sino que además como penalización este regresará al nivel 1.

El siguiente diagrama muestra el flujo de las acciones que se pueden realizar en el videojuego:

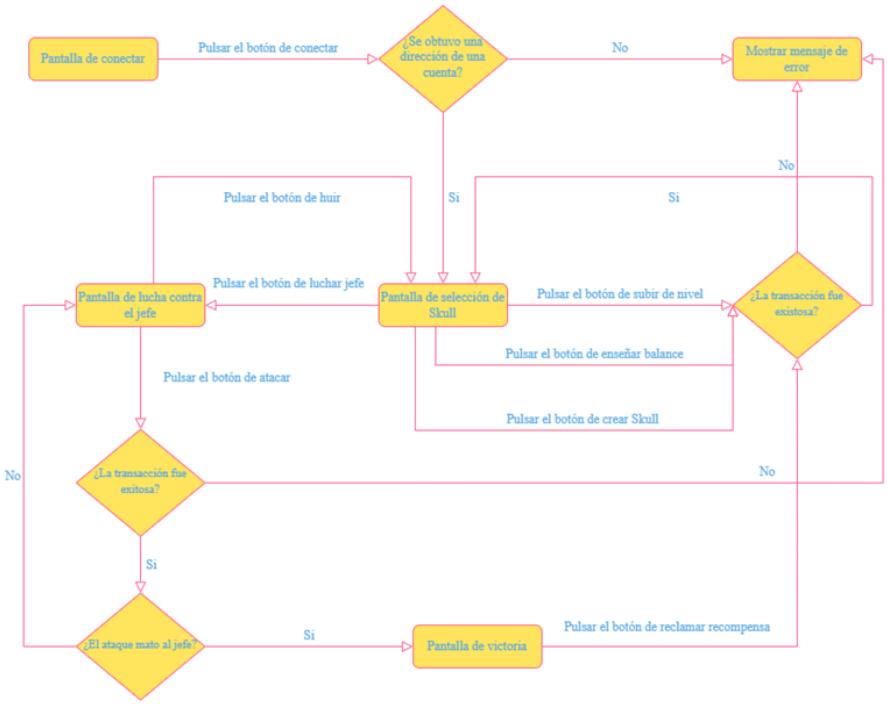

Figura 4.1.- Diagrama de acciones del juego.

En la Figura *4.1*, se puede ver la el diagrama de acciones en el cual se representan las diferentes pantallas de la aplicación y las acciones que llevan a cambiar de una pantalla a otra o a alguno de los estados que generan un error.

En el caso de los errores se le mostraría al usuario un mensaje informativo y por ejemplo si la transacción no tiene dinero suficiente como requiere el contrato para realizar

la acción que implica pues se le mostrara al usuario un mensaje informativo de que no envio dinero suficiente.

4.2.- Tecnologías empleadas en desarrollo del videojuego.

En el presente punto, se van a abordar las tecnologías empleadas en el videojuego, tanto para la implementación de los contratos inteligentes, como para la creación de la interfaz del juego.

4.2.1.- Tecnologías para el contrato inteligente.

Para el desarrollo del contrato inteligente del videojuego que en este caso se trata de un contrato inteligente se ha utilizado el lenguaje de programación *Solidity* ya que este es el más común y estandarizado para el desarrollo de este tipo de software.

Como proveedor se ha utilizado *MetaMask* a la hora de probar el contrato inteligente pero podría utilizarse indistintamente cualquier otro software que permitiese la interacción del navegador con una cadena de bloques.

Además, se ha empleado la aplicación web *Remix IDE* para desarrollar y testear inicialmente que el contrato compila y funciona correctamente antes de integrarlo en el proyecto final.

También se ha utilizado la librería de *openzeppelin* el cual es un proyecto que proporciona una serie de funciones para el desarrollo de aplicaciones descentralizadas siguiendo los estándares de seguridad y las prácticas de desarrollo de software más actuales.[39] [40]

Sus características principales citando la página del master en blockchain de la Universidad de Alcalá serían las siguientes: [41]

- Permite la ejecución de comandos interactivos para acelerar el desarrollo local.
- Permite realizar una sencilla corrección de errores e iteraciones rápidas a través de actualizaciones de los smart contracts.
- Permite una integración perfecta con todos los contratos que ofrece el proyecto OpenZeppelin.

[39] https://openzeppelin.com/
[40] https://github.com/OpenZeppelin
[41] https://masterblockchain.net/openzeppelin-master-blockchain-online/

- Da soporte a múltiples redes.
- Es totalmente compatible con Truffle.

Para nuestro videojuego se han utilizado en concreto funciones que se encuentran en los contratos prefabricados *ERC721.sol* y *Ownable.sol*.

4.2.2.- Tecnologías para generar imágenes.

Para la creación de las imágenes que se utilizaran en el videojuego se ha utilizado el software *MyPaint* con el cual se han generado varias imágenes a través de capas para que cuando se combinen entre si la imagen resultante tenga sentido estéticamente.[42]

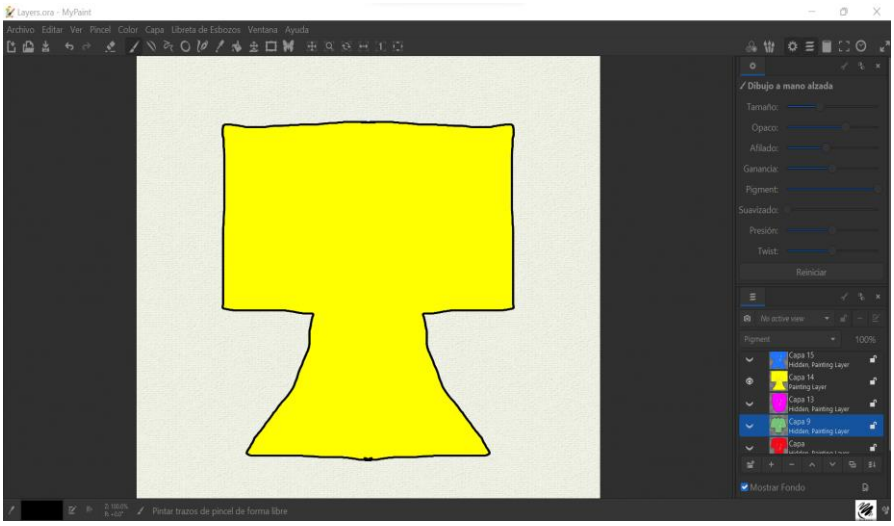

Figura 4.2.- Ejemplo de uso de MyPaint.

En la Figura *4.2*, se pueden ver los dibujos que se han realizado para los NFT son calaveras (*Skulls*) básicas teniendo como principal característica que hay diferentes tipos de ojos, narices, bocas y coronas para cada uno correspondiendo cada uno de estos elementos con una capa en particular.

[42] https://www.ecured.cu/MyPaint

4.2.3.- Tecnologías para el front-end.

En cuanto al aplicación que será el videojuego en sí mismo se han utilizado las tecnologías de Node.js, Yarn, Redux, Web3, Truffle en combinación con el framework React.

Se ha empleado React y no otras tecnologías más nativas del desarrollo de videojuegos como podría ser Unity porque esta última aún no tiene la suficiente adaptación para trabajar con *Web3* lo cual es esencial para este tipo de juegos.

Durante la creación de este trabajo se llevó a cabo una pequeña aproximación empleando Unity con un módulo de *WebGL*[43] pero debido a las limitaciones que terminaba causando en el desarrollo se descartó y no fue incluido en el trabajo final.

También se decidió no realizar el juego de forma nativa para móviles sino que se ha optado por otra aproximación consistente en desarrollar el juego para navegadores web y hacerlo lo suficientemente responsive como para poder ser utilizado correctamente en teléfonos inteligentes.

La razón principal detrás de esta decisión es que normalmente la mayoría de juegos NFT que se han desarrollado por el momento suelen hacerse así debido a que la forma más básica de utilizar proveedores como *MetaMask* es a través de una extensión en el navegador.

Node.js es un entorno de ejecución para la creación de aplicaciones web con el lenguaje de programación JavaScript y totalmente compatible con la programación orientada a eventos (los procesos no bloquean la ejecución del hilo principal, sino que se trabaja mediante *callbacks*) lo cual favorece la creación de sistemas más escalables.[44]

Yarn es un gestor de paquetes que ayuda a realizar instalaciones de otras librerías de código, para el desarrollo del videojuego NFT se ha utilizado Yarn tanto para la instalación de paquetes como Redux, openzeppelin o web3 como para la creación de la *build* del proyecto y la ejecución del mismo.[45]

[43] https://docs.unity3d.com/es/2018.4/Manual/webgl-gettingstarted.html#:~:text=Qu%C3%A9%20es%20Unity%20WebGL%3F,Unity%20en%20el%20explorador%20web.
[44] https://nodejs.org/es/about/
[45] https://yarnpkg.com/getting-started

Redux es un contenedor predecible del estado de aplicaciones JavaScript que ayuda a escribir aplicaciones que se comportan de manera consistente, corren en distintos ambientes (cliente, servidor y nativo), y son fáciles de probar.[46]

En resumen, Redux es una forma de administrar el estado (similar a una cache) al que pueden acceder todos los componentes de forma estructurada. Se tiene que acceder a través de acciones y reductores. En la Figura *4.3*, se puede ver su estructura:

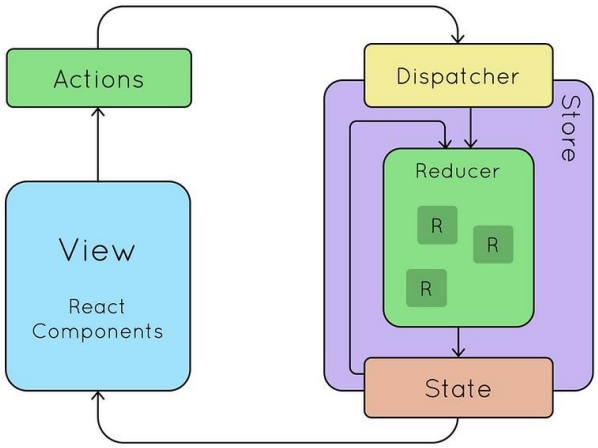

Figura 4.3.- Esquema de funcionamiento de Redux.[47]

Web3 es una colección de módulos que se utiliza para facilitar la interacción local o remota con el ecosistema de Ethereum.[48]

Truffle es un entorno de desarrollo, un marco de prueba y una canalización de activos para cadenas de bloques que utilizan la Máquina Virtual de Ethereum (EVM). Es ampliamente considerada la herramienta más popular para el desarrollo de aplicaciones de cadena de bloques con más de 1,5 millones de descargas.[49]

Las principales funciones de esta herramienta serían:

- Compilación, vinculación, implementación y administración binaria de contratos inteligentes.
- Pruebas automatizadas para contratos inteligentes.
- Marco de implementación y migración programable y extensible.

[46] https://es.redux.js.org/
[47] https://techclub.tajamar.es/react-con-redux/
[48] https://web3js.readthedocs.io/en/v1.7.3/getting-started.html
[49] https://trufflesuite.com/docs/truffle/overview

- Gestión de red para la implementación de cualquier número de redes públicas o privadas.
- Gestión de paquetes con NPM, utilizando el estándar ERC 190.
- Consola interactiva para la comunicación directa de contratos.
- Ejecutor de secuencias de comandos externo para la ejecución de secuencias de comandos dentro del entorno de Truffle.

Además de las tecnologías nombradas anteriormente también se ha empleado Ganache la cual es una blockchain personal para poder desplegar contratos inteligentes de manera local.[50]

El diagrama de comunicaciones de la aplicación sería el siguiente:

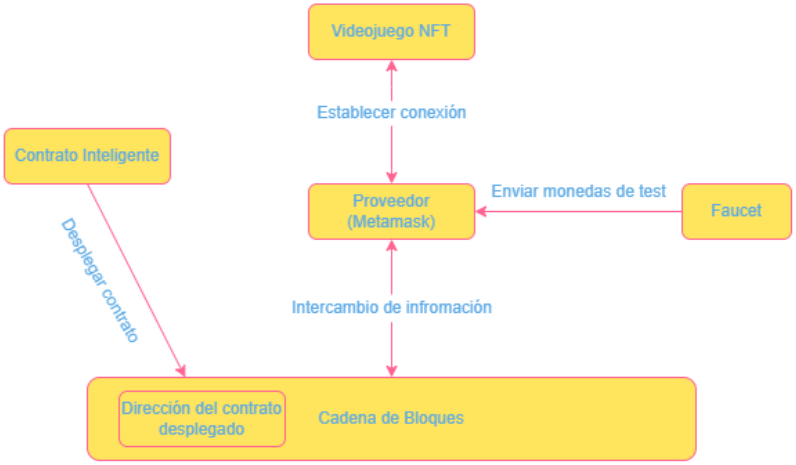

Figura 4.4.- Diagrama de comunicaciones.

En la Figura *4.4*, se puede ver como para realizar la conexión con la blockchain es necesaria la ayuda de un proveedor (*Provider*) que facilite la interacción entre el navegador web en el que se visualiza la aplicación y la propia blockchain.

Además se puede ver también un grifo de criptomonedas (*Faucet*) que son unos sitios web que proporcionan criptomonedas para las redes de prueba de las principales blockchains de la actualidad.

[50] https://trufflesuite.com/docs/ganache/

4.3.- Implementación del videojuego.

En el siguiente punto, se van a explicar la estructura del proyecto y todo lo referente a las implementaciones y configuraciones que se han llevado a cabo durante la creación del mismo.

4.3.1.- Estructura del proyecto.

La estructura que se ha utilizado para la creación del proyecto es la siguiente:

- Build: Esta carpeta contiene el resultado de crear una build final utilizando el comando *Yarn build*
- Contracts: Contiene los archivos de Solidity con el código de los contratos inteligentes. El archivo *GameContract.sol* contiene el código del juego mientras que *Migrations.sol* es el contrato de ejemplo que se suele dejar como base para la creación de contratos.
- Migrations: Contiene los archivos de JavaScript para realizar el despliegue del contrato hacia la blockchain mediante Truffle. Al igual que en la carpeta Contracts el archivo *1_initial_migration.js* es un contrato de ejemplo que se deja como ejemplo mientras que el contenido real para el despliegue se ubica en *2_initial_migration.js*.
- Node_modules: Es la carpeta que genera npm para almacenar todas las librerías y dependencias que tiene nuestro código. Esta carpeta se suele borrar antes de publicar o subir el contenido del proyecto para reducir su tamaño y se puede volver a generar realizando un *npm install* ya que npm extrae las dependencias del archivo *package.json*.
- Public: Esta carpeta contiene los archivos estáticos para montar la aplicación como por ejemplo el favicon que es el icono que se asigna a la página web.
- Src: Es la carpeta donde se almacena el código de React que se explicara en otro punto más en detalle.

4.3.2.- Declaraciones iniciales del contrato inteligente.

El archivo *GameContract.sol* contiene el contrato inteligente que el juego utilizara como back-end a través de la blockchain.

Lo primero que se encuentra sería el *pragma Solidity* que se utiliza para definir con que versiones de *Solidity* sea compatible nuestro *smart contract* y en el caso de este proyecto se ha elegido la versión 0.8.0 o superior para poder hacer el contrato compatible con muchas de las funciones que provienen de las librerías de *openzeppelin*.

Seguido a esa definición se encuentran los *imports* de los dos archivos de openzeppelin que se han empleado.

El archivo *ownable.sol* contiene funciones para la identificación del dueño de un contrato, trabajar con su dirección de forma más rápida y también contiene modificadores de seguridad para que solamente el dueño del contrato pueda acceder a ciertas funciones.

El *ERC721.sol* nos permite obtener funciones para la creación de un smart contract de tipo *ERC721* como pueden ser el constructor básico o la función *_safeMint ()* para asignar direcciones con id u objetos.

Mas adelante se define el contrato haciendo que este herede de los tipos *ERC721* y *Ownable* que están definidos en los archivos nombrados anteriormente para poder hacer uso de sus funciones como propias.

```
// Crear el smart contract para el videojuego NFT
contract GameContract is ERC721, Ownable {
    // Constructor de mi Smart Contract
    constructor(
        string memory _name,
        string memory _symbol,
        string memory treeBossName,
        uint256 treeBossMaxHpPoints,
        uint256 treeBossAttackPoints,
        string memory treeBossImageURI
    ) ERC721(_name, _symbol) {
        // Inicilización del jefe arbol
        treeBoss = TreeBoss({
            name: treeBossName,
            maxHpPoints: treeBossMaxHpPoints,
            hpPoints: treeBossMaxHpPoints,
            attackPoints: treeBossAttackPoints,
            imageURI: treeBossImageURI
        });
    }
```

Figura 4.5.- Constructor del contrato inteligente

En la Figura *4.5*, se puede ver el constructor del contrato al cual se le pueden pasar las propiedades que van a ser fijas para el contrato ya que solamente se inicializaran una vez, pero algunas de ellas sí que será posible modificarlas mediante algunas funciones como se explicara más adelante.

