Julen Lazkano

Liliana Ruiz Milla

Práctica 2: Hundir la flota

DII – MioT – 2022/23

Código desarrollado

Para esta práctica hemos desplegado un contrato de Ethereum escrito en Solidity, que representa el juego de hundir la flota.

Para llevar a cabo la implementación se debe tener en cuenta el juego constará de un propietario encargado de realizar todos los procedimientos de la compra de entradas para jugar y con un máximo de dos participantes.

Por otro lado, se deberán crear dos funciones fundamentalmente del tipo público. La primera función deberá permitir a los jugadores comprar una entrada para el juego y colocar sus barcos en el tablero. El coste que deberá pagar cada jugador será de 10 ethers, y se debe cumplir que ninguno de los participantes sea el propietario del juego. Así mismo, se deberá comprobar si hay espacio en el juego y si las coordenadas del barco son válidas. Si todas las condiciones se cumplen, el jugador quedará registrado y su barco se colocará en el tablero.

Paralelamente, se deberá crear otra función que permitirá a los jugadores realizar ataques a los barcos del oponente. El coste de esta operación será de 1 ether. Esta función deberá comprobar que las coordenadas del disparo y que el turno del jugador que realiza la acción sea correcto. Si el jugador consigue hundir el barco oponente, se le transfiere todo el dinero en el contrato al jugador.

En rasgos generales, el contrato deberá permitir a dos jugadores comprar entradas y colocar sus barcos en un tablero. Para que posteriormente se pueda llevar a cabo la interacción del juego, donde los participantes se turnarán para intentar hundir el barco del oponente. Llegando al final de la partida, cuando uno de los participantes consiga el objetivo y será este quien gana todo el dinero que se ha apostado en el contrato.

Teniendo en cuenta estas consideraciones se ha hecho el siguiente código:

```
pragma solidity 0.8.7;

contract Hundir_la_flota{

    address public ownerJuego;
    address public jugador_1;
    address public jugador_2;
    uint public participantes;
    uint public tablero = 3;    //El tablero sera cuadrado
    uint public turno = 1;  //Siempre empezara el jugador 1
    uint [2] private barco1;
    uint [2] private barco2;
    constructor(){
        ownerJuego = msg.sender;
        participantes = 0;
    }
```

Dentro de este contrato se declaran las variables públicas: **“ownerJuego”**, **“jugador_1”** y **“jugador_2”**. Las cuales almacenan las direcciones Ethereum del creador del juego, el jugador 1 y jugador 2 respectivamente.

También se estable la variable **"participantes"** que lleva la cuenta de cuantos jugadores que ha comprado una entrada. Y la variable **"tablero"** que corresponde a la medida del tablero, en este caso será un tablero cuadrado 3x3. Así mismo, se declara la variable **"turno"** que almacena el número del jugador a quien le toca jugar, siendo inicialmente el jugador 1 quien inicie la partida.

Seguidamente se declara los arreglos privados de dos elementos de tipo entero, **"barco1"** y **"barco2",** que almacenarán las coordenadas de los barcos del **"jugador_1"** y **"jugador_2"** respectivamente.

Posteriormente, se implementa la función ***constructor()*** para inicializar las variables de estado. De esta forma el código se implementa en un blockchain. Aquí se establece la dirección desde donde se originará la llamada de la función mediante la variable global *msg.sender*, el cual pertenece a la dirección del dueño del juego. Así mismo, se inicializa el número de participantes en 0.

```
//Los parametros de entrada van a ser la posicion del barco
function comprarEntrada(uint x, uint y) public payable{
    //Condiciones para participar
    require(msg.sender != ownerJuego, "El dueno del juego no puede participar");
    require(msg.value == 10 ether, "No tienes suficiente dinero como para comprar una entrada");
    require(participantes < 2, "El juego esta lleno, espera hasta que se termine.");
    require(x<=tablero, "No puedes colocar el barco fuera del tablero (3x3)");
    require(y<=tablero, "No puedes colocar el barco fuera del tablero (3x3)");
    require(jugador_1 != msg.sender, "No se puede participar dos veces!");

    if(participantes == 0) {
        jugador_1 = msg.sender;
        barco1[0] = x;
        barco1[1] = y;
    }
    else {
        jugador_2 = msg.sender;
        barco2[0] = x;
        barco2[1] = y;
    }
    participantes++;
}
```

La función "comprarEntrada" es una función pública y pagable que permite a un jugador comprar una entrada al juego y colocar su barco en una posición específica del tablero. La cual tiene dos parámetros de entrada "x" e "y" que representan las coordenadas del barco.

