Implementação do LUPA - Lowest-Unmatched Price Auctions

11/10/2024

Joao Otavio Rodrigues de Castro Manieri – 12021BSI263

Ana Roling Silvério – 12021BSI248

1. **Enum LUPAStates**: Define os três estados possíveis do leilão: Bid (lances abertos), Payment (em fase de pagamento) e Finished (leilão concluído).
2. **Struct BidValue**: Armazena o valor do lance, se é único (não repetido por outro licitante), e uma lista de endereços dos licitantes que fizeram lances com esse valor.
3. **Mapping bids**: Relaciona um valor de lance (uint) com sua estrutura correspondente (BidValue), permitindo o armazenamento e verificação dos lances feitos.
4. **Construtor**: Inicializa o contrato, atribui o dono, define o valor do prêmio e estabelece o tempo de leilão (em blocos).
5. **Função bid**: Permite que os usuários façam lances em Ether, verificando se o lance é único ou não e armazenando o licitante na lista de endereços.
6. **Função endBidding**: Permite que o dono do contrato encerre o leilão e distribua o prêmio ao menor lance único.
7. **Função distributePrize**: Transfere o valor do prêmio para o vencedor.
8. **Função withdrawFunds**: Após o término do leilão, permite que o dono do contrato retire os fundos restantes.

Fluxo do Contrato:

1. O dono "starta" o contrato e especifica o tempo de leilão.
2. Usuários fazem lances durante o período de leilão.
3. O dono encerra o leilão após o tempo determinado e o menor lance único recebe o prêmio.
4. O dono pode retirar o saldo remanescente após o leilão.

Código compilado no remix:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract LUPA {

    // Enum para definir os estados do leilão
    enum LUPAStates { Bid, Payment, Finished }

    // Estrutura para armazenar o valor do lance e os licitantes
```

```
    struct BidValue {
        uint value;  // O valor do lance
        bool isUnmatched;  // Se o lance é único (não foi duplicado por outro
licitante)
        address payable[] bidders;  // Lista de endereços que fizeram lances
com este valor
    }

    // Mapeamento de valores de lances para a estrutura BidValue
    mapping (uint => BidValue) bids;

    // Número de bloco após o qual o período de lances será encerrado
    uint blocklimit;

    // Estado atual do leilão (Bid, Payment ou Finished)
    LUPAStates public myState;

    // Valor total do prêmio (em Ether) que será concedido ao vencedor
    uint public prizeValue;

    // Dono do contrato (leiloeiro), que pode encerrar o leilão e retirar
fundos
    address payable public owner;

    // Modificador para garantir que apenas o dono do contrato possa executar
certas funções
    modifier onlyOwner() {
        require(msg.sender == owner, "Sorry, only the owner can perform this
action!");
        _;
    }

    // Modificador para verificar se o contrato está em um estado específico
antes de executar uma função
    modifier inState(LUPAStates _state) {
        require(myState == _state, "Invalid state for this action.");
        _;
    }

    // Construtor do contrato, define o tempo de leilão (em blocos) e o valor
do prêmio
    constructor(uint time) payable {
        owner = payable(msg.sender);  // Define o dono do contrato como o
endereço que o implantou
        prizeValue = msg.value;  // O valor enviado na criação do contrato é o
prêmio
```

```
        blocklimit = block.number + time;  // Define o número de blocos até o
fim do leilão
        myState = LUPAStates.Bid;  // O leilão começa no estado "Bid"
    }

    // Função para fazer lances (enviar Ether) durante o estado de "Bid"
    function bid() public payable inState(LUPAStates.Bid) {
        require(msg.value > 0, "Bid value must be greater than 0.");  //
Certifica que o lance é maior que 0

        // Armazena os lances no mapeamento baseado no valor enviado
        BidValue storage bidEntry = bids[msg.value];

        if (bidEntry.bidders.length == 0) {
            // Se é o primeiro lance com este valor, define como único
            bidEntry.value = msg.value;
            bidEntry.isUnmatched = true;
        } else {
            // Se outro lance já existe com o mesmo valor, marca-o como não
único
            bidEntry.isUnmatched = false;
        }

        // Adiciona o licitante à lista de licitantes com este valor de lance
        bidEntry.bidders.push(payable(msg.sender));
    }

    // Função para encerrar o período de lances (chamada pelo dono do
contrato)
    function endBidding() public onlyOwner inState(LUPAStates.Bid) {
        require(block.number >= blocklimit, "Bidding period has not ended
yet.");  // Verifica se o limite de blocos foi atingido

        myState = LUPAStates.Payment;  // Muda o estado para "Payment"
(pagamento)

        // Itera pelos valores de lance para encontrar o menor lance único
        for (uint i = 0; i <= address(this).balance; i++) {
            if (bids[i].isUnmatched) {
                // Se encontrar um lance único, paga o prêmio ao licitante
                distributePrize(bids[i].bidders[0]);
                return;
            }
        }
    }
```

```
    // Função interna para transferir o prêmio ao vencedor
    function distributePrize(address payable winner) internal {
        winner.transfer(prizeValue);  // Transfere o prêmio ao vencedor
        myState = LUPAStates.Finished;  // Muda o estado para "Finished"
(finalizado)
    }

    // Função para o dono retirar os fundos restantes após o leilão terminar
    function withdrawFunds() public onlyOwner inState(LUPAStates.Finished) {
        owner.transfer(address(this).balance);  // Transfere o saldo restante
para o dono
    }
}
```