

Dipartimento di Informatica
Corso di Laurea Magistrale in Informatica

Final Term:
Smart Auctions: Dutch, English and Vickrey
Auctions on the Ethereum blockchain

Studente:
Eugenio Paluella

Peer to Peer Systems and Blockchains
Anno Accademico 2018/2019

Indice

1. Introduzione

2. Implementazione

2.1. Codice

2.2. Scelte Implementative

2.2.1. Scelta Del Token

2.2.2. Dutch

2.2.3. Vickrey

3. Test

3.1. Test Dutch

3.2. Test Vickrey

4. Analisi del gas delle funzioni

4.1. Dutch

4.2. Vickrey

1 Introduzione

In questo Final Term l'obiettivo è quello di implementare delle smart auctions distribuite sulla blockchain di Ethereum. Nel caso specifico è stato scelto di implementare la Dutch Auction e la Vickrey Auction.

2 Implementazione

La simulazione è stata implementata utilizzando Solidity, Remix, python insieme a web3py, deployando lo smart contract sia sulla testnet Rinkeby che su ropsten.

Solidity è stato usato per scrivere gli smart contract, è stato usato anche lo standard ERC721 per creare un token e la libreria SafeMath per garantire la sicurezza delle operazioni.

Con Remix sono stati scritti e deployati gli smart contract.

Con python è stato implementato uno script che, dati nonce e bid, permette di generare un keccak256 come quello utilizzato su solidity.

2.1 Codice

Il codice è suddiviso in diversi smart contract in base alla auction presa in esame.

Dutch Auction:

- Dutch.sol
- StrategyInterface.sol
- PercDecrease.sol
- LinearDecrease.sol
- StaticDecrease.sol

Nel file Dutch.sol è stata implementata la logica dell'asta. StrategyInterface è un'interfaccia necessaria per l'utilizzo del Design Pattern strategy. Questo Design Pattern consiste nell'incapsulare un algoritmo all'interno di una classe, mantenendo un'interfaccia generica, che permette di avere un codice più scalabile.

PercDecrease, LinearDecrease e StaticDecrease sono tre distinti metodi che l'utente può scegliere per far decrescere (oppure non far decrescere nel caso di StaticDecrease) il prezzo della propria asta.

Vickrey Auction:

- Vickrey.sol

In Vickrey.sol è stata implementata la logica dell'asta.

In entrambe le aste sono stati utilizzati dei token non fungibili(ERC721) contenuti nei seguenti contratti:

ERC721:

- P2PToken.sol

- `erc721.sol`
- `erc721-token-receiver.sol`
- `erc721-metadata.sol`
- `erc721-enumerable.sol`
- `erc165.sol`
- `nf-token-enumerable.sol`
- `nf-token.metadata.sol`
- `nf-token.sol`
- `ownable.sol`
- `safe-math.sol`
- `supports-interface.sol`
- `address-utils.sol`

`P2PToken.sol` contiene la definizione del token, con nome, symbol e la funzione per creare nuovi token.

Le interfacce `erc721.sol`, `erc721-token-receiver.sol`, `erc721-metadata.sol`, `erc721-enumerable.sol`, `erc165.sol` definiscono le funzioni necessarie per l'utilizzo dei token.

`Nf-token.sol`, `nf-token-enumerable.sol`, `ownable.sol` contengono tutte le funzioni per lo scambio di token, per il controllo del balance o per controllare il proprietario di un dato token. `Nf-token.metadata.sol` contiene i metadati, come URI e descrizione del token.

La libreria `safe-math.sol` ridefinisce tutte le funzioni matematiche base (somma moltiplicazione ecc..) per proteggere il codice da overflow quando si utilizzano uint.

`Address-utils.sol` permette di riconoscere quando un address appartiene ad uno smart contract e quando ad un utente.

Per l'implementazione dei token è stato utilizzato in parte codice open source, altri file sono stati invece creati da zero o modificati in base alle necessità.

2.2 Scelte Implementative

2.2.1 SCELTA DEL TOKEN

Il contesto in cui agisce questo smart contract è quello di un'asta, in cui c'è un utente che mette in vendita qualcosa ed un compratore. Come possiamo rendere sicuro lo scambio se il compratore paga allo smart contract e il venditore spedisce il suo bene al di fuori della blockchain? A mio parere, l'unica garanzia in questo caso è che lo scambio avvenga completamente on-chain, quindi il venditore invia allo smart contract il token che vuole mettere in vendita (potenzialmente il token potrebbe rappresentare qualsiasi cosa, da un contratto per un bene immobile ad un `cryptokitties`). Mentre il compratore potrà vedere le informazioni contenute nel token ed inviare la quantità di denaro richiesta, in questo modo il contratto avrà funzione di escrow, nel momento in cui l'asta sarà conclusa effettuerà lo scambio. Lo standard scelto per il token è ERC721, un token infungibile, la scelta è ricaduta su questa tipologia in quanto viene messo all'asta un bene e non una valuta.