Después se definirán las declaraciones iniciales entre las que se tendrán dos variables numéricas llamadas *skullCounter* y cost.

La primera se utilizó para llevar la cuenta de cuantos NFT se han creado y para darles Id en el momento de su creación en base a este campo.

La segunda es el coste de las operaciones que no dependan de nada más que en el caso de nuestro juego será solamente la operación de crear un NFT. Esta variable tiene por defecto el valor de 0.001 ethers independientemente del blockchain en el que se esté desplegando.

Los tipos que se tratan en el juego son el *Skull* y el *TreeBoss* para representar a los NFT y al jefe respectivamente.

Las propiedades básicas del Skull son:

- Id: Identificador de cada NFT.
- Name: Nombre de cada NFT.
- Dna: Esta propiedad es un numero generado aleatoriamente que va a determinar el aspecto que tendrá nuestro NFT en el juego.
- Rarity: Es la rareza real del NFT la cual puede variar entre 0 y 100.
- Level: Es el nivel del NFT que condicionara sus propiedades de salud y ataque.
- MaxHpPoint: Define la vida máxima que puede llegar a tener un NFT y podrá ser modificada cuando este sube de nivel.
- AttackPoints: Define los puntos de ataque que tiene el NFT y aumenta con el nivel.
- HpPoints: Representa la salud que le queda al NFT actualmente.

Por su parte las propiedades del jefe serán las mismas que las del NFT con excepción de aquellas que se utilizan para que cada uno de ellos sea único ya que al solo tener un único jefe este se inicializa durante la creación del contrato y no fue necesario que tuviese características únicas.

Además, posee una propiedad *imageURI* para que almacene el *String* que contendrá la imagen que representa al jefe.

```
// Estructura de almacenamiento
Skull[] public skulls;
TreeBoss public treeBoss;
address winnerAddress;

// Declaración de un emisor de eventos
event CreateSkull(address indexed owner, uint256 id, uint256 dna);
event LevelUpSkull(address indexed owner);
event ReviveSkull(address indexed owner);
event BossAttacked(address indexed owner, bool isBossDefeated);
```

Figura 4.6.- Estructuras de almacenamiento

En la Figura *4.6*, se puede ver como se definieron las estructuras de almacenamiento que serían un array para almacenar todas las *Skulls* que se creen por los usuarios, un único objeto de tipo jefe puesto que se necesitó solamente uno y una variable para almacenar la dirección de la cuenta que gano en el juego lo cual se explicará más adelante cuando se hable de las funciones relativas a la victoria de un jugador.

También se definieron una serie de emisores de eventos para poder comunicar la finalización de las diferentes funciones que el contrato posee.

```
// ==================================== Modificadores ======================================
// Modificador para que solo el ganador pueda sacar dinero
modifier onlyWinner(address _direccion) {
    require(
        winnerAddress == _direccion,
        "You are not the one who defeated the boss!"
    );
    _;
}

// Modificador para que solo el dueño de la skull pueda actuar
modifier onlySkullOwner(address _direccion, uint256 _skullId) {
    require(ownerOf(_skullId) == _direccion);
    _;
}
```

Figura 4.7.- Modificadores de Solidity

En la Figura *4.7*, se pueden ver los modificadores los cuales son estructuras que se añaden a las funciones para enmendarlas de forma declarativa y normalmente son utilizados para realizar validaciones de datos simples antes de iniciar el código de una función.

La parte del código con _; es sustituida por el código de la función que utiliza el modificador gracias a una mezcla (merge) que se realiza durante la compilación del contrato.

En el caso del videojuego desarrollado para este trabajo se han creado dos modificadores, uno es para verificar que solamente la dirección que esta almacenada en el contrato como ganador pueda ejecutar la función que utilice el modificador *onlyWinner* mientras que la segunda es para verificar que se es propietario del NFT acceder (este modificador se verá en todas las funciones del contrato que tengan que modificar datos de los NFT).

En la Figura *4.8*, se puede ver como se crearon dos funciones únicas para el dueño del contrato que no son utilizadas como tal en el juego pero que podría necesitarse en un futuro en caso de querer implementar una versión más realista del juego en el cual la empresa creadora quiera obtener beneficios.

```
// ========================== Funciones del dueño del contrato ======================================
// Actualizador del precio del token NFT
function updateCost(uint256 _cost) external onlyOwner {
    cost = _cost;
}

// Extractor de los ethers del smart contract hacia el owner
function withdrawOwner() external payable onlyOwner {
    address payable _owner = payable(owner()); // Esta funcion owner de Ownable.sol permite obtener el owner
    _owner.transfer(address(this).balance);
}
```

Figura 4.8.- Funciones del dueño

Ambas funciones son de tipo externo para poder ser llamadas fuera del contrato e interactuar con ellas, pero utilizan un modificador propio del archivo *Ownable* para que solo puedan ser ejecutadas por el creador del contrato inteligente.

La primera se utiliza para modificar el valor del coste que tienen las operaciones simples del contrato mientras que la segunda es para extraer todo el dinero que se haya depositado en el contrato hacia la cuenta del creador del mismo.

Una opción recomendable sería obtener solo un porcentaje de los beneficios mientras que la otra parte se dejase como botín para el ganador, pero ya que esta función no se va a utilizar para la demo realizada no se consideró necesario entrar en más detalle en ese aspecto.

4.3.3.- Funciones internas del contrato inteligente.

La primera función interna del contrato es un generador de números aleatorios que utiliza la función *keccack256* que recibe como parámetros la marca de tiempo del bloque

actual y la dirección de la persona que está realizando la transacción en la que se llame a esta función y retorna un numero aleatorio basándose en esos datos.

La Figura *4.9* muestra como ha sido implementada dicha función:

```
// ========================== Funciones internas del contrato ================
// Asignador de numeros aleatorios
function _createRandomNumber(uint256 _mod) internal view returns (uint256) {
    uint256 randomNumber = uint256(
        keccak256(abi.encodePacked(block.timestamp, msg.sender))
    );
    return randomNumber % _mod;
}
```

Figura 4.9.- Función de generación de números aleatorios

La segunda función que se ha definido es para la creación de un NFT y queda representada en la Figura *4.10*:

```
// Creador del token NFT
function _createSkull(string memory _name) internal {
    uint8 randomRarity = uint8(_createRandomNumber(100));
    uint256 randomDna = _createRandomNumber(10**16);
    uint256 maxHp = _createRandomNumber(100) * randomRarity;
    uint256 attack = _createRandomNumber(10) * randomRarity;

    Skull memory newSkull = Skull(
        skullCounter,
        _name,
        randomDna,
        1,
        randomRarity,
        maxHp,
        maxHp, // Por defecto todos los nft empiezan con toda la vida
        attack
    );

    skulls.push(newSkull);
    _safeMint(msg.sender, skullCounter); // Función de ERC721.sol que asigna al msg sender que ejecute el token

    skullCounter++;
    emit CreateSkull(msg.sender, skullCounter, randomDna); // Emitimos el evento de que el token ha sido creado
}
```

Figura 4.10.- Función para crear NFTs

Lo primero que se hace es utilizar la función de generación de números aleatorios para asignar una rareza, un ADN, una salud y un ataque.

Después se crea un objeto de tipo Skull signándole las propiedades antes definidas y por defecto se le pone un id igual al contador que se explicó anteriormente, se le pone nivel 1 y se le pone una vida llena por lo que los puntos de salud son los mismos que los puntos máximos de salud.

Finalmente se añade el objeto al array de datos para almacenarlo y después se utiliza la función _safeMint del archivo *ERC721.sol* para generar un NFT asignado al msg.sender que sería la dirección de la cuenta que inició esta transacción (la cuenta que llamo a esta función) y queda vinculado al id del objeto recién creado (el cual es el mismo que el valor del contador). Tras esto se incrementa el contador para no repetir id en la siguiente creación de NFT y se emite un evento para informar que ha finalizado la creación del NFT.

La tercera función es para subir de nivel un NFT que ya existiese la cual se puede ver a continuación en la Figura *4.11*:

```
// Subir de nivel los tokens NFT
function _levelUpSkull(uint256 _skullId) internal {
    Skull storage skull = skulls[_skullId];
    skull.level++;
    skull.maxHpPoints = skull.maxHpPoints + 500;
    skull.attackPoints = skull.attackPoints + 50;
    skull.hpPoints = skull.maxHpPoints;
    emit LevelUpSkull(msg.sender);
}
```

Figura 4.11.- Función para subir de nivel

Se accede a la estructura donde se guardan los NFT y se obtiene aquel que pertenece al id que se nos proporciona. Después modifican las propiedades del mismo y se emite un evento para comunicar la finalización de la transacción.

En la Figura *4.12*, se puede ver la que afecta a las propiedades del NFT directamente la cual es la función de revivir que recupera toda la vida del NFT y al mismo tiempo lo devuelve al nivel inicial.

```
// Revivir un token NFT
function _reviveSkull(uint256 _skullId) internal {
    Skull storage skull = skulls[_skullId];
    skull.hpPoints = skull.maxHpPoints;
    skull.level = 1;
    emit ReviveSkull(msg.sender);
}
```

Figura 4.12.- Función para revivir un NFT

Por último, en la Figura *4.13* se puede observar la función de atacar al jefe la cual permite la interacción de combate dentro del juego:

```
// Iniciar un ataque contra el TreeBoss
function _attackBoss(uint256 _skullId) internal {
    // Muy importante usar el storage para poder modificar los valores del nft
    Skull storage skull = skulls[_skullId];

    // Comprobar que el NFT seleccionado tiene mas de 0 HP.
    require( skull.hpPoints > 0,"Error: La skull tiene que tener vida para poder atacar");

    // Comprobar que el jefe tiene mas de 0 HP.
    require(treeBoss.hpPoints > 0,"Error: El jefe tiene que tener vida para poder atacar");

    // Permitir que la skull seleccionada ataque al jefe
    if (treeBoss.hpPoints < skull.attackPoints) {
        treeBoss.hpPoints = 0;
        winnerAddress = msg.sender;
    } else {
        treeBoss.hpPoints = treeBoss.hpPoints - skull.attackPoints;
    }

    // Permitir que el jefe ataque a la skull seleccionada
    if (skull.hpPoints < treeBoss.attackPoints) {
        skull.hpPoints = 0;
    } else {
        skull.hpPoints = skull.hpPoints - treeBoss.attackPoints;
    }

    emit BossAttacked(msg.sender, treeBoss.hpPoints == 0);
}
```

Figura 4.13.- Función para atacar al jefe

En esta función no se han pasado todas las validaciones a un modificador ya que se trata de algo especifico de esta función y no de un código susceptible de ser reutilizado por otras varias funciones.

Se comprueba que tanto el jefe como el jefe como el NFT que va a iniciar el ataque no están muertos y después se procede a la interacción de combate.

En el caso de que algún golpe pudiera ser mortal y dejar la vida de alguno de los dos en números negativos se pone automáticamente a 0 para indicar que ha muerto y en el caso de que el que muera sea el jefe entonces se guarda la dirección de la cuenta del NFT que consiguió derrotarlo para que solo él pueda reclamar la recompensa.

Finalmente se emite un evento para comunicar la finalización de la función de combate pasando además un boolean que indique si el jefe fue derrotado durante el ataque en cuestión.

4.3.4.- Funciones externas del contrato inteligente.

Ahora se van a explicar las funciones externas, estas funciones serán directamente llamadas por el *front-end* y muchas de ellas se apoyarán en las funciones internas antes explicadas para llevar a cabo su funcionalidad.

La primera de ellas cumple la misión de obtener aquellos NFT que pertenezcan a la dirección que invoco la transacción y por lo tanto es la que se utilizara para la pantalla de selección de NFT una vez que ya se haya realizado la conexión con el proveedor *MetaMask* y por lo tanto ya se posea una dirección de cuenta como se explicó en el diagrama de acciones.

```
// Obtención de los tokens de un usuario concreto
function getUserSkulls(address _owner)
    public
    view
    returns (Skull[] memory)
{
    Skull[] memory result = new Skull[](balanceOf(_owner));

    uint256 loopCounter = 0;
    for (uint256 i = 0; i < skulls.length; i++) {
        if (ownerOf(i) == _owner) {
            result[loopCounter] = skulls[loopCounter];
            loopCounter++;
        }
    }

    return result;
}
```

Figura 4.14.- Obtención de los NFT de un usuario

En la Figura *4.14*, se puede ver la función para recorrer la estructura con los NFT y comparar el dueño de cada posición del NFT (es importante recordar que el id de cada NFT es el mismo que el contador que se le asigno durante su creación) con la dirección que se le pasa por parámetro la cual será la dirección de la cuenta que invoco la función.

En la Figura *4.15*, se pueden ver las funciones básicas que se han definido como externas son las siguientes:

```
// Obtención de todos los tokens NFT
function getSkulls() public view returns (Skull[] memory) {
    return skulls;
}

// Obtención de la información del jefe
function getBossInfo() public view returns (TreeBoss memory) {
    return treeBoss;
}

// Visualizar la dirección del smart contract
function addressSmartContract() public view returns (address) {
    return address(this);
}

// Visualizar el balance del smart contract
function moneySmartContract() public view returns (uint256) {
    return address(this).balance;
}
```

Figura 4.15.- Funciones para acceso a datos

Estas funciones simplemente dan acceso a datos básicos del contrato inteligente como serían el número total de NFT que se han creado, la información actual del jefe, la dirección del contrato (para poder hacerle transferencias) y el balance actual del contrato que sería la cantidad total de dinero que este haya acumulado por las diferentes acciones que los jugadores han ido haciendo.

La etiqueta *view* que se ha podido ver en todas estas funciones se utiliza para indicar que estas funciones son solo de acceso a datos y no van a producir modificaciones de los mismos.

La Figura *4.16* muestra cómo se obtendrían los beneficios del contrato inteligente:

```
// Extractor de los ethers del smart contract hacia la direccion del que derroto al jefe
function withdrawWinner() public payable onlyWinner(msg.sender) {
    // Transferimos el dinero al address del usuario que le dio el ultimo golpe al jefe
    address payable _winner = payable(msg.sender);
    _winner.transfer(address(this).balance);

    // Reseteamos la address del ganador
    winnerAddress = address(0);

    // Una vez reclamada la recompensa el boss revive
    TreeBoss storage boss = treeBoss;
    boss.hpPoints = boss.maxHpPoints;
}
```

Figura 4.16.- Función de obtención de beneficios

Esta función se ejecuta cuando el usuario decide reclamar la recompensa por haber derrotado a un jefe. Gracias al modificador *onlyWinner* que se explicó en puntos anteriores se pude conseguir que solamente el ganador ejecute la función.

Lo que se hace es utilizar la función transfer para pasar el balance desde la dirección del contrato hacia la dirección ganadora.

Como para este trabajo solo se ha realizado una demo, todo el dinero del contrato se le transferirá al jugador, pero lo más normal sería que solo un porcentaje se le diera como recompensa y que otra parte fuera transferida a la dirección del creador del contrato en su lugar para que así tanto creadores como jugadores tengan incentivos económicos.

Tras realizar esto se resetea la dirección del ganador y el jefe vuelve a su estado inicial para poder continuar con el juego.

En caso de que el juego estuviera en un entorno real lo más común seria tener varios diseños de múltiples jefes y que se fueran rotando periódicamente para añadir más variedad al videojuego.

Por último, las funciones externas que se utilizan para realizar verificaciones que no se hayan podido extraer en un modificador y que a su vez acaban llamando a las funciones internas de puntos anteriores como se puede ver en la Figura *4.17*:

```
// Verificación de crear un NFT
function createSkull(string memory _name) public payable {
    require(msg.value >= cost); // Comprobamos que el pago es suficiente
    _createSkull(_name);
}

// Verificación de subir de nivel un NFT
function levelUpSkull(uint256 _skullId) public payable onlySkullOwner(msg.sender, _skullId) {
    Skull storage skull = skulls[_skullId];
    require(msg.value >= (skull.level * skull.rarity) / 1000); // Comprobamos que el pago es suficiente
    _levelUpSkull(_skullId);
}

// Verificación de revivir un NFT
function reviveSkull(uint256 _skullId) public payable onlySkullOwner(msg.sender, _skullId) {
    Skull storage skull = skulls[_skullId];
    require(msg.value >= skull.rarity / 10); // Comprobamos que el pago es suficiente
    _reviveSkull(_skullId);
}

// Verificación de atacar al jefe
function attackSkull(uint256 _skullId) public payable onlySkullOwner(msg.sender, _skullId) {
    Skull storage skull = skulls[_skullId];
    require(msg.value >= (skull.rarity * skull.level) / 10000); // Comprobamos que el pago es suficiente
    _attackBoss(skull.id);
}
```

Figura 4.17.- Funciones externas para realizar comprobaciones

En cada una de estas funciones simplemente se debe verificar que el pago que se envía en la transferencia es suficiente para permitir la invocación de las mismas.

En el caso de crear un NFT, ya se comentó que la variable cost sería la que determinaría el coste de crear uno.

