



Blockchain  
Labwork 1 : Smart Contract de una Subasta

Adrián Robles Camporro, David Olivares Muñoz

November, 2023

# Contents

<b>1</b>	<b>Análisis y definición del escenario</b>	<b>3</b>
<b>2</b>	<b>Diseño</b>	<b>3</b>
2.1	Casos de uso . . . . .	3
<b>3</b>	<b>Implementación</b>	<b>6</b>
<b>4</b>	<b>Lecciones aprendidas</b>	<b>9</b>
<b>5</b>	<b>Pruebas</b>	<b>9</b>

# 1 Análisis y definición del escenario

Nos encontramos en un momento en el que las blockchains tienen una gran popularidad y se utilizan en numerosos escenarios por sus numerosas ventajas. En nuestro caso, se ha optado por utilizar una blockchain pública como Ethereum para implementar el Smart Contract para una subasta de arte, en la cual un usuario inicia la subasta con las obras de arte que quiera y otros usuarios pueden pujar por ellas. Entre los motivos que llevan a la necesidad de utilizar una blockchain pública para este caso, encontramos los siguientes:

- **Transparencia y Confianza:** Ethereum es una cadena de bloques pública y descentralizada, lo que significa que todas las transacciones son visibles y verificables por cualquiera. Esto aumenta la transparencia y la confianza en el proceso de subasta.
- **Inmutabilidad:** Una vez que se registra una oferta en Ethereum, no se puede cambiar ni eliminar, lo que garantiza la integridad del proceso de subasta y evita la manipulación de ofertas.
- **Contratos Inteligentes:** Ethereum permite la creación de contratos inteligentes, que son programas autónomos que ejecutan automáticamente las reglas de la subasta. Esto elimina la necesidad de un intermediario y garantiza que las reglas se sigan de manera imparcial.
- **Historial Público:** Todas las ofertas y transacciones relacionadas con la subasta se registran en la cadena de bloques, lo que proporciona un historial público que los participantes pueden verificar en cualquier momento.
- **Acceso Global:** Ethereum es una red global, lo que significa que cualquier persona en cualquier parte del mundo puede participar en la subasta sin restricciones geográficas.
- **Seguridad:** La seguridad de Ethereum se basa en su robusta infraestructura y su inmenso poder de cómputo, lo que hace que sea muy difícil de hackear.

## 2 Diseño

### 2.1 Casos de uso

A continuación se detallan los casos de uso del Smart Contract:

Table 1: Caso de Uso 1: Iniciar Subasta (CU-01)

Campo	Descripción
ID del Caso de Uso	CU-01
Nombre del Caso de Uso	Iniciar Subasta
Descripción	Permite a los propietarios de obras de arte iniciar la subasta de una obra de arte concreta.
Actores Involucrados	Propietario de las obras
Escenarios Principales	1. El propietario proporciona el título de la obra de arte y el precio inicial con el que sale a subasta. 2. Se crea un nuevo objeto <code>ObraDeArte</code> y se establece el momento de final de subasta. Además, se añade al propietario como participante de la subasta. 3. Se inicia el período de subasta.

Table 2: Caso de Uso 2: Realizar oferta (CU-02)

<b>Campo</b>	<b>Descripción</b>
ID del Caso de Uso	CU-02
Nombre del Caso de Uso	Realizar oferta
Descripción	Permite a los usuarios realizar ofertas de dinero por una obra de arte.
Actores Involucrados	Usuarios con Ethers
Escenarios Principales	<ol style="list-style-type: none"> <li>1. El usuario decidirá pujar una cantidad de Ethers por una obra de arte.</li> <li>2. Si es la primera vez que puja se añade al usuario a la lista de participantes.</li> <li>3. Si la cantidad de dinero apostada por este usuario es la mayor hasta el momento, se convierte en el ganador provisional.</li> <li>4. Si realiza una nueva apuesta, estará sumando esta nueva cantidad a lo que ya había apostado.</li> </ol>

Table 3: Caso de Uso 3: Finalizar Subasta (CU-03)

