Dipartimento di Informatica
Corso di Laurea Magistrale in Informatica

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

Studente:
Luca Corbucci

Peer to Peer Systems and Blockchains
Anno Accademico 2018/2019

Indice

1 Asta inglese

1.1 Codice del contratto

```
pragma solidity ^0.5.1;
import "github.com/OpenZeppelin/zeppelin-solidity/contracts/math/SafeMath.sol";

contract englishAuction {
    using SafeMath for uint;

    // Offerta minima che deve essere fatta
    uint public reservePrice;

    uint public minIcrement;
    // Prezzo per l'acquisto diretto senza asta
    uint public buyoutPrice;

    // Numero di blocchi che devono passare prima di terminare l'asta
    uint public minBlocks;

    // Indirizzo che ha fatto l'offerta pi  alta e offerta pi  alta ricevuta
    uint public highestBid;
    address payable public highestBidder;

    // Indirizzo che ha comprato direttamente senza asta
    address public buyer;
    // Creatore dell'asta
    address payable auctioneer;

    // Booleano usato per capire se l'asta    terminata o no
    bool ended = false;
    // booleano per capire se    stato fatto l'acquisto diretto senza asta
    bool buyoutEnded = false;

    // usato per capire se l'asta    iniziata
    bool started = false;

    // Blocco in cui    stata eseguita l'ultima offerta
    uint startingBlock;

    // Due eventi, uno per dire che ho aumentato l'offerta e uno
    // per indicare che l'asta    terminata
    event HighestBidIncreased(address bidder, uint amount);
    event AuctionEnded(address winner, uint amount);

    // titolo dell'asta e url di una foto del prodotto che voglio vendere
    string title;
    string URL;

    /*
        Parametri:
        - _title: nome del prodotto che vogliamo mettere in vendita
        - _URL: url di una immagine del prodotto che vogliamo vendere
        - _reservePrice: prezzo base dell'asta
        - _minIcrement: incremento minimo che deve esserci tra un'offerta e l'altra
        - _buyoutPrice: prezzo per l'acquisto diretto
```

```
53          - _minBlocks: numero minimo di blocchi che devono passare prima di decretare
                il vincitore dell'asta
54      */
55      constructor(string memory _title, string memory _URL, uint _reservePrice, uint
            _minIcrement, uint _buyoutPrice, uint _minBlocks) public payable{
56          require(_reservePrice > 0);
57          require(_buyoutPrice > 0);
58          reservePrice = _reservePrice;
59          minIcrement = _minIcrement;
60          buyoutPrice = _buyoutPrice * 1 wei;
61          minBlocks = _minBlocks;
62          auctioneer = msg.sender;
63          title = _title;
64          URL = _URL;
65      }
66
67      // Controllo che l'asta non sia terminata
68      modifier only_notEnded(){
69          require(!ended, "Asta terminata"); _;
70      }
71
72      // Controllo che sia ancora possibile acquistare il prodotto
73      // direttamente e senza fare l'asta
74      modifier buyout_available(){
75          require(buyoutEnded == false, "Non acquistabile direttamente"); _;
76      }
77
78      // Controllo che tra l'ultima offerta e quella che provo a fare
79      // non siano passati troppi blocchi
80      modifier bidAvailable(){
81          require(startingBlock < startingBlock.add(minBlocks), "Bid Phase terminata");
                _;
82      }
83
84      // Controlla che chi fa l'offerta abbia a disposizione un balance maggiore di
85      // quanto offre
86      modifier balanceAvailable(uint price){
87          require(msg.sender.balance >= price, "Balance non sufficiente"); _;
88      }
89
90      /*
91          Controlla che il chiamante della funzione sia il creatore del contratto
92          oppure quello che ha vinto l'asta.
93      */
94      modifier onlyAuthorized() {
95           require(msg.sender == auctioneer || msg.sender == highestBidder, "Non
                autorizzato"); _;
96      }
97
98      modifier only_when_FinalizePhase(){
99          require(startingBlock.add(minBlocks) < uint(block.number), "Asta non
                terminata"); _;
100     }
101
102     // Controlla che la funzione sia stata chiamata dal creatore dell'asta
103     modifier only_auctioneer(){
104         require(msg.sender == auctioneer, "Non sei l'auctioneer"); _;
```

```
105     }
106
107      // Controllo che l'asta sia stata avviata
108     modifier only_when_started(){
109         require(started, "Asta non avviata"); _;
