Peer to Peer Systems and Blockchains
Academic Year 2018/2019
Finalterm assignment
Smart Auctions: Dutch, English and Vickrey Auctions on the Ethereum blockchain

Report

Leonardo Frioli
580611

Index

Introduction

The main purpose of this report is to explain some choices done during the implementation of the final term, to show the gas usage of the most relevant functions and to provide some lists of operations in order to run the contracts.

Beside the VickreyAuction that was mandatory, the DutchAuction has been implemented because it's more interesting with respect to the common English one. Both the Auction have been implemented and tested in the Remix web environment using the version 0.4.22 of the compiler.

General Considerations

In this section we are going to discuss considerations concerning both the implemented auctions. In particular we are going to talk about the implementation of a basic escrow contract, the role of the auction house, the implementation of the addBlock function and of the grace period.

Escrow

A simple escrow contract (SimpleEscrow.sol) is provided. This contract offer a very basic service for resolving conflicts between the seller and the buyer (winner bidder). The contract must be (and indeed is) initialized inside one of the two Auction contracts. The auction contracts will then expose the SimpleEscrow functionalities providing wrappers to its functions. The SimpleEscrow requires in input the address of a trusted third party that will resolve, if needed, the conflict between the two parties and that is the only one to have the power to conclude the escrow (function conclude()). For simplicity this contract is initialized using as trusted third party the auction house.

Auction House

The concept of auction house has been modelled because reading the Final-Term specifications and the Final-Project ones it seems to be required. The auction house is that address which deploys the contract, calls the finalize function in the VickreyAuction, and acts as third trusted party in the escrow.
Nevertheless, I think that this kind of entity can be omitted because one of the aims of the distributed blockchain is to avoid a central authority. In this case (in my opinion) the auction house is acting as a central authority; we could have disregarded it and have given to the selller the duty of the deployment of the Auction.

AddBlock function

Since we are in Remix, we are the only ones putting blocks on the local blockchain. Some functionalities required need some time to pass, counted as blocks in the chain. To avoid commenting this functionalities to have a fast test phase, a dummy function called **addBlock** (which emits an event) has been implemented. This function can be used to add blocks in the blockchain simulating a real environment. Of course this function is just for testing puposes and if we think to a real deployment we should delete it.

Grace Period

The grace period has, more or less, the same problem stated in the addBlock function. We can’t set fixed mining rate of, for istance, 1 block every 15 seconds because in that way we need to add quite a lot of blocks before starting. The mining rate is taken as an input parameter in the contract constructor so that it’s easier to set it. In that way a rate of 1 block every 300 seconds(5 mins) can be passed so that the grace period is equal to 1 and it’s easier to test the Auction. This input parameter is anyway useful in a real deployment because allows to have more flexibility and to allign the contract mining rate parameter to that one of the Ethereum blockchain.

VickreyAuction

The VickreyAuction has been implemented as stated in the specifications. Only few functionalities need to be discussed, such as the **deposit** parameter and the **doKekkak** function.

Deposit

The deposit is taken as input parameter, but it must be at least two times the reservePrice. In this way a dishonest bidder must pay (at least) the reservePrice to another bidder to convince it to withdraw, or has to pay two times the reservePrice to convince him to not open his bid. Since the bad agent should do this for more than one bidder if he wants to increase his chances of winning, he will be forced to invest a lot of money. From the other hand a honest agent is encouraged to open his bid because in that way he can take back all the money and avoid to loose half of the deposit or the entire one.

doKekkak function

The **commitBid** requires a kekkak hash in input, so the contract provides a **pure** function called **doKekkak** to get the desired value. This function, as the **addBlock** one previously discussed is provided only to ease the testing phase, if we want to deploy the contract in a real blockchain maybe it's better to delete it.

Gas Estimation

The next table will show the gas cost provided by Remix during the execution of the most important functions. The functions considered are those one called in the examples present in the **Appendix [1]**.