<b>Campo</b>	<b>Descripción</b>
ID del Caso de Uso	CU-03
Nombre del Caso de Uso	Finalizar subasta
Descripción	Permite al propietario cerrar la subasta una vez haya finalizado el tiempo de subasta.
Actores Involucrados	Propietario
Escenarios Principales	<ol style="list-style-type: none"> <li>1. El propietario decide cerrar la subasta de una obra de arte concreta (el tiempo de pujas debe haber finalizado).</li> <li>2. Si hay un ganador se traspasan los ethers pujados por este usuario al propietario y se llama a la función recuperarFondos para que el resto de usuarios que hayan pujado recuperen su inversión.</li> </ol>

Table 4: Caso de Uso 4: Obtener participantes (CU-04)

<b>Campo</b>	<b>Descripción</b>
ID del Caso de Uso	CU-04
Nombre del Caso de Uso	Obtener participantes de una subasta
Descripción	Permite al propietario o a un usuario que puja saber qué personas están participando en la subasta.
Actores Involucrados	Propietario, Usuarios que pujan
Escenarios Principales	<ol style="list-style-type: none"> <li>1. Se solicita saber quienes están participando en la subasta y esta información es devuelta.</li> </ol>

Table 5: Caso de Uso 5: Obtener participantes (CU-05)

Campo	Descripción
ID del Caso de Uso	CU-05
Nombre del Caso de Uso	Obtener puja de un participante
Descripción	Permite al propietario o a un usuario que puja saber qué cantidad ha pujado un usuario concreto para una obra concreta.
Actores Involucrados	Propietario, Usuarios que pujan
Escenarios Principales	1. Se solicita saber que cantidad ha pujado un usuario para una obra concreta (se pasan como parámetros).

Table 6: Caso de Uso 6: Obtener título de una obra (CU-06)

Campo	Descripción
ID del Caso de Uso	CU-06
Nombre del Caso de Uso	Obtener título de una obra
Descripción	Permite al propietario o a un usuario que puja saber qué obra está en puja ahora mismo.
Actores Involucrados	Propietario, Usuarios que pujan
Escenarios Principales	1. Se solicita saber el título de la obra en subasta (Se pasa el id).

A continuación, se puede ver un diagrama de flujo del funcionamiento del Smart Contract así como el diagrama de los casos de uso, los cuales dan una visión más detallada del funcionamiento del contrato:

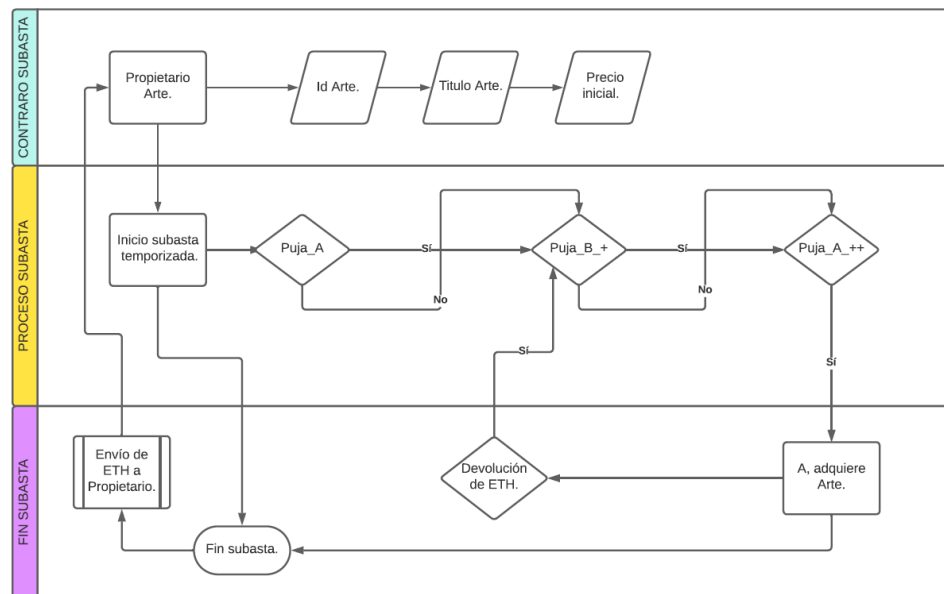


Figure 1: Diagrama de flujo del contrato

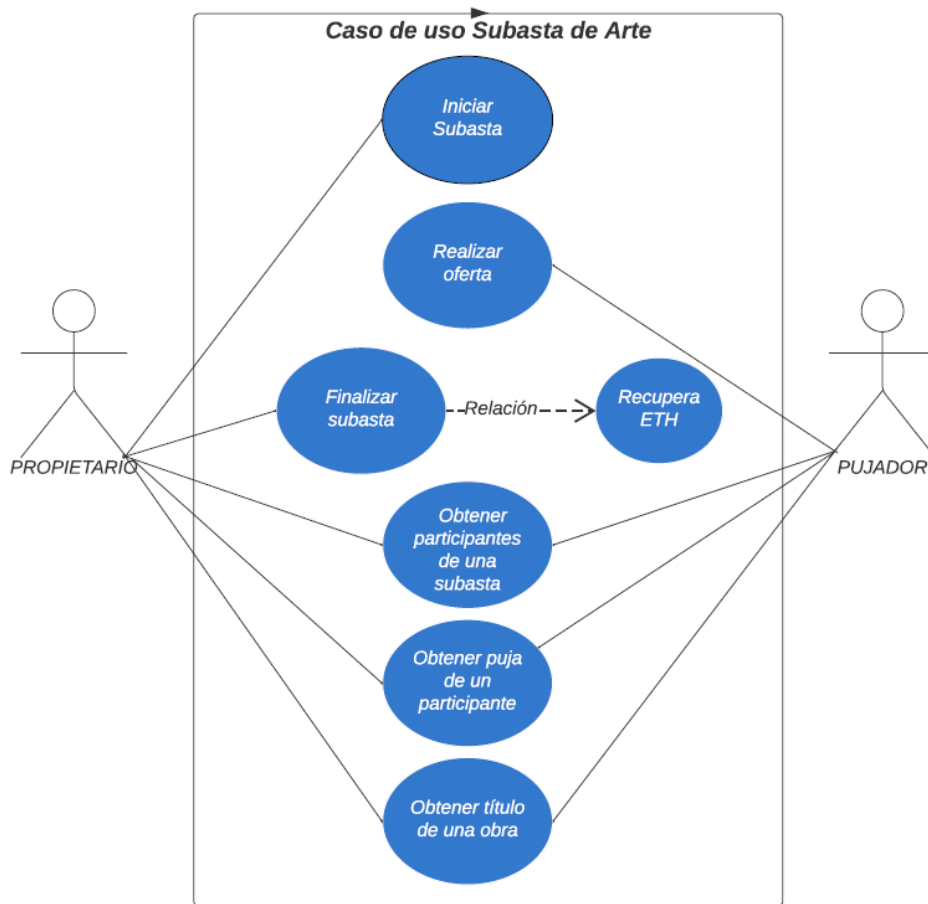


Figure 2: Diagrama de casos de uso del contrato

### 3 Implementación

A la hora de hablar de implementación, en primer lugar, se ha optado por definir unas constantes comunes para todas las subastas como: el propietario, la duración de la subasta, el precio mínimo de la obra de arte, el momento final de la subasta, un booleano para indicar si la subasta ha finalizado o no y un id de la obra actual siendo subastada.

A continuación, se define la estructura `ObraDeArte` que posee un título, el precio ganador de la subasta, la address del ganador y un booleano que indica si la obra ya ha sido vendida o no.

---

```

1  struct ObraDeArte {
2      string titulo;
3      uint precioGanador;
4      address ganador;
5      bool vendida;
6  }

```

---

También son necesarios tres mappings y dos eventos:

---

```

1  mapping(uint => ObraDeArte) public obras;
2  mapping(address => mapping(uint => uint)) ofertas;
3  mapping(uint => address[]) participantesPorObra;
4

```

---

```
5 event NuevaOferta(uint obraID, address participante, uint cantidad);
6 event SubastaFinalizada(uint obraID, address ganador, uint cantidad);
```

---

El mapping 'obras' almacena cada objeto ObraDeArte asociándolo a un ID, el mapping 'ofertas' almacena para cada usuario la cantidad que ha pujado para una obra concreta y por último, el mapping 'participantesPorObra' almacena para cada obra un array con las address de los usuarios participantes en la subasta de esa obra.

