Desenvolupament d'una aplicació distribuïda sobre Ethereum

Espígol Ayats, Aleix

Resum– Motivat per la contínua evolució i popularitat de la blockchain i les criptomonedes, he volgut endinsar-me en l'entorn d'Ethereum, una de les criptomonedes amb més renom. Per aquest motiu he desenvolupat una aplicació descentralitzada (DAPP) que treballa de forma autònoma sobre la blockchain d'Ethereum. L'aplicació en qüestió correspon al joc de la ruleta, un joc d'atzar molt conegut en centres recreatius i d'oci com els casinos. L'aplicació que he desenvolupat d'aquest joc, té la particularitat que les apostes es realitzen mitjançant criptomonedes, en aquest cas ethers (la moneda d'Ethereum). A causa de la transparència de la blockchain i amb l'objectiu de garantir aleatorietat i seguretat en l'elecció del nombre guanyador, s'ha elaborat una anàlisi exhaustiva sobre vulnerabilitats en el procés de la generació d'aleatorietat, per poder-ne triar la millor opció.

Paraules clau– Aleatorietat, Blockchain, Chainlink VRF, Criptomonedes, DAPP, EmailJS, Ethereum, MetaMask, ReactJS, Ruleta, SafeMath, Smart Contract, Solidity, Web3

Abstract– Motivated by the continuous evolution and popularity of the blockchain and cryptocurrencies, I wanted to delve into the environment of Ethereum, one of the most renowned cryptocurrencies. For this reason I have developed a decentralized application (DAPP) that works autonomously on the Ethereum blockchain. The application in question corresponds to the roulette game, a game of chance well known in recreational and leisure centers such as casinos. The application I have developed for this game has the peculiarity that bets are made using cryptocurrencies, in this case ethers (the currency of Ethereum). Due to the transparency of the blockchain and with the aim of guaranteeing randomness and security in the choice of the winning number, an exhaustive analysis has been elaborated on vulnerabilities in the process of generating randomness, in order to be able to choose the best option.

Keywords– Blockchain, Chainlink VRF, Cryptocurrencies, DAPP, EmailJS, Ethereum, MetaMask, Randomness, ReactJS, Roulette, SafeMath, Smart Contract, Solidity, Web3

1 Introducció

Amb el pas dels anys, la tecnologia blockchain ha guanyat una gran importància. El seu exponencial creixement ha beneficiat a ecosistemes com el d'Ethereum. Tot això és gràcies al seu disseny de base de dades distribuïda, formada per cadenes de blocs, que permet evitar la modificació de dades un cop són publicades.

Ethereum és una plataforma de codi obert, digital i descentralitzada sobre la qual s'executen contractes intel·ligents. L'ecosistema d'aquesta plataforma ha anat creixent aquests darrers anys, gràcies a la capacitat de crear noves monedes sobre ella mateixa i a la propietat d'executar programes que funcionin de manera autònoma. Aquesta propietat ha permès elaborar solucions descentralitzades que tenen l'objectiu d'eliminar intermediaris, per així simplificar processos i, d'aquesta manera, estalviar costos al consumidor.

Amb aquest projecte, he volgut desenvolupar una aplicació distribuïda sobre Ethereum. L'aplicació en qüestió, correspon al joc d'atzar de la ruleta. Per poder dur-lo a terme ha sigut necessari, d'una banda, una interfície web, amb la qual el client pugui interactuar, que formarà la part del *frontend*. I d'altra banda un *backend*, que correspon a tot el codi i programari necessari perquè l'aplicació funcioni correctament sobre Ethereum i pugui ser utilitzada pels diferents usuaris, en el qual hi haurà el servidor i el contracte intel·ligent (*smart contract*).

- E-mail de contacte: Aleix.espigol@e-campus.uab.cat
- Menció realitzada: Tecnologies de la Informació
- Treball tutoritzat per: Jordi Herrera Joancomarti (DEIC)
- Curs 2020/2021

En aquest informe esmentaré els aspectes fonamentals pel desenvolupament d'aquest projecte, començant per la descripció de què es vol fer i analitzant la situació inicial. A continuació, s'esmentaran els objectius que s'han d'aconseguir, detallant el seu nivell de prioritat i el fil d'execució. Detallaré la metodologia que he seguit pel correcte desenvolupament del projecte, així com la planificació que s'ha seguit durant aquest període de temps. També exposaré els resultats obtinguts de l'anàlisi de vulnerabilitats sobre l'aleatorietat, i comentaré els aspectes més importants del desenvolupament de l'aplicació. Finalment, per acabar, comentaré les línies futures per millorar l'aplicació i les conclusions extretes amb la realització del treball.

2 Estat de l'art

Per entendre d'una manera més senzilla com estarà estructurat el *backend* i com enllaçaré Ethereum amb l'aplicació explicaré el concepte d'*smart contract*.

Un *smart contract* [1] és un codi que conté un conjunt d'instruccions que són emmagatzemades a la blockchain, d'Ethereum en aquest cas, i té la capacitat d'autoexecutar accions d'acord amb una sèrie de paràmetres prèviament programats, tot això d'una manera immutable, completament segura i transparent. L'*smart contract* serà el responsable de definir la propietat d'aplicació distribuïda, ja que l'aplicació funcionarà sobre aquest conjunt d'instruccions que definiran les funcionalitats de l'aplicatiu. En tractar-se d'un programari que es troba penjat en un lloc públic, en el qual hi ha un moviment de diners, és molt important transmetre i garantir uns bons nivells de seguretat i transparència, perquè els usuaris puguin confiar a interactuar amb l'aplicació. Per aquest motiu hauré de ser molt curós amb el codi que programi, per evitar atacs o que els usuaris facin trampes amb l'aleatorietat de l'elecció del número de la ruleta, per afavorir els seus interessos econòmics.

Per aconseguir aquesta aleatorietat he fet ús de l'oracle *Chainlink VRF* [2] que em proporciona un nombre 100% aleatori a partir d'una llavor, que correspon a l'adreça de l'usuari que vol jugar. Aquest servidor també genera una prova criptogràfica que és publica i es verifica abans que pugui ser utilitzada per l'aplicació consumidora, així s'aconsegueix que els resultats no puguin ser alterats ni modificats.

3 Objectius

La finalitat del projecte és aconseguir programar completament el joc de la ruleta perquè els usuaris hi puguin jugar de manera online. El joc serà una aplicació distribuïda sobre Ethereum que contingui una interfície web bàsica i interactiva pels clients.