Functions	Transaction cost	**Execution Cost**	Comments
deploy Auction	3470830	**2632914**	
commitBid	46658	**23210**	We got always this costs
withdraw	21827	**15555**	We got always this costs
open	30812	**24348**	
finalize	831450	**810178**	The cost is high because we should consider the SimpleEscrow contract creation inside
acceptEscrow	32709	**11437**	
concludeEscrow	35282	**14010**	
total	4469568	**3531652**	

The execution cost is the relevant parameter, but also the transaction cost has been inserted. The values are derived from an average of the values of gas given by Remix output panel.

DutchAuction

The implementation of the DutchAuction is pretty much linear, the only thing to discuss is the implementation of the Decreasing Strategies

Decreasing Strategies

The strategies implemented are the ones in **[Fig1]**: the blue is linear, the yellow is logarithmic and the red is an inverse logarithm. The yellow curve and the red are calculated from the green mirroring the result w.r.t. the line x = 4 for the yellow, and y = 2 for the red.

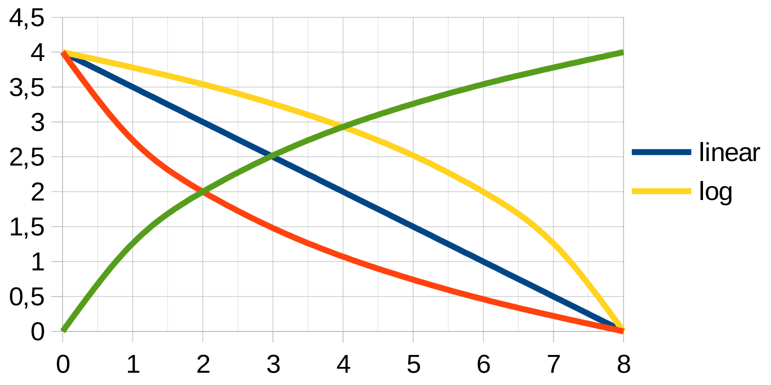

Fig1 - Decreasing Strategies

Gas Estimation

The next table will show the gas cost provided by Remix during the execution of the most important functions. The functions considered are those one called in the examples present in the **Appendix [2]**.

Functions	Transaction cost	**Execution Cost**	Comments
deploy Strategy	287017	**180157**	
deploy Auction	2310701	**1723485**	
bid (unsuccessful)	34851	**13579**	A correct bid but smaller the the current price
bid (suceessful)	825993	**804721**	The cost is high because we should consider the SimpleEscrow contract creation inside
acceptEscrow	32060	**10788**	
concludeEscrow	34794	**13522**	
total	3525416	**2746252**	

The execution cost is the relevant parameter, but also the transaction cost has been inserted. The values are derived from an average of the values of gas given by Remix output panel

Leonardo Frioli - 580611

Appendix

We will know provide some list of operations to perform in order to try the Auctions.

[1] List of operations to run the VickreyAuction

- VICKREY AUCTION – LIST 1-

- change the Remix gas limit to 4000000
- choose an address that will be the auction house and switch to it
1) Deploy the VickreyAuction contract using as parameters
   - _reservePrice = 2000000000000000000 // 2 ether
   - _commitmentPhaseLength = 3
   - _withdrawalPhaseLength = 2
   - _openingPhaseLength = 3
   - depositRequired = 4000000000000000000 // 2*reservePrice
   - _seller = **choose here the seller address**
   - miningRate = 300 // 1 block every 300 seconds

2) call <u>addBlock</u> 1 (grace period) time
- choose an addres that will be BIDDER1 and switch to it
3) call <u>doKekkak</u>(4000000000000000000,4)
4) call <u>commitBid</u> using the value retuned from the previous function,
passing as Value 4 ether (deposit)
- choose an addres that will be BIDDER2 and switch to it
5) call <u>doKekkak</u>(5000000000000000000,5)
6) call <u>commitBid</u> using the value retuned from the previous function,
passing as Value 4 ether (deposit)
- choose an addres that will be BIDDER3 and switch to it
7) call <u>doKekkak</u>(3000000000000000000,3)
8) call <u>commitBid</u> using the value retuned from the previous function,
passing as Value 4 ether (deposit)
9) call <u>addBlock</u> 2 (_withdrawalPhaseLength) times
   - no withdraw in this run