Hablando de los eventos, encontramos 'NuevaOferta' que se lanza cuando un usuario aporta una cantidad de dinero nueva y 'SubastaFinalizada' que se lanzará cuando el propietario cierre la subasta y se indicará el ganador de la misma si es que lo hay.

El constructor del contrato es bastante sencillo, quien cree el contrato debe indicar el tiempo de duración de la subasta y pasará a ser el propietario, por tanto sólo él podrá iniciar subastas de obras de arte. Además, se definen dos modifiers que se utilizarán en las funciones: uno para asegurar que sólo el propietario ejecuta una función y otro para asegurar que la subasta aún está activa (el tiempo de subasta no ha acabado y la obra no ha sido vendida). Tanto en el constructor como en algún modifier se utiliza el comando `block.timestamp` que devuelve el momento de creación del bloque (segundos desde una fecha concreta).

---

```
1 constructor(uint _duracionSubasta) {
2     propietario = msg.sender;
3     duracionSubasta = _duracionSubasta;
4     momentoFinalSubasta = block.timestamp + duracionSubasta;
5     subastaFinalizada = true;
6     obraActual = 0;
7 }
8
9 modifier soloPropietario() {
10     require(msg.sender == propietario, "Solo el propietario
11     puede llamar a esta funcion.");
12     _;
13 }
14
15 modifier subastaActiva(uint obraID) {
16     require(block.timestamp < momentoFinalSubasta, "La subasta ha finalizado.");
17     require(!obras[obraID].vendida, "La obra de arte ya ha sido vendida.");
18     _;
19 }
```

---

A continuación, se detalla la implementación de las funciones:

Hay tres funciones básicas que se utilizan para obtener información, ya sean los participantes en una subasta, la cantidad pujada por una persona para una obra o el título de una obra de arte que se haya subastado. Estas funciones simplemente acceden a los mappings para devolver la información deseada por el usuario.

---

```
1 function obtenerParticipantes(uint _obraID) public view
2 returns (address[] memory) {
3     return participantesPorObra[_obraID];
4 }
5
6 function obtenerSaldo(address participante, uint _obraID) public view
7 returns (uint) {
8     return ofertas[participante][_obraID];
9 }
10
11 function obtenerTituloObra(uint _obraID) public view returns (string memory) {
```

```

12     return obras[_obraID].titulo;
13 }

```

---

La función 'iniciarNuevaSubasta' recibe como parámetros el título de una obra de arte y el precio con el que sale a subasta. En primer lugar, se comprueba que sea el propietario quién ejecuta esta función y que la subasta de la obra anterior haya finalizado (en una subasta se va producto a producto) así como que la obra anterior se encuentre en estado vendida (esto quiere decir que el propietario haya cerrado la subasta). Tras esto se suma uno al ID para identificar esta obra, se crea el objeto 'ObraDeArte' correspondiente que se añade al mapping obras y se establece el momento de finalización de la subasta.

```

1  function iniciarNuevaSubasta(string memory _titulo, uint _precioInicial) public
2  soloPropietario {
3      require(subastaFinalizada == true, "La subasta anterior no ha finalizado.");
4      require(obraActual == 0 || obras[obraActual].vendida, "La obra de arte actual
5  no se ha vendido.");
6
7      obraActual++;
8      subastaFinalizada = false;
9      precioMinimo = _precioInicial;
10     obras[obraActual] = ObraDeArte(_titulo, 0, address(0), false);
11     participantesPorObra[obraActual].push(proprietario);
12     momentoFinalSubasta = block.timestamp + duracionSubasta;
13
14 }