Para subir de nivel un NFT o para atacar al jefe el coste escalará con la rareza y el nivel de dicho NFT.

Para revivirlo dependerá solamente de la rareza, pero en una proporción mucho mayor que si lo que se quiere es subir de nivel.

4.3.5.- Funciones básicas para el despliegue.

Para el despliegue del contrato se utiliza el archivo *2_initial_migration* donde se contiene el código necesario para que el contrato inteligente se inicialice en la red blockchain que se haya determinado en el archivo de configuración de *Truffle* (se explicará en puntos posteriores):

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

module.exports = function (deployer) {
    deployer.deploy(
        GameContract,
        "SkullGame",
        "SG",
        "Strong Tree",
        100000,
        1000,
        "https://i.pinimg.com/564x/db/b3/d1/dbb3d1bc432dae5098b07edb449c87b9.jpg"
    );
};
```

Figura 4.18.- Código para desplegar el contrato inteligente

Lo primero que se hace es obtener el contrato que se quiere que se despliegue que en este caso es el *GameContract.sol* y después se define un desplegador (deployer) que recibe como primer parámetro el contrato inteligente que debe desplegar.

Opcionalmente se le añaden los parámetros a utilizar durante la inicialización del contrato, es decir, los parámetros que finalmente se acabaran pasando al constructor que se define dentro de *Game Contract*.

La vida y ataque del jefe se han decido teniendo en cuenta que un NFT de nivel 10 de la rareza máxima necesitaría aproximadamente 10 ataques (lo cual tendría un coste muy elevado) para poder derrotar al jefe muriendo en el proceso.

4.3.6.- Archivos estáticos de React.

Al haber empleado el framework React para la generación de la estructura básica de la aplicación web la estructura básica se ha creado dentro del archivo *index.html* el cual es donde se definen datos básicos de la página web como pueden ser el idioma principal que en este caso se definió en inglés, el título de la página y la raíz (root) de la aplicación.

Los otros archivos estáticos que React necesita son los siguientes:

- El *favico.ico*, es el icono que se visualizara en la pestaña de la página web cuando esta se inicie en el navegador.
- El *manifest.json*, contiene datos de estilos como pueden ser el fondo, el tema oscuro o claro o la URL de partida de la aplicación.
- El archivo *robots.txt*, contiene las restricciones que se quieren aplicar a los rastreadores de los buscadores dentro de nuestra página web, aunque en el caso de esta demo se han dejado los valores por defecto.

4.3.7.- Gestión de las imágenes y Styled Components.

Para la gestión de estilos de la aplicación se ha decidido emplear una aproximación mediante *Styled Components* que es una librería que permite escribir código *CSS* en *JavaScript*.

Estos componentes son una manera de estructurar mejor el css de una aplicación a través de un archivo *JavaScript* y que es una forma muy utilizada en *React* para mejorar la utilización de estilos.

Algunas de las principales ventajas que aporta utilizar esta forma de crear componentes ya estilizados son:

- Se evitan errores a la hora de asignar las clases css a los elementos *HTML*
- Reduce la cantidad de archivos css necesarios para aplicar el diseño deseado a la aplicación.
- Facilita el mantenimiento de la aplicación por equipos posteriores ya que se centraliza la mayoría del css en un archivo (aunque no es obligatorio que todo el css este aquí).
- Facilita la creación de estilos globales para los componentes de la aplicación.

La gestión de las imágenes se realiza mediante los archivos de la carpeta *Src* siendo el archivo *index.js* el cual contiene la renderización del Modelo de Objetos del Documento (DOM) y utilizando un provider de *Redux* se le pasa la información almacenada en el archivo *store.js*.

En la Figura *4.19*, se pueden ver las importaciones realizadas y como se emplea el store de redux dentro del archivo antes explicado:

```
import './index.css';
import "./styles/reset.css";

import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import reportWebVitals from './reportWebVitals';
import store from "./redux/store";

import { Provider } from "react-redux";

ReactDOM.render(
  <Provider store={ store }>
    <App />
  </Provider>,
  document.getElementById('root')
);

reportWebVitals();
```

Figura 4.19.- Renderizado del DOM de React

El archivo *index.css* contiene los estilos básicos de la aplicación como pueden ser el tipo de letra o el tamaño de la misma.

La verdadera funcionalidad de la aplicación se encuentra dentro del archivo *app.js* que se utiliza en forma de componente dentro del *index.js* básico.

Antes de empezar a explicar los archivos *App.js* y *App.css* se ha considerado más adecuado comenzar por otras partes de la aplicación para facilitar la comprensión de estos dos archivos.

La carpeta *assets* contiene todas las imágenes que se han generado para utilizar en el juego. Dentro de esta carpeta están otras tres que contienen el fondo del videojuego, los iconos de los NFT según su rareza y las imágenes para formar un NFT según su ADN.

Todas las imágenes de la carpeta anterior son utilizadas por el archivo *skullParts.js* que se encuentra dentro de la carpeta *parts.*

Este archivo simplemente se encarga de generar una constante que posea una serie de arrays para cada una de las partes que podría tener un NFT.

Tomando como ejemplo la parte *Crown* que sería la corona que puede tener uno de los NFT de este proyecto en la cabeza. En este caso todas las posibles coronas se almacenan en un array dentro de la constante a exportar para que cuando se vaya a renderizar el NFT, en función del ADN del mismo se le asigne una corona y otra de las que estén almacenadas en el array.

En la Figura 4.20, se puede ver el código de la estructura para el almacenamiento de las imágenes que luego se usaran para renderizar el NFT:

```
import n_2 from "../assets/parts/noses/2.png"
import n_3 from "../assets/parts/noses/3.png"
import n_4 from "../assets/parts/noses/4.png"
import n_5 from "../assets/parts/noses/5.png"

import e_0 from "../assets/parts/eyes/0.png"
import e_1 from "../assets/parts/eyes/1.png"
import e_2 from "../assets/parts/eyes/2.png"
import e_3 from "../assets/parts/eyes/3.png"
import e_4 from "../assets/parts/eyes/4.png"
import e_5 from "../assets/parts/eyes/5.png"

import c_0 from "../assets/parts/crowns/0.png"
import c_1 from "../assets/parts/crowns/1.png"
import c_2 from "../assets/parts/crowns/2.png"
import c_3 from "../assets/parts/crowns/3.png"
import c_4 from "../assets/parts/crowns/4.png"
import c_5 from "../assets/parts/crowns/5.png"

export const parts = {
    backs: [b_0, b_1, b_2, b_3, b_4, b_5],
    faces: [f_0, f_1, f_2, f_3, f_4, f_5],
    mouths: [m_0, m_1, m_2, m_3, m_4, m_5],
    noses: [n_0, n_1, n_2, n_3, n_4, n_5],
    eyes: [e_0, e_1, e_2, e_3, e_4, e_5],
    crowns: [c_0, c_1, c_2, c_3, c_4, c_5]
}
```

Figura 4.20.- Generación del vector de imágenes

La carpeta *contracts* es donde se almacenan los json que genera *Truffle* cuando se realiza la compilación de los contratos inteligentes que se tenga en el proyecto. Por defecto *Truffle* guarda esos ficheros leyendo del archivo *truffle-config.js* la propiedad:

```
contracts_build_directory: contractDirectory,
```

Figura 4.21.- Definición del directorio de salida de la compilación del contrato inteligente

Como se puede ver en la Figura 4.21, en el videojuego creado para esta investigación el *contractDirectory* es una variable de entorno que se ha configurado para que lleve a *./src/contracts/* para así tener una única carpeta donde almacenar todos estos archivos.

En la carpeta *styles* se almacenan todos los estilos que se van a usar en la aplicación más allá de los estilos iniciales que ya se han comentado anteriormente. Aquí se encuentran los archivos *reset.css* y *theme.css*.

El primero contiene estilos para utilizar en *index.js* en combinación con los de *index.css* y el segundo archivo contiene variables básicas para usar colores dentro de otros archivos css. Por último, el punto más importante de esta carpeta es *styles.js* que contiene los *styled components* de toda la aplicación.

4.3.8.- Gestión de datos mediante Redux (Reducers y Actions).

La carpeta *redux* contiene las carpetas en las que se ha implementado la estructura de Redux para el tratamiento de la información y las comunicaciones entre el front-end y la cadena de bloques de la aplicación.

El *archivo store.js* es donde se ha creado el store único de Redux se puede ver en la Figura *4.22*:

```
import { applyMiddleware, compose, createStore, combineReducers } from "redux"
import thunk from "redux-thunk";
import blockchainReducer from "./blockchain/blockchainReducer";
import dataReducer from "./data/dataReducer";

const rootReducer = combineReducers({
    blockchain: blockchainReducer,
    data: dataReducer
});

const middleware = [thunk];
const composeEnhancers = compose(applyMiddleware(...middleware));

const configureStore = () => {
    return createStore(rootReducer, composeEnhancers);
}

const store = configureStore();

export default store;
```

Figura 4.22.- Store de Redux para la gestión de la información

Lo primero que se hace es combinar el *reducer* de blockchain con el de data para generar así un reducer raíz. Después de esto se crea un *composeEnhancers* utilizando el middleware de Redux.

Una vez creados todos los valores anteriores ya se puede crear una función que genere el store de redux en base a los parámetros anteriores y exportar el store para poder utilizarlo en el código de *App.js.*

Una vez explicado cómo se crea el store de redux ahora se va a explicar cómo se han creado los reducers para cada uno de los dos tipos de datos que se tienen que manejar, así como las comunicaciones a realizar.

A continuación, se pasará a explicar que son los archivos *dataReducer.js* y *blockchainReducer.js*.

En el primero de ellos se define como se realiza la gestión de las peticiones que se realizan para la obtención de datos de la blockchain una vez ya se estableció la conexión con esta. Aquí lo que se hace es tomar las estructuras de la respuesta de la petición realizada a la cadena de bloques para actualizar la información que previamente tenía el front-end.

En el segundo se define como realizar la conexión a la cadena de bloques la primera vez que se inicia el juego y es donde se obtiene la información de la cuenta que se utilizó para conectar, el contrato del juego y la información general de *Web3*.

En la Figura *4.23*, se puede visualizar un *switch* que se emplea para gestionar los diferentes tipos de peticiones que se pueden realizar y como se debe manejar cada uno de ellos.

```
switch (action.type) {
    case "CONNECTION_REQUEST":
        return {
            ...initialState,
            loading: true
        };

    case "CONNECTION_SUCCESS":
        return {
            ...initialState,
            loading: false,
            account: action.payload.account,
            gameContract: action.payload.gameContract,
            web3: action.payload.web3

        };

    case "CONNECTION_FAILED":
        return {
            ...initialState,
            loading: false,
            errorMsg: action.payload
```

Figura 4.23.- Manejo de los diferentes tipos de peticiones

Los reducers solamente se encargan de gestionar las peticiones una vez que son enviadas o recibidas, pero para que una petición se desencadene primero se debe producir una acción en el juego.

La gestión de la creación de la petición se ha hecho a través de los archivos *dataActions.js* y *blockchainActions.js*.

En dataActions.js lo único que se hace es obtener la información que el contrato inteligente almacenaba de las estructuras de datos que contienen todo lo referente al jefe y a los NFT.

En este caso concreto se ha usado la función *call()* ya que los métodos del contrato se marcaron como view lo cual quiere decir que no van a modificar ningún dato del contrato, sería el equivalente a hacer una petición de tipo *GET* en *HTTP*, es decir, su función es solamente la obtención de información.

```
export const fetchData = (account, skullId) => {
  return async (dispatch) => {
    dispatch(fetchDataRequest());

    try {
      const allSkulls = await store.getState().blockchain.gameContract.methods.getSkulls().call();
      const allUserSkulls = await store.getState().blockchain.gameContract.methods.getUserSkulls(account).call();
      const bossInfo = await store.getState().blockchain.gameContract.methods.getBossInfo().call();
      const selectedSkull = allSkulls[skullId ? skullId.toString() : skullId];

      dispatch(fetchDataSuccess({ allSkulls, allUserSkulls, bossInfo, selectedSkull }));
    }

    catch (error) {
      dispatch(fetchDataFailed("Error al cargar los datos del smart contract"));
    }

  }
```

Figura 4.24.- Obtención de información para el Store

En la Figura *4.24*, es importante comentar que las acciones incluyen accesos al store para conseguir el contrato puesto que la ejecución de las acciones clasificadas dentro del archivo de data es siempre posterior a la conexión inicial con la cadena de bloques que se realiza siempre mediante el archivo *blockchinActions.js* que explicare a continuación.

En *blockchainActions.js* lo que se maneja es la creación de las peticiones de conexión a la cadena de bloques y no de las funciones propias del contrato que esta desplegado en esta (de esa parte se encarga *dataActions.js*).

Lo primero que alberga este archivo son las funciones de gestión de los estados de las peticiones como se puede ver en la Figura *4.25*:

```
const connectRequest = () => {
    return {
        type: "CONNECTION_REQUEST"
    };
}

const connectSuccess = (payload) => {
    return {
        type: "CONNECTION_SUCCESS",
        payload: payload
    };
}

const connectFailed = (payload) => {
    return {
        type: "CONNECTION_FAILED",
        payload: payload
    };
}

const updateAccountRequest = (payload) => {
    return {
        type: "UPDATE_ACCOUNT",
        payload: payload
    };
}
```

Figura 4.25.- Gestión de las peticiones de conexión

La única función que se exporta de este archivo y que por tanto será utilizada fuera de él es *connect* ya que será la función a llamar cuando el usuario pulse en el botón de conectar.

Una vez eso ocurra lo primero que hay que hacer es indicar que se ha pasado al estado de pedir conexión lo cual se gestiona con la función *connectRequest* del propio archivo.

```
if (window.ethereum) {
    let web3 = new Web3(window.ethereum);

    try {
        await window.ethereum.enable();
        const accounts = await window.ethereum.request({ method: "eth_accounts" });
        const networkId = await window.ethereum.request({ method: "net_version" });
```

Figura 4.26.- Conexión con el provider

Después lo que se hace es comprobar que se posee un proveedor de acceso a blockchain que normalmente se espera que sea *MetaMask* y en caso de no tenerlo se retorna un error como se puede ver en la Figura *4.26*.

También es necesario activar el proveedor ya que, aunque esto no es obligatorio hay veces en que la conexión no se consigue de forma automática y causa problemas por incompatibilidades.

Durante el desarrollo de este trabajo se utilizó Windows 11 en su *versión 21H2* y en este caso sí que detecta la existencia del proveedor, pero no llega a realizar su activación de manera correcta por lo que es necesario añadir esa línea de código.

Tras eso se obtienen las cuentas que estén conectadas al proveedor (provider) y el id de la red a la que se encuentre conectado. En la Figura *4.27*, se pueden ver las cuentas de ejemplo que se han empleado durante el desarrollo de este trabajo:

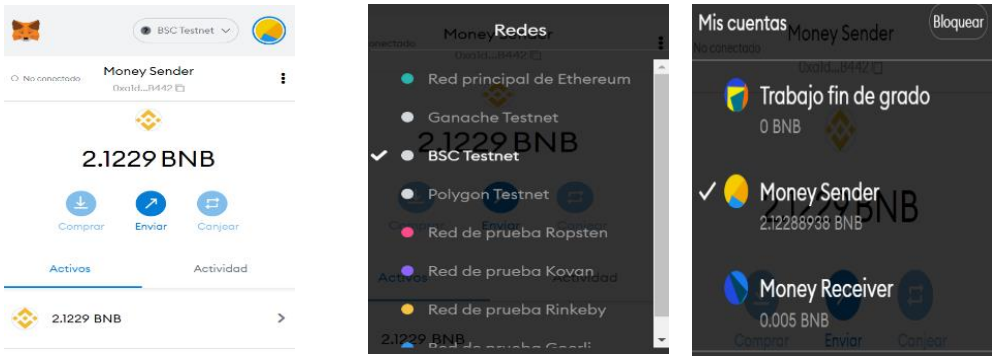

Figura 4.27.- Ejemplo de uso de provider

Tomando como ejemplo el proveedor, red y cuenta utilizados para este proyecto lo que se obtendría en este caso serian todas las redes y cuentas que se encuentren vinculadas a esta cuenta de *MetaMask.*

Después de obtener esos datos se debe limitar el acceso a que red se quiere ir utilizando el id de la red en la que se desplego en contrato inteligente como se ve en la Figura *4.28*:

```
// Ganache -> 5777
// BSC Testnet -> 97
// Polygon Testnet -> 80001
if (networkId === '97') {
    const networkData = GameContract.networks[networkId]; // Obtenemos 
    const abi = GameContract.abi;
    const address = networkData.address;
    const deployedContract = new web3.eth.Contract(abi, address);

    dispatch(
        connectSuccess({
            account: accounts[0],
            gameContract: deployedContract,
            web3: web3
        })
    );
```

Figura 4.28.- Gestión de la información para el despliegue

Una vez obtenidos todos los datos necesarios para poder acceder a la dirección del contrato y almacenarlo en una variable, lo que se hace es enviarle esa información al reducer junto con la confirmación de que la conexión fue exitosa.

Además de esto se crean funciones que escuchen posibles cambios que el provider comunique. Todo esto se puede ver en la Figura *4.29*:

```
window.ethereum.on("accountsChanged", (accountsChanged) => {
    dispatch(updateAccountRequest(accountsChanged[0]));
});

window.ethereum.on("chainChanged", () => {
    window.location.reload();
});
```

Figura 4.29.- Funciones para cuando se produce un cambio de cuenta o de blockchain

4.3.9.- Componentes genéricos del juego.

En la carpeta de componentes se encuentran tres componentes genéricos que se pueden reutilizar en varias pantallas y son:

- HealthBar: Este componente recibirá dos parámetros opcionales que serán la vida máxima y la vida actual para poder generar una barra de vida en base a ellos.
- Loading Indicator: Aquí estaría el componente que se utilizó en la pantalla de cargando y que contiene el JavaScript y css necesario para poner la animación del icono de cargando.
- Renders: Aquí estarían los renders del NFT y del jefe que generan el contenido de las imágenes de esos dos elementos del juego.

En la Figura *4.30*, se puede ver como se ha implementado el componente *SkullRenderer* el cual se usa para la renderización de la imagen en base al ADN del NFT:

```
import React from "react";
import { parts } from "../../parts/skullParts";

import _r1 from "../../assets/rarity/1.png";
import _r2 from "../../assets/rarity/2.png";
import _r3 from "../../assets/rarity/3.png";

const SkullRenderer = ({ skull = null, size = 200, style }) => {
    if (!skull) return null;

    const rarity = assignRarity(skull.rarity);
    const dna = assignDna(skull.dna);
    const skullDetail = {
        back: dna.substring(0, 2) % 5,
        face: dna.substring(0, 2) % 5,
        mouth: dna.substring(0, 2) % 5,
        nose: dna.substring(0, 2) % 5,
        eye: dna.substring(0, 2) % 5,
        crown: dna.substring(0, 2) % 5,
        name: skull.name
    }
    const skullStyle = {
        width: "100%",
        height: "100%",
        position: "absolute"
    }
```

Figura 4.30.- Obtención de las propiedades del NFT

Lo que se hace es combinar los primeros valores del ADN para obtener un número que como máximo será de 5 el cual es el mayor valor del array de posibles imágenes para mostrar para cada una de las propiedades como se ve en la Figura *4.31*.

