pragma solidity ^ 0.5.1;

import "./Auction.sol";

/// @title Vickrey Auction contract

/// @author Andrea Bruno 585457

/// @notice This contract implements the Vickrey auction functionalities.

/// @dev The following comments are written using the Solidity NatSpec Format.

contract VickreyAuction is Auction {

/// @dev Those are the phases of the auction.

enum Phase {

GracePeriod,

Commitment,

Withdrawal,

Opening,

Finished

}

Phase public phase;

/// @dev This variable contains the block number of the current phase.

uint startPhaseBlock;

/// @dev The reserve price decided by the seller

uint reservePrice;

/// @dev The minumum deposit decided by the seller

uint min\_deposit;

/// @dev The commitment phase lenght (number of blocks)

uint commitment\_len;

/// @dev The withdrawal phase lenght (number of blocks)

uint withdrawal\_len;

/// @dev The opening phase lenght (number of blocks)

uint opening\_len;

/// @dev The address of the highest bidder (see `open` function)

address payable highestBidder;

/// @dev The value of the highest bid

uint highestBid;

/// @dev The value of the second highest bid

uint secondHighestBid;

/// @dev This boolean is to understand if during the opening is the first bid opened

bool firstOpen = true;

///@dev The structure of a Bid.

struct Bid {

// An integer value of the bid

uint value;

// A nonce generated by the bidder

bytes32 nonce;

// The hash of the tuple <value,nonce>

bytes32 hash;

// The deposit given during the bid

uint deposit;

}

/// @dev This mapping buond the address of the bidder to his bid

mapping(address => Bid) bids;

/// @notice This event communicate the beginning of the Withdrawal phase.

event withdrawalStarted();

/// @notice This event communicate the beginning of the Opening phase.

event openingStarted();

/// @notice This event communicate that a bidder has been refunded by half of his deposit.

/// @notice The other half is sent to the seller of the item.

/// @param bidder is the address of the bidder

/// @param value is the value sent to the bidder

/// @param seller is address of the seller

/// @param val is the value sent to the seller

event withdrawalExecuted(address bidder, uint value, address seller, uint val);

/// @notice The constructor of the Dutch auction.

/// @dev The `msg.sender` will be the seller.

/// @param \_itemName is a short description of what is going to be sold.

/// @param \_reservePrice is the reserve price decided by the seller.

/// @param \_min\_deposit is the minimum deposit decided by the seller.

/// @param \_commitment\_len the lenght of the commitment phase.

/// @param \_withdrawal\_len the lenght of the withdrawal phase.

/// @param \_opening\_len the lenght of the opening phase.

constructor(

string memory \_itemName,

uint \_reservePrice,

uint \_min\_deposit,

uint \_commitment\_len,

uint \_withdrawal\_len,

uint \_opening\_len

) public {

// Before deplying the contract a few controls are needed.

require(\_reservePrice > 0, "Reserve price should be greater than zero.");

require(\_min\_deposit >= \_reservePrice, "The deposit should be greater than the reserve price");

require(\_commitment\_len > 0, "The lenght of commitment should be greater than zero.");

require(\_withdrawal\_len > 0, "The lenght of withdrawal should be greater than zero.");

require(\_opening\_len > 0, "The lenght of opening should be greater than zero.");

// Set into the description both the seller

description.seller = msg.sender;

// and the name of the item

description.itemName = \_itemName;

// Set state into "grace period"

phase = Phase.GracePeriod;

reservePrice = \_reservePrice;

min\_deposit = \_min\_deposit;

commitment\_len = \_commitment\_len;

withdrawal\_len = \_withdrawal\_len;

opening\_len = \_opening\_len;

// Update the block number of the current phase.

startPhaseBlock = block.number;

}

/// @dev This modifier allow to invoke the function olny during the Commitment phase.

modifier duringCommitment {

require(phase == Phase.Commitment, "Commitment phase not started yet");

require((block.number - startPhaseBlock) <= commitment\_len, "Commitment phase is ended");

\_;

}

/// @dev This modifier allow to invoke the function olny during the Withdrawal phase.

modifier duringWithdrawal {

require(phase == Phase.Withdrawal, "Withdrawal phase not started yet");

require((block.number - startPhaseBlock) <= withdrawal\_len, "Withdrawal phase is ended");

\_;

}

/// @dev This modifier allow to invoke the function olny during the Opening phase.

modifier duringOpening {

require(phase == Phase.Opening, "Opening phase not started yet");

require((block.number - startPhaseBlock) <= opening\_len, "Opening phase is ended");

\_;

}

/// @notice This function will activate the auction.

/// @dev Only the seller can invoke it.

function activateAuction() public onlySeller {

// In order to activate the auction we need to be in the "grace period"

require(phase == Phase.GracePeriod, "To activate the contract you must be in the Grace Period");

// and also 5 minutes (20 blocks) must be elapsed.

require(block.number - startPhaseBlock > 20, "Grace period is not finished yet");

// Set the current phase to "Commitment".

phase = Phase.Commitment;

// Update the start block into the descritpion

description.startBlock = block.number;

// and also the start phase block

startPhaseBlock = block.number;

// Communicate that the auction is started.

emit auctionStarted();

}

///@notice This function allow people to make bid.