Esta función incluye varios requisitos que deben cumplirse para que un jugador pueda comprar una entrada mediante la función *require()*:

- El primer requisito es que el remitente del mensaje actual (el usuario que está intentando comprar una entrada) no sea el mismo que el propietario del juego.
- El segundo requisito es que el usuario ha enviado 10 ethers como pago para la entrada.
- El tercer requisito es que el número de participantes es menor a 2
- El cuarto es que el valor de x es menor o igual al tamaño del tablero.
- El quinto es que el valor de y es menor o igual al tamaño del tablero.
- La sexta es que el remitente del mensaje actual (el usuario que está intentando comprar una entrada) no sea el mismo que el jugador 1.

Si estas condiciones no se cumplen, se lanzará una excepción con mensajes que informan sobre el problema.

Posteriormente, una vez los requisitos se han cumplido, se establece unas condiciones para que el jugador pueda color su barco en el tablero. Si es el turno del primer jugador que ha comprado una entrada, se le asigna la dirección del remitente a la variable "jugador_1" y los valores x e y como la posición x e y de su barco. Y se incrementa en 1 el número de participantes.

En caso contrario, se le asignará la dirección del remitente y las posiciones de las coordenadas x e y al jugador_2 y su barco.

```
    //Empieza el jugador 1
    function disparar (uint x, uint y) public payable{
        require(participantes == 2, "Para jugar tiene que haber dos participantes.");
        require(msg.value == 1 ether, "Cada intento vale un ether.");
        require(x<=tablero, "Las coordenadas son erroneas (3x3)");
        require(y<=tablero, "Las coordenadas son erroneas (3x3)");
        if(turno == 1) require(msg.sender == jugador_1, "Es turno del jugador 1, espera.");
        if(turno == 2) require(msg.sender == jugador_2, "Es turno del jugador 2, espera.");

        if(turno == 1 && msg.sender == jugador_1) {     //Turno del jugador 1
            if(barco2[0] == x && barco2[1] == y){
                payable(jugador_1).transfer(address(this).balance);
            }
            turno = 2;
        } else if (turno == 2 && msg.sender == jugador_2){
            if (barco1[0] == x && barco1[1] == y){
                payable(jugador_2).transfer(address(this).balance);
            }
            turno = 1;
        }
    }
}
```

Por último, la función "disparar" permite a un jugador atacar una posición específica del tablero. Esta función tiene dos parámetros de entrada "x" e "y" que representan las coordenadas del ataque.

Esta función también incluye varios requisitos que deben cumplirse para que un jugador pueda atacar al barco del oponente mediante la función *require()*:

- El primer requisito es que haya dos participantes en el juego.
- El segundo requisito es que el mensajero (quien llama a la función) esté haciendo el pago de 1 ether para realizar la acción.
- El tercer requisito es que las coordenadas x e y se encuentren dentro del tablero.
- El cuarto requisito es que sea el turno del jugador que llama la función.

Si estas condiciones no se cumplen, se lanzará una excepción con mensajes que informan sobre el problema.

