Peer to Peer Systems and Blockchains
final-term assignment

Giacomo De Liberali

Smart Auctions: Dutch and Vickery Auctions
on the Ethereum blockchain

June 15, 2019

Contents

1 Introduction

This project involves the implementation of two types of auction systems using Ethereum blockchain. One is Vickery and the other, to be chosen between Dutch and English, is the first one. The Dutch auction consists in a auction where the good price is initially set to an high value than gradually decreased until a given threshold. The first bidder who sends a sufficient amount will be the winner of the auction. The Vickery instead is a blind auction where bidders initially submits a commitment of payment, and only after the auction end the bidder who submitted the highest valid commitment will be the winner and will pay the second price.

2 Auctions

2.1 Dutch auction

This contract requires several parameters in the constructor, but the most interesting is the `IDecrease _strategy` which is a reference object that refers to another contract. Since it is required to support different price descending strategies I adopted the strategy pattern, which fits well this scenario. In order to deploy this contract we need so to first deploy a contract that implement the interface *IDecrease* and then deploy the *DutchAuction* passing as value the address where the *IDecrease* contract instance is actually deployed.

```
constructor(uint _reservePrice,
            uint _initialPrice,
            uint _lastForBlocks,
            IDecrease _strategy) public {
    // ...
}
```

The *IDecrease* interface has only a single method that returns the price:

```
function getPrice(uint initialAmount,
                  uint reservePrice,
                  uint startBlockNumber,
                  uint endBlockNumber,
                  uint currentBlockNumber) external pure returns(uint);
```

This contract has only four public methods and one event:

- `function getCurrentPrice() public view returns(uint)`: as the name suggests, it returns the current price of the auction calculated by querying the strategy object described above.
- `function makeBid() payable external`: allows to a bidder to submit a payment to win the auction. If the bid if not valid, it gets immediately refund.
- `function isClosed() public view returns(bool)`: returns a flag indicating if this auction is still open or it has been closed (either by a winner bidder or by the elapsed number of blocks).
- `function terminate() external`: can be called only by the owner of the contract (who deployed it) to terminate the auction before any bidder submits a valid bid.
- `event HasBidderEvent(address, uint, uint)`: it is fired after a valid *makeBid()* that elects a winner. It contains three parameters, the address of the winner, the current price of the good and the amount of ether effectively sent by the winner.

To properly deploy this contract we need an already deployed contract that implements the *IDecreased* interface. I chose to implement a *LinearDecreaseStrategy* which decreases the price of the auction down to the threshold in a linear way based on the elapsed blocks from its deployment. The gas cost of the *DutchAuction* deployment is reported in the below figure.

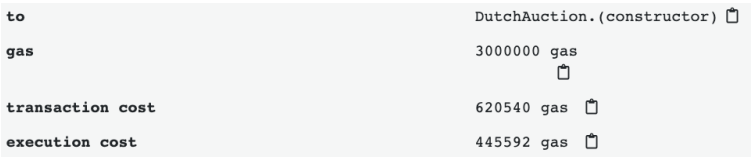

Figure 1: Gas cost of *DutchAuction* deployment

A bidder who wants to participate to this auction must first of all check that it is still open by calling the *isClosed()* method, then obtain the current price by querying the *getCurrentPrice()* and finally submit a payment with the *makeBid()*. The total gas cost for this three operations is:

- *isClosed()*: zero gas if called by an externally owned account, 22848 gas if called by a contract.
- *getCurrentPrice()*: zero gas if called by an externally owned account, 25068 gas otherwise.
- *makeBid()*: 56322 gas, since it modifies the state

The total amount of gas used by an externally owned account that participates and wins this auction is 25068 gas, while if the participant is another contract the total cost will be of 104238 gas. If the owner wants to end the auction it can all the *terminate()* function that will cost 26989 gas.

2.1.1 Testing

Since the verification of correctness is fundamental in this environment I wrote some unit and interaction tests using *Truffle* framework. It allows to test in a local blockchain solidity contracts while tests are written in javascript, leveraging *Mocha* as testing framework. An minmal example of a unit test is the following:

```
const DutchAuction = artifacts.require("DutchAuction");
const LinearStrategy = artifacts.require("LinearStrategy");
contract("DutchAuction", accounts => {
  let linearStrategy;
  let dutchAuction;
  beforeEach(async () => {
    // reinitialize contracts inside each test
    linearStrategy = await LinearStrategy.new();
    dutchAuction = await DutchAuction.new(100, 1000, 5, linearStrategy.address);
  });
  it("Should be deployed correctly", async () => {
```

```
    assert.equal(
      await dutchAuction.isClosed(),
      false,
      "Should be opened"
    );
  });
  // other tests...
});
```

Exploiting *Truffle*'s testing environment the testing speed increases with respect to the manual functional testing that can be done in *Remix IDE*.

```
giacomodeliberali@Giacomos-MBP ~/Code/personal/unipi/smart-auctions (master) $ truffle test test/dutch.js
Using network 'development'.


Compiling your contracts...
===========================
> Compiling ./contracts/dutch/DutchAuction.sol
> Compiling ./contracts/dutch/IDecrease.sol
> Compiling ./contracts/dutch/LinearStrategy.sol
> Compiling ./contracts/vickery/HashGenerator.sol
> Compiling ./contracts/vickery/VickeryAuction.sol
> Artifacts written to /var/folders/zg/y687rmh16zq91j_jhm8g3yjh0000gn/T/test-119515-2448-phttk8.knei
> Compiled successfully using:
   - solc: 0.5.8+commit.23d335f2.Emscripten.clang



  Contract: DutchAuction
    ✓ Should be deployed correctly
    ✓ Should not be terminated by someone who is not the owner (76ms)
    ✓ Should be terminated by the owner (47ms)
    ✓ Should reject invalid bids (82ms)
    ✓ Should accept valid bids (93ms)
    ✓ Should accept first valid bid (123ms)


  6 passing (1s)
```