```
return (
    <div style={{ minWidth: size, minHeight: size, background: "blue", position: "relative", ...style }}>
        <img alt={"back"} src={parts.backs[skullDetail.back]} style={skullStyle} />
        <img alt={"face"} src={parts.faces[skullDetail.face]} style={skullStyle} />
        <img alt={"mouth"} src={parts.mouths[skullDetail.mouth]} style={skullStyle} />
        <img alt={"nose"} src={parts.noses[skullDetail.nose]} style={skullStyle} />
        <img alt={"eye"} src={parts.eyes[skullDetail.eye]} style={skullStyle} />
        <img alt={"crown"} src={parts.crowns[skullDetail.crown]} style={skullStyle} />
        <img alt={"rarity"} src={rarity} style={skullStyle} />
    </div>
);
```

Figura 4.31.- Generación de la imagen del NFT

La rareza asignara una imagen en función de si se supera valores limite que están definidos en 80% y 95% de rareza. Esta funcionalidad se puede ver representada en el código de la Figura *4.32*:

```
function assignRarity(skullRarity) {
    if (skullRarity >= 95) {
        return _r3;
    }

    if (skullRarity >= 80) {
        return _r2;
    }

    return _r1;
}

function assignDna(skullDna) {
    let functionDna = String(skullDna);

    while (functionDna.length < 16) functionDna = "0" + functionDna;

    return functionDna;
}
```

Figura 4.32.- Asignación de la rareza del NFT

4.3.10.- Renderización e interacciones con el juego.

El archivo *App.js* es el que contiene toda la gestión de la renderización del juego, así como la asignación de las funciones a los diferentes elementos visuales que se muestran como por ejemplo los botones.

Lo primero que se hace es iniciar constantes para la gestión de variables globales del proyecto como *loading*, *bossFightState* y *bossDefeatedState* cuyo valor será false por defecto y que se usaran para saber cuándo es necesario pasar de una pantalla a la siguiente:

```
// Carga de la información
const dispatch = useDispatch();
const data = useSelector((state) => state.data);
const blockchain = useSelector((state) => state.blockchain);
const [loading, setLoading] = useState(false);
const [bossFightState, setBossFightState] = useState(false);
const [bossDefeatedState, setBossDefeatedState] = useState(false);
```

Figura 4.33.- Inicialización de estructuras y servicios necesarios para obtener información

También se inicializan las constantes *data* y *blockchain* desde el estado en que este la aplicación el cual como ya se explicó antes, será gestionado por Redux.

Al final de este archivo se retornar la vista que se tiene que mostrar lo cual será gestionado por la función *renderView* y se colocará dentro de un styled component que mostrar el fondo del juego con la imagen que este en la carpeta *background* dentro de *assets*:

```
// De forma similar a componentDidMount y componentDidUpdate
useEffect(() => {
  if (blockchain.account !== "" && blockchain.gameContract !== null) {
    dispatch(fetchData(blockchain.account));
  }
}, [blockchain.gameContract]); // eslint-disable-line react-hooks/exhaustive-deps

// Visualización del videojuego
return (
  <globalStyle.Screen image={_background}>
    {
      renderView()
    }
  </globalStyle.Screen>
);
```

Figura 4.34.- Código para poner el fondo y seleccionar pantalla

En la Figura *4.34*, se muestra como el *useEffect* se utiliza para producir efectos secundarios dentro del ciclo de vida de una aplicación en React y en este caso se utiliza para actualizar la información de la cuenta conectada al videojuego.

La función *renderView* retorna la vista que corresponde según las variables globales del juego como se ve en la Figura *4.35*:

```
// Función para seleccionar que vista se debe renderizar
const renderView = () => {
  // Renderizar la animacion de cargando
  if (loading) {
    return renderLoadingScreen();
  }

  // Renderizar la pantalla de conexion
  if (blockchain.account === "" || blockchain.gameContract === null) {
    return renderConnectScreen();
  }

  // Renderizar la pantalla de seleccion
  if (!bossFightState && !bossDefeatedState) {
    return renderCharacterSelectionScreen();
  }

  // Renderizar la pantalla de pelea
  if (bossFightState && !bossDefeatedState) {
    return renderFightBossScreen();
  }

  return renderDefeatedBossScreen();
};
```

Figura 4.35.- Gestión de las pantallas según el estado del juego

En caso de que la variable loading este activa se debe mostrar la animación de carga del videojuego, en caso de no tener cuenta conectada o contrato desplegado entonces se mostrará la pantalla de conectar, si no se está peleando con el jefe ni se lo ha derrotado se mostrar la pantalla se selección de NFT, si se está peleando con el jefe pero

aún no ha sido derrotado se mostrará la pantalla de pelea con el jefe y en caso de que el jefe este derrotado y se esté peleando con el entonces se debe mostrar la pantalla de victoria.

La función de renderizar la pantalla de conexión retornara una pantalla con el botón que activara la función de conexión que se exporto en *blockchainActions.js* como se ve en la Figura *4.36*.

```
// Funcion para renderizar la pantalla de conectar cuenta
const renderConnectScreen = () => {
  return (
    <globalStyle.Container flex={1} ai={"center"} jc={"center"}>
      <globalStyle.TextTitle>Welcome to Skull Game</globalStyle.TextTitle>
      <globalStyle.SpacerSmall />

      <globalStyle.Button1 onClick={(e) => { e.preventDefault(); dispatch(connect()); }}>Connect</globalStyle.Button1>
      <globalStyle.SpacerSmall />

      {blockchain.errorMsg !== "" ? (<globalStyle.TextDescription> {blockchain.errorMsg} </globalStyle.TextDescription>) : null}
    </globalStyle.Container>
  );
}
```

Figura 4.36.- Renderizado de la pantalla de conexión

El aspecto visual de la pantalla de conexión se puede ver en la Figura *4.37*:

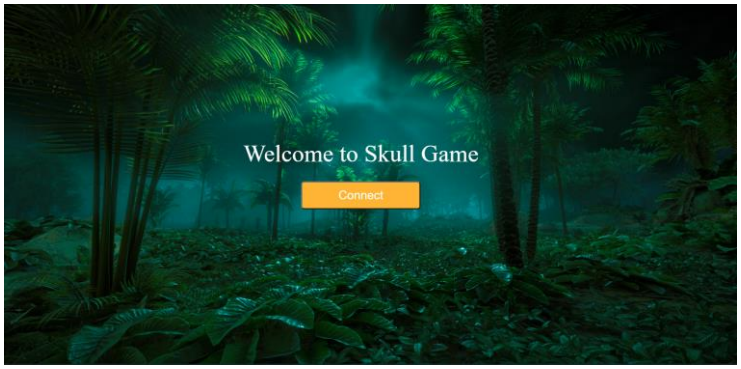

Figura 4.37.- Pantalla de conexión inicial

Mientras alguna transacción o petición se esté cargando se llamará en su lugar a *renderLoadingScreen* donde se utiliza el componente loading indicator el cual es un componente genérico. Esto se puede ver en la Figura *4.38*.

```
// Funcion para renderizar la pantalla de loading
const renderLoadingScreen = () => {
  return (
    <globalStyle.LoadingContainer ai={"center"} style={{ padding: "24px" }}>
      <globalStyle.TextTitle>Processing Transaction</globalStyle.TextTitle>
      <LoadingIndicator />
    </globalStyle.LoadingContainer>
  );
}
```

Figura 4.38.- Código para renderizar la pantalla de cargando

El aspecto final de la pantalla de carga se puede ver en la Figura *4.39*:

Figura 4.39.- Pantalla de procesar transacciones

Una vez se procese la transacción se pasará a la pantalla se selección de NFT la cual se puede visualizar en la Figura *4.40*.

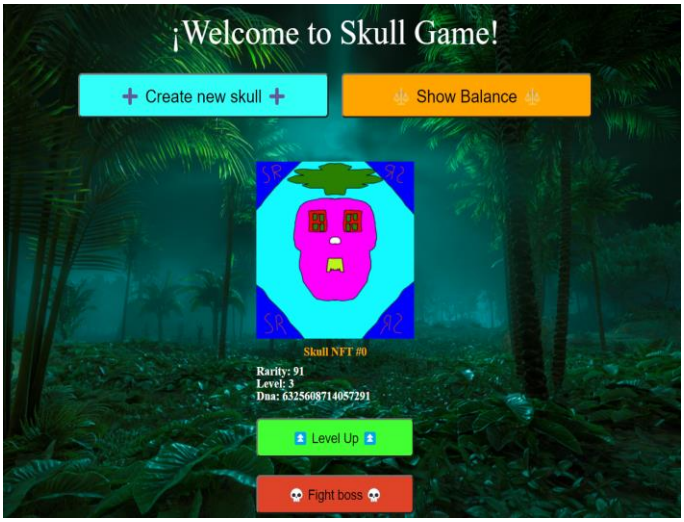

Figura 4.40.- Pantalla de selección de NFT

La función para generar esta pantalla seria *renderSelectionCharacterScreen* la cual queda representada en la Figura *4.41* y en la Figura *4.42*.

```
return (
  <globalStyle.Container ai={"center"} style={{ padding: "24px" }}>
    <globalStyle.TextTitle> ¡Welcome to Skull Game! </globalStyle.TextTitle>
    <globalStyle.SpacerSmall />

    <globalStyle.ButtonContainer>
      <globalStyle.Button2 onClick={(e) => {
        const name = "Skull NFT"
        e.preventDefault();
        mintNFT(blockchain.account, name)
      }}
      >
        ➕ Create new skull ➕
      </globalStyle.Button2>

      <globalStyle.Button4 onClick={(e) => { e.preventDefault(); balanceSmartContract(); }}>🏦 Show Balance 🏦</globalStyle.Button4>

    </globalStyle.ButtonContainer>

    <globalStyle.SpacerMedium />

    <globalStyle.Container jc={"center"} fd={"row"} style={{ flexWrap: "wrap" }} >
      {data.allUserSkulls.map((skull, index) => {

        return (
          <globalStyle.Container key={index} style={{ padding: "15px" }} >
            <SkullRenderer skull={skull} />
            <globalStyle.SpacerXSmall />

            <globalStyle.Container
```

Figura 4.41.- Estructura para la parte superior de la pantalla de selección de NFT

```
<globalStyle.Container jc={"center"} fd={"row"} style={{ flexWrap: "wrap" }} >
  {data.allUserSkulls.map((skull, index) => {

    return (
      <globalStyle.Container key={index} style={{ padding: "15px" }} >
        <SkullRenderer skull={skull} />
        <globalStyle.SpacerXSmall />

        <globalStyle.Container>
          <globalStyle.nameTitle>{skull.name} #{skull.id}</globalStyle.nameTitle>
          <globalStyle.SpacerXSmall />

          <globalStyle.TextDescription>Rarity: {skull.rarity}</globalStyle.TextDescription>

          <globalStyle.TextDescription>Level: {skull.level}</globalStyle.TextDescription>

          <globalStyle.TextDescription>Dna: {skull.dna}</globalStyle.TextDescription>
          <globalStyle.SpacerXSmall />

          {
            skull.hpPoints === "0" ? <div></div> : <globalStyle.Button3 onClick={(e) => { e.preventDefault(); levelUpSkull(blockchain.account,
          }
          {
            skull.hpPoints === "0" ?
              <globalStyle.Button9 onClick={(e) => { e.preventDefault(); reviveSkull(blockchain.account, skull); }}> 🔥 Revive Skull 🔥 </glob
              :
              <globalStyle.Button5 onClick={(e) => { e.preventDefault(); startBossFight(blockchain.account, skull); }}> 💀 Fight boss 💀 </gl
          }
```

Figura 4.42.- Estructura para la parte inferior de la pantalla de selección de NFT

Con esta estructura se generan primero dos botones en la parte superior, uno para crear un nuevo NFT y otro para mostrar el balance actual del contrato.

Después se recorre el array de NFT para generar cada uno de ellos utilizando primero el *skullRenderer* para crear la imagen que lo representa y después se muestran debajo algunos datos básicos como el nombre o su ADN.

Los botones que tiene cada NFT dependen del estado de este, en caso de que no tenga puntos de vida solo se podrá intentar revivirlo. En caso de tener vida se podrá tanto atacar al jefe como subir de nivel el NFT.

En el botón superior para crear el NFT llama a la función *mintNFT* en la cual se utiliza la función send porque es necesario enviar el pago que se requiere para realizar dicha acción y por ello se envía un from con la dirección de la cuenta a la que se le debe retirar el dinero y el valor a retirar, para esto último se utiliza la función *toWei* de *Web3* a la que se le pasa una cantidad de Ether y la convierte a Wei.

Cuando se realiza la llamada también se le asigna valor true a la variable global de loading mediante su función de set para que aparezca la pantalla de cargando mientras se realiza la transacción.

Por otra parte, el segundo botón superior simplemente hace una llamada para comprobar el balance por lo que en lugar de send se utiliza call y simplemente se muestra el valor por pantalla en una alerta.

Pasando a explicar las funciones de los botones del NFT que son las siguientes:

- LevelUpSkull: Esta función determina el coste que tiene que enviarle al contrato en base al NFT que la invoque y después hace una petición al método de subir de nivel del contrato.
- ReviveSkull: Funciona de la misma manera que *LevelUpSkull*, pero en lugar de llamar a la función de subir de nivel lo hace con la de revivir al NFT.
- StartBossFight: Esta función marca el NFT que la invoco como seleccionado y cambia el estado de la variable global de combate a true para que la pelea con el jefe inicie.

Cuando el usuario elige pelear con el jefe se inicia la pantalla de pelea la cual se visualiza de la siguiente manera:

Figura 4.43.- Ejemplo de batalla contra jefe

En la Figura *4.43*, se puede ver que el NFT tiene un botón para atacar que llevara a una transacción y uno de huir que simplemente retornara la variable de pelea a false y retornara a la selección de personaje dejando reseteado el valor del personaje seleccionado puesto que se debe seleccionar de nuevo en caso de querer pelear con el jefe.

En la figura 5.48, se puede ver la función para atacar al jefe hará una llamada a la cadena de bloques y recibirá si el jefe fue derrotado durante el combate cambiando el valor de la variable global para mostrar la batalla de victoria en caso de que el jefe muriera:

```
// Iniciar un ataque contra el boss
const attackBoss = async (_account, _item) => {
  const attackBossCost = _item.rarity * _item.level / 10000;

  setLoading(true);

  blockchain.gameContract.methods.attackSkull(_item.id).send(
    {
      from: _account,
      value: blockchain.web3.utils.toWei(attackBossCost.toString(), "ether")
    }
  )
    .once("error", () => setLoading(false))
    .then((response) => {
      setLoading(false);
      setBossDefeatedState(response.events.BossAttacked.returnValues.isBossDefeated);
      dispatch(fetchData(blockchain.account, _item.id));
    });
};
```

Figura 4.44.- Función asignada al botón de atacar al jefe

Además de la renderización del NFT que es muy similar a la ya usada anteriormente durante la selección, la novedad en esta pantalla es el jefe el cual se genera

mediante el bossRenderer que básicamente toma la imagen de la url que forma parte de los datos del jefe y la muestra adaptada al tamaño del contenedor en que se encuentra.

Otra novedad aquí serían las barras de salud que se han creado mediante el componente genérico *HealthBar*.

Lo último a comentar sobre la renderización de pantallas seria que la pantalla de victoria simplemente será la misma que la de pelea con el jefe, pero mostrando un único botón en la parte de arriba para reclamar el premio por derrotar al jefe para después devolver al jugador a la pantalla de selección de personaje.

4.3.11.- Gestión de variables de entorno y configuración de Truffle.

La gestión de las variables de entorno se realiza en el archivo *.env* que se encuentra en la raíz del proyecto contiene los valores de las variables de entorno a utilizar. En caso de hacer un despliegue en algún entorno de preproducción o producción sería necesario crear archivos nuevos para utilizar en cada caso.

Se ha decidido que la gestión de los entornos se haga a través de la librería *Dotenv* mientras que la gestión de las carteras se ha usado el *HDWalletProvider* de *Truffle* para cuando se quiera desplegar en una cadena de bloques que no sea local como *Ganache* sino por ejemplo en la red de prueba de *Binance Smart Chain* (BSC). En la Figura *4.45*, se puede visualizar la definición de las mismas:

```
// Requerir dotenv para la gestion de variables de entorno
const dotenv = require('dotenv');

// Instalación HDWalletProvider: npm i @truffle/hdwallet-provider
const HDWalletProvider = require('@truffle/hdwallet-provider');

// Activamos la configuración de dotenv para la gestion de las variables de entorno
dotenv.config();

// Clave secreta: 12 palabras de la Wallet
const mnemonic = process.env.REACT_APP_MNEMONIC;

const ganacheTestnetUrl = process.env.REACT_APP_GANACHE_TESTNET_URL;
const bscTestnetUrl = process.env.REACT_APP_BSC_TESTNET_URL;
const maticTestnetUrl = process.env.REACT_APP_MATIC_TESTNET_URL;
const contractDirectory = process.env.REACT_APP_CONTRACTS_DIRECTORY;
```

Figura 4.45.- Utilización de las variables de entorno para la configuración de Truffle

Todo esto se trata dentro del archivo *Truffle-config.js* que es el que utiliza *Truffle* cada vez que se hace un despliegue del contrato y que nos permite también gestionar las redes posibles, en el caso de este proyecto se han configurado una red local de *Ganache*, la red de pruebas de BSC y la red de pruebas de Polygon (MATIC).

5. Propuesta de despliegue

En este capítulo se van a tratar los cambios que el juego necesita para poder ser desplegado en un entorno real y para complementar el estudio del funcionamiento de una blockchain real se va a crear una blockchain de prueba aunque la integración con el juego no se realizara de forma práctica.

5.1.- Cambios necesarios para un despliegue

Para el despliegue del videojuego se ha considerado *Docker* como la mejor alternativa ya que permite la creación de imágenes dentro de un contenedor. [51]

La problemática que normalmente se tiene es que para probar y desplegar la aplicación lo normal será tener un servidor propio que haga de host para el videojuego aunque luego el "guardado de datos" se haga a través de blockchain y en ese servidor es difícil replicar exactamente el mismo entorno que se utilizó para el desarrollo de la aplicación lo cual puede causar problemas.

Otra opción en caso de no querer utilizar *Docker* podría ser *Heroku* la cual es un PasS (Platform as a Service) que permite la realización de despliegues de aplicaciones aunque cabe resaltar que podría causar ciertos problemas debido a que este *PasS* está pensado para funcionar con una estructura de aplicación web clásica con un front-end y un back-end en el cual este último utiliza una base de datos clásica y no una cadena de bloques.[52]

Además de esto para un despliegue real sería necesario emplear una red principal (mainnet) de alguna blockchain o crear una blockchain propia.

Tomando como ejemplo la *Binance Smart Chain* cuya red de test se ha utilizado para este trabajo, los cambios a realizar serian:

- Sustituir la RPC url a utilizar por https://bsc-dataseed.binance.org/
- Sustituir el id de la cadena a usar por 56 el cual es el identificador de la red principal de *Binance*.

[51] https://www.docker.com/
[52] https://www.heroku.com/

- Cambiar el explorador de bloques para utilizar el *bscscan* de la red principal.

También habría que utilizar dinero real puesto que las faucet como las empleadas para este trabajo dejarían de ser validas y sería recomendable modificar el juego para que el creador del contrato obtuviera parte de los beneficios cuando un jugador ganase.

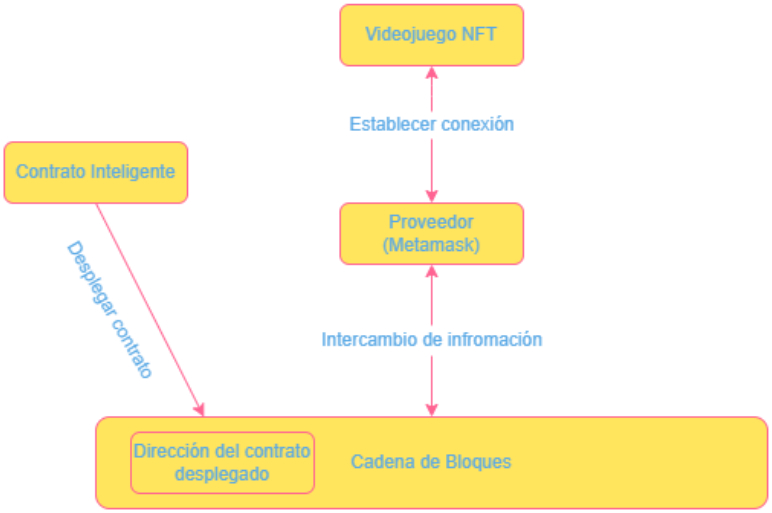

Figura 5.1.- Diagrama de comunicaciones en una blockchain real

En la Figura *5.1*, se puede ver que el diagrama de comunicaciones del juego no varía mucho respecto del planteado para la demo más allá de lo comentado antes con las *faucets*.

Para la obtención de monedas necesarias para trabajar en un entorno real se pueden utilizar muchos métodos siendo los más comunes el comprarlas con dinero Fiat empleando algún mercado de intercambios (*Exchange*) o mediante minería directa aunque para este último método se necesitaría un ordenador con gran capacidad de cómputo.

5.2.- Creación de una blockchain

Para complementar el punto anterior referente a un potencial despliegue en un entorno real, se ha desarrollado una cadena de bloques de ejemplo utilizando el lenguaje de programación Python puesto que es un lenguaje de programación que se caracteriza por ser sencillo y fácil de aprender. Además de esto también posee una gran cantidad de librerías que facilitan el desarrollo de una blockchain.

Esta blockchain utiliza el protocolo de prueba de trabajo ya que es el mecanismo de consenso pionero dentro de esta tecnología como se comentó en la parte de protocolos de consenso.

5.2.1.- Librerías e importaciones necesarias

Para la creación y utilización de la blockchain se han utilizado las siguientes librerías:

- *Datetime*: Se emplea para el manejo de fechas y horas. En la blockchain desarrollada se ha empleado para la generación del timestamp que necesita cada bloque.
- Hashlib: Se utiliza para tener acceso a cifrados desde Python. En este caso se utilizó para poder emplear el algoritmo SHA-256.
- Flask: Este microframework se emplea para facilitar el desarrollo de aplicaciones web en Python y en el caso de este proyecto se usará para crear una pequeña aplicación de ejemplo que permita probar la blockchain.
- Flask-ngrock: Permite utilizara desde internet aplicaciones que se tengan desplegadas en nuestro dispositivo local. Se utilizará para poder simular que se tienen varios nodos funcionando.
- Json: Esta librería nos da acceso a funciones para el formateo de objetos en formato Json lo cual será necesario para enviar información a través del protocolo HTTP.
- Urllib: Se utilizará para parsear adecuadamente las URL de los diferentes nodos que se conecten a la cadena de bloques.
- Uuid: Se empleará para la generación de la dirección del nodo que funcionará como aplicación de prueba.

- Requests: Para facilitar el trabajo con peticiones HTTP dentro de la aplicación de prueba de la blockchain.

5.2.2.- Definición de clases de la blockchain

Para facilitar el trabajo con los campos de la blockchain se han definido previamente las clases que se pueden ver en la Figura *5.2*:

```
[ ]  class SkullTransaction:

        def __init__(self, sender, receiver, amount):
            self.sender = sender
            self.receiver = receiver
            self.amount = amount

        def toJSON(self):
          return json.dumps(self, default=lambda o: o.__dict__, sort_keys=True)
```

```
[ ]  class SkullBlock:

        def __init__(self, sender, receiver, amount, previous_hash, transactions):
          self.index = sender
          self.timestamp = receiver
          self.nounce = amount
          self.previous_hash = previous_hash
          self.transactions = transactions

        def toJSON(self):
            return json.dumps(self, default=lambda o: o.__dict__, sort_keys=True)
```

Figura 5.2.- Clases de la blockchain

La clase *SkullTransaction* representa las transacciones de la cadena de bloques que deben tener como mínimo una dirección de destino, una dirección de origen y una cantidad a transferir.

Se ha definido un constructor para cuando se necesiten crear este tipo de transacciones y un método para convertir el objeto a formato JSON lo cual será necesario para cuando se quiera enviar las respuestas mediante HTTP.

Por su lado, la clase *SkullBlock* representa cada uno de los bloques que se conforma la cadena de bloques y sigue el formato que se explicó en el punto 2.4 de este trabajo.

Por esto es necesario que cada bloque contenga un índice que indica su posición dentro de la cadena, el hash del bloque anterior (para el bloque génesis el hash anterior

será 0), el nounce, las transacciones que se van a llevar a cabo en ese bloque y la estampa de tiempo para dificultar la prueba de trabajo.

También se ha incluido el mismo método para transformar a formato JSON que en la clase de SkullTransaction ya que también será necesario enviar respuestas con el contenido del bloque.

5.2.3.- Constructor de la blockchain y generación de bloques

La clase *SkullBlockchain* representará la cadena de bloques que se va a crear teniendo como propiedades un array de objetos tipo *SkullBlock*, un array de objetos *SkullTransaction* que serán las transacciones pendientes y que funcionará como una piscina (*pool*) donde se guardaran las transacciones que se han recibido por parte de la blockchain pero que aún no se han introducido en un bloque minado y por último un set que contendrá las direcciones de los nodos que hay conectados a la blockchain.

```python
class SkullBlockchain:
    skull_chain = []
    pending_transactions = []
    nodes = set()

    def __init__(self):
        # Generacion del bloque genesis
        self.generate_block(nounce=1, previous_hash='0')


    def generate_block(self, nounce, previous_hash) -> SkullBlock:
        # Inicializamos un bloque nuevo
        new_block = SkullBlock(len(self.skull_chain) + 1, str(datetime.datetime.now()), nounce, previous_hash, self.pending_transactions)

        # Añadimos el bloque a la cadena
        self.skull_chain.append(new_block)

        # Limpiamos las transacciones pendientes ya que se han metido al bloque minado
        self.pending_transactions = []

        return new_block
```

Figura 5.3.- Constructor y función generadora de bloques

En la Figura *5.3*, se pueden ver las propiedades antes explicadas así como el constructor propio de la clase el cual se encarga de generar el bloque original de la cadena para lo cual utiliza la función *generate_block*.

Esta función se emplea para generar los bloques de la cadena y en ella se utiliza el constructor de la clase *SkullBlock* pasándole la longitud de la cadena más uno como índice ya que será el siguiente bloque, como estampa de tiempo se emplea la hora actual al momento de crear el bloque y se introducen las transacciones que el bloque va a tener.

También se añaden el *nounce* y el hash del bloque anterior pero es importante tener en cuenta que estos datos se le pasan como parámetro a la función ya que se calcularan previamente mediante funciones que se explicaran más adelante.

Finalmente se añade el bloque a la cadena y se limpia el *pool* de transacciones pendientes ya que esas transacciones acaban de ser introducidas en el bloque.

5.2.4.- Funciones para obtener los datos del bloque

Para poder obtener parámetros necesarios para la creación del bloque se han definido las siguientes funciones auxiliares:

```python
def generate_hash_block(self, block) -> str:
    encoded_block = block.toJSON().encode()
    hash_block = hashlib.sha256(encoded_block).hexdigest()
    return hash_block

def get_previous_block(self) -> SkullBlock:
    return self.skull_chain[-1]

def get_nounce(self, previous_nounce) -> int:
    new_nounce = 1

    while True:
        hash_result = hashlib.sha256(str(new_nounce**2 - previous_nounce**2).encode()).hexdigest()

        if hash_result[:5] != '00000':
            new_nounce += 1

        else:
            return new_nounce
```

Figura 5.4.- Funciones para obtener el nounce y los hashes

Como se puede ver en la Figura *5.4*, la función *get_previous_block* retorna el bloque se la última posición de la cadena el cual es el anterior al siguiente que se quiera generar.

La función *get_nounce* se utiliza para obtener el número para el cual el hash resultante tenga el número objetivo de 0 al inicio que en este caso será siempre 5 ya que en blockchains más complejas ese valor varía según la cantidad de usuarios que intentan minar bloques.

Esta función emplea un bucle de búsqueda de números en el cual se va probando sucesivamente con posibles nounce hasta encontrar aquel que genera un hash objetivo con el suficiente numero de 0 iniciales.

Para la obtención del hash objetivo se está utilizando la diferencia de cuadrados como función para pasarle al algoritmo SHA-256 pero podría ser cualquier función no simétrica en su lugar. También se utiliza la función *hexdigest* para obtener un String con los dígitos hexadecimales del hash obtenido y así poder comprobar el número inicial de ceros.

5.2.5.- Funciones para añadir transacciones y nodos

Toda blockchain tiene que tener la posibilidad de añadir transacciones y nuevos nodos que se conecten para trabajar con la cadena. En la blockchain que se ha desarrollado esto se hace a través de las funciones *add_transaction* y *add_node*.

```
def add_transaction(self, sender, receiver, amount) -> int:
    # Inicializamos una transaccion nueva
    new_transaction = SkullTransaction(sender, receiver, amount)

    # Añadimos la transaccion a la lista de pendientes
    self.pending_transactions.append(new_transaction)

    previous_block = self.get_previous_block()

    return previous_block.index + 1

def add_node(self, address):
    parsed_url = urlparse(address)
    self.nodes.add(parsed_url.netloc)
```

Figura 5.5.- Funciones para añadir nodos y transacciones

Como se puede ver en la Figura *5.5*, añadir una transacción no es más que crear un objeto SkullTransaction con los parámetros recibidos y añadirla al array de transacciones pendientes.

Para verificar su correcto funcionamiento se va a retornar el índice del bloque actual para que el usuario pueda saber a qué número de bloque será añadida la transacción que ha añadido.

Por otro lado, la función de *add_node* utiliza *urlparse* para obtener la url del nodo que se quiere añadir para luego añadirla al set de nodos que contiene la blockchain donde se almacenaran para su posterior utilización.

5.2.6.- Validación de la blockchain

En la tecnología blockchain cada nodo tiene su propia versión de la cadena de bloques y para poder saber si esa cadena es válida para cuando tenga que ser tomada por el resto de nodos como la nueva cadena más larga, como ya se explicó en detalle en el punto 2.4.4.

```python
def is_valid_chain(self, chain) -> bool:
    previous_block = chain[0]
    current_block = chain[1]

    for block_index in range(2, len(chain)):
        if current_block.previous_hash != self.generate_hash_block(previous_block):
            return False

        hash_operation = hashlib.sha256(
            str((current_block.nounce)**2 - (previous_block.nounce)**2).encode()).hexdigest()

        if hash_operation[:4] != '0000':
            return False

        previous_block = current_block
        current_block = chain[block_index]

    return True
```

Figura 5.6.- Función de validación de cadenas

En la Figura *5.6*, se puede ver la función *is_valid_chain* donde se realizará la comprobación de que la cadena recibida por parámetro es correcta.

Para ello lo que se hace es obtener el primer bloque como bloque previo y el segundo bloque como bloque actual para comparar si el hash previo que está registrado en el bloque actual se corresponde con el hash que resultaría de utilizar bloque previo.

Una vez terminado este proceso el segundo bloque pasara a ser el bloque previo y el tercer bloque pasaría a ser el bloque actual repitiendo este proceso de manera recursiva para todos los bloques de la cadena.

5.2.7.- Sustitución de la blockchain

Cuando se está trabajando con tecnologías descentralizadas como lo es la tecnología blockchain cada nodo es independiente y se plantea el problema de que el resto de nodos se sincronicen entre sí.

Por esta razón se necesita la función que en la blockchain desarrollada para este trabajo se llama *replace_chain* la cual se usa para que un nodo pida al resto sus cadenas

de bloques para después comparar su cadena con la del resto de nodos y así poder comprobar que cadena es la más larga y por tanto la que debe ser tenida en cuenta siempre y cuando sea válida.