Una vez los requisitos se han cumplido, se establece unas condiciones para que el jugador atacar al barco oponente. Si es el turno del jugador 1 y llama la función, se verifica si las coordenadas de ataque corresponden al barco del jugador 2. Si es así, se transfiere el saldo del contrato al jugador 1 con *"payable().transfer(address(this).balance)".*

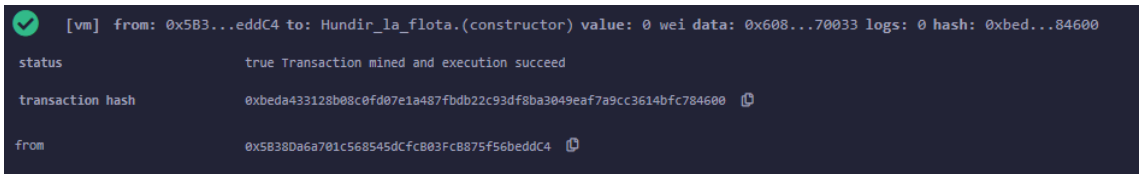
Por otro lado, si es el turno del jugador 2 y llama la función, se verifica si las coordenadas de ataque corresponden al barco del jugador 1. Si es así, se transfiere el saldo del contrato al jugador 2 con *"payable(jugador_2).transfer(address(this).balance)".*

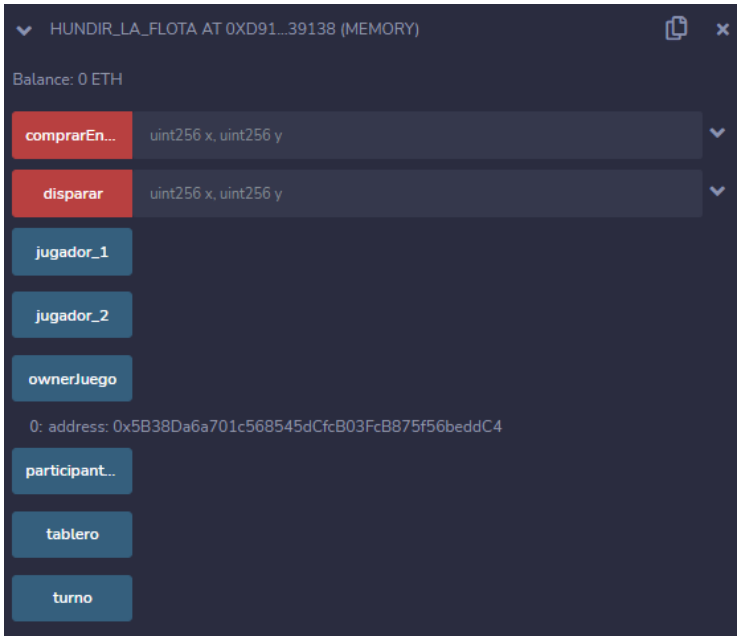
Finalmente, se cambia el turno a la otra persona con "turno = 2;" o "turno = 1;" dependiendo de quien realizó el disparo.

Pruebas realizadas

Existen varias pruebas que se pueden realizar para verificar que el código del contrato "Hundir_la_flota" funciona correctamente, entre ellas hemos realizado las siguientes:

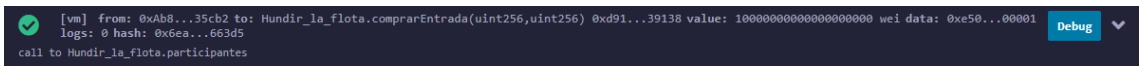
1. Prueba de despliegue.

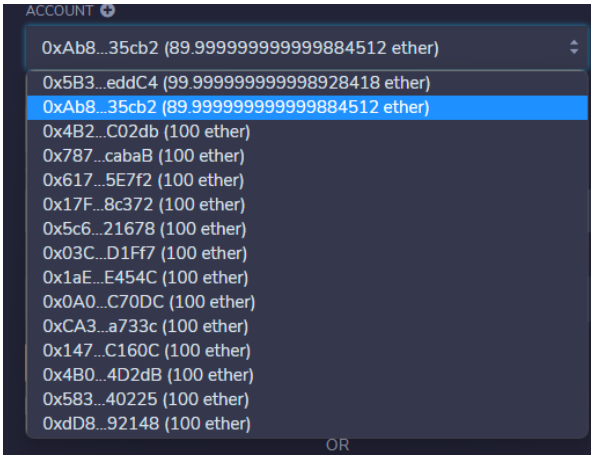




El contrato se despliega en la red de pruebas de Ethereum y aquí podemos ver como el contrato se crea correctamente, así mismo vemos que la cuenta del propietario del juego coincide con la cuenta que hace el despliegue. Siendo este el 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4.