110     }
111
112
113     /*
114         Inizio dell'asta, solamente il creatore dell'asta pu  avviarla
115     */
116     function openAuction() public only_auctioneer(){
117         require(started == false);
118         started = true;
119     }
120
121
122     /*
123         Funzione che permette di acquistare direttamente il prodotto senza
124         dover svolgere l'asta pagando il buyoutPrice.
125         Pu  essere chiamata solamente se:
126         - l'asta non    terminata
127         -    ancora disponibile la possibilit  di acquistare direttamente
128         - ho un balance sufficiente per acquistare
129         - Il valore di buyout    uguale a quello che invio
130
131         Alla fine il valore che invio viene trasferito al creatore dell'asta.
132     */
133     function acquistoDiretto() public payable only_notEnded() buyout_available()
        balanceAvailable(buyoutPrice) only_when_started(){
134
135         require(msg.value == buyoutPrice);
136
137         emit AuctionEnded(msg.sender, msg.value);
138         ended = true;
139         buyer = msg.sender;
140         auctioneer.transfer(buyoutPrice);
141
142     }
143
144     /*
145         Funzione che permette di eseguire un'offerta.
146         Viene eseguita se:
147         - l'asta non    gi  terminata
148         - chi invia l'offerta ha un balance maggiore di quello che offre
149         - Se tra il bid precedente e il successivo non sono passati troppi blocchi
150     */
151     function bid() public payable only_notEnded() balanceAvailable(msg.value)
        only_when_started(){
152
153         // Prima offerta che arriva, in questo caso posso fare un'offerta pari al
            minimo
154         // e non devo controllare che sia terminata la fase di bid
155         if(buyoutEnded == false){
156
157             require(msg.value >= reservePrice);
158             buyoutEnded = true;
```

```
159         }
160         else{
161             // offerte successive alla prima, in questo caso devo fare un'offerta
                    maggiore della precedente pi  il minimo incremento
162             // una volta ricevuta l'offerta devo anche restituire i soldi al
                    precedente bidder
163             require(block.number <= startingBlock.add(minBlocks));
164             require(msg.value >= highestBid.add(minIcrement));
165         }
166 
167         uint value = highestBid;
168         address payable receiver = highestBidder;
169 
170         highestBid = msg.value;
171         highestBidder = msg.sender;
172         startingBlock = uint(block.number);
173         emit HighestBidIncreased(highestBidder, highestBid);
174 
175         // Serve l'if perch  alla prima offerta che ricevo non devo restituire soldi
176         if(value != 0 && receiver != address(0)){
177             receiver.transfer(value);
178         }
179     }
180 
181     /*
182         Funzione che pu  essere chiamata solamente se:
183         - l'asta non    gi  terminata
184         - solo dal vincitore o da chi ha creato l'asta
185         - solamente dopo che non posso fare altre offerte
186         Qua trasferisco i soldi che sono stati offerti al creatore del contratto e
187         termino l'asta emettendo anche un evento.
188     */
189     function finalize() external payable only_notEnded() onlyAuthorized()
            only_when_FinalizePhase() only_when_started(){
190         ended = true;
191         emit AuctionEnded(highestBidder, highestBid);
192         auctioneer.transfer(highestBid);
193     }
194 
195     /*
196         Funzione che restituisce l'address che ha fatto l'offerta pi  alta
197     */
198     function getHighestBidder() public view returns(address){
199         return highestBidder;
200     }
201 
202     /*
203         Funzione che restituisce l'offerta pi  alta
204     */
205     function getHighestBid() public view returns(uint){
206         return highestBid;
207     }
208 
209     /*
210         Funzione che restituisce il prezzo dell'acquisto diretto
211     */
212     function getBuyoutPrice() public view returns(uint){
```

```
213         return buyoutPrice;
214     }
215
216     /*
217         Funzione che restituisce l'incremento minimo da fare rispetto all'offerta
                pi   alta
218     */
219     function getMinIncrement() public view returns(uint){
220         return minIcrement;
221     }
222
223 }
```