```

---

La función 'realizarOferta' se trata de una función payable (se descuentan ethers de los usuarios que la llaman). Recibe como parámetro el ID que identifica a una obra y comprueba que la subasta aún esté activa (modifier) y que la oferta del usuario sea mayor que el precio inicial de la obra puesto por el propietario. tras esto se añade la cantidad pujada por este usuario al mapping ofertas y si se trata de la primera oferta que hace, se le añade al mapping 'participantesPorObra'. Además, se comprueba si este usuario ya ha superado la mayor puja y se convierte en ganador provisional. También se emite el evento 'NuevaOferta'. Nótese que el usuario va pujando añadiendo dinero a lo que ya ha aportado, esto se ha hecho así porque en caso de hacerlo como en las películas (aumentando la puja anterior), se tendría que devolver al usuario la cantidad pujada anteriormente y guardar la nueva puja, lo cual no resulta muy eficiente.

```

1  function realizarOferta(uint _obraID) public payable subastaActiva(_obraID) {
2      require(msg.value > precioMinimo, "La oferta debe superar el precio minimo.");
3      ofertas[msg.sender][_obraID] += msg.value;
4
5      if (ofertas[msg.sender][_obraID] > obras[_obraID].precioGanador) {
6          obras[_obraID].precioGanador = ofertas[msg.sender][_obraID];
7          obras[_obraID].ganador = msg.sender;
8      }
9
10     if (ofertas[msg.sender][_obraID] == msg.value) {
11         participantesPorObra[_obraID].push(msg.sender);
12     }
13
14     emit NuevaOferta(_obraID, msg.sender, msg.value);
15 }

```

---

La función 'finalizarSubasta' recibe como parámetro el ID de la obra en subasta y es una función que sólo puede ser llamada por el propietario. Además, se requiere que el tiempo de subasta haya acabado y que la obra de arte no haya sido vendida ya. Tras esto, cambia el estado de la obra a vendida y el



de la subasta a finalizada y si el ganador de la obra no es la address 0 (nadien pujó), se realiza una transferencia al propietario de los ethers que han resultado ser el precio ganador y se lanza el evento de 'subastaFinalizada'. Por último, se llama a la función 'recuperarFondos' para que el resto de usuarios que no han ganado pero si han pujado recuperen su dinero.

---

```
1  function finalizarSubasta(uint _obraID) public soloPropietario {
2      require(block.timestamp >= momentoFinalSubasta, "La subasta aun esta activa.");
3      require(!obras[_obraID].vendida, "La obra de arte ya ha sido vendida.");
4
5      obras[_obraID].vendida = true;
6      subastaFinalizada = true;
7
8      if (obras[_obraID].ganador != address(0)) {
9          address ganador = obras[_obraID].ganador;
10         uint cantidad = obras[_obraID].precioGanador;
11         payable(proprietario).transfer(cantidad);
12         emit SubastaFinalizada(_obraID, ganador, cantidad);
13     }
14
15     recuperarFondos(_obraID);
16 }
```

---

La función 'recuperarFondos' es privada y no se puede llamar desde fuera. En ella se realiza un for para recorrer los elementos del mapping participantes para esta obra y, para cada participante que no haya resultado ser el ganador, se le transfiere la cantidad que haya pujado.

---

```
1  function recuperarFondos(uint _obraID) private {
2      for (uint i = 0; i < participantesPorObra[_obraID].length; i++) {
3          address participante = participantesPorObra[_obraID][i];
4          if (participante != obras[_obraID].ganador) {
5              uint cantidad = ofertas[participante][_obraID];
6              if (cantidad > 0) {
7                  ofertas[participante][_obraID] = 0;
8                  payable(participante).transfer(cantidad);
9              }
10         }
11     }
12 }
```

---

## 4 Lecciones aprendidas

La principal lección aprendida durante el desarrollo del contrato es el hecho de implementar la subasta de una forma no demasiado convencional pero sí más efectiva. Es decir, en vez de ir aumentando la cantidad pujada, lo que supondría tener que devolver lo pujado al usuario y guardar la nueva puja, se ha optado por ir aumentando la cantidad pujada sobre lo que ya se ha pujado. De esta forma, se devuelve lo pujado a los usuarios perdedores una vez ha terminado la subasta.