2.2.2 DUTCH

- `Constructor()`:

Quando il contratto viene deployato vengono definiti prezzo di riserva e prezzo di partenza controllando che entrambi siano maggiori di 0 con il modifier minPrice. Viene poi definita la durata dell'asta, espressa in numero di blocchi, controllando anche qui che sia maggiore di 0 attraverso il modifier minDuration. Infine vengono definiti il creator del contratto, che coincide con il creator dell'asta, l'id del token e il metodo di decremento scelto.

- **UpdateContractDecreaseMethod():**
È possibile, in seguito alla creazione del contratto modificare il metodo per far decrementare il prezzo.
Sono stati aggiunti però dei vincoli per garantire la sicurezza, infatti sarà modificabile solo nel periodo che va dalla creazione del contratto all'inizio effettivo dell'asta, perché sennò sarebbe possibile per il creator modificare il metodo e conseguentemente modificare il prezzo durante il corso dell'asta. Inoltre sarà possibile aggiornare il metodo solo con altri metodi esistenti e solo da parte del creator dell'asta.
- **StartAuction():**
Questa fa iniziare l'asta, quindi il blocco di inizio dell'asta diventa quello in cui questa funzione è stata invocata, mentre il blocco di fine viene impostato come: blocco di inizio + durata.
Anche qui sono stati inseriti dei vincoli, infatti la funzione può essere invocata solo dal creator. Può essere invocata solo una volta, infatti viene controllato che il blocco di inizio non sia ancora settato. Infine l'asta non può partire finché non viene inviato il token allo smart contract. Questo vincolo è stato inserito per garantire lo scambio, infatti in questo modo il contratto avrà funzione di escrow, prenderà il token del venditore e riceverà il pagamento dall'offerente, nel momento in cui l'asta sarà conclusa effettuerà lo scambio. Inoltre l'utilizzo del token permette all'utente che entrerà a far parte dell'asta come offerente di poter controllare cosa sta effettivamente comprando.
- **ActualPrice():**
In base alla modalità di decremento del prezzo scelta dall'utente viene chiamata la funzione e viene restituito il prezzo al momento della chiamata.
Per evitare che i concorrenti dell'asta conoscano il prezzo esatto questo metodo può chiamarlo solo il creator.
- **Bid():**
Funzione che può essere chiamata da chiunque, richiede il pagamento immediato della bid, il pagamento viene accettato però solo se è maggior o uguale del prezzo attuale(prezzo considerato il decremento). Inoltre è accettato solo se l'asta non è stata già vinta da un altro utente o se il tempo non è scaduto.
- **PayOut():**
Se l'asta non ha avuto partecipanti restituisce il token al creator, altrimenti trasferisce i soldi della bid al creator ed il token all'utente che si è aggiudicato l'asta.
Se il periodo dell'asta non è concluso non può essere invocata.
Non sono stati inseriti vincoli su chi può invocarla in quanto l'asta è terminata, token ed offerta sono stati inviati allo smart contract, quindi restringere il campo degli utenti

che possono chiamare questa funzione, potrebbe portare solo ad un eventuale stallo in situazioni particolari.

- `StringToBytes()`:
Trasforma una string in byte
- `CloseContract()`:
Distrugge il contratto.