1.2 Descrizione funzioni

All'interno del contratto dell'asta inglese sono state definite le seguenti funzioni:

- **openAuction()**: Questa funzione è stata implementata per sostituire il "Grace Period" in modo da semplificare i test del contratto all'interno di Remix. Può essere chiamata solamente dal creatore del contratto e viene utilizzata per avviare l'asta.
- **acquistoDiretto()**: Questa funzione permette l'acquisto diretto del prodotto senza dover effettuare alcuna offerta. L'interessato invia la quantità di denaro richiesta dal creatore dell'asta. L'esecuzione di questa funzione comporta la fine dell'asta e l'impossibilità di chiamare altre funzioni. Prima di eseguire questa funzione si controlla che sia ancora possibile eseguire un acquisto diretto, che l'asta sia stata avviata e che l'utente intenzionato ad acquistare il prodotto abbia effettivamente la quantità di denaro necessaria. Ho inserito quest'ultimo controllo perchè su Remix, nel momento in cui un address offre più di quanto posseduto, vengono aggiunti all'address altri soldi per coprire la spesa.
- **bid()**: Questa funzione implementa l'invio di un'offerta da parte di un utente che vuole acquistare il prodotto messo in vendita. La funzione può essere eseguita solamente se l'asta è stata avviata e non è terminata. Anche in questo caso, per lo stesso motivo spiegato in precedenza, controllo che il balance dell'account che fa l'offerta sia sufficiente. Prima di eseguire la funzione controllo anche se quella che stiamo cercando di fare è la prima offerta o una delle successive. Se è la prima offerta allora registro l'offerta come "più alta" ed emetto un evento. Se in precedenza ci sono già state altre offerte invece devo per prima cosa controllare che quanto offerto sia maggiore della somma tra l'attuale offerta più alta e l'incremento, poi verifico anche che non sia terminata la bidPhase. Se supero l'offerta più alta modifico la variabile HighestBid e devo anche restituire al precedente "HighestBidder" il denaro che aveva offerto.
- **finalize()**: Questa funzione viene chiamata per ultima dal creatore del contratto o dal vincitore dell'asta. Possiamo chiamarla solamente dopo che è terminata la fase di invio delle offerte. All'interno di questa funzione emettiamo un evento che indica che l'asta è terminata e trasferiamo il denaro pagato dal vincitore al creatore dell'asta.

1.3 Eventi definiti

All'interno del contratto ho definito due eventi:

- HighestBidIncreased: viene emesso all'interno della funzione bid() quando arriva una nuova offerta che supera la precedente.

- AuctionEnded: viene emesso quando termina l'asta, lo possiamo richiamare sia nel momento in cui avviene un acquisto diretto sia quando viene chiamata la funzione finalize().

1.4 Analisi del gas consumato

Per il calcolo del gas utilizzato delle funzioni ho sfruttato in alcuni casi le indicazioni di Remix. Per alcune funzioni però Remix non è stato in grado di fornire una stima e indicava un costo "infinito". In questi casi ho calcolato la stima prendendo in considerazione i costi indicati nello Yellow Paper [1] di Ethereum. In particolare le operazioni più frequenti sono:

- Confronti: hanno un costo di 3 gas;
- Assegnamenti "in memoria": hanno un costo di 20.000 gas;
- Trasferimenti di denaro: hanno un costo di 2100 gas;
- Emissione di eventi: il costo dipende dal numero dei parametri che vengono passati alla funzione che emette l'evento.

Le funzioni definite hanno i seguenti costi, "Transaction Cost" ed "Execution Cost" sono stati calcolati da remix dopo le chiamate alle funzioni.

Funzione	**Stima dei costi**	**Transaction Cost**	**Execution Cost**
openAuction()	21.008	27.319	6047
acquistoDiretto()	43.615	58.226	36.954
bid()	82.118	90.939	69.667
finalize()	23.800	38.184	16.912

- **openAuction()**: per questa funzione remix indica che il gas necessario per l'esecuzione è 21.008. All'interno della funzione infatti viene eseguito un assegnamento, che ha un costo di 20.000 gas e poi vengono fatti 2 confronti che hanno un costo di 6 gas.
- **acquistoDiretto()**: se considero tutta la funzione al completo remix indica un costo di esecuzione infinito. Considerando le singole operazioni possiamo stimare il gas necessario all'esecuzione di questa funzione, sono presenti due assegnamenti che costano 40.000 gas, 5 confronti e un trasferimento di denaro che costa 1500 gas.
  Stima complessiva: 43.615
- **bid()**: in questa funzione abbiamo 4 assegnamenti (uno viene effettuato solamente alla prima chiamata della funzione), il costo complessivo è di 80.000 gas a cui vanno aggiunti i 6 confronti che comportano un costo di 18 gas e il trasferimento finale.
  Stima complessiva: 82.118
- **finalize()**: la funzione finalize contiene un assegnamento che quindi ha un costo di 20.000, una emissione dell'evento che costa 1500 gas e il trasferimento finale di soldi.
  Stima complessiva: 23.800

1.5 Test

Per provare il funzionamento del contratto dell'asta inglese possono essere eseguite le seguenti operazioni:

- Deploy del contratto fornendo i seguenti parametri:
  - Titolo dell'oggetto da mettere in vendita;
  - URL di una immagine dell'oggetto da mettere in vendita;
  - Reserve Price espresso in Wei;
  - MinIncrement espresso in Wei;
  - buyoutPrice espresso in Wei;
  - minBlocks espresso in numero di blocchi
- Lo stesso address che ha creato il contratto deve chiamare la funzione openAuction
- Un qualsiasi address può chiamare la funzione "acquistoDiretto" fissando come value da inviare al contratto il valore che è stato utilizzato come buyoutPrice. In questo caso l'asta termina
- Un qualsiasi address può chiamare la funzione bid proponendo un prezzo per il prodotto. Il prezzo deve essere maggiore dell'offerta più alta di almeno minIncrement.
- Quando saranno passati minBlocks dall'ultima offerta il creatore del contratto o il vincitore può chiamare la funzione "finalize".