- switch to the address of BIDDER2
10) call <u>open</u>(5) passing as Value 5 ether
- switch to the address of BIDDER1
11) call <u>open</u>(4) passing as Value 4 ether
- switch to the address of BIDDER3
12) call <u>open</u>(3) passing as Value 3 ether
   - BIDDER3 will get compensated of 3 + 4 (deposit) ether

- switch to auction house address
13) call the <u>finalize</u> function
   - BIDDER1 will get 4 + 4 (deposit) ether
   - BIDDER2 wille get 1 (difference between its bid and BIDDER1 bid) + 4 (deposit) ether

- switch to BIDDER2 address
14) call the <u>acceptEscrow</u> function
- switch to seller address
15) call the <u>acceptEscrow</u> function
- switch to auction house address
16)call the <u>concludeEscrow</u> function
   - the seller will get the money (4 ether)

- VICKREY AUCTION – LIST 2-

- change the Remix gas limit to 4000000
- choose an address that will be the auction house and switch to it
1) Deploy the VickreyAuction contract using as parameters
   - _reservePrice = 2000000000000000000 // 2 ether
   - _commitmentPhaseLength = 3
   - _withdrawalPhaseLength = 2
   - _openingPhaseLength = 3
   - depositRequired = 4000000000000000000 // 2*reservePrice
   - _seller = **choose here the seller address**
   - miningRate = 300 // 1 block every 300 seconds

2) call addBlock 1 (grace period) time
- choose an addres that will be BIDDER1 and switch to it
3) call doKekkak(4000000000000000000,4)
4) call commitBid using the value retuned from the previous function,
passing as Value 4 ether (deposit)
- choose an addres that will be BIDDER2 and switch to it
5) call doKekkak(5000000000000000000,5)
6) call commitBid using the value retuned from the previous function,
passing as Value 4 ether (deposit)
- choose an addres that will be BIDDER3 and switch to it
7) call doKekkak(3000000000000000000,3)
8) call commitBid using the value retuned from the previous function,
passing as Value 4 ether (deposit)
- switch to the address of BIDDER2
10) call withdraw
   - BIDDER2 will get back 2 ether (half of the deposit)

- switch to the address of BIDDER1
11) call withdraw
   - BIDDER1 will get back 2 ether (half of the deposit)

- switch to the address of BIDDER3
12) call open(3) passing as Value 3 ether
   - it is the only one remained

13) call addBlock 2 (remaining openingPhase period) time
- switch to auction house address
14) call the finalize function
   - BIDDER3 wille get 1(difference between its bid and reservePrice) + 4 (deposit) ether

- switch to BIDDER3 address
15) call the acceptEscrow function
- switch to seller address
16) call the acceptEscrow function
- switch to auction house address
17)call the concludeEscrow function
   - the seller will get the money (2 ether)

Leonardo Frioli - 580611

- VICKREY AUCTION – LIST 3-

- change the Remix gas limit to 4000000
- choose an address that will be the auction house and switch to it
1) Deploy the VickreyAuction contract using as parameters
- _reservePrice = 2000000000000000000 // 2 ether
- _commitmentPhaseLength = 3
- _withdrawalPhaseLength = 2
- _openingPhaseLength = 3
- depositRequired = 4000000000000000000 // 2*reservePrice
- _seller = **choose here the seller address**
- miningRate = 300 // 1 block every 300 seconds

2) call addBlock 1 (grace period) time
- choose an addres that will be BIDDER1 and switch to it
3) call doKekkak(4000000000000000000,4)
4) call commitBid using the value retuned from the previous function,
passing as Value 4 ether (deposit)
- choose an addres that will be BIDDER2 and switch to it
5) call doKekkak(5000000000000000000,5)
6) call commitBid using the value retuned from the previous function,
passing as Value 4 ether (deposit)
- choose an addres that will be BIDDER3 and switch to it
7) call doKekkak(3000000000000000000,3)
8) call commitBid using the value retuned from the previous function,
passing as Value 4 ether (deposit)
9) call addBlock 2 (_withdrawalPhaseLength) times
- no withdraw in this run

- switch to the address of BIDDER3
12) call open(3) passing as Value 3 ether
- switch to the address of BIDDER2
10) call open(5) passing as Value 5 ether
11) call addBlock 1(remaining openingPhase period) times
- switch to auction house address
13) call the finalize function
- BIDDER1 didn’t call the withdraw function and didn’t opened the bid
- BIDDER3 will get 3 + 4 (deposit) ether
- BIDDER2 wille get 2 (difference between its bid and BIDDER3 bid) + 4 (deposit) ether

- switch to BIDDER2 address
14) call the acceptEscrow function
- switch to seller address
15) call the acceptEscrow function
- switch to auction house address
16)call the concludeEscrow function
- the seller will get the money (3 ether)

[2] List of operations to run the DutchAuction

- DUTCH AUCTION - LIST 1 -
- choose an address that will be the auction house and switch to it
0) Deploy the LinearDecreasingStrategy
1) Deploy the DutchAuction contract using as parameters
    - _reservePrice = 1000000000000000000 // 1 ether
    - _initalPrice = 5000000000000000000
    - _openedForLength = 8
    - _seller = **choose here the seller address**
    - _decrStrategy = the address of the deployed linear strategy
    - miningRate = 300 // 1 block every 300 seconds

2) call addBlock 1 (grace period) + 2 times
- choose a bidder address and switch to it
3) set Value as 2 ether and call bid
    - this call will be mined but you will get compensated because your bid is too low

4) call addBlock two times
5) set Value as 2 ether and call bid
    - you are the winner of the auction, check it in the logs

6) call the acceptEscrow function
- switch to the seller address
7) call the acceptEscrow function
- switch to the auction house address
8) call the concludeEscrow function
    - the seller will get the money because the escrow finished in a good way.

- DUTCH AUCTION - LIST 2 -
- choose an address that will be the auction house and switch to it
0) Deploy the LogarithmicDecreasingStrategy
1) Deply the DutchAuction contract using as parameters
    - _reservePrice = 1000000000000000000 // 1 ether
    - _initalPrice = 5000000000000000000
    - _openedForLength = 8
    - _seller = **choose here the seller address**
    - _decrStrategy = the address of the deployed logarithmic strategy
    - miningRate = 300 // 1 block every 300 seconds

2) call addBlock 1(grace period) + 7 times
- choose a bidder address and switch to it
3) set Value as 1 ether and call bid
    - you are the winner of the auction

4) call the acceptEscrow function
- switch to the auction house address
5) call the acceptEscrow function
6) call the concludeEscrow function
    - the winner will get the money back because the seller didn’t accept the escrow.

- DUTCH AUCTION - LIST 3-
- choose an address that will be the auction house and switch to it
0) Deploy the InverseLogarithmicDecreasingStrategy
1) Deply the DutchAuction contract using as parameters
- _reservePrice = 1000000000000000000 // 1 ether
- _initalPrice = 5000000000000000000
- _openedForLength = 8
- _seller = **choose here the seller address**
- _decrStrategy = the address of the deployed inverse logarithmic strategy
- miningRate = 300 // 1 block every 300 seconds

2) call addBlock 1 (grace period) time
- choose a bidder address and switch to it
3) set Value as 5 ether and call bid
- you are the winner of the auction

- switch to the seller address
4) call the acceptEscrow function
- switch to the auction house address
5) call the acceptEscrow function
6) call the concludeEscrow function
- the seller will get the money anyway because the winner didn’t accept the escrow.