Perquè es pugui consolidar com un projecte real, serà necessari complir amb unes funcionalitats mínimes requerides a més de contenir alguna característica diferent i innovadora. Aquesta característica innovadora és la particularitat que tant les apostes com els premis es gestionaran amb ethers. A més a més, els usuaris podran veure i comprovar el codi de l'aplicació perquè es trobarà en el *smart contract* i aquest estarà penjat a la blockchain d'Ethereum, de manera que això suposarà un major nivell de confiança cap a l'usuari.

TAULA 1: Objectius

Objectiu	Prioritat
Smart contract que contingui les funcionalitats suficients per jugar al joc de la ruleta	Essencial
Smart contract que s'executi de manera autònoma i aporti la propietat d'aplicació distribuïda	Essencial
Relacionar el pagament i les apostes amb la blockchain utilitzant Ethereum	Essencial
Smart contract que s'executi de manera autònoma i aporti la propietat d'aplicació distribuïda	Essencial
Aconseguir una aleatorietat imprevisible i correcta per garantir una millor qualitat de servei	Essencial
Analitzar les vulnerabilitats i possibles atacs que pot patir l'aplicació	Essencial
Elaborar la documentació tècnica de suport al projecte	Essencial
Programar un bon codi per millorar la seguretat i evitar vulnerabilitats	Essencial
Desenvolupar una interfície web bàsica, senzilla i intuïtiva	Essencial
Aconseguir desenvolupar una bona interfície gràfica per millorar l'experiència de joc del client	Opcional

Com es pot veure a la Taula 1 podem veure el conjunt dels objectius plantejats pel correcte funcionament del projecte amb el seu respectiu nivell de prioritat. Els objectius es troben classificats en quatre grans blocs:

- Desenvolupar un *smart contract*
- Estudiar vulnerabilitats
- Programar codi de qualitat
- Desenvolupar una interfície web

4 Metodologia i planificació

4.1 Metodologia

Pel desenvolupament d'aquest projecte s'ha decidit utilitzar una metodologia *agile*, ja que el projecte té unes funcionalitats molt concretes. Per emprar aquesta metodologia s'ha fet servir *KanbanFlow* [3], que m'ha permès obtenir una millor organització i planificació de les tasques i activitats a realitzar tant diàriament com setmanalment. En segon lloc, s'ha utilitzat *Sourcetree* [4], per tenir tot el repositori del projecte actualitzat i guardat, eina que a més m'ha permès fer un seguiment i control de versions, modificacions del codi i la documentació elaborada.

El taulell organitzatiu *KanbanFlow* utilitzat conté quatre columnes diferents:

- To do: Llista de tasques que s'han de realitzar durant l'sprint.
- Do today: Llista de les diferents tasques o activitats que s'han de desenvolupar avui.
- In progress: Tasca o tasques que s'estan realitzant i es troben en procés, amb un màxim de tres a la vegada.
- Done: Tasques i activitats finalitzades i supervisades.

Sourcetree m'ha permès tenir un repositori complet, format per dues carpetes principals. La primera carpeta, *codi*, conté tota la informació respecte a la part pràctica del projecte, és a dir, l'*smart contract*, el server, index.html, les llibreries necessàries, etc. La segona, *documentació*, conté tota la informació respecte a la documentació del projecte, com són els diferents informes i qualsevol informació rellevant pel projecte.

Per agilitzar i facilitar el desenvolupament del projecte, s'ha desenvolupat el projecte en un entorn local, per així comprovar el correcte funcionament d'una manera més ràpida i senzilla.

4.2 Planificació

La planificació ha sigut una tasca molt important, tenint en compte que les funcionalitats del projecte tenien una gran dependència entres elles. Per agilitzar aquesta tasca, es va dissenyar un diagrama de Gantt que podem veure a la Figura 7.

El projecte s'ha dividit en dues grans activitats. La primera d'elles fa referència a totes les tasques relacionades amb l'*smart contract* [5] que serà la base del projecte. La segona activitat fa referència a la interfície web [6], que serà la plataforma web amb la qual interactuaran els usuaris.

Durant el desenvolupament del projecte s'ha seguit la planificació prevista sense la necessitat de realitzar canvis importants, aconseguint finalitzar els pertinents sprints durant els terminis planificats. Tot i així, he dividit les diferents tasques en subtasques més específiques i he modificat el fil d'execució de les tasques en alguns moments, per evitar perdre molt de temps en tasques amb les quals m'havia trobat amb una dificultat que m'impedia avançar correctament.

5 Anàlisi de vulnerabilitats d'aleatorietat

Un dels factors més rellevants pel correcte funcionament d'aquest projecte, és la correcta implementació i utilització d'un mètode que garanteixi l'obtenció d'un nombre aleatori i imprevisible. A simple vista, sembla una tasca relativament senzilla, però s'ha de tenir en compte que Ethereum és incorruptible, descentralitzada, transparent i determinista. Aquestes característiques comporten unes certes dificultats a l'hora de programar un generador de nombres pseudoaleatoris. Cal remarcar la particularitat de determinista, la qual ens fa requerir d'un mètode que proporcioni un mateix nombre aleatori als diferents nodes de la xarxa.

Segons un estudi realitzat el 2018, dels quasi 4000 *smarts contracts* implementats a la blockchain d'Ethereum, 72 d'aquests contenen implementacions úniques d'aleatorietat i el 50% d'aquests se'ls va identificar com a vulnerables. A partir d'aquest estudi, podem arribar a la conclusió que hi ha poques implementacions que siguin segures i cal dedicar molt d'esforç en implementar-ne alguna que ho sigui.

5.1 Model d'adversari

La seguretat és un concepte relatiu, ja que hem de tenir en compte que aquesta depèn d'uns certs factors. Un d'aquests, és que hem de tenir clar contra qui competim o volem fer front, perquè no és el mateix implementar un *smart contract* segur contra simples usuaris que competir contra miners o usuaris experts. Per aquest motiu he volgut analitzar diferents implementacions i les seves possibles vulnerabilitats tenint en compte el possible atacant.

5.1.1 Simple Usuari

A la xarxa hi ha un gran nombre d'usuaris amb grans coneixements que poden detectar qualsevol vulnerabilitat per atacar-la i treure profit d'ella. La majoria d'aquests atacs consisteixen en contractes atacants que són específics cap a la funcionalitat vulnerable, que en aquest cas seria la de generar el nombre aleatori.