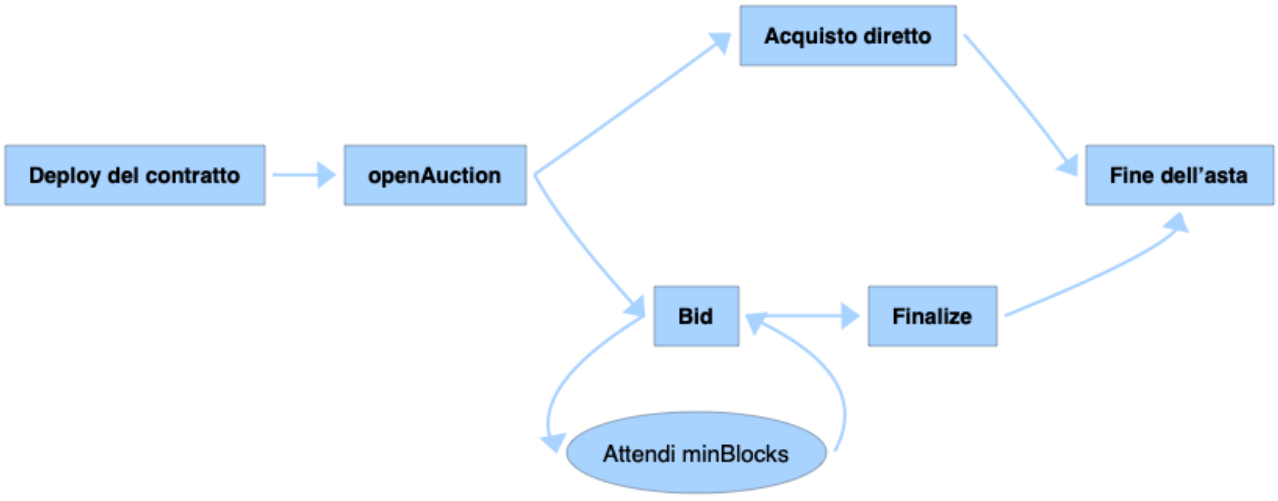

Figura 1: Ordine in cui chiamare le varie funzioni del contratto

2 Asta Vickrey

2.1 Codice del contratto

```
224 pragma solidity ^0.5.1;
225 import "github.com/OpenZeppelin/zeppelin-solidity/contracts/math/SafeMath.sol";
226
227
228 contract Vickrey {
229     using SafeMath for uint;
230
231     // Prezzo minimo dell'asta
232     uint reservePrice;
233     // Deposito minimo
234     uint public bidDeposit;
235     // Prezzo che viene restituito in caso di ritiro
236     uint refundPrice;
237     // Creatore dell'asta
238     address payable auctioneer;
239
240     // blocco in cui inizia l'asta e durata della bid Phase
241     uint bidTimeStart;
242     uint bidTime;
243
244     // tempo (in blocchi) della withdrawal phase
245     uint withdrawalTime;
246
247     // tempo (in blocchi) della fase di apertura delle offerte
248     uint bidOpeningTime;
249
250     // variabili usate per memorizzare l'offerta pi  alta e chi l'ha presentata
251     // stesso discorso con la seconda offerta pi  alta.
252     uint public highestBid;
253     uint public secondHighestBid;
254     address payable highestBidder;
255     address payable secondhighestBidder;
256
257     // Memorizzo la coppia (indirizzo, hash offerta)
258     mapping (address => bytes32) bids;
259
260     // Un booleano per capire se l'asta    iniziata e uno per capire se    finita
261     bool started = false;
262     bool ended = false;
263
264     // Nome del prodotto che viene messo in vendita e URL di una foto
265     string title;
266     string URL;
267
268     /*
269         Parametri:
270         - _title = nome del prodotto che viene messo in vendita con l'asta
271         - _URL = url di una foto del prodotto
272         - _reservePrice = prezzo minimo che deve essere pagato
273         - _bidTime = tempo (misurato in numero di blocchi) della fase in cui
                possibile
274         fare le offerte
```

```
275         - _withdrawalTime = tempo (misurato in numero di blocchi) della fase in cui
                possibile
276         ritirare le offerte
277         - _bidOpeningTime = tempo (misurato in numero di blocchi) della fase in cui
                possibile
278         aprire le "buste" con le offerte
279         - _bidDeposit = deposito che deve essere lasciato da chi fa un'offerta
280     */
281     constructor(string memory _title, string memory _URL, uint _reservePrice, uint
            _bidTime, uint _withdrawalTime, uint _bidOpeningTime, uint _bidDeposit) public
             payable{
282         require(_reservePrice > 0, "reserve > 0");
283         reservePrice = _reservePrice;
284         auctioneer = msg.sender;
285         bidTime = _bidTime;
286         withdrawalTime = _withdrawalTime;
287         bidOpeningTime = _bidOpeningTime;
288         bidDeposit = _bidDeposit;
289         // a refundPrice assegno la met  del _bidDeposit perch  quando ritiro l'
                offferta
290         // dobbiamo restituire la met  di quanto    stato lasciato come deposito.
291         refundPrice = _bidDeposit.div(2);
292         secondHighestBid = reservePrice;
293         title = _title;
294         URL = _URL;
295     }
296
297     // Controlla che l'asta sia stata avviata e che la fase di invio delle offerte
            non
298     // sia ancora completata
299     modifier only_when_BidPhase(){
300         require(started == true, "Asta non iniziata");
301         require(block.number <= bidTimeStart.add(bidTime), "Bid Phase ended"); _;
302     }
303
304     // Controlla che la fase di invio delle offerte sia terminata, che siamo nella
            fase Withdrawal
305     // e che l'asta    stata avviata
306     modifier only_when_WithdrawalPhase(){
307         require(started == true, "Asta non iniziata");
308         require(block.number > bidTimeStart.add(bidTime), "Bid Phase non terminata");
309         require(block.number <= (bidTimeStart.add(bidTime)).add(withdrawalTime), "
                Withdrawal Phase terminata"); _;
310     }
311
312     // Controlla che l'asta sia iniziata, che la fase di ritiro delle offerte non sia
             terminata
313     // e che la fase di apertura delle buste sia iniziata
314     modifier only_when_openBid(){
315         require(started == true, "Asta non iniziata");
316         require(block.number > (bidTimeStart.add(bidTime)).add(withdrawalTime), "
                Withdrawal Phase non terminata");
317         require(block.number <= ((bidTimeStart.add(bidTime)).add(withdrawalTime)).add
                (bidOpeningTime), "Open Bid Phase terminata"); _;