También se ha decidido usar tantos mappings para poder almacenar una mayor cantidad de información, lo cual resulta muy útil para una subasta y para poder acceder de manera más sencilla a la información.

## 5 Pruebas

A continuación se muestran una serie de pruebas que se han realizado para comprobar el funcionamiento del contrato.

En primer lugar, el usuario con la address 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 lanza el contrato con una duración de subastas de 120 segundos (2 min) como se ve en la imagen siguiente:

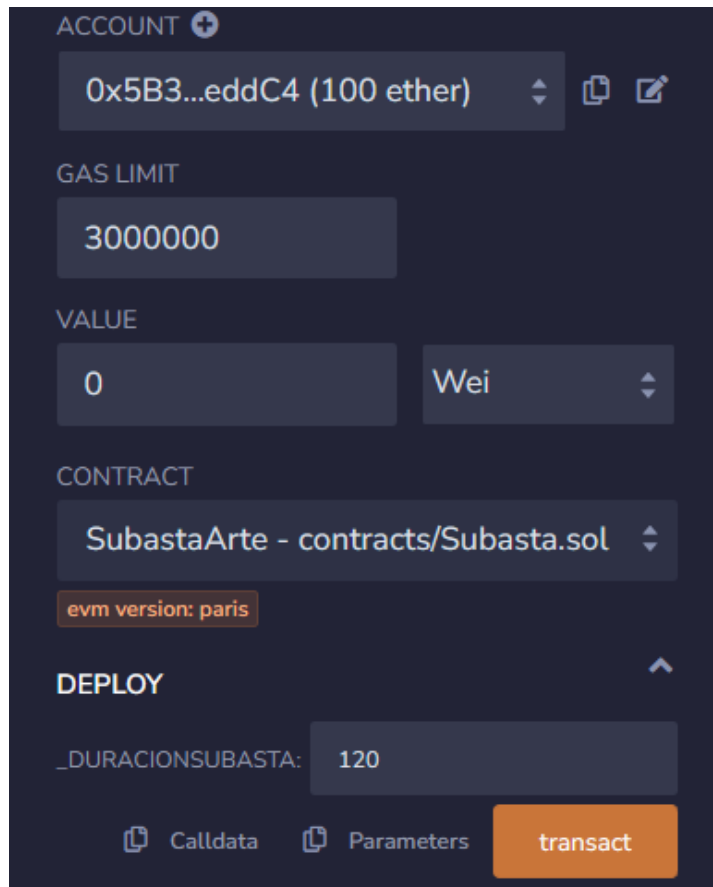


Figure 3: Deploying del contrato

Tras esto, inicia la subasta de la obra de arte 'mona lisa' con un precio de salida de 10 ethers llamando a la función 'iniciarNuevaSubasta'.

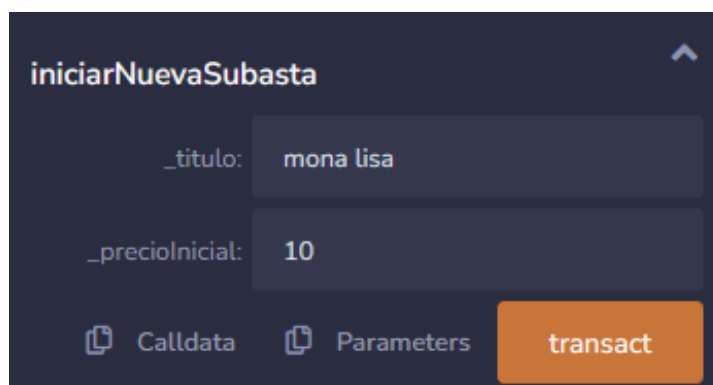


Figure 4: Llamada a función 'iniciarNuevaSubasta'