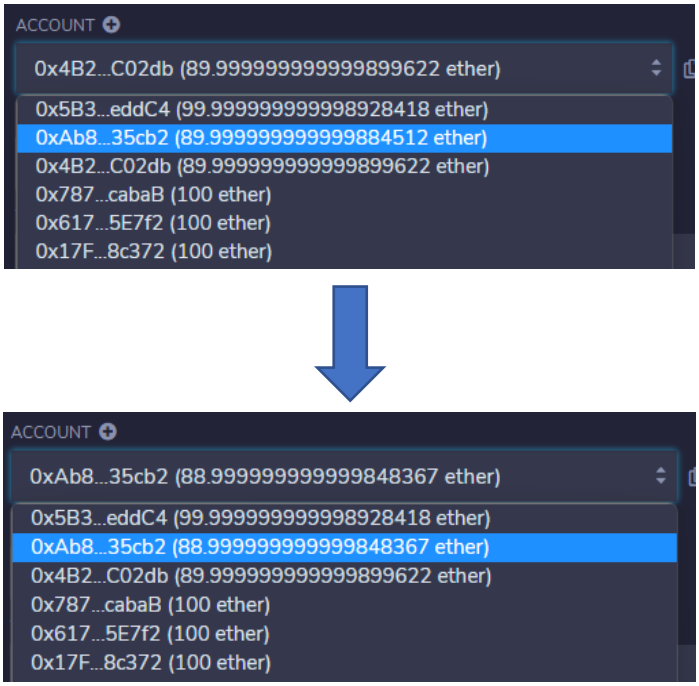
2. Prueba de compra de entrada.



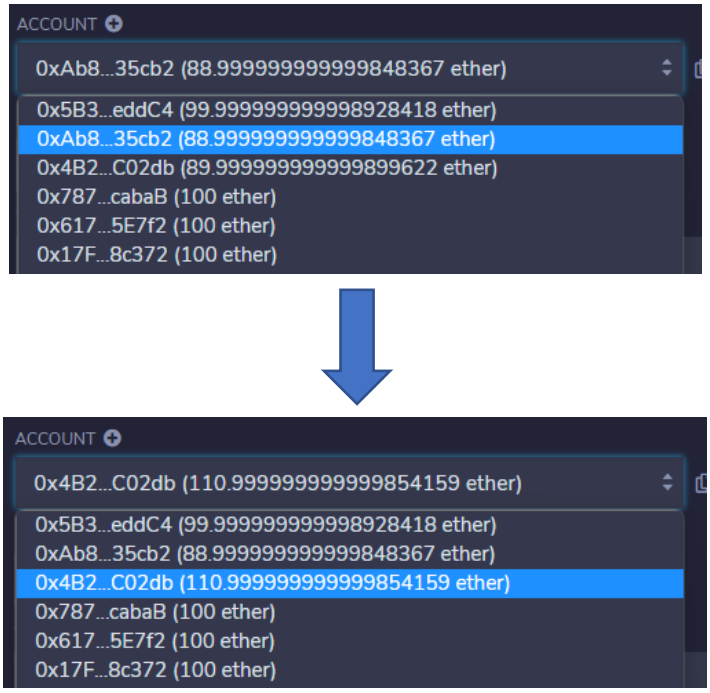


Al utilizar "comprarEntrada" para comprar una entrada al juego y colocar un barco en el tablero podemos ver como el saldo de la cuenta que compró la entrada se ha reducido en la cantidad correcta (10 ethers). Así mismo, vemos que la variable "participantes" del desplegable del contrato Hundir_la_flota aumenta en 1.