318     }
319
320      // Controlla che l'asta sia stata avviata
```

```
321     modifier only_when_auctionStarted(){
322         require(started == true, "asta non iniziata"); _;
323     }
324
325     // Controlla che la funzione sia stata chiamata dal creatore dell'asta
326     modifier only_auctioneer(){
327         require(msg.sender == auctioneer, "non sei l'auctioneer"); _;
328     }
329
330     // Controlla che l'asta sia iniziata, che non sia terminata e che la fase di
            apertura delle buste sia iniziata
331     modifier only_when_openBidPhaseEnded(){
332         require(started == true, "Asta non iniziata");
333         require(ended == false, "Asta terminata");
334         require(block.number > ((bidTimeStart.add(bidTime)).add(withdrawalTime)).add(
                bidOpeningTime), "Open Bid Phase non terminata"); _;
335     }
336
337     // Controlla che chi fa l'offerta abbia a disposizione un balance maggiore di
338     // quanto offre
339     modifier balanceAvailable(uint price){
340         require(msg.sender.balance >= price, "Balance non sufficiente"); _;
341     }
342
343     // Eventi che vengono emessi nelle funzioni, uno lo uso per segnalare che l'
            offerta massima
344     //    stata aumentata e una per segnalare che l'asta    terminata
345     event HighestBidIncreased(address bidder, address secondBidder);
346     event AuctionEnded(address winner, uint amount);
347
348     /*
349         Inizio dell'asta, solamente il creatore dell'asta pu  avviarla
350     */
351     function openAuction() public only_auctioneer(){
352         require(started == false);
353         started = true;
354         bidTimeStart = block.number;
355     }
356
357     /*
358         Aggiunta di una nuova offerta, possiamo aggiungere l'offerta solamente dopo
359         che l'asta    stata avviata e solamente prima che inizi la fase di ritiro
                delle offerte.
360         Prima di far aggiungere una nuova offerta devo anche contorllare che il
                deposito
361         che viene inviato sia corretto.
362         Un utente pu  fare pi  di una offerta (modificando la precedente) lasciando
                 solamente
363         una volta il deposito.
364
365         Parametri:
366         - Hash dell'offerta che vogliamo inviare (Nonce + valore denaro da inviare)
367     */
368     function addBid(bytes32 bid) public payable only_when_BidPhase() balanceAvailable
            (bidDeposit){
369         require(msg.value == bidDeposit, "bidDeposit errato");
370
```

```
371         // Se ho gi  fatto un'offerta restituisco il value inviato con questa
                transazione
372         if(bids[msg.sender] == 0){
373             bids[msg.sender] = bid;
374         }
375         else{
376             bids[msg.sender] = bid;
377             msg.sender.transfer(msg.value);
378         }
379     }
380
381
382     /*
383         Ritiro dell'offerta, viene restituito met  del bidDeposit.
384         Questa funzione la posso chiamare solamente prima che finisca la fase di
                withdrawal e dopo
385         che    finita quella di invio delle offerte. Inoltre si controlla che l'
                utente che
386         chiede il rimborso abbia effettivamente inviato un'offerta.
387
388         Parametri:
389         - Hash dell'offerta che    stata precedentemente inviata
390     */
391     function withdrawal(bytes32 bid) public only_when_WithdrawalPhase(){
392         require(bid == bids[msg.sender], "Non puoi ritirare");
393
394         delete bids[msg.sender];
395         msg.sender.transfer(refundPrice);
396     }
397
398
399     /*
400         Apertura delle buste con le offerte. Questa funzione la posso chiamare
401         solamente dopo che sono finite le precedenti due fasi e prima che termini
                questa.
402         Devo controllare che l'hash che ho inviato quando ho fatto l'offerta sia lo
                stesso che
403         calcolo qua usando il nonce e il valore che invio.
404         Devo controllare anche che la mia offerta sia maggiore del reservePrice.
405         Se ho fatto un'offerta minore del reservePrice perdo anche il deposito.
406         Parametri:
407         - Nonce usato quando abbiamo calcolato l'hash
408     */
409     function openBid(uint _nonce) public payable only_when_openBid()
            balanceAvailable(msg.value){
410         require(keccak256(abi.encode(_nonce, msg.value)) == bids[msg.sender], "
                Impossibile aprire");
411
412         // Indirizzo a cui va inviato il rimborso
