FreeTON Auctions

1. Introduction

This document will describe the submission proposed by the Origin Labs / OCamlPro team for solving the DevEx contest about on-chain auctions. The goal of this contest was to provide smart-contracts (SC) for deploying and managing different type of auctions (English & Dutch) with multiple variations (Forward, Reversed & Blind). These SCs must be compatible with TON's Crystals, TIP3 & DePools. Also, they must be written in Solidity or C++, the FreeTON SC's languages.
This submission includes

- The auction source code (contracts/)
- Deployment scripts using `ft`
- A specification of each auction in TLA+ (tla+/)
- A high-level specification in an UML fashion (spec/)
- A debot for interacting with the SC framework (contracts/debot/)

2. Auction specification

We divided the six auctions into three categories, English, Dutch, which can either be Forward or Reverse and Clear (by default) or Blind. The logic behind each auction is described in the TLA+ files.

- English Forward: the auction is initialized with a *starting price*, a *max tick* (time unit) and a *max time* (time unit). The auction always end when *max time* is reached. As long as no bid is done, nothing happens. When a bid is done, the auction ends after *max tick* without any other bid **higher** than the last one (or if *max time* is reached).

- English Reverse: same than English Forward, except a new bid must be **lower** than the last one.

- Dutch Forward: the auction is initialized with a *starting price*, a *limit price*, a *price delta* and a *time delta* (time unit). The *bidding price* is a function of the time spend since the beginning of the auction, the *price delta* and the *time delta*. After each *time delta*, the bidding price is **decremented** of *price delta* until it reaches *limit price*. If the price reaches *limit price*, the auction ends after *time delta*. If anyone bids a value **higher** than the *bidding price*, he automatically wins the auction.

- Dutch Reverse: same than Dutch Reverse, except the *bidding price* is **incremented** and the auction ends when anyone bids a value **lower** than the *bidding price*.

Blind auctions work the same way as a **Clear** auctions, except the commitment is hidden until validation of the bidder.
These auction definitions hare the pleasant property that they **eventually ends**, either with or without a winner. Also note that the difference between a Forward & Reverse auction is quite small. The SC implementation takes into account this similarities to limit their size.

3. Infrastructure

Here is the list of the main contracts (including their interface) with a short description.

- *AuctionRoot*: the main contract of the infrastructure ; this contract will deploy the auctions and the bid managers.

- *BidBuilder*: the bid manager ; an instance of this contract is built when an auction is created and will be in charge of creating Bid contracts and validating bids.

- *Bid*: the bid contract ; when a user wants to bid to an auction, the auction sends a request to its BidBuilder to create a new Bid – this Bid is in charge of checking the status of a generic “Vault” contract. If the status of the “Vault” is correct, then the Bid is valid and tells the BidBuilder that it is correct.

- ReverseBid : the bid for reverse auctions

- BlindBid : the bid for blind auctions

- *VEnglishAuction*: the logic behind any English auction ; this abstract contract encodes the bidding process of an English Auction with a virtual function *newBidIsBetterThan*.

- *VDutchAuction*: the logic behind any Dutch auction ; this abstract contract encodes the bidding process of an Dutch Auction with a virtual function *betterPriceThanCurrent*.

- *Constants* : defines miscellaneous constants, modifiers and events used through the project.

The actual Auction contracts are defined in *EnglishAuction.sol, EnglishReverseAuction.sol, DutchAuction.sol* and *DutchReverseAuction.sol*. They inherit their respective abstract contract (*VEnglishAuction* or *VDutchAuction*) and only define the virtual function (*newBidIsBetterThan* or *betterPriceThanCurrent*).

Few more interfaces are defined. These contracts have no static implementation and must be defined by the Auctioneer.

- *IVault:* a contract holding the bidding funds. When a bid is done, the user must satisfy the “Vault” constraint (for example, transferring a given amount of crystals on its balance). The funds on the “Vault” are locked and managed by the Bid contract ; this last contract will either send it back to the Bidder if he lost the auction or to the Auctioneer if he won.

- *IRootWallet:* the manager of Vaults. They define a function *getWalletAddress* that returns a Vault address (mostly used for TIP3 compatibility)

- *IProcessWinner.* a contract that should only be called by an auction. It has two functions: *acknowledgeWinner* and *acknowledgeNoWinner* that are respectively called when there is a winner / there is no winner.

4. Description of the Auction process

The file *spec/bid_diagram.pdf* describes the different steps of an auction.

1. Auction deployment

The auctioneer defines a Winner Processor. In Forward auctions, the auctioneer will sell the contract ownership while in Reverse auctions, he simply provides the contract code he wants to take control of. Then, he deploys an auction through the AuctionRoot contract. AuctionRoot deploys 2 contracts : the Auction & its associated BidBuilder. The Auctioneer then must initialize the BidBuilder with the address of the Auction (this step could be avoided by calculating in advance the auction address). The Auction can then be started.

2. Bidding process

Any user can commit a bid by calling the *bid* function of an Auction. The Auction requests the BidBuilder to create a Bid contract. This Bid contracts emits the details of its associated Vault (in Forward, a vault is created for every contract while in Reverse, there is a single vault containing the auctioneer funds). The user then transfers funds to the Vault (Forward) or deploys a winner processor contract (Reverse); when this is done, the bidder asks to check the correctness of the bid. If it is correct with respect to the Bidder's commitment, Bid validates its status to BidBuilder, who checks the origin of the validation. If the validation comes from a correct Bid contract, it transfers the validation to the Auction. The Auction has now a new bid, the potential old bid transfers back its vault to the previous bidder.

3. Ending an auction

a) Auction winner

The function *acknowledgeWinner* is called with the best bidder information (address, amount, Bid address & Vault address) by the auction. The Vault content is transferred to the Auctioneer (Forward) or the buyer (Reverse).

b) No winner

The function *acknowledgeNoWinner* is called.