```python
def replace_chain(self) -> bool:
    longest_chain = None
    max_length = len(self.skull_chain)
    for node in self.nodes:
        response = requests.get(f'http://{node}/get_chain')
        if response.status_code == 200:
            length = response.json()['Length']
            jsonChain = response.json()['SkullBlockchain']
            chain = []

            for jsonBlock in jsonChain:
                objectBlock = json.loads(jsonBlock)
                newBlock = SkullBlock(objectBlock["index"], objectBlock["timestamp"], objectBlock["nounce"],
                                      objectBlock["previous_hash"], objectBlock["transactions"])
                chain.append(newBlock)
                print(chain)

            if length > max_length and self.is_valid_chain(chain):
                max_length = length
                longest_chain = chain

    if longest_chain:
        self.skull_chain = longest_chain
        return True
    return False
```

Figura 5.7.- Función de reemplazo

En la Figura *5.7*, se puede ver como se obtiene la cadena más larga de entre todos los nodos que se encuentren registrados dentro de la cadena de bloques y para ello se hace una petición al método *get_chain* de cada nodo el cual se explicara más adelante cuando se definan las comunicaciones para probar el correcto funcionamiento de la cadena de bloques.

Una vez obtenida la cadena del otro bloque se comparan las longitudes de las cadenas validas hasta obtener la longitud de la cadena más larga y finalmente se sustituye la cadena actual por la más larga y se retorna true para informar de que la cadena ha sido sustituida.

En el caso de que se recorran las cadenas de todos los nodos y no haya ninguna válida más larga que la actual entonces se retornara false para informar al nodo de que su cadena es actualmente la más larga.

5.2.8.- Comunicación con la blockchain

En este punto se va a tratar la creación de las comunicaciones necesarias para probar el funcionamiento de la blockchain para lo cual se utilizará *Flask*.

Lo primero que se tiene que hacer es definir la aplicación de *Flask* y establecer que se lanzará con *ngrock* lo cual nos permitirá ejecutarla de manera local pero acceder a ella como si fuera una página web.

Después se inicializa la cadena de bloques utilizando el constructor que se definió en la clase *SkullBlockchain* en el cual se crea el bloque génesis de la cadena.

A continuación se utilizará la notación de *Flask* para generar funciones que se ejecutaran cuando se realice una petición HTTP a la url que *ngrock* creará cuando se lance la aplicación.

```
@app.route('/mine_block', methods=['GET'])
def mine_block():
  previous_block = mySkullBlockchain.get_previous_block()
  previous_hash = mySkullBlockchain.generate_hash_block(previous_block)
  current_nounce = mySkullBlockchain.get_nounce(previous_block.nounce)
  block = mySkullBlockchain.generate_block(current_nounce, previous_hash)

  # Hay que enviar la respuesta en formato json
  responseTransactions = []

  for transaction in block.transactions:
    responseTransactions.append(transaction.toJSON());

  response = {
              'message'       : 'New block has been mined',
              'index'         : block.index,
              'timestamp'     : block.timestamp,
              'nounce'        : block.nounce,
              'previous_hash' : block.previous_hash,
              'transactions'  : responseTransactions
            }

  return jsonify(response), 200
```

Figura 5.8.- Función de minado

La primera función que se ha definido es una petición GET para comenzar el minado de un bloque. Como se puede ver en la Figura *5.8*, en esta función se obtiene la información necesaria para crear un bloque como son el hash del bloque anterior y el nounce el cual se calcula mediante la función que se explicó en puntos anteriores.

Finalmente como respuesta se enviará al usuario todas las transacciones que se han minado al incluir el bloque minado a la cadena de bloques para lo cual fue necesario utilizar la función de parse que se añadió a la clase *SkullTransaction* para poner el objeto en un formato JSON adecuado para ser enviado por HTTP.

```python
@app.route('/get_chain', methods=['GET'])
def get_chain():
  # Hay que enviar la respuesta en formato json
  responseChain = []

  for block in mySkullBlockchain.skull_chain:
    responseChain.append(block.toJSON());

  response = {'SkullBlockchain': responseChain}

  return jsonify(response), 200


@app.route('/is_valid', methods = ['GET'])
def is_valid():
  if (mySkullBlockchain.is_valid_chain(mySkullBlockchain.skull_chain)):
      response = {'message' : 'Valid chain'}

  else:
      response = {'message' : 'Non valid chain'}

  return jsonify(response), 200
```

Figura 5.9.- Funciones de validación y obtención de la cadena

En la Figura *5.9*, se pueden ver las siguientes funciones a explicar que son *get_chain* y *is_valid*. En la primera de ellas se retorna una respuesta que contiene todos los bloques de la cadena formateados adecuadamente mientras que en la segunda lo que se hace es utilizar la función de la blockchain para validar si la cadena actual es válida y retorna un mensaje informativo al respecto.

La siguiente función que se ha creado es para añadir transacciones a la cadena aunque estas no se harán efectivas hasta que algún nodo decida minar el bloque en el que serán introducidas y por lo tanto estas transacciones quedaran en el pool de pendientes hasta que esto ocurra.

```python
@app.route('/add_transaction', methods = ['POST'])
def add_transaction():
  json = request.get_json()
  mandatoryParams = ['sender', 'receiver', 'amount']

  if not all(key in json for key in mandatoryParams):
      return 'There are mandatory params left on the request sended', 400

  index = mySkullBlockchain.add_transaction(json['sender'], json['receiver'], json['amount'])

  response = {'message': f'Adding transaction to the block {index}'}

  return jsonify(response), 201
```

Figura 5.10.- Función para añadir transacciones

En la Figura *5.10*, se puede ver que esta vez la función es un POST debido a que debe recibir por parte del cliente que haga la petición la información básica de la transacción como la persona que envía el dinero, la persona que lo recibe y la cantidad enviada.

También se realiza una validación de que se han recibido los parámetros necesarios para la creación de una transacción y en caso de no ser así se retornará un error 400 con un mensaje de que faltan parámetros.

Para que cada blockchain pueda funcionar, cada nodo que esté conectado tendrá que definir los otros nodos que van a estar trabajando e interactuando con la cadena. Para esto se ha creado la función *connect_node* que será un POST que recibe las direcciones de los otros nodos que interactuaran con la cadena para simular que hay varios usuarios usando la cadena creada.

```
@app.route('/connect_node', methods = ['POST'])
def connect_node():
  json = request.get_json()
  nodes = json.get('nodes')

  if nodes is None:
      return 'There are no new nodes', 400

  for node in nodes:
      mySkullBlockchain.add_node(node)

  response = {'message': 'All the nodes received are now connected', 'total_nodes' : list(mySkullBlockchain.nodes)}

  return jsonify(response), 201
```

Figura 5.11.- Conexión de nodos

En la Figura *5.11*, se puede ver que en esta función se añaden a la cadena los nodos y se retorna al usuario el número total de nodos para que sepa si se han podido añadir correctamente a la cadena.

```
@app.route('/replace_chain', methods = ['GET'])
def replace_chain():
  # Hay que enviar la respuesta en formato json
  responseChain = []

  for block in mySkullBlockchain.skull_chain:
    responseChain.append(block.toJSON());

  if (mySkullBlockchain.replace_chain()):
      response = {'message' : 'SkullBlockchain has been replaced with the new longest chain', 'new_chain': responseChain}

  else:
      response = {'message': 'SkullBlockchain was already the longest chain in all nodes', 'actual_chain': responseChain}

  return jsonify(response), 200
```

Figura 5.12.- Reemplazo de cadena

En la Figura *5.12*, se puede ver la última función que será necesario exponer la cual será la función de sustitución de la cadena (*replace_chain*) ya que en caso de que algún nodo no tenga la cadena más larga este tendrá que sustituir su cadena por la nueva más larga.

En caso de que se haya sustituido la cadena se informará al usuario con un mensaje de respuesta indicando que se ha producido la sustitución y en caso de que la cadena ya fuera la más larga también se le indicará con otra respuesta.

Una vez definidas todas las funciones anteriores solamente falta lanzar la aplicación utilizando *app.run()* para que ngrock lance la aplicación y suministre una dirección a la cual enviar peticiones utilizando algún cliente HTTP como puede ser Postman o en el caso de este proyecto la extensión de Visual Studio Code llamada Thunder Client.

En la Figura *5.13*, se muestra un ejemplo de una petición enviada mediante esta extensión:

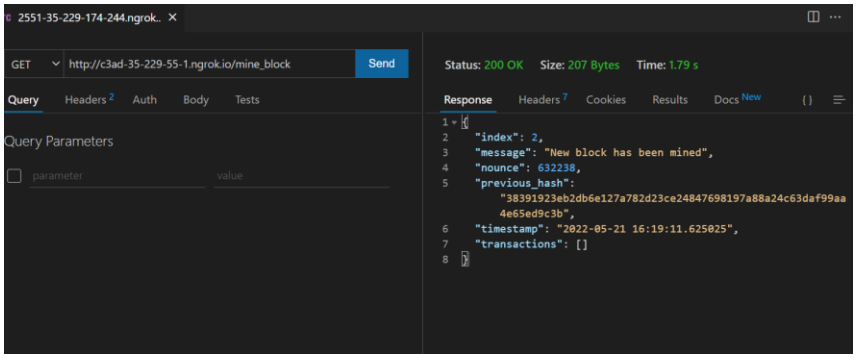

Figura 5.13.- Ejemplo de petición

En la ejecución de la cadena realizada en un cuaderno de *Google Colab* se puede ver que la petición queda reflejada en la cadena de bloques. Esto queda reflejado en la Figura *5.14*:

Figura 5.14.- Resultado en Google Colab

6. Conclusiones y Trabajos futuros

6.1.- Conclusiones

En el presente Trabajo Fin de Grado se ha realizado un estudio pormenorizado sobre la tecnología blockchain y su aplicación al campo de los videojuegos.

Para llevar a cabo esto, se ha efectuado una revisión amplia de la literatura existente sobre la materia obteniendo como principal conclusión que en la actualidad el mundo del desarrollo de videojuegos de tipo blockchain es un mundo en auge.

Además de esto, se ha realizado el diseño e implementación de un prototipo de videojuego que use tecnología blockchain para verificar de forma empírica la aplicación directa de la tecnología en este campo.

Como una de las principales conclusiones y resultado de este Trabajo de Fin de Grado se ha podido demostrar la gran importancia que blockchain tiene en el presente para el desarrollo de videojuegos y también se han comprobado las ventajas y desventajas que la aplicación de esta tecnología puede implicar.

El resultado más relevante de este trabajo es la creación de un prototipo de juego que cumple las principales características que suelen tener los videojuegos de tipo NFT que son la forma más común de aplicar la tecnología blockchain en este sector.

En este punto, me ha resultado de gran interés el poder inmiscuirme en el mundo del desarrollo de videojuegos haciendo uso de tecnologías, hasta ahora desconocidas para mí, y ampliando de esta forma mi conocimiento y aptitudes tanto personales como profesionales.

Cabe destacar aquí que blockchain tiene problemas graves que aún no se han podido solucionar cuando esta se aplica a videojuegos de gran popularidad debido a los problemas evidentes de escalabilidad así como la falta de estabilidad en el precio de los tokens que generalmente se utilizan como moneda dentro de estos.

De no ser solucionados estos problemas cabe la posibilidad que la aplicación de esta tecnología quede relegada solamente a su utilización de forma residual para aportar valor a ítems dentro de juegos, pero no para el desarrollo completo de juegos basados en esta tecnología.

6.2.- Trabajos futuros

La proyección más directa de este Trabajo de Fin de Grado seria la creación de un videojuego que extienda lo realizado aquí con un enfoque centrado en la ingeniería del software y el desarrollo de forma práctica del despliegue del juego en un "entorno real" en el que se utilizase dinero real que aquí solo se planteó de forma teórica.

Además se podrían realizar ampliaciones de las funcionalidades actuales del juego así como añadir más contenido como por ejemplo nuevos jefes, más estilos para los NFT actuales, distintos tipos de ataques, posibilidad de defenderse o la posibilidad de añadirle estados al jefe como envenenado, dormido o paralizado.

Otro aspecto que podría dar lugar a un trabajo futuro seria la realización de una mejora estética del juego empleando diseños 3D o incluso dibujos realizados a mano por un grupo de diseñadores para potenciar el apartado visual del videojuego.

Otro posible trabajo futuro seria la realización de estudios sobre el rendimiento del juego y comparativas entre su comportamiento en un navegador tradicional frente al navegador de un dispositivo móvil.

Una posible línea de investigación que también sería muy interesante es la creación de una versión del juego nativa para teléfonos móviles aplicando tecnologías propias de esos dispositivos como por ejemplo *Flutter* [53].

También se debe continuar investigando la tecnología blockchain ya que la extensión de esta tecnología es cada vez mayor no solo en el mundo de los videojuegos sino a casi todos los ámbitos del desarrollo de software.

[53] https://flutter.dev/

7. Bibliografía

[1] S. Nakamoto, «Bitcoin: A peer-to-peer electronic cash system», *Decentralized Business Review*, p. 21260, 2008.
[2] A. Merino González, «El bitcoin en la sociedad de hoy», 2019.
[3] S. S. Sabry, N. M. Kaittan, y I. Majeed, «The road to the blockchain technology: Concept and types», *Periodicals of Engineering and Natural Sciences (PEN)*, vol. 7, n.º 4, pp. 1821-1832, 2019.
[4] J. Käll, «Blockchain Control», *Law Critique*, vol. 29, n.º 2, pp. 133-140, jul. 2018, doi: 10.1007/s10978-018-9227-x.
[5] M. K. Shrivas y D. T. Yeboah, «The disruptive blockchain: Types, platforms and applications», 2018.
[6] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng, y V. C. M. Leung, «Decentralized Applications: The Blockchain-Empowered Software System», *IEEE Access*, vol. 6, pp. 53019-53033, 2018, doi: 10.1109/ACCESS.2018.2870644.
[7] M. A. López, «Cómo desarrollar confianza en entornos complejos para generar valor de impacto social.», *Banco Interamericano de Desarrollo*, 2018.
[8] F. L. Alonso Fernández, «El bitcoin: análisis y evolución», 2017.
[9] I. Nieves Jiménez, «Bitcoin: Análisis y comparación con sus principales competidoras», 2020.
[10] S. Zhang y J.-H. Lee, «Analysis of the main consensus protocols of blockchain», *ICT express*, vol. 6, n.º 2, pp. 93-97, 2020.
[11] P. R. Tiwari y M. Green, «Algorithm-Substitution Attacks on Cryptographic Puzzles», 477, 2022. Accedido: 2 de mayo de 2022. [En línea]. Disponible en: http://eprint.iacr.org/2022/477
[12] B. Cohen y K. Pietrzak, «The chia network blockchain». 2019.
[13] F. Regner, N. Urbach, y A. Schweizer, «NFTs in practice–non-fungible tokens as core component of a blockchain-based event ticketing application», 2019.
[14] Q. Wang, R. Li, Q. Wang, y S. Chen, «Non-fungible token (NFT): Overview, evaluation, opportunities and challenges», *arXiv preprint arXiv:2105.07447*, 2021.
[15] M. Dowling, «Fertile LAND: Pricing non-fungible tokens», *Finance Research Letters*, p. 102096, 2021.
[16] L. Kugler, «Non-fungible tokens and the future of art», *Communications of the ACM*, vol. 64, n.º 9, pp. 19-20, 2021.
[17] U. W. Chohan, «Non-Fungible Tokens: Blockchains, Scarcity, and Value», *Critical Blockchain Research Initiative (CBRI) Working Papers*, 2021.
[18] T. Min, H. Wang, Y. Guo, y W. Cai, «Blockchain games: A survey», en *2019 IEEE Conference on Games (CoG)*, 2019, pp. 1-8.
[19] O. Kharif, «Cryptokitties mania overwhelms ethereum network's processing», *Bloomberg (4 Dec. 2017)*, 2017.
[20] X.-J. Jiang y X. F. Liu, «CryptoKitties transaction network analysis: The rise and fall of the first blockchain game mania», *Frontiers in Physics*, vol. 9, p. 57, 2021.
[21] T. Sihvonen, A. Serada, y J. T. Harviainen, «CryptoKitties and the New Ludic Economy: How blockchain introduces value ownership and scarcity in the digital world», 2019.
[22] A. Panachev, V. Shcherbitsky, y M. A. Medvedev, «Application of blockchain technologies and game approach in the educational process of universities», *AIP*

Conference Proceedings, vol. 2333, n.º 1, p. 100004, mar. 2021, doi: 10.1063/5.0042076.
[23] A. Pfeiffer, S. Kriglstein, y T. Wernbacher, «Blockchain Technologies and Games: A Proper Match?», en *International Conference on the Foundations of Digital Games*, New York, NY, USA, sep. 2020, pp. 1-4. doi: 10.1145/3402942.3402996.
[24] J. Nyyssölä, «Assessing the effects of blockchains in video games: Case IkuneRacers», p. 49.
[25] T. Felin y K. Lakhani, «What problems will you solve with blockchain?», *MIT Sloan Management Review*, vol. 60, pp. 32-38, oct. 2018.
[26] D. A. Aguila y J. M. Bartolata, *COLLEGE OF LIBERAL ARTS AND SCIENCES AXEing the Axie Infinity (AI): The AI of Modern Gaming, Business Model Strategem, and Global Economy Towards Cryptocurrency Era In partial fulfillment of the requirements for The Degree Bachelor of Arts Major in International Development Studies*. 2022. doi: 10.13140/RG.2.2.10609.56162.
[27] R. C. Culannay, «Analysis on the Factors that Influence the Investment on Online Crypto Games», *International Journal of Arts, Sciences and Education*, vol. 3, n.º 1, Art. n.º 1, mar. 2022.
[28] A. Dutra, A. Tumasjan, y I. M. Welpe, «BLOCKCHAIN IS CHANGING HOW MEDIA AND ENTERTAINMENT COMPANIES COMPETE», p. 8.
[29] J. Al-Jaroodi y N. Mohamed, «Blockchain in Industries: A Survey», *IEEE Access*, vol. 7, pp. 36500-36515, 2019, doi: 10.1109/ACCESS.2019.2903554.
