FinalTerm Peer2Peer Systems and blockchains

Smart Auctions

Lorenzo Bellomo, 531423

Attached Files Structure

Attached with the project comes the auction files. The chosen auction is the Dutch one.
The final term code consists of three files:

- *dutchAuction.sol*: This file contains the Dutch auction logic. More details are explained in section Dutch Auction.
- *decreaseLogic.sol*: This file, to be associated with *dutchAuction.sol*, contains the *DecreaseLogic* interface contracts, and the 3 implemented logics: *Linear*, *Exponential* and *Logarithmic*.
- *vickreyAuction.sol*: This file contains the logic of the Vickrey Auction. Further details are explained in section Vickrey Auction.

Notes Common to Both Auctions

Handling of Time Time was handled in the same way between the two auctions. All the durations (except for the one of grace time) are expressed in *number of blocks* and are passed as parameter to the constructor of the auction. Grace time, instead, is constant and "hard coded" in the contract. To respect the specifications of the assignment this value should be around 20/23 in order to respect the request of a time close to 5 minutes for the grace time. This is computed assuming an average mining time of 13/15 seconds per block on the Ethereum blockchain (according to the values in `https://etherscan.io/chart/blocktime`). However, in the actual implementation, the constant value for graceTime is set to 2 in order to ease the burden of testing.
The time passage is implemented in a *lazy* way. What this means is that every time a major function, like *bid()*, is called, the contract checks the current block number, and by confronting it to the first one (the block number when the auction was generated), and by checking the various durations specified, it computes the current phase. So no central entity is required to synchronize the auction (no auctioneer is required to dictate the passing of time).
The main pros of this approach (lazy way), with respect to the one with the auctioneer that dictates the passage of time are:

- *Completely decentralized*: No central entity (auctioneer) is required to make the time pass.
- *Guaranteed time correctness (on average)*: The auctioneer model requires the this entity to make the time flow. This means that he might be late while calling the phase switch (but not early supposing that the contract checks the correctness of the auctioneer calls). This is not the case of the lazy implementation, where the checks are made for each call by the contract.
  The time is guaranteed to be respected on average, with possible fluctuations given by the variability in the mining process. This could have been avoided by using the timestamp of the block of the function call transaction, but this hypothesis was discarded in order to stick to the project requirements and implement time with respect to number of blocks.

Instead the main cons of this choice are:

- *Events variability*: Both of the auction implementations emit events related to the passage of time. Those events are fired whenever a phase switch is recognized. Since the phase switch process is lazy, the events may fire late with respect to the actual time in which the phase switched. This issue is inevitable in this kind of implementation, and it can be eased by providing a special method that checks the current phase of the auction and updates the state of the contract if enough time has passed. This method is *checkIfAuctionEnded()* for the Dutch Auction and *updateCurrentPhase()* for the Vickrey one.
- *Events cost*: The cost of those events emissions (which is very high with respect to normal EVM instructions) are in the general case at the expenses of the users (the bidders), which fire those events by executing the main methods of the contracts. The same "solution" of the first point is adopted, but this problem is evident, particularly when dealing with "almost empty auctions".

Dutch Auction

Main Functions The file has two main methods:

- *bid()*: This method simply allows a user to make a bid if everything respects the correctness parameters.
- *checkIfAuctionEnded()*: This method is only allowed to auction owner and allows to check if the auction has ended with no bidder.

In addition to these two the method provides some getter functions, in the form of *Solidity View Methods*.

Events The only event that is generated by this auction is the one emitted when it ends. This is generated when either the owner discovers that no offers were given or when someone tries to bid but discovers that the limit time was reached.

Decrease Logic The three implemented decrease logics are:

- *Linear*: This contract simply provides the linear decrease logic, which goes linearly with the passing of time from *startPrice* to *reservePrice*.
- *Exponential*: This contract provides the exponential decrease logic, which means that the first price drops are huge only to later decrease the price drops in order to reach the reservePrice
- *Logarithmic*: This contract is the opposite of the exponential one, meaning that the first price drops are low, only to increase with time.

The $\lceil \log_2 x \rceil$ function, used in order to implement both the logarithmic and the exponential decrease logic, was copied from a forum online. It uses inline assembly code and has a fixed cost of 757 *wei*.

Testing

Vickrey Auction

Main Functions The main functions and modifiers provided by this file are:

- *changeAuctionPhase(blockNumber)*: This is a modifier that checks the current time (current block number) with respect to the parameters passed when constructing the auction contract. This modifier is used before any of the following methods are called.
- *bid(hash)*: This function takes a commitment as parameter and tries to submit it if all the parameters are respected (correct time range for making bids, paid deposit...).
- *withdraw()*: This function, if all the parameters are respected, undoes the commitment of the function caller.
- *open(nonce)*: This function is both used to open the envelopes of the bids, and to refund losers immediately (before the owner calls finalize). The case of a non valid commitment (which means that the commitment hash and the hash computed with the received nonce do not coincide), is handled as a non valid bid, which means that the deposit fund is not refunded. This is the same treatment of an opening which reveals that the original payment was below the reserve price threshold.
- *finalize*: This function can only be called by the owner of the auction, and only when the auction state is *ENDED*. This method simply refunds all the not yet refunded accounts.
- *changeAuctionTime()*: This method simply calls the modifier specified to change the auction phase.

Events The events that are generated by this auction are:

- *Time related events*: Those are simple events that follow the passing of time (phases). They are *GraceTimeOver*, *CommitmentOver*, *WithdrawalOver* and *AuctionEnded*. All those events carry some informations, like the number of current bidders (so the number of valid commitments that were submitted), and the duration of the next phase. As discussed in section Notes Common to Both Auctions, all those events are emitted in a *lazy* way.
- *Informative*: Those events (*NewCommitment* and *NewWithdrawal*) are emitted every time a new valid commitment is made or withdrawn.

Testing