413         address payable toRefund;
414         // rimborso da inviare
415         uint val;
416
417
418         // Caso in cui supero l'highest Bid, devo vedere anche se    la prima offerta
419         // che lo supera o no, altrimenti rischio di azzerare il secondhighestBidder
420         if(msg.value > highestBid) {
```

```
421             // se    la prima offerta che supera l'highest Bidder non devo
422             // rimborsare nessuno perch  il secondhighestBidder    il valore del
423             // reservePrice. Faccio questo controllo per evitare che Highest = X
424             // e secondHighest = 0
425             if(highestBid == 0){
426                 highestBid = msg.value;
427                 highestBidder = msg.sender;
428             }
429             // se invece sono offerte successive allora devo fare un rimborso
430             else{
431                 toRefund = secondhighestBidder;
432                 val = secondHighestBid.add(bidDeposit);
433                 secondhighestBidder = highestBidder;
434                 secondHighestBid = highestBid;
435                 highestBid = msg.value;
436                 highestBidder = msg.sender;
437             }
438
439             emit HighestBidIncreased(msg.sender, secondhighestBidder);
440         }
441         // Caso in cui faccio un'offerta che supera solamente la seconda, in questo
                caso
442         // il precedente bidder viene rimborsato dell'offerta e del deposito.
443         else if(msg.value > secondHighestBid){
444             toRefund = secondhighestBidder;
445             val = secondHighestBid.add(bidDeposit);
446             secondhighestBidder = msg.sender;
447             secondHighestBid = msg.value;
448         }
449         /*
450             La mia offerta non supera la prima o la seconda quindi dovr  essere
451             rimborsato sia dell'offerta che del deposito. Qua ci finisco anche se l'
                    offerta
452             non supera il reservePrice perch  inizialmente il reservePrice
                    fissato come
453             seconda offerta pi  alta, quindi se non supero la seconda non supero
                    nemmeno il reservePrice,
454             questo serve quando ad esempio abbiamo tutte le offerte sotto al
                    reservePrice e non vogliamo
455             che la vendita avvenga comunque
456         */
457         else{
458             toRefund = msg.sender;
459             val = msg.value.add(bidDeposit);
460         }
461
462         // Eseguo il rimborso
463         if(toRefund != address(0)){
464             toRefund.transfer(val);
465         }
466
467     }
468
469
470
471     /*
472         Funzione che chiude l'asta, viene chiamata solamente da chi ha creato
```

```
473         l'asta o da chi ha vinto, solamente quando le fasi precedenti sono terminate.
                Serve per segnare l'asta come chiusa e anche per
474         fare il rimborso a chi ha vinto.
475     */
476     function finalize() public payable only_when_openBidPhaseEnded() {
477         require(msg.sender == highestBidder || msg.sender == auctioneer, "Non hai i
                permessi");
478
479         emit AuctionEnded(highestBidder, secondHighestBid);
480         ended = true;
481
482         // Controlliamo che sia stata eseguita un'offerta valida.
483         if(highestBidder != address(0)){
484             highestBidder.transfer((highestBid.sub(secondHighestBid)).add(bidDeposit)
                    );
485         }
486
487         // Restituisco quanto offerto pi  il bidDeposit al secondo classificato
488         if(secondhighestBidder != address(0)){
489             secondhighestBidder.transfer(secondHighestBid.add(bidDeposit));
490         }
491
492
493         // trasferisco il balance del contratto al creatore dell'asta
494         if(address(this).balance != 0){
495             auctioneer.transfer(address(this).balance);
496         }
497     }
498
499
500     /*
501         Funzione che restituisce l'address che ha fatto l'offerta pi  alta
502     */
503     function getHighestBidder() public view returns(address){
504         return highestBidder;
505     }
506
507     /*
508         Funzione che restituisce l'offerta pi  alta
509     */
510     function getHighestBid() public view returns(uint){
511         return highestBid;
512     }
513
514     /*
515         Funzione che restituisce il bidDeposit
516     */
517     function getDeposit() public view returns(uint){
518         return bidDeposit;
519     }
520
521 }
```