Figure 2: *DutchAuction* unit test results

2.2 Vickery

In this auction all bids are blind and the winner must be chosen only after all bids have been received and processed. The auction can be in different states and in each one it behaves differently. All the states a represented by the following enum:

```
enum PhaseType {
    Commitment, // Can only accept commitment of blind bids
    Withdrawal, // Can withdrawal half deposit
    Opening,    // Can only accept bid opening requests
    Closed      // The auction has ended
}
```

All these states have a defined duration expressed in terms on number of elapsed block from deployment. Since each methods in the contract must be available only in determined states, I choose to create a modifier that updates the state of the contract before the actual method invocation:

```
// Ensure that the state property is updated up to the current block.number
modifier ensureFreshState() {
    require(!isFinalized, "This auction has already been finalized.");
    PhaseType currentState = // calculate current state based on block.number
    if(state != currentState) {
        updateState(currentState);
    }
    _; // decorated method body goes here
}
```

In this way each method can be decorated with this modifier in order to be sure that the state property is properly updated.

2.2.1 Methods

Public methods of this contracts are:

- `function makeBid(bytes32 hash) external ensureFreshState payable`: allows to submit a commitment of payment with an hash representing the concatenation between the bid amount and a nonce. This transaction must also meet the min deposit requirement.
- `function withdrawal() external ensureFreshState payable`: after the commitment phase any bidder can request a withdrawal in order to receive back half of the deposit they submitted in the commitment phase.
- `function openBid(uint32 nonce) external ensureFreshState payable`: after the withdrawal phase each bidder that has not withdrawn can submit its nonce and the amount. If the calculated hash matches the first submitted, the bid is considered valid., otherwise the deposit is lost.
- `function finalize() external ensureFreshState`: called by the owner of the auction after the opening phase, it calculates the winner and the second price. All loosing bidders are refund of their bid and deposit, and the winner will be charged of the second price.
- `function generateHash(uint amount, uint32 nonce) public pure returns(bytes32)`: it's a debug function with which a user can calculate easily the hash that it has to submit in the commitment phase. Obviously this function can't appear on the real contract since the input values are public.

2.2.2 Events

Each state change in this contract triggers an event:

- `event StateUpdatedEvent(PhaseType indexed state)`: fired each time the state of the contract changes from value to another.

- `event WithdrawalEvent(address, uint)`: fired after a bidder requested and executed a withdrawn.
- `event BidEvent(address)`: fired after a valid bid commitment.
- `event OpenBidEvent(address, uint, uint)`: fired after a valid bid opening, so whenever a bidder correctly submits an amount and a nonce that matches the initial submitted hash.
- `event InvalidNonceEvent(address, uint, uint)`: fired when a bidder tries to open its bid providing an invalid nonce or amount (hash mismatch).
- `event FinalizeEvent(uint)`: fired after the finalize method is called and the winner has been chosen and losers have been refunded.
- `event RefundEvent(address, uint indexed amount, string)`: fired whenever a refund is emitted from the contract to a bidder. The string parameter is useful the better identify the cause of the refund.
- `event NotEnoughValidBiddersEvent(uint)`: if in the finalize method there are less than 2 valid bidders the auction is canceled and they get a full refund.

2.2.3 Gas evaluation

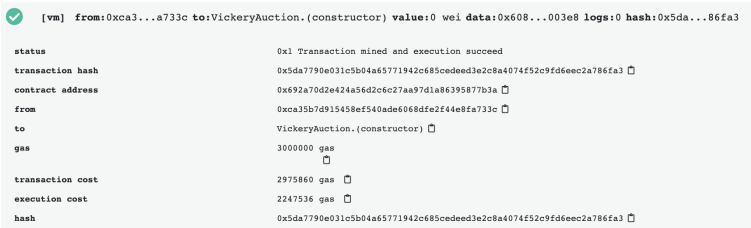

Figure 3: Deployment cost: 2975860 gas.

Figure 4: *makeBid()* cost: 150315 gas.

Figure 5: *withdrawal()* cost: 75492 gas.

Figure 6: *openBid()* cost: 97048 gas.

The cost of the *finalize()* called in an auction of 3 valid bidders have a cost of 90960 gas.

2.2.4 Testing

Also in this contract i preferred to use *Truffle* to perform some unit testing rather than manually execute transaction on *Remix*.

```
giacomodeliberali@Giacomos-MBP ~/Code/personal/unipi/smart-auctions (master) $ truffle test test/vickery.js
Using network 'development'.


Compiling your contracts...
===========================
> Compiling ./contracts/dutch/DutchAuction.sol
> Compiling ./contracts/dutch/IDecrease.sol
> Compiling ./contracts/dutch/LinearStrategy.sol
> Compiling ./contracts/vickery/HashGenerator.sol
> Compiling ./contracts/vickery/VickeryAuction.sol
> Artifacts written to /var/folders/zg/y687rmh16zq91j_jhm8g3yjh0000gn/T/test-119515-2476-1nywqb0.s8nn
> Compiled successfully using:
   - solc: 0.5.8+commit.23d335f2.Emscripten.clang


  Contract: VickeryAuction
    ✓ Should be deployed correctly in Commitment phase
    ✓ Should only allow makeBid (118ms)
    ✓ Should check deposit in makeBid (50ms)
    ✓ Should ensure only one commitment is made by same address (109ms)
    ✓ Should accept bids (525ms)
    ✓ Should refund all if not enough bidders (391ms)


  6 passing (2s)
```

Figure 7: *VickeryAuction* unit test results