2.2.3 Vickrey

- `Constructor()`:
Vengono definite le durate delle 3 fasi dell'asta, controllando che siano tutte maggiori di 0, viene settato il creator e vengono impostati il prezzo di riserva ed il `depositBid` coontrollando che anche questi siano maggiori di 0. Inoltre viene definito l'id del token inviato ed il `charityaddress`, l'address a cui vengono inviato il balance in eccesso alla fine dell'asta
- `StartAuction()`:
Da inizio all'asta, quindi vengono impostati blocchi di inizio e fine di ogni fase. Sono stati inseriti dei vincoli, infatti la funzione può essere invocata solo dal creator. Può essere invocata solo una volta, infatti viene controllato che il blocco di inizio non sia ancora settato. Infine l'asta non può partire finché non viene inviato il token allo smart contract, questo vincolo è stato inserito per garantire lo scambio, infatti in questo modo il contratto avrà funzione di escrow, prenderà il token del venditore e riceverà il pagamento dall'offerente, nel momento in cui l'asta sarà conclusa effettuerà lo scambio inoltre l'utilizzo del token permette all'utente che entrerà a far parte dell'asta come offerente di poter controllare cosa sta effettivamente comprando.
- `BidCommitment()`:
In questa fase gli utenti che partecipano all'asta devono inviare l'hash della propria offerta+nonce, pagando anche il deposito settato in precedenza dal creator. L'hash non viene calcolato nello smart contract per evitare che vengano fatte transazioni con l'offerta "in chiaro", altrimenti gli altri utenti saprebbero quali sono le offerte da superare.
Viene controllato che l'utente non abbia già inviato un'offerta, che questa funzione venga chiamata solo nella sua fase e che il deposito che l'utente sta inviando coincida con quello impostato dall'utente.
- `Withdrawal()`:
In questa fase l'utente può decidere di tirarsi indietro ed ottenere la metà del deposito versato inizialmente. Viene controllato che sia stato fatto il deposito precedentemente dall'utente e che questa funzione venga chiamata solo nella giusta fase.
- `RevealBid()`:
In questa fase gli utenti rivelano la propria offerta. Viene chiamata questa funzione passando come parametro il nonce e pagando il value inserito nell'hash precedentemente. L'offerta viene accettata solo se l'hash del nonce + il value dell'offerta coincidono con l'hash mandato precedentemente. Viene inoltre controllato che l'offerta non sia già stata rivelata, in modo da non permettere di mandare più soldi

del previsto al contratto, se l'utente ha mandato precedentemente l'hash e che la funzione venga chiamata solo nella giusta fase.

Gli utenti che con le loro offerte non rientrano nei primi 2 "classificati" vengono immediatamente rimborsati dell'amount dell'offerta e del deposito iniziale.

- **Finalize():**

Questa funzione permette di finalizzare l'asta. Vengono considerati vari casi:

- Nel caso in cui nessuno faccia offerte il creator viene rimborsato del suo token.
- Nel caso gli offerenti dell'asta abbiano fatto offerte minori del prezzo di riserva vengono tutti rimborsati delle loro offerte, del loro deposito ma nessuno vince, quindi il token viene inviato al creator
- Nel caso in cui ci sia un solo partecipante e la sua offerta sia maggiore del prezzo di riserva, viene mandato al creator la seconda offerta più alta, in questo caso non essendoci offerenti coincide con il prezzo di riserva, all'offerente viene inviato il token e viene anche rimborsato del deposito e dell'eccedenza della sua offerta rispetto al prezzo di riserva.
- Nel caso in cui ci sia più di un partecipante e che tutti superino il prezzo di riserva:
 - Viene mandato al creator dell'asta la seconda offerta più alta;
 - Il token viene inviato all'offerente;
 - Viene rimborsato all'offerente il valore del deposito e dell'eccedenza della sua offerta rispetto alla seconda offerta;
 - Al secondo offerente viene rimborsata la sua offerta più il deposito iniziale.
- In ogni caso alla fine dell'asta se il balance del contratto non è a 0 viene mandato tutto il balance rimasta al charityaddress impostato dall'utente.

- **StringToBytes():**

Trasforma una string in byte

- **CloseContract():**

Distrugge il contratto.

3 Test

3.1 Test Dutch

1. Deploy Dutch su ropsten

The screenshot shows the 'Dutch' contract deployment interface. It includes a 'Deploy' section with the following fields:
- `_resPrice`: 10000000000000000
- `_sPrice`: 10000000000000000
- `_duration`: 50
- `_decreaseMethod`: 1
- `_tokenId`: 1
At the bottom right, there is a 'transact' button.

2. Recuperare P2pToken

The screenshot shows the 'P2pToken' contract interface. It has a 'Deploy' button and an 'At Address' section with a 'Load contract from Address' input field. A blue arrow points from a text box to this input field.
Text box:
Inserire address
0xcfc03245ca98b26455da
b6f27c69de6f10c1c3c4

3. Generare Token dal contratto P2pToken (Nel caso in cui si dovessero generare più token è necessario incrementare il numero del tokenId, in quanto viene controllato se un certo token è stato creato precedentemente)

The screenshot shows the 'mint' section of the P2pToken contract interface. It includes the following fields:
- `_to`: ADDRESS DUTCH
- `_tokenId`: 1
- `_uri`: TEST
- `_descr`: TEST
At the bottom right, there is a 'transact' button. A blue arrow points from a text box to the `_to` field.
Text box:
Inserire address ottenuto
deployando dutch

4. Aspettare che il token arrivi allo smart contract

5. Funzione Start Auction

6. Funzione Bid (Va inserita l'offerta, che deve essere minore o uguale del prezzo attuale, volendo si può inserire il prezzo di partenza)

7. Funzione Payout

3.2 Test Vickrey

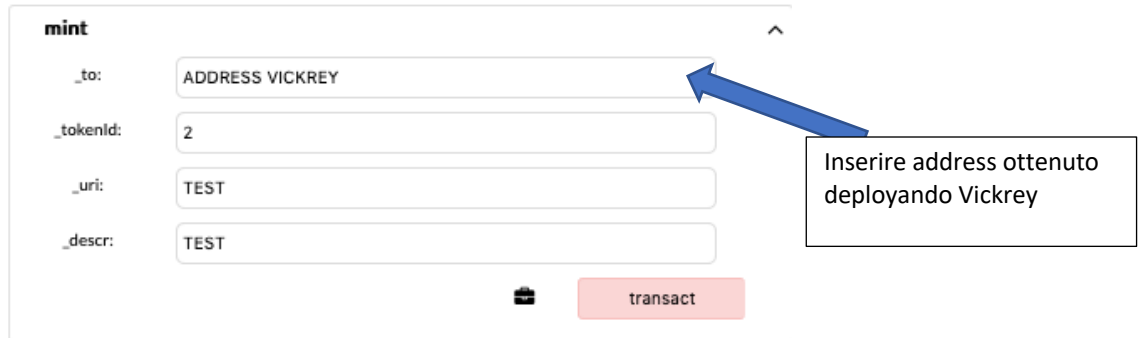
1. Deploy Vickrey su ropsten

The screenshot shows the 'Vickrey' contract deployment page. It features a 'Deploy' section with several input fields:
- `_phase1:`, `_phase2:`, and `_phase3:` are all set to '4'.
- `_depositBid:` is set to '1000000000000'.
- `_reservePrice:` is set to '1000000000000000'.
- `_charityAddress:` is highlighted with a blue border and contains the text 'ADDRESS A SCELTA'.
- `_tokenId:` is set to '2'.
Below the inputs is a 'transact' button. At the bottom, there is an 'or' section with a button labeled 'At Address' and a text input field containing 'Load contract from Address'.

2. Recuperare P2pToken

The screenshot shows the 'P2pToken' contract interface. It has a 'Deploy' button and an 'or' section with an 'At Address' button and a text input field containing 'Load contract from Address'. A blue arrow points from a text box on the right to the 'Load contract from Address' field. The text box contains the address:
Inserire address
0xcfc03245ca98b26455da
b6f27c69de6f10c1c3c4

3. Generare Token dal contratto P2pToken (Nel caso in cui si dovessero generare più token è necessario incrementare il numero del tokenId, in quanto viene controllato se un certo token è stato creato precedentemente)



mint

_to: ADDRESS VICKREY

_tokenId: 2

_uri: TEST

_descr: TEST

transact

Inserire address ottenuto deployando Vickrey

4. Aspettare che il token arrivi allo smart contract

5. Funzione Start Auction

6. Lanciare lo script python getKeccakHash.py per ottenere l'hash. In alternativa utilizzare i seguenti dati:

Nonce: 123
Bid: 110000000000000000
Hash: 0x51ebdbf79a7e049c761c759023a4246fa44c58890937e43eeb7d858efde74303

7. Funzione BidCommitment passando come parametro l'hash generato precedentemente e inserendo come value della transazione esattamente il valore del deposito
8. Aspettare la fase 2(si può controllare in che fase ci troviamo attraverso la call getPhase)
9. Se necessario chiamare la funzione withdrawal, una volta chiamata non sarà possibile proseguire con l'asta
10. Aspettare la fase 3
11. Funzione revealBid, passare come parametro il nonce utilizzato per generare l'hash e inserire come value della transazione il value inserito nella creazione dell'hash
12. Aspettare il termine della fase 3
13. Funzione finalize

4 Analisi del gas delle funzioni

4.1 Dutch

	Exec	Tx
StartAuction	44522	65794
Bid	5233	90049
Payout	55419	46691
Closecontract	5233	13253
Safetransferfrom	45241	38895
actualPrice	8771	31771

4.2 Vickrey

	Exec	Tx
Bidcommitment	20896	44582
Revealbid	61268	83014
Finalize	54721	45669
Withdrawal	10896	30151