2.2 Descrizione funzioni

- **openAuction()**: Questa funzione è stata implementata per sostituire il "Grace Period" in modo da semplificare i test del contratto all'interno di Remix. Può essere chiamata solamente dal creatore del contratto e viene utilizzata per avviare l'asta.
- **addBid()**: Questa funzione viene utilizzata per aggiungere una nuova offerta. Prende come parametro l'hash che rappresenta l'offerta (hash(nonce+offerta)). La funzione può essere eseguita solamente quando l'asta è stata avviata e quando abbiamo una quantità di balance nell'account che è maggiore del deposit minimo. È stata implementata la possibilità di eseguire più di una offerta per ogni utente, quando viene inviata la nuova offerta quella già presente viene sovrascritta. In questo caso il deposito viene trattenuto solamente alla prima offerta e non alle successive.
- **withdrawal()**: Questa funzione viene utilizzata per ritirare l'offerta che è stata presentata nella "Bid Phase". La funzione può essere chiamata solamente se la "Bid Phase" è terminata e solo da chi ha fatto un'offerta in precedenza.
- **openBid()**: Questa funzione serve per implementare l'apertura delle "buste" contenenti le offerte. Viene eseguita solamente quando abbiamo terminato la fase precedente e se il chiamante ha in precedenza inviato un'offerta. Prima di eseguire la funzione si controlla anche che l'offerta sia minore del balance a disposizione di quell'address. Eseguendo questa funzione abbiamo alcuni casi da considerare:
  - Se la busta contiene un'offerta maggiore di "Highest Bid" e non abbiamo ancora aperto altre buste dobbiamo semplicemente memorizzare l'offerta più alta e chi l'ha inviata.
  - Se la busta contiene l'offerta più alta e ne abbiamo già aperte altre in precedenza allora l'Highest Bid attuale diventa il second Highest Bid e l'address che fino a quel momento aveva proposto la seconda cifra più alta viene risarcito di quanto offerto più il deposito.
  - Se la busta contiene un'offerta più alta dell'offerta che attualmente è seconda allora assegno il nuovo valore e risarcisco l'address che aveva proposto la seconda offerta più alta.
  - In tutti gli altri casi risarcisco chi ha proposto l'offerta.
- **finalize()**: Questa funzione può essere chiamata solamente quando la fase precedente è terminata, soltanto da chi ha creato il contratto o da chi ha fatto l'offerta più alta. In questa fase dobbiamo risarcire chi ha vinto l'asta e chi è arrivato secondo. Chi è arrivato secondo deve essere risarcito perchè non lo rimborso nel momento in cui apro le buste, avevo provato a fare tutto quanto nella funzione di apertura delle buste ma con alcuni input particolari si verificavano dei comportamenti non attesi del contratto, quindi l'ho spostato nella funzione finalize.

2.3 Eventi definiti

All'interno di questo contratto ho definito i seguenti eventi:

- HighestBidIncreased: viene emesso all'interno della funzione openBid() quando una nuova busta che è stata aperta contiene un'offerta che supera quella che fino a quel momento era la più alta.

- AuctionEnded: viene emesso all'interno della funzione finalize() per segnalare che l'asta è terminata.