S'ha de tenir en compte que qualsevol decisió que prengui un usuari que afecti el resultat, li està oferint a aquest mateix usuari un avantatge injust. També s'ha de saber que tot el que el contracte veu, ho veu també la resta d'usuaris. Així que qualsevol mètode que utilitzi valors, els quals pot accedir qualsevol usuari, comporta una vulnerabilitat, ja que aquests utilitzaran un contracte atacant per cridar al mètode del contracte atacat, per així obtenir el mateix resultat. Tot i això, cal pensar, que només seràs objectiu d'atacs si comportes un gran interès pels atacants.

5.1.2 Miners

Els miners tenen un paper molt important a Ethereum. Són els responsables de minar els blocs, i per aquest motiu, d'afegir les diferents transaccions en un bloc. A causa de la seva funcionalitat, poden influenciar en aquells mètodes que utilitzin variables del bloc com a font d'entropia, com per exemple, data en la que es va extreure el bloc, l'altura del bloc actual, l'adreça del miner o la seva dificultat. Totes aquestes variables poden ser manipulades pels miners, i a més, qualsevol contracte atacant pot cridar al contracte perquè així els generi el mateix valor.

Tot i així, el cas dels miners és un cas particular, ja que aquests busquen obtenir el màxim benefici. Per minar un bloc, obtenen una recompensa de 2 ethers, més les comissions que obtenen per cadascuna de les transaccions d'aquest. Per aquest motiu no posaran en risc aquesta recompensa de minat, si la quantitat d'ethers que poden obtenir manipulant l'entropia d'un *smart contract* és menor.

5.1.3 Usuaris externs

Com més gran sigui la quantitat d'ethers que mou un *smart contract* més quantitat d'atacs pot patir aquest. Una solució per generar aleatorietat és l'ús d'oracles externs, que són uns serveis per aplicacions distribuïdes que proporcionen un pont entre la cadena de blocs i l'entorn extern. Aquest servei ens oferiria un nombre aleatori de manera segura, ja que els usuaris no podrien predir-lo. El problema que comporta aquest tipus de serveis és la centralització, ja que hauríem de confiar en la plataforma i en tota la seva infraestructura subjacent, i en que aquests no alteraran els resultats pel seu propi interès. Un fet que pot ser poc probable si s'estan manipulant grans quantitats d'ethers.

5.2 Mètodes per aconseguir aleatorietat

El mètode més senzill és utilitzant variables dels blocs com a font d'entropia, com l'altura del bloc o la dificultat, informació a la que tothom té accés i per aquest motiu no és segura. Per aconseguir una dificultat més elevada s'utilitza el *hash* o identificador de cada bloc de la cadena, d'aquesta manera l'obtenció d'aquesta aleatorietat no és trivial, sinó que està codificada. Tot i així qualsevol usuari pot obtenir aquest *hash* de verificació.

Una altra opció és la utilització d'oracles externs, els quals són servidors externs que et generen un nombre aleatori a partir d'una llavor. El problema d'aquesta implementació és que té un enfocament molt centralitzat i s'ha de confiar en aquest servidor.

Un altre mètode és el de compromís-revelació, en el qual hi ha una etapa de compromís en què els usuaris envien secrets protegits criptogràficament a l'*smart contract*. Tot seguit, l'etapa de revelació en què els usuaris anuncien les llavors de text sense xifrar, l'*smart contract* verifica que siguin correctes i les llavors s'utilitzen per generar el nombre aleatori.

Finalment, hi ha l'algoritme *Signidice* que consisteix en què cada usuari fa una aposta, aquesta és firmada amb la clau privada i s'envia a l'*smart contract*. Aquest verificarà la firma utilitzant la clau pública coneguda, i al final aquesta firma s'utilitzarà per generar un nombre aleatori.

5.3 Solució adoptada

Un cop finalitzat l'anàlisi de possibles vulnerabilitats, he pogut observar que la complexitat de generar un nombre pseudoaleatori va augmentant a mesura que creix el nivell d'implicació dels usuaris, als que haig de fer front, i a partir del seu nivell de coneixement.

Generar un nombre suficientment aleatori és molt complex, per aquest motiu he volgut establir contra qui haurà de fer front el meu *smart contract*. La quantitat d'ethers que manipularà l'*smart contract* no és excessivament elevada, perquè l'aposta mínima és de 0'1 ether, i la màxima de 4. Tot i així, es pot arribar a la casuística que s'arribi a manipular una gran quantitat d'ethers, ja que no es realitzarà una extracció d'aquests fons fins a assolir 144 ethers, que correspondria a la màxima recompensa que pot aspirar un usuari. Per aquest motiu, no només hauré de fer front a simples usuaris, sinó que es pot donar el cas que els miners i possibles usuaris externs actuïn per manipular i obtenir el nombre aleatori depenen del saldo de l'*smart contract*.

Per aquests motius, he decidit utilitzar l'oracle *Chainlink VRF* com a servidor que m'aportés aquesta font d'entropia, perquè és una font d'aleatorietat justa i verificable, dissenyada per l'ús a *smart contracts*. *Chainlink* genera a partir d'una llavor un nombre aleatori i una prova criptogràfica de com s'ha determinat aquest nombre, a partir de criptogràfia d'una clau pública. La prova es publica, i verifica la cadena abans que pugui ser utilitzada per l'aplicació consumidora, d'aquesta manera es garanteix que els resultats no puguin ser alterats ni manipulats per ningú. La part més important, és que l'*smart contract* determini una llavor que sigui difícil d'influir i predir, perquè aquesta és la que s'utilitzarà per generar el nombre aleatori.

5.4 Funcionament de Chainlink VRF

Chainlink VRF és una implementació de la funció aleatòria verificable (VRF) de *Goldberg* [7], la qual és completament imprevisible per qualsevol persona que desconegui la llavor o clau secreta. La clau secreta VRF és un nombre que l'oracle tria de la distribució uniforme *secp256k1* [8], que és la corba el·líptica utilitzada per la criptografia d'Ethereum. Associada a la clau secreta hi trobem també una clau pública, que s'utilitza per identificar l'oracle.

Quan un contracte consumidor sol·lícita aleatorietat, aquest proporciona una llavor, és important utilitzar algun valor difícil de predir. A més a més, la maquinària VRF, proporciona certa protecció bàsica en barrejar la llavor amb altres dades per evitar atacs de repetició.

Una vegada determinada la llavor, la maquinària transmet un registre d'Ethereum sol·licitant la sortida VRF corresponent de l'oracle especificat en la sol·licitud del consumidor. Al veure el registre, l'oracle construeix la sortida de la següent manera:

En primer lloc, aplica un *hash* a l'entrada de la corba, obtenint una mostra aleatòria criptogràficament segura de *secp256k1*, a partir de la llavor i la clau pública com a valors d'entrada. Realitza aquest procediment mitjançant el *hash* recursiu de les entrades utilitzant *keccak256* [9], fins que la sortida és menor a l'ordre del cos base de *secp256k1*. El resultat hauria de ser una funció de format $y=x+p$, en la qual *(x,y)* és el *hash* de l'entrada de la corba.

A continuació, multiplica *(x,y)*, com un punt de corba *secp256k1* per la seva clau secreta per obtenir un punt γ. El *hash keccak256* de γ, en format *uint256*, és la sortida VRF. Un cop obtinguda la sortida, genera una prova que γ és el mateix múltiple de *(x,y)* que la clau pública de l'oracle del generador *secp256k1*. Aquesta prova és molt similar a una firma de *Schnorr* [10]: primer, es mostra de forma segura un *nounce n* de *secp256k1*, com ja s'ha fet amb la clau secreta, llavors calcula $u = n*g$, on g és el generador *secp256k1* i pren l'adreça Ethereum de u. Després calcula $v = n*$ *(x,y)* i c com a resultat del *hash* de la clau pública, γ, *(x,y)* i la direcció u en mòdul *secp256k1*. A continuació, calcula $s = nc * k$ *mod(secp256k1)*, on k és la clau secreta. La prova

correspondrà a la clau pública, γ, c, s i la llavor d'entrada. Finalment, un cop generada la prova criptogràfica, es retorna a la maquinària VRF en la cadena, perquè verifiqui la prova i enviï la part corresponen de la sortida al contracte consumidor.

Per acabar, el contracte s'ha d'encarregar de comprovar la prova que li han subministrat. Per fer-ho, comprova que la clau pública y γ són punts vàlids de *secp256k1*, verifica que l'adreça del punt $c * pk + s * g$ coincideixi amb l'adreça d'u. Finalment calcula el *hash* a la corba (x,y) de la clau pública i la llavor, i verifica que aquest *hash*, la seva clau pública, γ, l'adreça d'u i $c * \gamma + s * (x,y)$ coincideixin amb c.

6 APLICACIÓ

6.1 Entorn

En aquest apartat comentaré i explicaré en detall les diferents tecnologies i llibreries que m'han facilitat el desenvolupament del projecte. Els diferents programaris són *ReactJS*, *Web3*, *EmailJS*, *SafeMath*, *Etherscan* i finalment *MetaMask*.

6.1.1 ReactJS

Per enllaçar el *backend* amb el *frontend*, i per desenvolupar la interfície web, s'ha creat un projecte amb *ReactJS*. *ReactJS* [11] és un dels *frameworks* més utilitzats i populars en els darrers anys, a causa de la facilitat en el procés d'escriptura de components. La programació orientada a components permet dividir el codi en diferents classes que hereten de la classe Component. Aquests components segueixen un cicle de vida en el qual els components van canviant d'estat. Una altra característica a remarcar és que *ReactJS* introdueix l'ús del *Document Object Model (DOM)* virtual, això juntament amb la característica de l'isomorfisme, permet garantir una major rapidesa entre la interacció de l'usuari i la interfície web. Finalment un altre factor que em va ajudar a decantar-me per l'elecció d'aquest framework és el seu ecosistema i la seva gran comunitat, que contenen moltes llibreries i funcionalitats que et faciliten la programació.

6.1.2 Web3

Una d'aquestes llibreries que és de molta utilitat és *Web3* [12]. Aquesta és una *Application Programming Interface (API)* en Javascript compatible amb Ethereum que implementa les especificacions genèriques *Remote Procedure Call (RPC)* en format JSON. La llibreria internament es comunica amb un node local a través de crides RPC, la qual cosa permet comunicar-se amb qualsevol node d'Ethereum. Aquesta llibreria el que ens permet és quan vulguem interactuar amb l'*smart contract*, aquesta es comunica amb un node de la blockchain d'Ethereum i especificant-li la direcció del *smart contract*, la funció que volem cridar i les variables que volem passar per paràmetre, tot això d'una manera senzilla.

6.1.3 EmailJS

Una altra llibreria a destacar, ha sigut *EmailJS*. *EmailJS* [13] és una llibreria que permet enviar correus electrònics utilitzant únicament la tecnologia de la part del client. No és necessari la implementació d'un servidor, sols és necessari connectar *EmailJS* a un dels servidors de correu electrònic, com per exemple Gmail, crear una plantilla de correu i efectuar la crida des de la part del client. Aquesta llibreria m'ha permès establir una comunicació del *smart contract* amb el *manager*, ja que aquest avisarà via correu electrònic al *manager*, quan no es disposin de suficients fons per continuar amb el correcte funcionament. A la Figura 1 es pot veure un exemple.

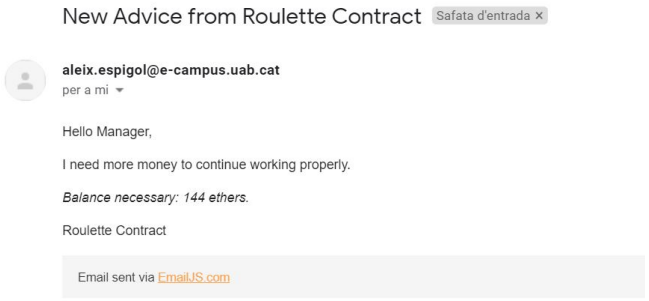


Fig. 1: Avís via correu electrònic del contracte

6.1.4 SafeMath

M'agradaria remarcar la importància de *SafeMath* [14], una biblioteca matemàtica de *Solidity*, dissenyada per suportar operacions segures matemàtiques. El seu objectiu consisteix a prevenir el desbordament quan s'està treballant amb variables de tipus enter (*int*). El desbordament es produeix quan una operació aritmètica intenta crear un valor numèric que està fora del rang en què es pot representar amb un nombre determinat de dígits, ja sigui perquè és major que el màxim o menor que el valor mínim representable. Gràcies a l'ús d'aquesta biblioteca es podrà evitar el desbordament en les operacions bàsiques matemàtiques com la suma, resta, multiplicació i divisió, i d'aquesta manera programar un *smart contract* més segur i fiable.

6.1.5 Etherscan

També voldria parlar sobre *Etherscan* [15], la plataforma d'anàlisi líder per Ethereum, una plataforma descentralitzada de contractes intel·ligents. Aquest explorador de blocs, consisteix en un motor de cerca que permet als usuaris buscar, confirmar i validar fàcilment transaccions que han tingut lloc a la blockchain d'Ethereum. Gràcies a aquesta eina, he pogut validar el correcte funcionament de l'*smart contract* i m'ha ajudat a solucionar algunes dificultats, com la mala estimació de gas que realitzava *MetaMask* en executar la funcionalitat de calcular el nombre aleatori. A la Figura 2 es pot veure la informació i les dades referent a l'*smart contract* de l'aplicació.

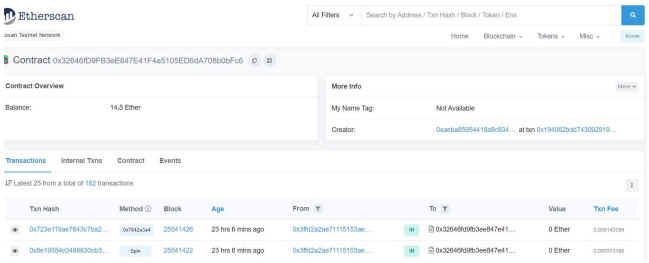


Fig. 2: Informació i detalls de l'smart contract

6.1.6 MetaMask

Finalment trobem *MetaMask* [16], un software que permet la interacció dels usuaris amb les DAPPs de blockchains. *MetaMask* és una extensió o puglin per navegador web que permet als usuaris comunicar-se fàcilment i de manera segura amb les DAPPs. Aquest funciona gràcies a l'ús de la llibreria *Web3*, que com ja hem comentat anteriorment permet crear aplicacions web que puguin interactuar amb Ethereum. *MetaMask* és l'únic requisit previ per la utilització de l'aplicació que s'ha desenvolupat, ja que sense aquest, no es pot disposar d'una wallet per Ethereum, ni es pot interactuar amb la DAPP.

6.2 Funcionalitats

Com ja s'ha dit anteriorment, l'aplicació consisteix en el joc de la ruleta i aquesta conté un conjunt de funcionalitats bàsiques que permeten el seu correcte funcionament. A la Figura 3 es pot veure el diagrama de casos d'ús que resumeix el funcionament. En l'aplicació es poden distingir dos tipus d'actors diferents, un simple jugador i el *manager* que correspondria al creador del joc. Aquest últim disposa d'una sèrie de privilegis i de funcionalitats que els altres usuaris no tenen accés, ni poden executar, com seria el cas que poden extreure diners del *smart contract*, el poden eliminar i poden modificar alguns paràmetres de les apostes. A continuació es detalla en què consisteix cada funcionalitat.

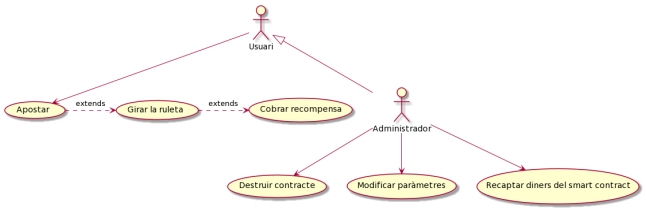


Fig. 3: Diagrama de casos d'ús

6.2.1 Apostar

Aquesta és la funcionalitat més important del joc, és la que permet a l'usuari realitzar una aposta. En aquest joc, els usuaris tenen la possibilitat de realitzar diferents tipus d'apostes. En aquest cas com a catàleg d'apostes l'usuari podrà triar entre tres tipus diferents. Podrà apostar a un nombre concret del 0 al 36, a un color concret (negre o vermell) i un nombre parell o imparell.

El procés per apostar consisteix en un conjunt de requeriments previs que validen si l'aposta pot ser acceptada. Aquestes validacions prèvies consisteixen a comprovar que l'usuari disposa de la quantitat que vol apostar, que el format d'aposta és correcte i que el valor apostat és major a l'aposta mínima i menor a l'aposta màxima. En cas de complir amb aquestes condicions, es fa una última comprovació interna, que consisteix a calcular si l'*smart contract* disposa de suficients fons per pagar el premi en el cas que fos una aposta guanyadora. Si es complís aquesta casuística que no es pot acceptar l'aposta per faltar de fons, l'*smart contract* enviarà un correu al *manager* informant que no hi ha suficients fons i són necessaris X ethers per seguir funcionant correctament.

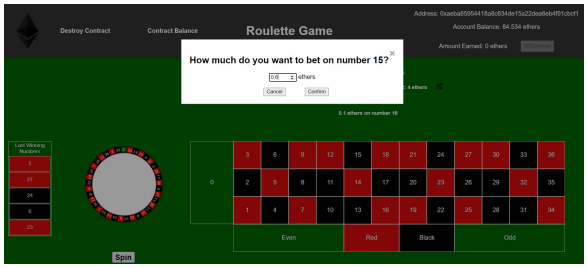


Fig. 4: Interfície web quan es vol apostar

6.2.2 Girar la ruleta

Una de les funcionalitats bàsiques i importants, és la de girar la ruleta, aquesta sols es podrà executar en cas que s'hagi fet alguna aposta. Aquesta funció compleix una labor molt important, ja que és l'encarregada d'obtenir el nombre guanyador i a partir de totes les condicions, calcular quines apostes són guanyadores i quines no.

En primer lloc ens centrarem a calcular el nombre guanyador. Com ja s'ha comentat anteriorment per obtenir una font d'entropia segura i fiable s'ha optat per utilitzar un oracle extern que ens aprovisiona aquesta aleatorietat. Aquest oracle es crida a partir d'un segon *smart contract*, que conté unes funcions específiques que permeten obtenir aquesta aleatorietat. Així que aquesta funció crida a aquest segon *smart contract* aprovisionant-li la llavor que s'utilitzarà per calcular el nombre aleatori, i espera al fet que el segon *smart contract* cridi a l'oracle i li retorni el nombre aleatori. Per cridar a l'oracle, el segon *smart contract* ha de disposar de tokens *LINK*, que són la moneda de pagament per obtenir aquesta aleatorietat.