3. Prueba de ataque.

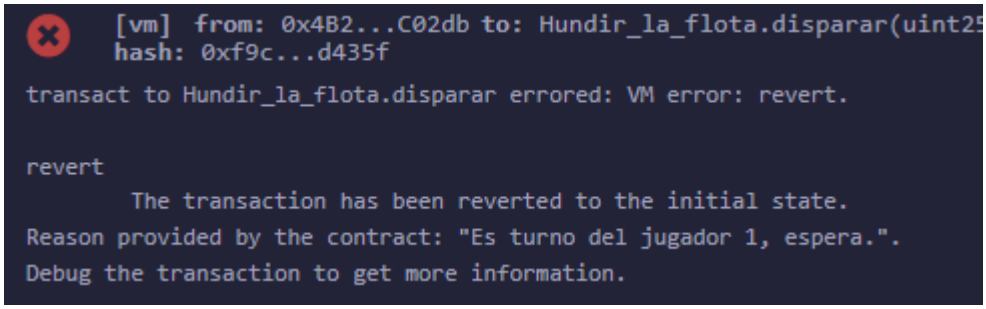


Al utilizar la función "disparar" para atacar la posición del barco del oponente. Vemos como el saldo de la cuenta que realizó el ataque se ha reducido en la cantidad correcta.

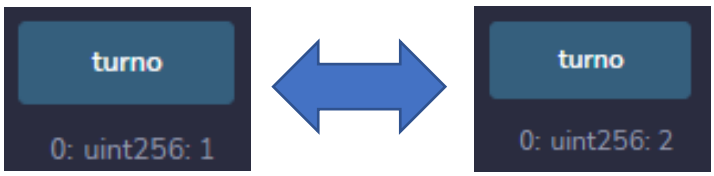


Por otro lado, si el jugador acierta y hunde el barco del oponente, vemos como el saldo del contrato se ha transferido a la cuenta que acertó el ataque.

4. Prueba de turnos.

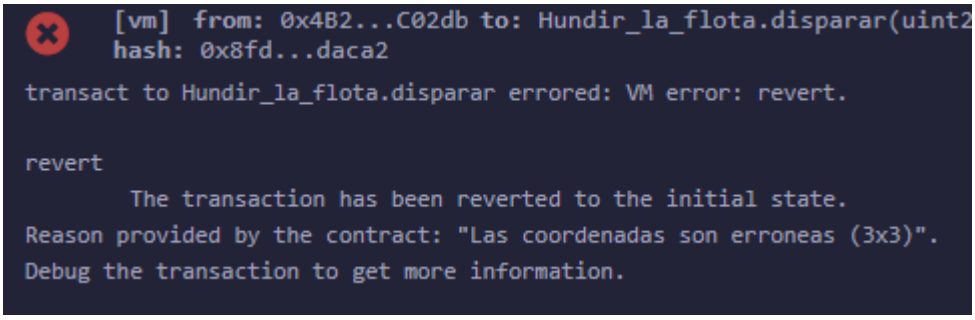


Cuando uno de los jugadores realiza un disparo en el turno equivocado podemos ver como se muestra el mensaje con la información de que es el turno del oponente y que tiene que esperar.



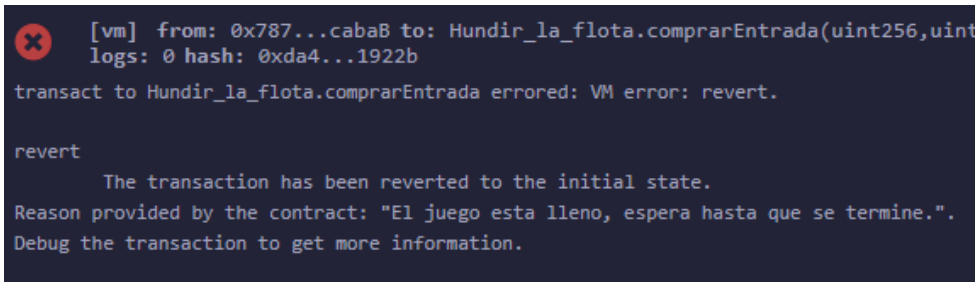
Así mismo en la variable “turno” podemos ver cómo va cambiando correctamente después de cada ataque.

5. Prueba de límites.



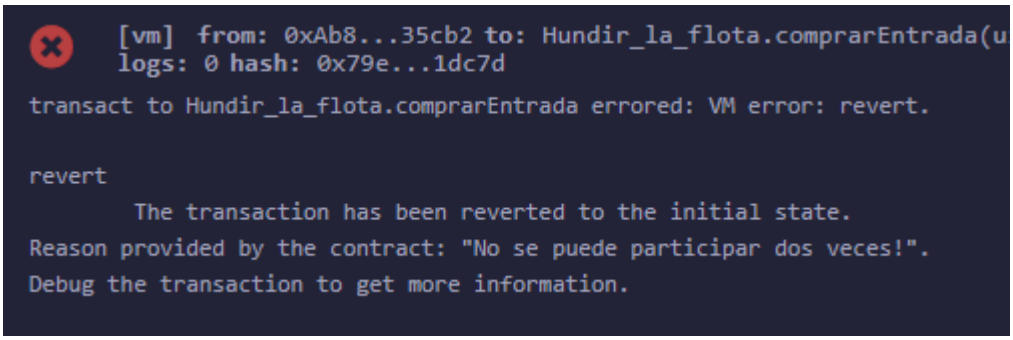
En el caso que se escojan coordenadas fuera de rango del tablero, podemos ver como se muestra un mensaje de alerta indicando que las coordenadas son erróneas y que el tablero es de 3x3.

6. Prueba de juego lleno.



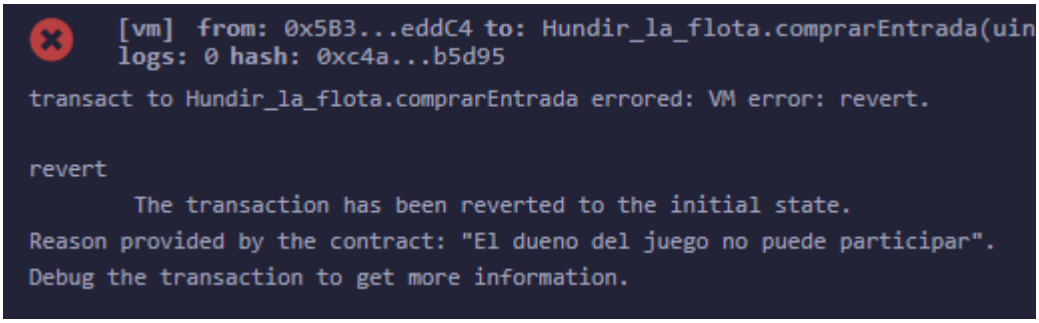
En el caso que un tercer usuario quiera unirse a la partida, cuando ya hay dos jugadores, podemos ver como se lanza un mensaje de alerta que la partida está completa.

7. Prueba de jugador duplicado.



En el caso que un mismo jugador quiera comprar otra entrada para la misma partida, se lanzará un aviso para informar que solo se permite un participante por cuenta.

8. Prueba de propietario no puede jugar.

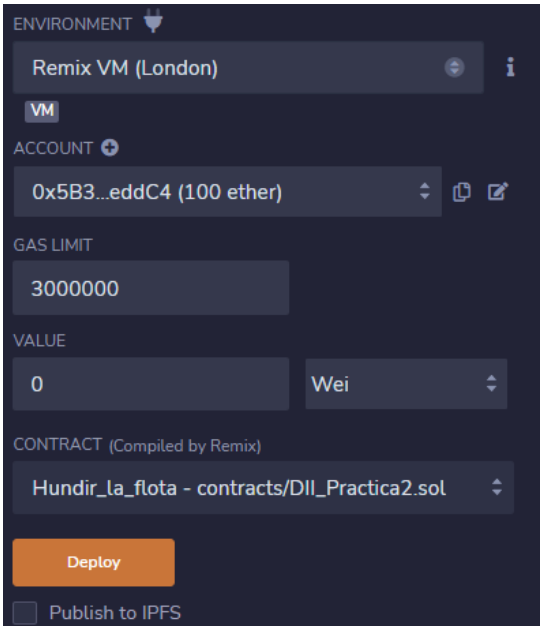


En el caso que el propietario quiera participar en el juego, podemos ver como se muestra un mensaje de que no es posible.

Guía de usuario

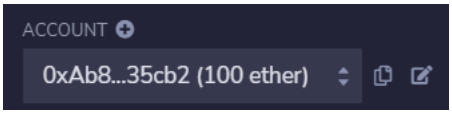
Respecto al **dueño del sorteo,** este se encargará de abrir la partida para que los usuarios puedan comprar una entrada para participar en el juego.

Para ello primero tiene que compilar el script del contrato del juego “Hundir_la_flota” para posteriormente realizar el deploy de dicho script. De esta manera el juego queda abierto para que se puedan empezar a comprar las entradas.



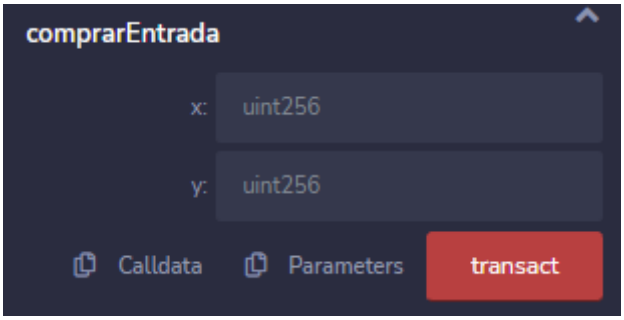
Una vez realizado la apertura de la partida, el propietario del juego no tiene que realizar ningún paso más hasta que haya un ganador. Cuando la partida ha finalizado porque ha habido un ganador, el propietario del juego tiene que cerrar la partida eliminando el despliegue y abriendo un nuevo desplegable dándole al Deploy nuevamiente.

Por otro lado, respecto a los **participantes del juego**, su participación empieza una vez el dueño a iniciado el contrato del juego. Para poder participar en el sorteo, el usuario tiene que seleccionar su cuenta en el apartado ACCOUNT del entorno.

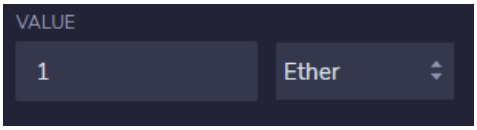


Una vez la cuenta esta seleccionada tiene que poner el valor del precio de una entrada para el juego en el apartado de VALUE del entorno, el cual corresponde a 10 Ether.

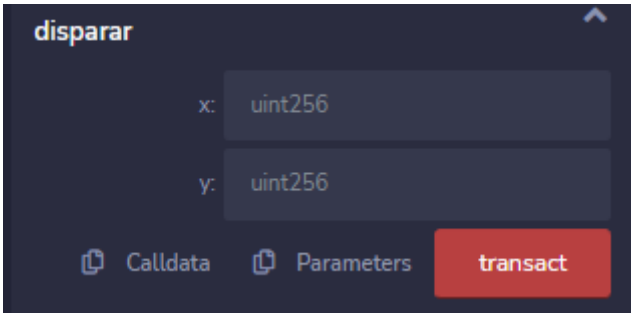
Para hacerse con una entrada para el juego hace falta que en el desplegable del contrato del juego hundir la flota se elija la posición x e y de su barco en el apartado comprarEntrada. Y posteriormente seleccionar transact para que se haga efectivo la acción. Este procedimiento solo se podrá hacer una vez por jugador.



Posteriormente, cuando sea su turno de ataque, tiene que poner el valor del precio de realizar una acción de disparo en el apartado de VALUE del entorno, el cual corresponde a 1 Ether.



Luego, en el aparato disparar se tiene que poner la posición x e y de donde se crea que puede estar en el tablero el barco de su oponente. Y una vez seleccionado transact se comprobará si ha acertado.



El procedimiento que se hace para realizar un ataque, llamando a la función disparar, se tiene que repetir hasta que uno de los participantes acierte la posición del barco oponente en el tablero. Una vez esto suceda, el dinero que se ha apostado en el contrato se ingresará en la cuenta del ganador.

Guía de usuario

Lo último que hemos hecho en el proyecto ha sido incluir el código en un repositorio de github y ponerle una licencia MIT.

Referencias

1. https://solidity-es.readthedocs.io/es/latest/introduction-to-smart-contracts.html
2. https://vsupalov.com/ethers-call-payable-solidity-function/