2.4 Analisi del gas consumato

Per il calcolo del gas utilizzato delle funzioni ho sfruttato in alcuni casi le indicazioni di Remix. Per alcune funzioni però Remix non è stato in grado di fornire una stima e indicava un costo "infinito". In questi casi ho calcolato la stima prendendo in considerazione i costi indicati nello Yellow Paper [1] di Ethereum.

Funzione	**Stima dei costi**	**Transaction Cost**	**Execution Cost**
openAuction()	40.876	62.254	40.982
addBid()	42.112	45.940	22.492
withdrawal()	7106	23.566	15.118
openBid()	162.121	69.310	45.862
finalize()	27.818	48.342	27.070

- **openAuction()**: anche per questa funzione Remix è stato in grado di fornire una stima del costo indicando un valore pari a 40.876. All'interno della funzione infatti abbiamo due assegnamenti che costano in tutto 40.000 gas e due confronti che costano 6 gas.
- **addBid()**: per questa funzione abbiamo 4 confronti tra interi che comportano un costo di 12 gas e due assegnamenti che hanno un costo di 40.000 gas. Alla fine della funzione viene eseguito anche un trasferimento di denaro.
  Stima complessiva: 42.112 gas
- **withdrawal()**: in questo caso abbiamo due confronti, una cancellazione dalla map che costa 5000 gas perchè fissiamo a 0 un certo valore e il trasferimento finale di denaro.
  Stima complessiva: 7106 gas
- **openBid()**: in questa funzione sono presenti 7 confronti che quindi costano 21 gas. Abbiamo inoltre 8 assegnamenti, in alcuni casi questi vengono svolti solamente in determinate condizione, complessivamente questi hanno un costo di 160.000 gas Nella funzione viene anche emesso un evento ed effettuato un trasferimento di denaro.
  Stima complessiva: 162.121 gas
- **finalize()**: nell'ultima funzione del contratto abbiamo 6 confronti che costano 18 gas e un assegnamento che ne costa 20.000. Abbiamo poi tre trasferimenti (2100 gas ciascuno) e una emissione di un evento (approssimativamente 1500 gas).
  Stima complessiva: 27.818 gas

2.5 Test

Per provare il funzionamento del contratto dell'asta Vickrey possono essere eseguite le seguenti operazioni:

- Deploy del contratto fornendo i seguenti parametri:
  - Titolo dell'oggetto da mettere in vendita;

  - URL di una immagine dell'oggetto da mettere in vendita;
  - Reserve Price espresso in Wei;
  - BidTime espresso in numero di blocchi;
  - WithdrawalTime espresso in numero di blocchi;
  - bidOpeningTime espresso in numero di blocchi;
  - bidDeposit espresso in Wei;
- Lo stesso address che ha creato il contratto deve chiamare la funzione openAuction() per avviare l'asta
- Gli address possono inviare le loro offerte chiamando la funzione addBid. Per provare il contratto possono essere inviate le seguenti offerte, per ognuna va anche impostato il corretto valore del bidDeposit:
  * Hash: 0x750a8608a49d3eb68fd58fd0b016664ea1fba267dcfa93a16e5dcd7ce03d6d15
  * Hash: 0xc614b9b9dc59665b769dbbaa6073199c950aff0f8d4bdbacd00a1296adf74efe
  * Hash: 0x982d466923086be4c16c13f12c9c26cdb1ad8a3785363639599fa6964a29fdf4
  * Hash: 0x79353e13acc34fdf725fbbc9f3bef86db7c5212dae183901d449340b24937485
- Gli utenti possono chiamare la funzione withdrawal e indicare come parametro l'hash che hanno precedentemente inviato quando hanno presentato l'offerta
- Gli utenti chiamano la funzione openBid indicando il nonce usato per il calcolo dell'hash con l'offerta che sono intenzionati a fare.
  * Msg.value: 1
  * Nonce: 0x55d643f37bec66db03fd36ba60d8d685264c7e050964eef18c046227ae1d9bee
  * Msg.value: 2
  * Nonce: 0xb2d799532c83f44fb8dab17c735f8b04b0311cd69a459dc458591729b4d5e5d0
  * Msg.value: 3
  * Nonce: 0x982d466923086be4c16c13f12c9c26cdb1ad8a3785363639599fa6964a29fdf4
  * Msg.value: 4
  * Nonce: f8b9919d8068374bfd5d664cf5fec32b84ffb4f1962ba04406ae0e042ea9c961

  Testando il contratto ho provato ad aprire le buste in vari ordini in modo da provare le possibili combinazioni di risultati cercando di evitare dei comportamenti inattesi del contratto.
- Il vincitore o il creatore dell'asta chiama la funzione finalize per mettere fine all'asta.

Figura 2: Sequenza delle chiamate delle funzioni del contratto Vickrey

Riferimenti bibliografici

[1] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger.