Memoria Proyecto Final

1. Contrato QuadraticVoting

1.1 Variables del contrato

Enfocadas en poder realizar el proyecto sin iteraciones innecesarias y protegiendo los contratos de cualquier ataque

address public **owner**: Address del creador

address private **token**: Address del token

uint256 public **tokenPrice**: Precio token

uint256 private **maxNumToken**: Máximo número de tokens que se pueden comprar

bool public **voteOpen**: Votaciones abiertas o cerradas

uint private **currentEther**: Ether que tenemos para pagar propuestas

struct activeParticipant { Estructura para cada participante en cada propuesta

uint numVotes: Número de votos del participante

bool active: booleano para comprobar que es un participante que se ha dado de alta

}

mapping(**address** => **mapping**(**uint** => **activeParticipant**)) private **numVotesParticipants**:

Mapping para saber el participante “x” (address) a la propuesta “y” (uint) cuantos votos ha depositado y si es participante, en el mapping del address de cada participante clave 0 almacenaremos el número de votos totales del participante ya que no voy a tener ninguna propuesta como clave el 0 y si se ha dado de alta o no. Ejemplo:

numVotesParticipant[address][0].numVotes: Numero de votos totales

numVotesParticipant[address][0].active: Se ha dado de alta o no este address

numVotesParticipant[address][1].numVotes: Numero de votos a la propuesta 1

numVotesParticipant[address][1].active: Ya no se le da uso a este campo

uint private **proposalIds**: Id de las propuestas que las empiezo a asignar desde el 1 y va aumentando con cada propuesta nueva.

mapping(**uint** => **Proposal**) private **proposals**: Mapping de todas las propuestas

struct Proposal{ Struct para cada propuesta

uint proposalId: Id propuesta

string title: Título de la propuesta

string description: Descripción de la propuesta

uint256 budget: Cantidad necesaria para aprobar la propuesta

uint256 totalBudget: Cantidad de ether actual disponible

uint256 numVotes: Número de votos totales de la propuesta

uint256 numTokens: Numero de tokens que se han depositado

address dest: Dirección del contrato en donde hay que depositar la cantidad que pedia la propuesta al llegar a su objetivo

address creator: Creador de la propuesta

uint state: estado de la propuesta 1 = pending, 2 = approved, 3 = signaling

address[] voters: array con el address de los votantes si uno retira sus voto no se le saca porque en el mapping “numVotesParticipant” veo que a esta propuesta tiene 0 votos y me sale mas barato que estar constantemente quitando y agregando participantes en el array

}

uint public **numProposals**: Número de propuestas no aprobadas

uint public **numParticipants**: Número de participantes totales formula

uint[] private **pendingProp**: Array de los ids de las propuestas pending

uint[] private **approvedProp**: Array de los ids de las propuestas aprobadas

uint[] private **signalinProp**: Array de los ids de las propuestas signaling

mapping(**address** => **bool**) private **locks**: mapping con un lock para cada usuario para que no puedan realizar ataques de reentrada por ejemplo entrar dos veces seguidas en stake y pagando lo mismo en ambas votaciones cuando la última debe ser mas cara.

bool private **lock**: Lock general para las funciones más vulnerables que solo pueda acceder a ellas una única persona.

```
modifier onlyOwner{
    require(msg.sender == owner);
    _;
} modificador para comprobar si es el creador del contrato
modifier proposalExists(uint proposalId){
    require(proposalId < proposalIds && (proposals[proposalId].state == 1 ||
proposals[proposalId].state == 3));
    _;
} modificador para ver que ha elegido una propuesta valida y si es pending o
signaling
modifier isParticipant {
    require (numVotesParticipants[msg.sender][0].active == true);
    _;
} modificador para ver si es participante o no el usuario que ejecuta
modifier voteIsOpen(){
    require(voteOpen == true);
    _;
} modificador para ver si la votación está abierta
```

1.2 Funciones del contrato

Todas con coste constante menos las que obligatoriamente necesitan realizar una iteración en algún bucle o mapping, esto nos permite tener un precio de gas lo mas barato posible.

constructor(uint prices, uint maxTokens):

Si nos proporciona el creador un precio o máximo número de tokens no válido hago un revert.

Empezamos a asignar ids desde el 1.

Creo el contrato que gestiona mis tokens y guardo su address para realizar desde ahí las operaciones que yo quiera.

function addParticipant() external payable:

Compruebo que la cantidad proporcionada es mayor o igual que un token y que no se había añadido ya antes a este participante

Miramos que no exceden el máximo número de tokens ya definido.

Creamos los tokens necesarios a través de la interfaz con la función mint.

El ether que sobre al sacar el número de tokens nos lo quedamos como "tip" del participante para financiar otras propuestas.

Damos de alta al participante

Función con conste constante.

function addProposal(string memory titulo, string memory descripcion, uint256 presupuesto, address destino) voteIsOpen isParticipant external returns (uint):

Solo la pueden ejecutar los participantes y si la votación está abierta.

Fijo el máximo presupuesto a pedir en 1e30 porque ya es un valor muy alto y evito problemas de vulnerabilidad con overflow.

Función con conste constante.

function cancelProposal(uint propoId) voteIsOpen proposalExists(propoId) external:

Comprueba que la votación está abierta y si la propuesta existe.

Compruebo que sea el creador y que su lock está abierto para evitar que este entre dos veces en la función y recuperen x2 sus votos los participantes que hayan votado a esta propuesta y nos saquen dinero

Función con conste lineal.

function buyTokens() isParticipant voteIsOpen external payable:

Compruebo que la votación esté abierta y sea participante.

Compruebo que no supere el máximo de tokens establecido

Función con coste constante.

function sellTokens() isParticipant voteIsOpen external:

Compruebo que sea participante y que la votación esté abierta

Compruebo que me haya dado el approve de los tokens que tiene en su cuenta.

Compruebo que el lock esté abierto para evitar que el mismo usuario entre dos veces, venda dos veces y nos saque beneficio.

Si al vender no tiene ningún voto a ninguna propuesta deja de ser participante.

Función con coste constante.

function stake(uint proposId, uint votes) isParticipant voteIsOpen proposalExists(proposId) external:

Compruebo que sea participante, la votación esté abierta y la propuesta exista.

Compruebo que el lock del usuario esté abierto para evitar que el mismo usuario deposite dos votos en dos llamadas distintas y le salgan por el mismo precio sin aplicar la ley cuadrática y luego me saque beneficio al devolverlos y venderlos.

Compruebo que los votos que me ha pasado por parámetro no hacen overflow mediante SafeMath ya que pueden pasarme un valor de 2**256 e intentar aprovechar esa vulnerabilidad

Compruebo que el participante tiene los tokens suficientes para realizar este voto y que me ha dado approve

Después de que el participante deposite el voto llamo desde esta función a _checkAndExecuteProposal para comprobar si hay que ejecutar la propuesta

Función con coste constante.

function withdrawFromProposal(uint votes, uint proposId) isParticipant voteIsOpen proposalExists(proposId) external:

Compruebo que sea participante, que la votación esté abierta y que la propuesta existe.

Calculo la cantidad de tokens que le corresponden para realizar el transfer

Compruebo que son validos el número de votos que quiere retirar

Compruebo que los votos que me ha pasado por parámetro no hacen underflow mediante SafeMath ya que pueden pasarme un valor de 2**256 e intentar aprovecharse.

Compruebo que el lock del usuario esté abierto para evitar que el mismo usuario retire dos votos en dos llamadas distintas y le devuelvan más tokens de los que le corresponden.

Función con coste constante.

function _checkAndExecuteProposal(uint proposalId) internal:

Calculo el threshold y miro si se cumplen las dos condiciones para ejecutar una propuesta, si se cumple activo el lock para que al ejecutar la función “executeProposal” no intente entrar en “withdrawFromProposal” a la vez y saquen tokens de la propuesta que se acaba de aceptar aprovechándose.

Función con coste constante.

function closeVoting() voteIsOpen onlyOwner external:

Compruebo que lo ejecute el creador del contrato y que la votación esté abierta.

Función con coste lineal.

2. Contrato ERC20

Las modificaciones que he realizado en mi contrato que gestiona los tokens son:

Al constructor le paso un address property, que es el único que puede llamar a las funciones mint, burn y burnFrom este address corresponde al contrato QuadraticVoting por lo tanto solo este puede llamar a las funciones dichas.