Un cop obtingut el nombre aleatori, es procedeix a realitzar el procés de calcular les apostes guanyadores. El procediment consisteix a revisar una per una totes les apostes, per comprovar el tipus d'aposta que s'ha realitzat, i verificar si es correspon amb el nombre guanyador. En el cas d'esdevenir una aposta guanyadora, aquesta es marca per diferenciar-les de les altres. Finalment es calcula el premi que li correspon a cada aposta, segons el tipus d'aposta realitzada i la quantitat apostada i es procedeix a actualitzar els valors actuals de les variables i a comprovar si és necessari extreure diners a causa de la quantitat elevada de fons de l'*smart contract*. A la Figura 5 es pot veure el diagrama de seqüència que resumeix quins actors interactuen i com funciona aquesta funcionalitat.

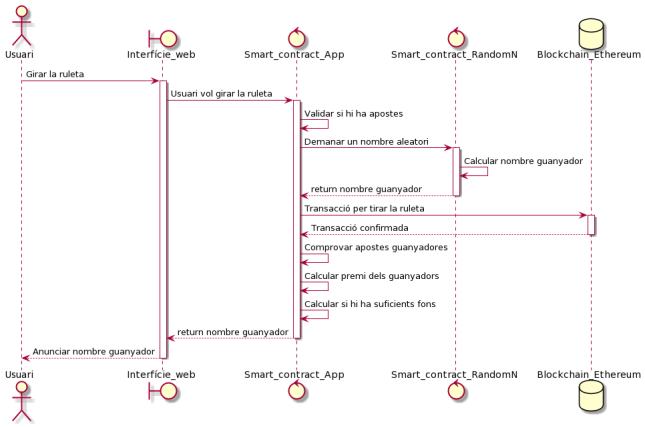


Fig. 5: Diagrama de seqüència de girar la ruleta

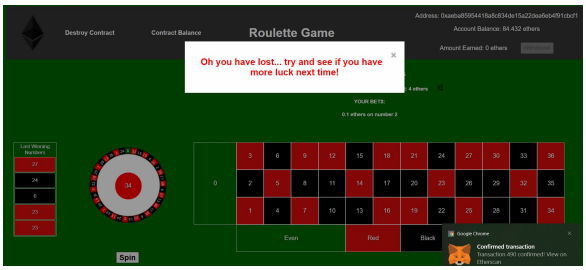


Fig. 6: Interfície web casuística d'aposta no guanyadora

6.2.3 Cobrar premi

L'usuari un cop s'ha girat la ruleta i ha sortit un nombre guanyador té la possibilitat de seguir jugant o d'extreure el premi obtingut. En cas de seguir jugant i guanyant el premi s'anirà acumulant. La recompensa obtinguda s'esdevé segons el tipus d'aposta, i d'aquesta manera de la probabilitat que hi ha d'encert. És a dir, en el cas d'apostar a un color o a la paritat del nombre guanyador, el premi obtingut consisteix en el doble de la quantitat apostada perquè sols hi ha un 50% de possibilitats d'encertar. En canvi en el cas d'apostar a un nombre concret el premi obtingut seria la quantitat apostada multiplicada per 36, que són el conjunt de possibles nombres que poden sortir.

6.2.4 Recaptar diners

Aquesta funcionalitat és la que regula la quantitat de fons que disposa l'*smart contract*, per evitar acumular grans quantitats d'ethers i esdevenir una possible font d'atacs. D'aquesta manera el *manager* té la possibilitat de cridar aquesta funció per extreure fons de l'*smart contract* i en cas que no, aquesta funció es crida de manera automàtica en superar un llindar fixat. Aquest llindar consisteix en una quantitat calculada que és necessària sempre en l'*smart contract* perquè aquest funcioni correctament, en el cas de no superar aquest llindar no es permet l'extracció de diners. El llindar té una relació directament proporcional amb l'aposta màxima, ja que aquest es calcula a partir de la màxima recompensa a la qual pot aspirar un usuari, que en aquest cas correspondria a realitzar una aposta màxima a un nombre concret.

6.2.5 Modificar paràmetres

El *manager* té la possibilitat de modificar la quantitat establerta d'aposta mínima i màxima, segons l'interessi. Aquest interès pot venir donat per l'historial d'apostes. En el cas que les apostes realitzades siguin de quantitats molt altes, seria interessant en un futur augmentar els valors de les apostes, en el cas que siguin baixes s'hauria de produir una disminució dels valors. Aquesta funció és simple, consisteix a modificar aquests paràmetres per part del *manager* i d'aquesta manera modificar de manera automàtica tots els paràmetres que estiguin relacionats amb aquestes variables, com per exemple el llindar per extreure diners.

6.2.6 Eliminar smart contract

Com ja he comentat anteriorment, l'aplicació funciona de manera descentralitzada a partir d'un *smart contract* i aquest es troba penjat a la blockchain de forma permanent i pública. Per aquest motiu vaig desenvolupar una funció que consisteix a eliminar l'*smart contract*, és a dir, destruir-lo per si algun dia vull eliminar l'aplicació. Aquesta funció el que fa és autodestruir l'*smart contract* i transferir tot el seu saldo actual al creador de l'aplicació. Aquest procés sols pot ser executat pel *manager* .

7 LÍNIES FUTURES

Un cop finalitzat el projecte objectiu, voldria comentar una sèrie de millores que serien interessants incorporar en un futur, per millorar la qualitat de l'aplicació.

En primer lloc, seria interessant modificar l'estructura del projecte *ReactJS*, ja que he descobert que programar el codi amb *Hooks*, en comptes de components i classes, ofereix un millor rendiment del *scope* del *framework* i una estructura més organitzada.

En segon lloc, seria important investigar i utilitzar algun mètode més complex per obtenir el nombre aleatori. D'aquesta manera s'aconseguiria una major aleatorietat i seguretat en l'obtenció del nombre guanyador. Aquesta millora juntament amb l'anterior comentada, agilitzarien l'experiència de joc de l'usuari, ja que a vegades el procés és molt lent.

Referent a la velocitat de l'experiència de joc, seria interessant buscar si existeix algun mètode que permetis autoconfirmar totes les transaccions, per així evitar que l'usuari hagi de confirmar una per una cada transacció que vulgui realitzar.