///@notice Note that a bid will be taken into account if the value sent is >= the minimum deposit.

///@dev This function can be invoked only during the commitment phase.

///@param \_bidHash this is the hash of the tuple <value,nonce>. See `GenerateBid` contract for more info.

function bid(bytes32 \_bidHash) public duringCommitment payable {

// The bidder should send an appropriate value.

require(msg.value >= min\_deposit, "The value sent is not sufficient");

// The bidder should not be able to send another bid.

require(bids[msg.sender].value == 0, "You have already submitted a bid");

// Create a temporary bid

Bid memory tmp\_bid;

tmp\_bid.hash = \_bidHash;

tmp\_bid.deposit = msg.value;

// Save this bid into the mapping called `bids`

bids[msg.sender] = tmp\_bid;

}

///@notice This function start the withdrawal phase.

///@dev This function can be invoked only by the seller.

function startWithdrawal() public onlySeller {

require(phase == Phase.Commitment, "You can't start withdrawal before commitment");

require((block.number - startPhaseBlock) > commitment\_len, "Commitment period is not finished yet");

phase = Phase.Withdrawal;

startPhaseBlock = block.number;

// Communicate that the Withdrawal phase is started.

emit withdrawalStarted();

}

///@notice This function allow people to retire their bid.

///@notice Note that only half of the deposit will be refunded.

///@notice The other half goes to the seller

///@dev The security pattern Checks-Effects-Interaction is respected.

function withdrawal() public duringWithdrawal {

//1. Checks

require(bids[msg.sender].deposit > 0, "You don't have any deposit to withdraw");

uint bidderRefund = bids[msg.sender].deposit / 2;

uint sellerRefund = bids[msg.sender].deposit - bidderRefund;

//2. Effects

bids[msg.sender].deposit = 0;

emit withdrawalExecuted(msg.sender, bidderRefund, description.seller, sellerRefund);

//3. Interaction

description.seller.transfer(sellerRefund);

msg.sender.transfer(bidderRefund);

}

///@notice This function activate the Opening phase

///@dev This function can be invoked only by the seller.

function startOpening() public onlySeller {

require(phase == Phase.Withdrawal, "You can't start opening before withdrawal");

require((block.number - startPhaseBlock) > withdrawal\_len, "Commitment period is not finished yet");

phase = Phase.Opening;

startPhaseBlock = block.number;

// Communicate that the Opening phase is started.

emit openingStarted();

}

///@notice This function allow people to open their bid.

///@param \_nonce is the nonce of the bid previously generated.

function open(bytes32 \_nonce) public duringOpening payable {

// Control the correctness of the bid

require(keccak256(abi.encodePacked(msg.value, \_nonce)) == bids[msg.sender].hash, "Wrong hash");

// If the `msg.sender` has been honest, then refund him with his deposit

uint deposit = bids[msg.sender].deposit;

bids[msg.sender].deposit = 0;

msg.sender.transfer(deposit);

// Update the bid status

bids[msg.sender].value = msg.value;

bids[msg.sender].nonce = \_nonce;

// If it is the first opening

if (firstOpen) {

// Update the highest bidder

highestBidder = msg.sender;

// and the bid

highestBid = msg.value;

//if there is only one bid, the winner have to pay at least the reservePrice

secondHighestBid = reservePrice;

firstOpen = false;

// If it is NOT the first opening

} else {

// If the msg.value is more than the highest bid

if (msg.value > highestBid) {

// The highest bid becomes the second highest bid

secondHighestBid = highestBid;

// Now we need to refund the bidder of the (old) highest bid

refund(highestBidder, highestBid);

// Set the new highest bidder and its own bid

highestBidder = msg.sender;

highestBid = msg.value;

// If the msg.value is NOT more than the highest bid

} else {

// Check whether the msg.value is higher than the second highest bid

if (msg.value > secondHighestBid) secondHighestBid = msg.value;

// Since the current opening is not the highest we can refund the bidder

refund(msg.sender, msg.value);

}

}

}

///@notice This function allows to refund an address

///@param \_dest is the destination address

///@param value is the value of this refund.

///@dev The function is declare internal, so no one externally can invoke it.

function refund(address payable \_dest, uint value) internal {

\_dest.transfer(value);

}

///@notice This function finalize and close the contract.

///@notice In case of surplus, this ether will be sent to a charity address.

///@dev The function is declare internal, so no one externally can invoke it.

function finalize() public onlySeller {

require(phase == Phase.Opening, "You can't finalize the contract before opening");

require((block.number - startPhaseBlock) > opening\_len, "Opening period is not finished yet");

//If there is a winner (at least one bid)

if (highestBidder != address(0)) {

description.winnerAddress = highestBidder;

description.winnerBid = secondHighestBid;

//Refund the winner

highestBidder.transfer(highestBid - secondHighestBid);

//Send ehter to the seller of the item

description.seller.transfer(description.winnerBid);

}

// A random charity address

address payable charity = 0x64DB1B94A0304E4c27De2E758B2f962d09dFE503;

uint surplus = address(this).balance;

// Set the current phase to Finished

phase = Phase.Finished;

// Communicate the end of the auction

emit auctionFinished(description.winnerAddress, description.winnerBid, surplus);

// Send the surplus to the charity address.

charity.transfer(surplus);

}

}