Ahora, el resto de usuarios pueden empezar a pujar durante los 2 min, para ello, deben llamar a la función 'realizarOferta' y tras finalizar esos dos minutos, ya no se admitirán más subastas y el propietario puede llamar a la función 'finalizarSubasta'. Con la subasta finalizada, los usuarios no ganadores reciben lo que habían pujado y el ganador transfiere sus ethers al propietario. Por eso podemos ver lo siguiente:

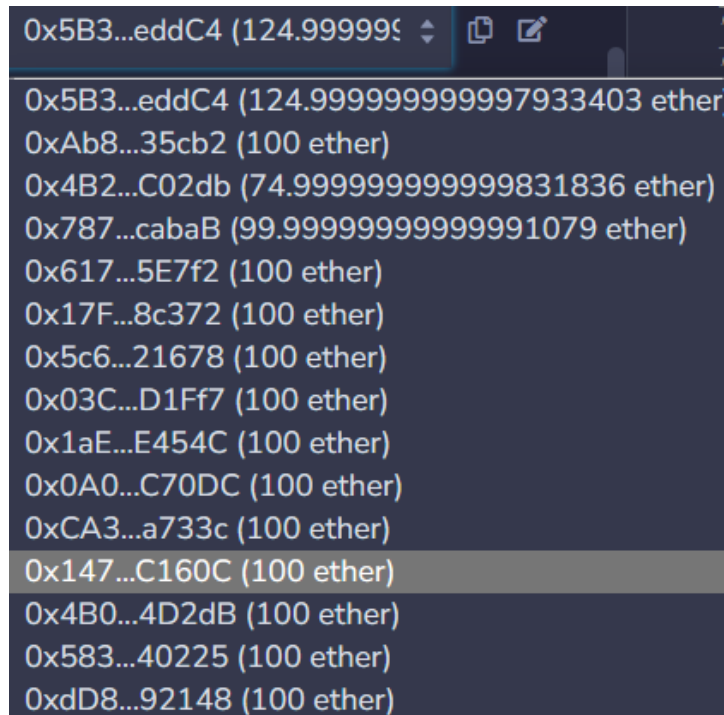


Figure 5: Transacciones de Ethers tras acabar la subasta

Podemos ver que el propietario (0x5B38Da6a701c568545dCfcB03FcB875f56beddC4) ha sumado los ethers de la apuesta ganadora (25 menos el gas de ejecutar las operaciones), el ganador de la apuesta (0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db) ha perdido esos 25 ethers y el otro participante en la puja (0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB), se ha quedado como estaba.

Por último si llamamos a las otras funciones públicas que proporcionan información, podemos ver quiénes han participado en la puja (función obtenerParticipantes) y qué cantidad ha pujado el ganador (función obtenerSaldo).

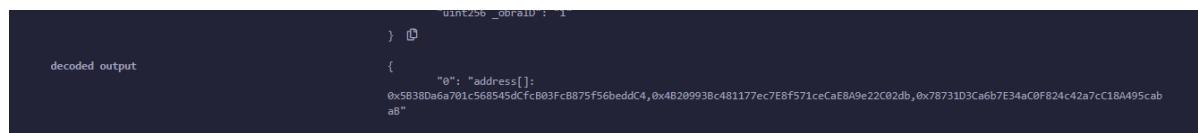


Figure 6: Participantes en la subasta

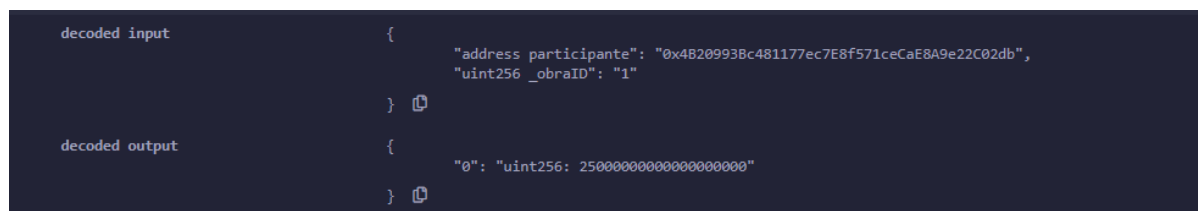


Figure 7: Cantidad pujada por el ganador de la subasta