També seria interessant, crear un mètode que actualitzes els valors d'aposta màxima i mínima de forma autònoma a partir dels valors de les apostes realitzades pels usuaris. Així s'aconseguiria reduir aquesta dependència cap al *manager*.

Finalment, una bona idea per millorar la qualitat de l'experiència de joc de l'usuari, seria programar una interfície web més visual i atractiva pels jugadors.

8 CONCLUSIONS

Per concloure aquest projecte podem afirmar que s'han pogut complir exitosament els objectius plantejats en el començament del treball, tot i les dificultats amb les quals m'he trobat. Crec que he sabut organitzar-me correctament i no se m'ha acumulat la càrrega de treball, també crec que he sabut gestionar els imprevistos amb els quals m'he topat i he sabut solucionar-los correctament. Totes aquestes dificultats m'han ajudat a aprendre d'elles i evitar-les en un futur, a més a més, m'han ajudat en la meva capacitat de buscar informació i solucions per internet.

Haig d'admetre que un cop vaig finalitzar la part referent a l'*smart contract*, vaig pensar que la major part de la feina ja estava feta, ja que és la part que considero que és més complexa i difícil del projecte i que té més importància. Em vaig confiar, pensant que el treball ja estava quasi acabat, però em vaig adonar de la dificultat que comporta programar totes les possibles casuístiques que pot realitzar l'usuari i de desenvolupar la connexió entre la interfície i l'*smart contract*.

En aquest punt, vull remarcar la importància que té la interfície web, ja que sense aquesta, l'*smart contract* no tindria sentit perquè no se'n podria fer ús. La interfície actua d'intermediària entre l'usuari i l'aplicatiu, ja que realitza les crides a les funcionalitats de l'*smart contract* i en facilita al seu ús, perquè un usuari inexpert desconeixeria com executar i interactuar amb un *smart contract*.

Finalment voldria concloure parlant sobre un dels temes principals del treball, l'anàlisi de vulnerabilitats d'aleatorietat, gràcies a aquest treball m'he adonat que la complexitat de generar un nombre pseudoaleatori va augmentant a mesura que creix el nivell d'implicació dels usuaris, als que haig de fer front, i a partir del seu nivell de coneixement. També he après la importància que té la seguretat en treballar amb programes i informació que es troba penjada públicament per tothom, com seria el cas dels *smart contracts*.

AGRAÏMENTS

En primer lloc, vull donar les gràcies al meu tutor del treball de final de grau, Jordi Herrera, per totes les recomanacions, suport i ajuda que m'ha ofert durant tot el procés i que m'ha animat a investigar tot allò que desconeixia.

També vull agrair a tota la família i amics pels consells i suport moral que m'han donat durant tot aquest període. Finalment voldria agrair a l'Adrià Massanet per ajudar-me amb aquelles dificultats que no sabia fer front.

REFERÈNCIES

[1] Bit2Me Academy, "Smart Contracts: ¿Qué son, cómo funcionan y qué aportan?," Jan. 30, 2021. https://academy.bit2me.com/que-son-los-smart-contracts/ (accessed Feb. 20, 2021).

[2] Chainlink Developers, "Getting Started - Chainlink," 2018. https://docs.chain.link/docs (accessed Apr. 21, 2021).

[3] KanbanFlow, L-Lean project management. Simplified.," Feb. 10, 2011. https://kanbanflow.com/ (accessed Feb. 25, 2021).

[4] Sourcetree, "Free Git GUI for Mac and Windows," Mar. 12, 2013. https://www.sourcetreeapp.com/ (accessed Feb. 25, 2021).

[5] Quora, "How long does it take to program a Smart Contract?", Feb. 8, 2020. https://www.quora.com/How-long-does-it-take-to-program-a-Smart-Contract. (accessed Feb. 20, 2021).

[6] Toni, "How Long Does It Take to Develop a Web App," Oct. 24, 2020. https://gbksoft.com/blog/how-long-does-it-take-to-develop-a-web-app/ (accessed Feb. 20, 2021).

[7] D. Papadopoulos et al., "Making NSEC5 Practical for DNSSEC," Feb. 2017. https://eprint.iacr.org/2017/099.pdf. (accessed May 30, 2021).

[8] Bitcoin Wiki, "Secp256k1 - Bitcoin Wiki," Apr. 2019. https://en.bitcoin.it/wiki/Secp256k1 (accessed Jun. 3, 2021).

[9] Wikipedia, "SHA-3 - Wikipedia." https://en.wikipedia.org/wiki/SHA-3 (accessed Jun. 3, 2021).

[10] Wikipedia, "Schnorr signature - Wikipedia." https://en.wikipedia.org/wiki/Schnorrsignature (accessed Jun. 3, 2021).

[11] J. Walke, "React – A JavaScript library for building user interfaces," May 29, 2014. https://reactjs.org/ (accessed Apri. 25, 2021).

[12] G. Wood, "web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation," 2016. https://web3js.readthedocs.io/en/v1.3.4/ (accessed Mar. 21, 2021).

[13] EmailJS, "EmailJS," 2017. https://www.emailjs.com/ (accessed May. 25, 2021).