[30] O. J. Scholten, N. G. J. Hughes, S. Deterding, A. Drachen, J. A. Walker, y D. Zendle, «Ethereum Crypto-Games: Mechanics, Prevalence, and Gambling Similarities», en *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, Barcelona Spain, oct. 2019, pp. 379-389. doi: 10.1145/3311350.3347178.
[31] D. Charrieras, «A CRITIQUE OF VIDEOGAME PRODUCTION ON THE BLOCKCHAIN: GAME DESIGN, ETHICS, AESTHETICS, MONEY», presentado en 21st International Congress of Aesthetics, Possible Worlds of Contemporary Aesthetics, ICA 2019: Aesthetics Between History, Geography and Media, jul. 2019. Accedido: 23 de abril de 2022. [En línea]. Disponible en: https://scholars.cityu.edu.hk/en/publications/a-critique-of-videogame-production-on-the-blockchain(2821336d-c7e8-418c-9f79-31e1d307098a).html
[32] A. Carvalho, «Bringing transparency and trustworthiness to loot boxes with blockchain and smart contracts», *Decision Support Systems*, vol. 144, p. 113508, may 2021, doi: 10.1016/j.dss.2021.113508.
[33] «Blockchain-SimGame: How can Simulation be Used for Facilitating Corporate Learning of Blockchain Technology? - ProQuest». https://www.proquest.com/openview/48ec8c9b84f4152a7dc1a3ac00110063/1?pq-origsite=gscholar&cbl=396495 (accedido 25 de abril de 2022).
[34] A. Manzoor, M. Samarin, D. Mason, y M. Ylianttila, «Scavenger Hunt: Utilization of Blockchain and IoT for a Location-Based Game», *IEEE Access*, vol. 8, pp. 204863-204879, 2020, doi: 10.1109/ACCESS.2020.3037182.

Anexo A. Manual de usuario de SkullGame

A lo largo de este anexo se va a explicar cómo se jugaría al videojuego *SkullGame* desde el punto de vista de un usuario y utilizando la red de test de *Binance Smart Chain* para realizar el despliegue del contrato inteligente.

Lo primero que el usuario debe tener previamente es una cuenta de *MetaMask* en la cual tenga BNB de test la cual es la moneda utilizada en la red de *Binance*, para lo cual puede emplear una faucet como se muestra en la Figura A *1*:

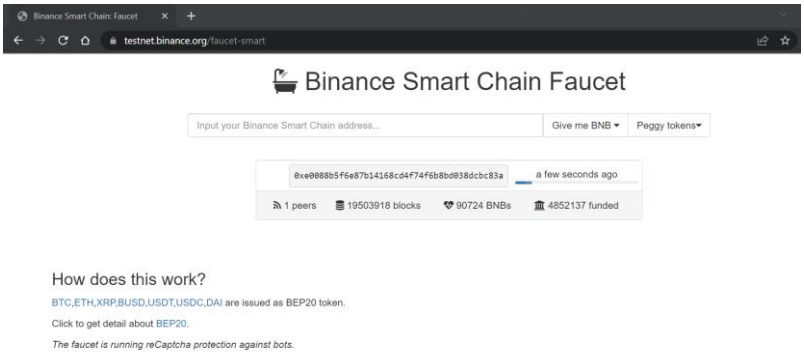

Figura A 1.- Faucet de ejemplo para usuarios

Lo único que hay que hacer seria pegar la dirección de la wallet a la que se quiere recibir un BNB (criptomoneda de *Binance*) de test y pulsar en el botón de recibir tokens aunque esto solo se puede hacer una vez al día como se puede ver en la Figura A *2*.

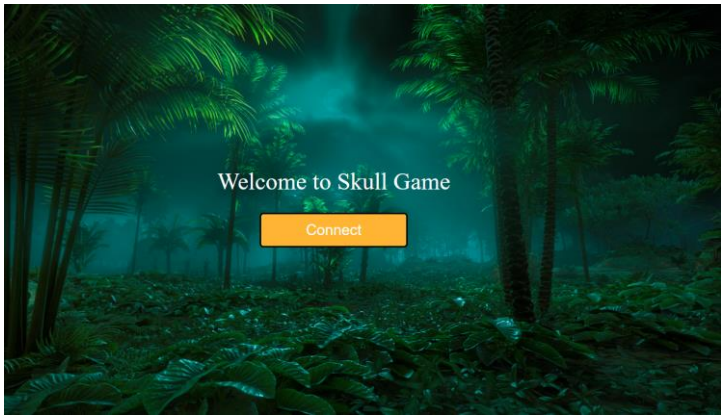

Figura A 2.- Pantalla de conexión

Para comenzar a jugar lo primero que el usuario tiene que hacer es pulsar el botón de conexión que automáticamente detectara la extensión de *MetaMask* o del provider que se esté usando en el navegador y conectara la cuenta que se esté usando con el juego.

Una vez establecida la conexión con la cuenta se podrá visualizar un menú con todas las *Skulls* que el usuario tiene asociadas a la dirección de su cuenta en particular y podrá seleccionar cual usar.

Si es la primera vez que el usuario entra entonces tendrá que usar el botón de crear *Skull* para que se lance una transacción a su cuenta de *MetaMask* que deberá autorizar para poder obtener su primera *Skull*.

Algo que cualquier usuario puede hacer incluso si no tiene ninguna *Skull* es comprobar cuánto dinero se ha almacenado actualmente para saber que si derrota al jefe se llevara esa cantidad de dinero.

El usuario puede decidir querer subir de nivel su *Skull* lo cual le supondrá un coste en forma de transacción y que hará que su *Skull* suba de nivel mejorando así todas sus estadísticas.

La transacción se visualizará de forma similar a este ejemplo de la Figura A *3*:

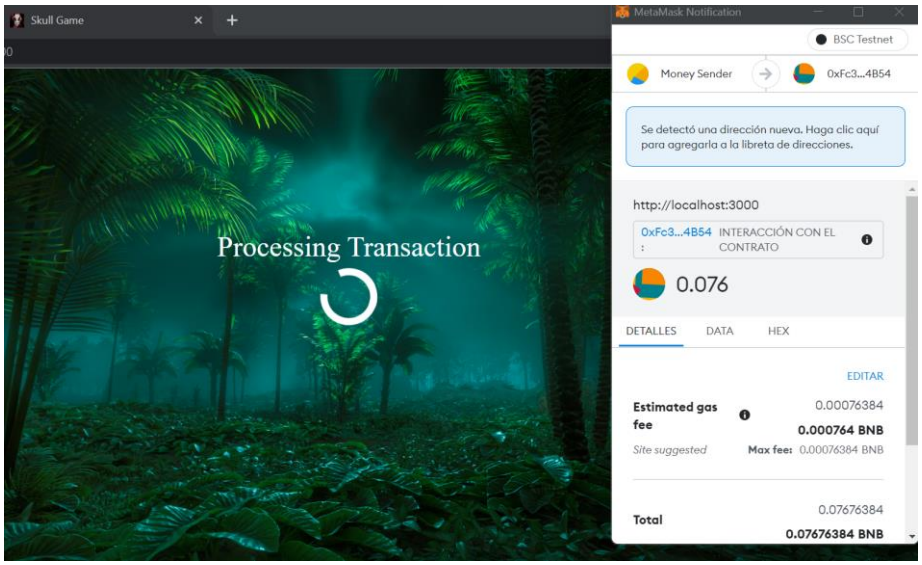

Figura A 3.- Ejemplo de transacción

Por otra parte si el usuario pulsa el botón de pelear con el jefe se le seleccionará automáticamente la *Skull* correspondiente y se le redirigirá a la pantalla de lucha que se puede ver en la Figura A *4*:

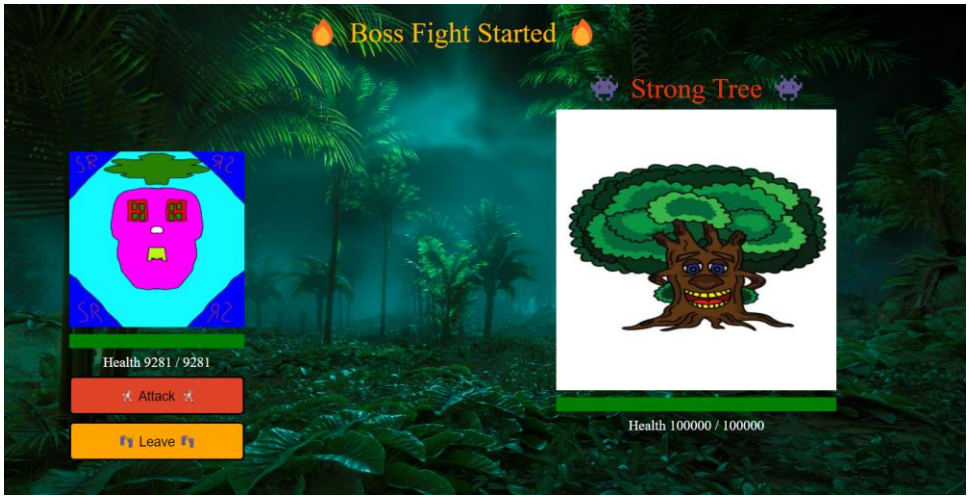

Figura A 4.- Batalla de ejemplo

Aquí se plantean dos opciones, o bien atacar haciendo daño al jefe y recibiendo como contraparte daño que podría matar a la *Skull* elegida o huir lo cual nos llevaría de regreso a la pantalla de selección de *Skull*. Esto queda reflejado en la Figura A *5*.

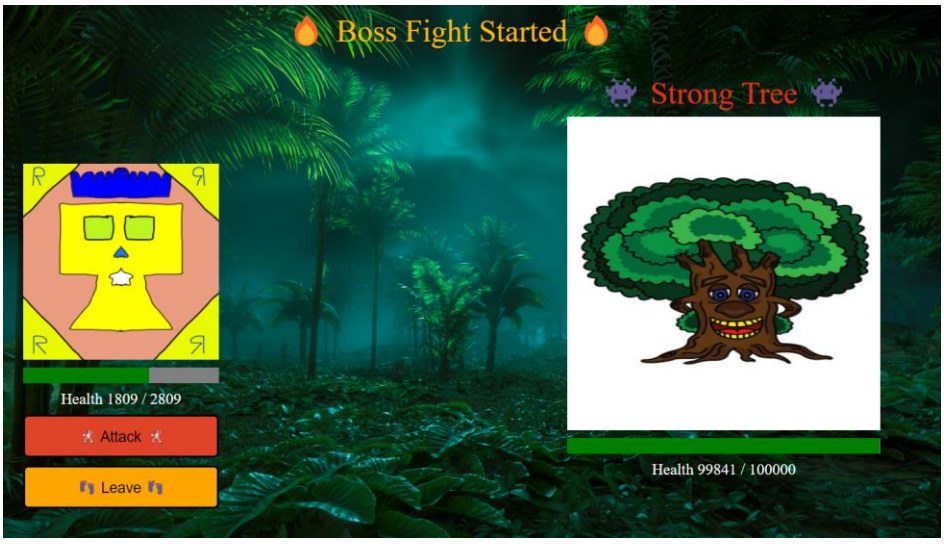

Figura A 5.- Perdida de vida del jefe

En caso de que una *Skull* haya muerto en combate por el ataque del jefe la única opción que nos quedara será huir para regresa al menú de selección de *Skull* pero este NFT ya no podrá pelear ni subir de nivel y solo podrá ser revivido lo cual también tiene un coste para el usuario además de que como castigo por morir la Skull regresara al primer nivel.

En la Figura A *6* se puede ver como quedaría un NFT después de haber perdido todos sus puntos de vida.

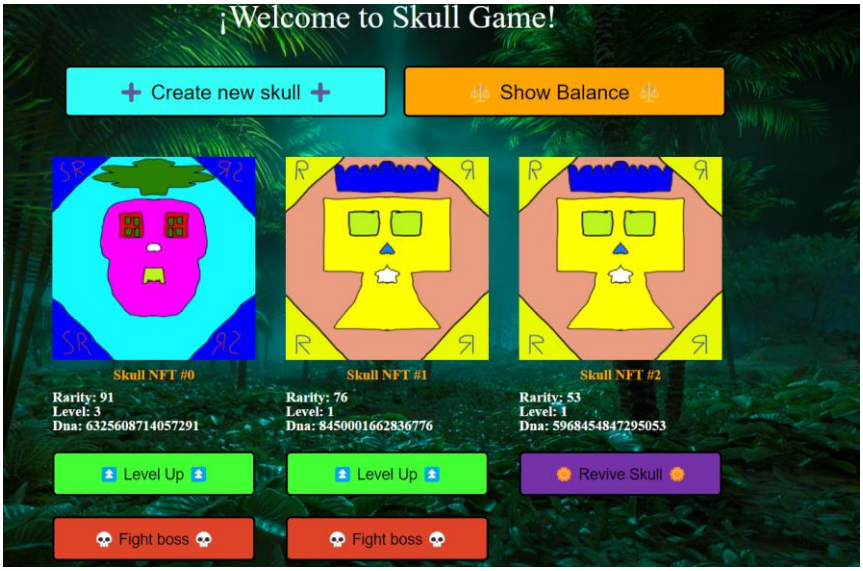

Figura A 6.- Skull derrotada

Si el usuario consigue que el ataque de su *Skull* sea el que mate al jefe este podrá pulsar un botón de obtener recompensa y todo el dinero que se haya ido generando a lo largo del juego con las diferentes compras de NFT, subidas de nivel y ataques al jefe le será entregado como premio.

Una vez esto ocurra el jefe recuperara su vida y el juego comenzara de nuevo para que otros jugadores puedan obtener la recompensa.

Anexo B. Manual de utilización de SkullBlockchain

A lo largo de este anexo se va a explicar cómo utilizar correctamente la blockchain que se ha creado de prueba para complementar la investigación teórica en el despliegue del videojuego desarrollado.

Durante este manual se va a ejecutar el cuaderno de Google Colab que contiene el código de la cadena en tres instancias diferentes para simular que son 3 ordenadores los que están trabajando con la blockchain.

Lo primero que se va a hacer es ver que la cadena ha generado el bloque génesis correctamente y cómo se puede comprobar en el primer nodo la cadena es correcta como se ve en la Figura B *1*:

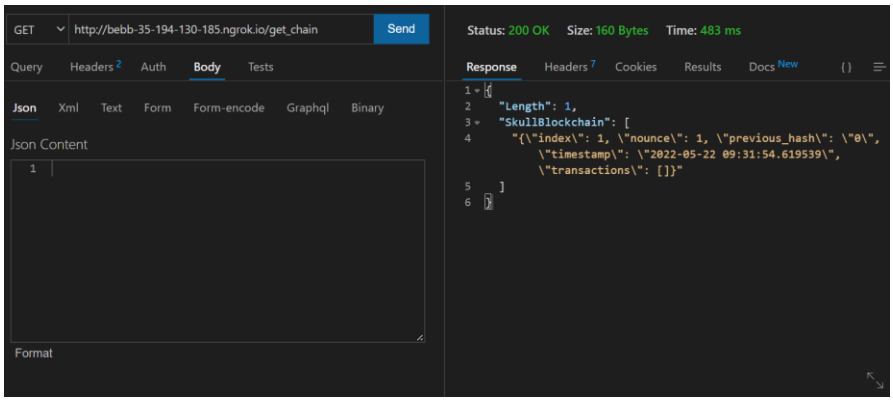

Figura B 1.- Obtención de cadena en el primer nodo

Después, cada nodo tendrá que conectar al resto de nodos para poder saber que también están operando con la cadena de bloques y sincronizarse con ellos.

Como ejemplo de conexión de nodos se pueden ver los realizados en la Figura B *2*, en la Figura B *3* y en la Figura B *4*.

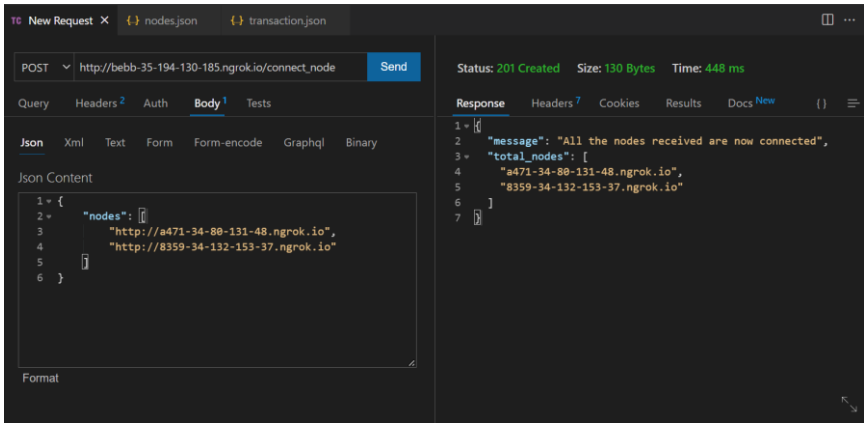

Figura B 2.- Ejemplo de conexión de nodos 1

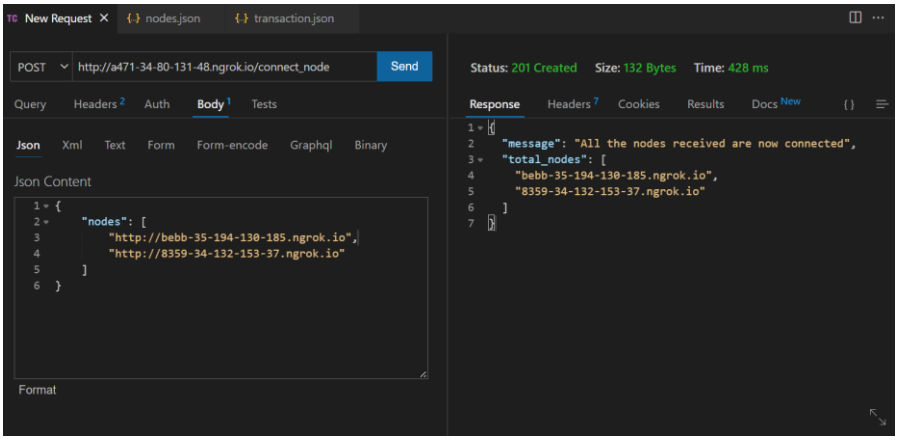

Figura B 3.- Ejemplo de conexión de nodos 2

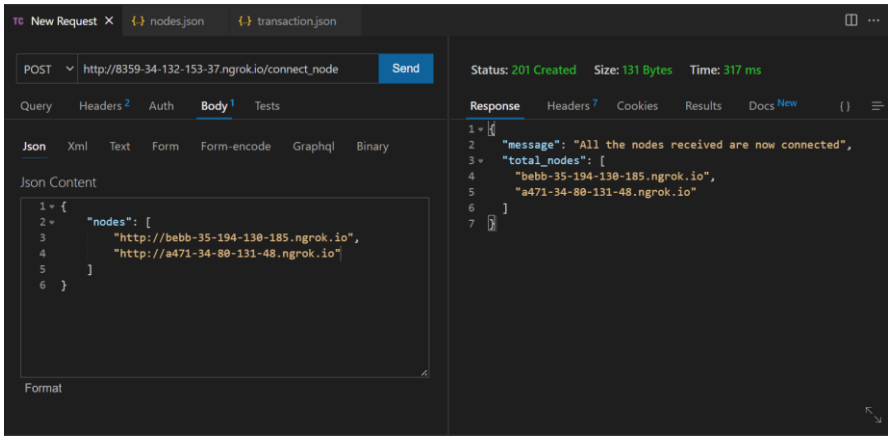

Figura B 4.- Ejemplo de conexión de nodos 3

Una vez que todos los nodos tienen entre sus datos al resto, ya se puede empezar a enviar transacciones a la cadena sin causar problemas de desincronización.

En este caso se ha utilizado el primer nodo definido para enviar dos transacciones de prueba como se puede ver en la Figura B *5* y en la Figura B *6*.

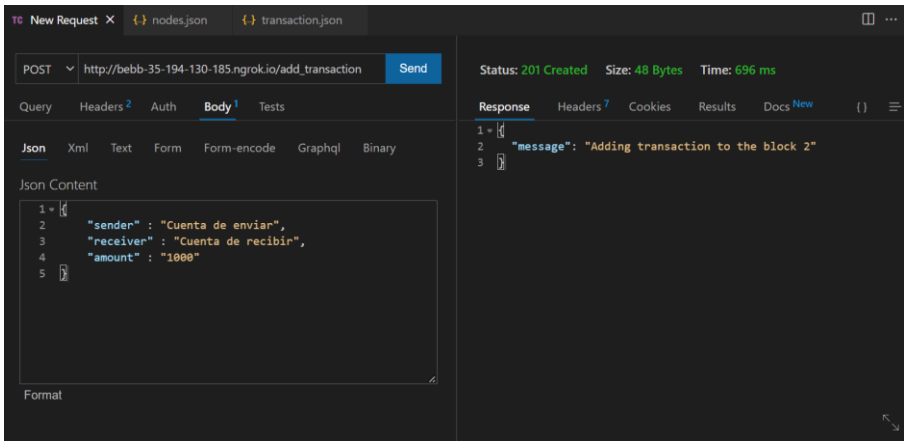

Figura B 5.- Añadir una transacción inicial

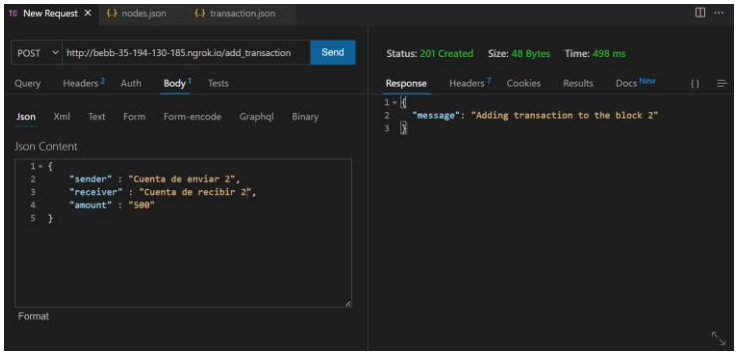

Figura B 6.- Añadir una transacción complementaria

Si en este punto se hace obtiene la cadena se puede ver que no hay nada puesto que las transacciones estarán en el almacenadas como pendientes hasta que se mine el siguiente bloque y se incluyan en el cómo se puede ver en la Figura B *7*:

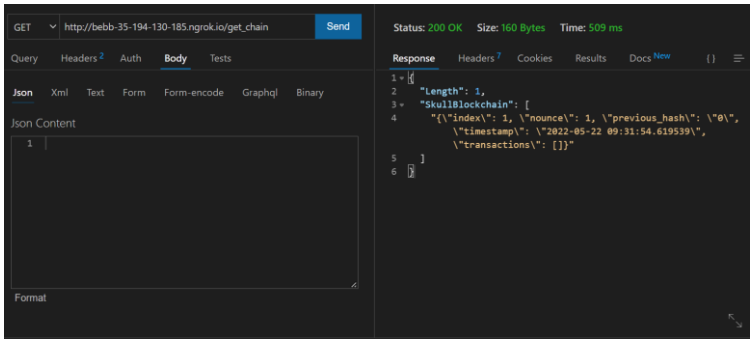

Figura B 7.- Comprobación de la cadena

Si en este punto se hace obtiene la cadena se puede ver que no hay nada puesto que las transacciones estarán en el almacenadas como pendientes hasta que se mine el siguiente bloque y se incluyan en el cómo se ve en la Figura B *8*:

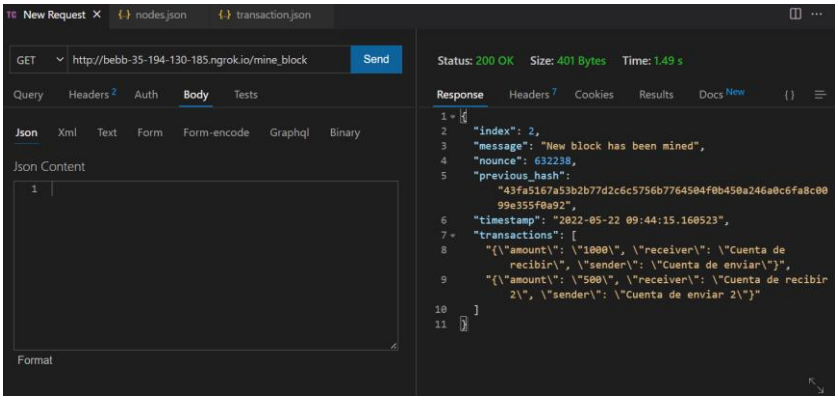

Figura B 8.- Minado de bloque de prueba

Tras haber minado el bloque se puede realizar una obtención de la cadena como comprobación de que todo ha ido bien y como se puede ver en la figura la cadena ya tiene un nuevo bloque con las correspondientes transacciones en su interior.

En la Figura B *9* se puede observar cómo se produce la obtención de la cadena con bloques nuevos:

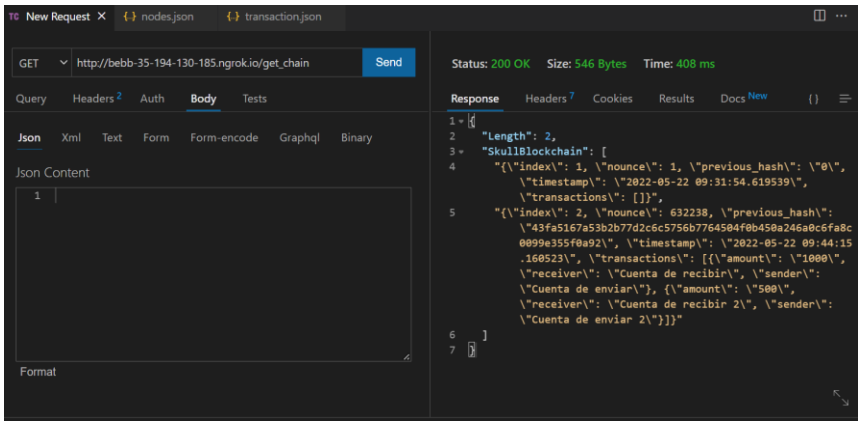

Figura B 9.- Obtención de la cadena con bloques nuevos

Por otro lado si se va al segundo nodo y se obtiene su cadena se puede comprobar que este no tiene aún el bloque puesto que esta cadena no es la más larga actualmente.

La comprobación que se ha realizado durante el ejemplo se puede ver en la Figura B *10*.

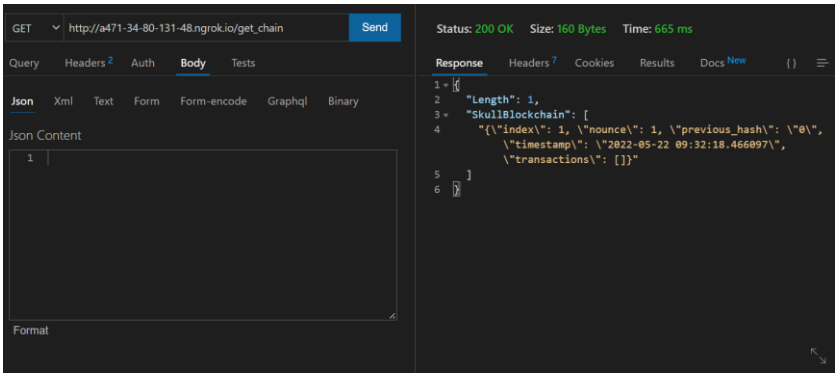

Figura B 10.- Comprobación de estado de cadena en nodo secundario

Para poder obtener la nueva cadena más larga que es la del primer nodo se tiene que hacer un *replace_chain* en el segundo nodo para que este sustituya su cadena por la nueva cadena más larga como se ve en la Figura B *11*.

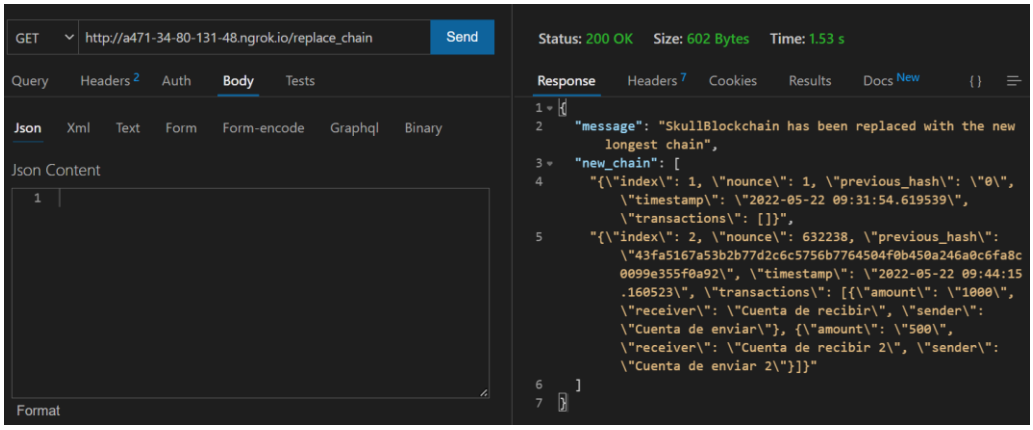

Figura B 11.- Reemplazo de cadena

Haciendo un *get_chain* para comprobar que todo está correcto se puede ver que el segundo nodo se ha sincronizado correctamente con el primero ya que la cadena más larga en el momento del *replace_chain* era la del primer nodo por lo que el segundo nodo debería actualizarse sustituyendo su cadena.

La petición para la obtención de la nueva cadena que se ha realizado durante el ejemplo se puede ver en la Figura B *12*.

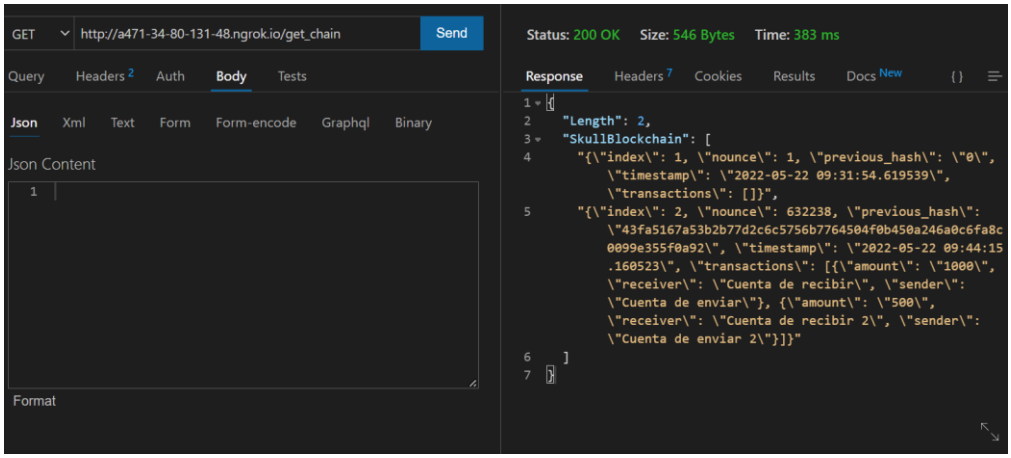

Figura B 12.- Obtención de la cadena nueva

Si se comprueba en Google Colab el estado de ejecución de la aplicación desplegada con *ngrock* en el primer nodo se puede ver que todas las peticiones que se han ido haciendo a lo largo de esta prueba han quedado reflejadas correctamente como se puede ver en la Figura B *13*.

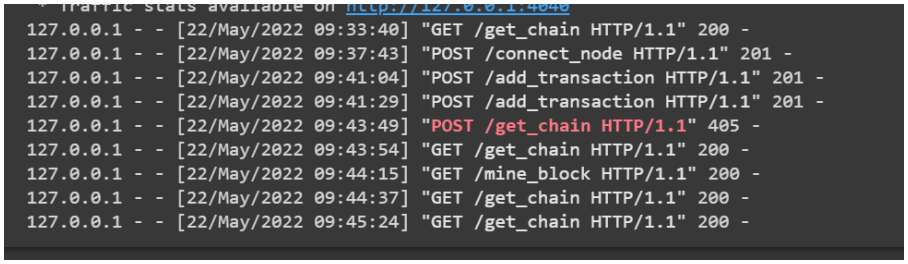

Figura B 13.- Resultado de las peticiones