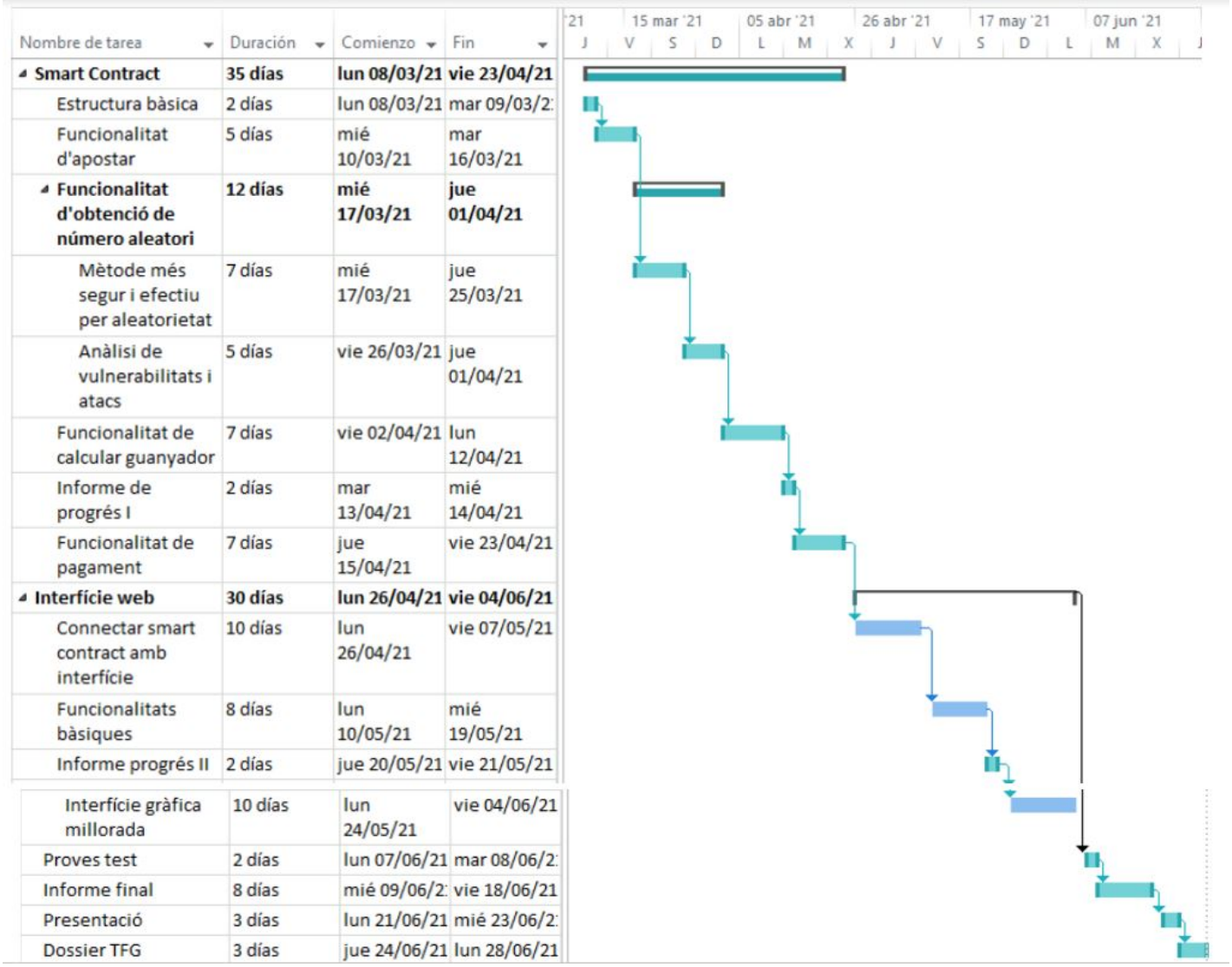
[14] Jdourlens, "Using Safe Math library to prevent from overflows - EthereumDev," Apr. 05, 2020. https://ethereumdev.io/using-safe-math-library-to-prevent-from-overflows/ (accessed Apr. 21, 2021).

[15] M. Tan, "Ethereum (ETH) Blockchain Explorer," Jul. 2007. https://etherscan.io/ (accessed Mar. 15, 2021).

[16] A. Davis and D. Finlay, "MetaMask," 2016. https://metamask.io/ (accessed Mar. 10, 2021).

[17] J. Spolsky, "Stack Overflow - Where Developers Learn, Share, Build Careers," 2008. https://stackoverflow.com/ (accessed Mar. 10, 2021).

APÈNDIX



Nombre de tarea	Duración	Comienzo	Fin
Smart Contract	**35 días**	**lun 08/03/21**	**vie 23/04/21**
Estructura bàsica	2 días	lun 08/03/21	mar 09/03/2
Funcionalitat d'apostar	5 días	mié 10/03/21	mar 16/03/21
Funcionalitat d'obtenció de número aleatori	**12 días**	**mié 17/03/21**	**jue 01/04/21**
Mètode més segur i efectiu per aleatorietat	7 días	mié 17/03/21	jue 25/03/21
Anàlisi de vulnerabilitats i atacs	5 días	vie 26/03/21	jue 01/04/21
Funcionalitat de calcular guanyador	7 días	vie 02/04/21	lun 12/04/21
Informe de progrés I	2 días	mar 13/04/21	mié 14/04/21
Funcionalitat de pagament	7 días	jue 15/04/21	vie 23/04/21
Interfície web	**30 días**	**lun 26/04/21**	**vie 04/06/21**
Connectar smart contract amb interfície	10 días	lun 26/04/21	vie 07/05/21
Funcionalitats bàsiques	8 días	lun 10/05/21	mié 19/05/21
Informe progrés II	2 días	jue 20/05/21	vie 21/05/21
Interfície gràfica millorada	10 días	lun 24/05/21	vie 04/06/21
Proves test	2 días	lun 07/06/21	mar 08/06/2
Informe final	8 días	mié 09/06/2	vie 18/06/21
Presentació	3 días	lun 21/06/21	mié 23/06/2
Dossier TFG	3 días	jue 24/06/21	lun 28/06/21

Fig. 7: Diagrama de Gantt

```
import "@chainlink/contracts/src/v0.6/VRFConsumerBase.sol";

contract RandomNumber is VRFConsumerBase {

    //variables for random number
    bytes32 internal keyHash; //hash of public key
    uint256 internal fee;
    uint256 public randomResult; //result of random number
    bytes32 public reqId;

    constructor()
        VRFConsumerBase(
            0xdD3782915140c8f3b190B5D67eAc6dc5760C46E9, // VRF Coordinator
            0xa36085F69e2889c224210F603D836748e7dC0088  // LINK Token
        ) public
        {
            //constructor of oracle
            keyHash = 0x6c3699283bda56ad74f6b855546325b68d482e983852a7a82979cc4807b641f4;
            fee = 0.1 * 10 ** 18; // 0.1 LINK
        }

    //Requests randomness from a user-provided seed
    function getRandomNumber(uint256 userProvidedSeed) public returns (bytes32 requestId) {
        require(LINK.balanceOf(address(this)) >= fee, "Not enough LINK - fill contract with faucet");
        return requestRandomness(keyHash, fee, userProvidedSeed);
    }

    //Callback function used by VRF Coordinator
    function fulfillRandomness(bytes32 requestId, uint256 randomness) internal override {
        reqId = requestId;
        randomResult = randomness;
    }

    function getRandomResult() public returns (uint256){
        return randomResult;
    }
}
```

Fig. 8: Codi del Random Contract