CMPE 483 Blockchain Programming HW1

Volkan Öztürk 2019400033
Deniz Büyüktaş 2022700018
Çağrı Gülbeycan 2020400369

June 3, 2023

Contents

1 Introduction

We aim to construct a decentralized autonomous lottery using Solidity Smart Contracts in this project. The project consists of a Solidity Smart Contract file named "lottery.sol" and test file written in JavaScript. Thus, the documentation mainly consists of the explanation of the Smart Contract Code, the explanation of our testing methodology, and our test results.

2 Solidty Smart Contract

2.1 Functions in The Project Description

2.1.1 function depositEther() public payable

It is used to add the value sent by the message to the account of the user.

2.1.2 function withdrawEther(uint amnt) public

Users can withdraw the amount stated as "amnt" using this function. This amount denotes the amount of Wei the user wants to withdraw. There must be at least the desired amount of Wei in the balance of the user.

2.1.3 function buyTicket(bytes32 hash_rnd_number, uint ticket_type) public returns(uint ticket_no)) public

Users can buy tickets using this function. hash_rnd_number denotes the 32 bytes-hashed version of the random number that the user wants to purchase to participate in the current lottery round. ticket_type denotes whether the ticket is full ticket, half ticket, or quarter ticket. If the ticket_type is 1, the user wants to buy a full ticket. If the ticket_type is 2, the user wants to buy a half ticket. If the ticket_type is 4, the user wants to buy a quarter ticket. The balance of the user must be greater than or equal to the cost of the ticket that this user wants to buy. Besides, the hashed random number must not have been sold before. Every ticket has a unique ticket number even if a ticket is refunded. The function returns this unique ticket number.

2.1.4 collectTicketRefund(uint ticket_no) public

The users can refund their tickets using this function. However, the lottery during which the corresponding ticket denoted in the ticket_no variable must be in its "purchase" state.

2.1.5 revealRndNumber(uint ticket_no, uint rnd_number) public

The users can reveal the random numbers of their tickets. However, the random number they provide and the variable "rnd_number" given while calling the function must be the same. Furthermore, the lottery for which the ticket is bought must be in its "reveal" stage.

2.1.6 function checkIfTicketWon(uint lottery_no, uint ticket_number) public view returns (uint amount)

It receives two parameters: ticket_no and lotter_no. It checks whether the given ticket number has won any orize from the corresponding loettery number which is also given. It returns the gained amount if any prize was won otherwise 0.

2.1.7 function getLastOwnedTicket(uint lottery_no) public view returns (uint ticket_no, uint status)

It returns the last purchased ticket of the user and its status. It accepts the lottery number and checks whether the sender has any ticket for the given lottery. Status 0 means the ticket is refunded or not revealed yet. Status 1 means the ticket is revealed but its prize is not collected. Status 2 means the ticket is revealed and the prize of the ticket is collected.

2.1.8 function collectTicketPrize(uint lottery_no, uint ticket_no) public

A winning user may withdraw their reward money from the chosen lottery round using this method. It requires two inputs: ticket_no, the winning ticket number that the user wishes to claim their prize for, and lottery_no, the lottery round number. The reward sum is sent to the sender's account and the won field on the ticket is set to false if the given ticket is held by the sender and has won in the specified lottery round.

2.1.9 function getIthWinningTicket(uint i, uint lottery_no) public view returns (uint ticket_no, uint amount)

The i-th winning ticket in the specified lottery round is identified by its ticket number, which is returned by this function. It requires two parameters: lottery_no, the lottery round number, and i, the index of the winning ticket to be retrieved. The function first verifies the validity of the supplied lottery round and index. If yes, it pulls the winning ticket information from the chosen lottery round's winningTickets mapping and divides the total reward money by the number of winning tickets to determine the prize amount. The function then produces a tuple that includes the reward amount and ticket number.

2.1.10 function getLotteryNos(uint unixtimeinweek) public pure returns (uint lottery_no1, uint lottery_no2)

The given Unix timestamp is used to determine the start and finish lottery round numbers for a two-week period. Unixtimeinweek, the Unix timestamp for any day within the two-week period, is the only parameter required. By reducing the timestamp to the nearest multiple of two weeks, the function first determines the beginning and ending times of the two-week period. The start and finish times are then divided by two weeks, and the start lottery round number is increased by 1, to get the start and end lottery round numbers.

2.1.11 getTotalLotteryMoneyCollected(uint lottery_no) public view returns (uint amount)

The total amount of money raised for the chosen lottery round is returned by this function. It just requires one argument, lottery_no, which stands for the round number of the lottery. The function first verifies the legality of the selected lottery round. If yes, it provides the total amount collected after retrieving it from the chosen lottery round's totalMoneyCollected field.

2.1.12 function getIthOwnedTicketNo(uint i,uint lottery_no) public view returns(uint,uint8 status)

It returns the ticket number of i-th purchased ticket of the user and its status. It accepts the lottery number and the number given by the user for the i-th ticket. Status 0 means the ticket is refunded or not revealed yet. Status 1 means the ticket is revealed but its prize is not collected. Status 2 means the ticket is revealed and the prize of the ticket is collected.

2.2 Additional Functions

This section includes the functions in our Solidity Smart Contract that are not in the description of the homework.

2.2.1 function advanceLottery() private

This function is used for updating the lottery time in the reveal stage.

2.2.2 function determineWinners(uint_current_RevealLottery_no) private

This function is used for determining winners in the lottery. It makes a modula operation to do that. This operation is made three times for determining the first, the second and the third winner respectively. Then based on the ticket type of winning tickets, the prize is determined as well. The total collected money is reduced by the amount of total prize. Finally winning tickets are stored in the lotteries.

2.2.3 function getBalance() public view returns (uint balance)

This function is coded for testing purposes. It returns returns the balance of the address in the message as Wei.

3 Scheme of User Functions

In this scheme, the functions which the user can interact have been represented. These are the basic functions for the flow of the lottery:

- depositEther
- withdrawEther
- buyTicket
- buyTicket
- revealRndNumber
- checkifTicketWon
- collectTicketPrize

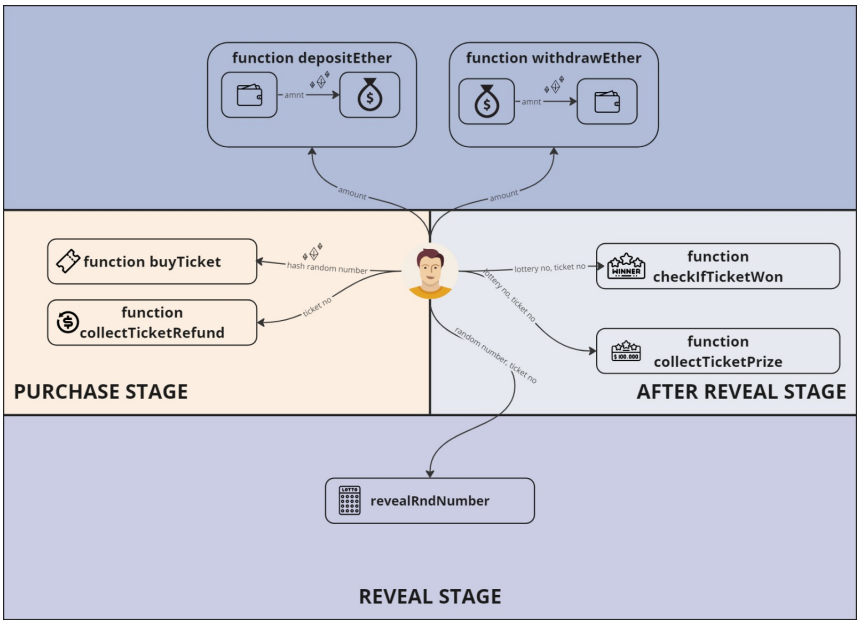

Figure 1: Scheme of user functions

4 Testing

For the testing of our whole contract, we have simulated whole two weeks period of lottery 1.

4.1 Testing Environment

We tested the contract on our local environment using 'ganache client' and 'truffle'. To test the code:

- First run ganache with: **ganache -a <number-of-addresses-you-want>**
- Open a new terminal and enter the directory: **cd Lottery**
- Then compile the contract: **truffle compile**
- Then deploy the contract with truffle: **truffle migrate**
- Then open truffle console with: **truffle console**
- Finally run the code on truffle console: **truffle exec test.js**

Note: Our code simulates two weeks period with manipulating time using the method evm_increaseTime. Geth client does not allow such manipulation, please do not use it for executing the script. Please use ganache instead.

4.2 Testing Method

First all accounts in the blockchain is unlocked:

```
// Unlock all accounts in the blockchain
const accounts = await web3.eth.getAccounts();
for(let i = 0; i < accounts.length; i++){
  await web3.eth.personal.unlockAccount(accounts[i]);
}
```

Figure 2: code snippet 1

depositEther function is tested with all accounts sending 2 ETH to the lottery:

```
// Test of depositEther function
for(let i = 0; i < accounts.length; i++){
  const txHash = await contract.methods.depositEther().send({from: accounts[i], value: web3.utils.toWei('2', 'ether')});
}

for(let i = 0; i < accounts.length; i++){
  let balance = await contract.methods.getBalance().call({from: accounts[i]});
  console.log('Adress: ', accounts[i], 'Balance: ', balance)
}
```

Figure 3: code snippet 2

getLotteryNos function is tested:

```
// Test of getLotteryNos function
var block = await web3.eth.getBlock("latest");
var time = block.timestamp;
currentLotteryNo = await contract.methods.getLotteryNos(time).call();
console.log('Result: lottery_no1:', currentLotteryNo[0], 'lottery_no2:', currentLotteryNo[1]);
currentLotteryNo = currentLotteryNo[0];
```

Figure 4: code snippet 3

buyTicket function is tested. All accounts buy three tickets as follows: To determine the ticket type Math.random() is used. To determine the random number web3.utils.randomHex() is used. Then buyTicket is called accordingly. Finally event is listened to get the returned value of ticket number.

```
// buyTicket function is called because this call will trigger the determineWinners function for the first lottery
let random_hex = web3.utils.randomHex(4);
let random_number = web3.utils.hexToNumberString(random_hex);
let random_hash = web3.utils.solidityShа3(random_number);
const txHash2 = await contract.methods.buyTicket(random_hash, 1).send({from: accounts[0], gas: 1000000});
```

Figure 5: code snippet 4

getLastOwnedTicket and getIthOwnedTicketNo functions are tested:

```
// Test of getLastOwnedTicket and getIthOwnedTicketNo functions
for(let i = 0; i < accounts.length; i++){
  let result = await contract.methods.getLastOwnedTicket(currentLotteryNo).call({from: accounts[i]});
  console.log('Account: ', accounts[i], 'Lottery No: ', currentLotteryNo, 'Ticket No: ', result[0], 'Status: ', result[1]);
  let result2 = await contract.methods.getIthOwnedTicketNo(2, currentLotteryNo).call({from: accounts[i]});
  console.log('Account: ', accounts[i], 'Lottery No: ', currentLotteryNo, 'Ticket No: ', result2[0], 'Status: ', result2[1]);
}
```

Figure 6: code snippet 6

Time is increased by 1 week plus 1 second: Since one week has been passed, we

```
// Time is increased in order to test other functions. The direct usage of web3.currentProvider.send() does not provide callback() function for the promise, triggers a warning on the console.
// Thus, I created a new promise to also have the callback() function to avoid this warning.
const increaseTime = async (duration) => {
  return new Promise((resolve, reject) => {
    web3.currentProvider.send(
      {
        jsonrpc: "2.0",
        method: "evm_increaseTime",
        params: [duration],
        id: new Date().getTime(),
      },
      (error, response) => {
        if (error) {
          reject(error);
        } else {
          resolve(response);
        }
      }
    );
  });
};

try {
  await increaseTime(604801);
  console.log('Time increased successfully.');
} catch (error) {
  console.error('Error occurred while increasing time:', error);
}
```

Figure 7: code snippet 7

can now test revealRndNumber:

```
// Test of revealRndNumber function
for(let i = 0; i < roundTickets.length; i++) {
  let ticket = roundTickets[i];
  await contract.methods.revealRndNumber(ticket.ticket_no, ticket.random_number).send({from: ticket.adress, gas: 500000})
}
```

Figure 8: code snippet 8

getTotalLotteryMoneyCollected is tested:

```
// Test of totalLotteryMoneyCollected function
var totalMoney = await contract.methods.getTotalLotteryMoneyCollected(currentLotteryNo).call();
console.log('Total Lottery Money Collected:', totalMoney);
```

Figure 9: code snippet 9

Time is again increased by 1 week plus 1 second

```
try {
  await increaseTime(604801);
  console.log('Time increased successfully.');
} catch (error) {
  console.error('Error occurred while increasing time:', error);
}
```

Figure 10: code snippet 10

Since two weeks have been passed from the start of the first lottery, the call of buyTicket will trigger advanceLottery() function and advanceLottery function will trigger determineWinnners() function:

```
// buyTicket function is called because this call will trigger the determineWinners function for the first lottery
let random_hex = web3.utils.randomHex(4);
let random_number = web3.utils.hexToNumberString(random_hex);
let random_hash = web3.utils.soliditySha3(random_number);
const txHash2 = await contract.methods.buyTicket(random_hash, 1).send({from: accounts[0], gas: 1000000});
```

Figure 11: code snippet 11

Now we can test checkIfTicketWon function:

```
// Test of checkIfTicketWon function
roundWinners = []
for(let i = 0; i < roundTickets.length; i++) {
  let ticket = roundTickets[i];
  winAmount = await contract.methods.checkIfTicketWon(currentLotteryNo, ticket.ticket_no).call({from: ticket.adress})
    if (winAmount > 0) {
      console.log('Winner ticket number:', ticket.ticket_no, 'Prize:', winAmount);
      roundWinners.push(ticket)
    }
}
```

Figure 12: code snippet 12

Test of getIthWinningTicket function:

```
// Test of getIthWinningTicket function
for(let i = 1; i < 4; i++) {
  let result = await contract.methods.getIthWinningTicket(i, currentLotteryNo).call();
  console.log(i, 'th Winner:', result[0], 'Amount:', result[1]);
}
```

Figure 13: code snippet 13

Test of collectTicketPrize function:

```
// Test of collectTicketPrize function. Observe the differences between getBalance calls
for(let i = 0; i < roundWinners.length; i++) {
  let ticket = roundWinners[i];
  beforeBalance = await contract.methods.getBalance().call({from: ticket.adress});
  console.log('Before Balance of Address:', ticket.adress, beforeBalance);
  const txHash = await contract.methods.collectTicketPrize(currentLotteryNo, ticket.ticket_no).send({from: ticket.adress});
  afterBalance = await contract.methods.getBalance().call({from: ticket.adress});
  console.log('After Balance of Address:', ticket.adress, afterBalance);
}
```

Figure 14: code snippet 14

Test of withdrawEther function:

```
// Test of the withdrawEther function. Observe the differences between getBalance calls
for(let i = 0; i < accounts.length; i++) {
  beforeBalanceChain = await web3.eth.getBalance(accounts[i]);
  beforeBalanceContract = await contract.methods.getBalance().call({from: accounts[i]});
  console.log('Before Balance of Address:', accounts[i], 'In the Contract:', beforeBalanceContract, 'In the Chain:', beforeBalanceChain);
  const txHash = await contract.methods.withdrawEther(10**18).send({from: accounts[i]});
  afterBalanceChain = await web3.eth.getBalance(accounts[i]);
  afterBalanceContract = await contract.methods.getBalance().call({from: accounts[i]});
  console.log('After Balance of Address:', accounts[i], 'In the Contract:', afterBalanceContract, 'In the Chain:', afterBalanceChain);
}
```

Figure 15: code snippet 15

4.3 Gas Usages

depositEther:

```
Transaction: 0xebc425c03949e4a45cd14f82233b9c4cc1fc95138dd26df275faf0ba682c924b
Gas usage: 43676
Block number: 2
Block time: Thu Jun 01 2023 05:44:28 GMT+0300 (GMT+03:00)
```

Figure 16: gas usage

buyTicket:

```
Transaction: 0x3bf8e54a02d0e0d9be57e600ea7cf73129a79225488bba54c85ecfa2f89aeb29
Gas usage: 188624
Block number: 19
Block time: Thu Jun 01 2023 05:44:28 GMT+0300 (GMT+03:00)
```

Figure 17: gas usage

Note: If a user triggers advanceLottery() (and possibly determineWinners()) the gas will be much higher (This happens when the time-sensitive contract variables(i.e. current_RevealLottery_no and current_PurchaseLottery_no) are outdated. Same applies for revealRndNumber and collectTicketRefund. SS below for buyTicket + advanceTime + determineWinnners

```
Transaction: 0x26c99efdf31b191349dc551d9061c7146750f02b9ae57b50e9469540dbda6b90
Gas usage: 833392
Block number: 37
Block time: Thu Jun 15 2023 05:44:31 GMT+0300 (GMT+03:00)
```

Figure 18: gas usage

collectTicketRefund:

```
Transaction: 0x758bca94525f7fd04fd0314a92aee6b6e65ce4004de98c76a22dd68b86c0ae6f
Gas usage: 54587
Block number: 22
Block time: Thu Jun 01 2023 05:44:28 GMT+0300 (GMT+03:00)
```

Figure 19: gas usage

revealRndNumber:

```
Transaction: 0x0c271d8880cdc4a8367c17572d36357186d17146f4602957413ca1c14040020f
Gas usage: 192405
Block number: 23
Block time: Thu Jun 08 2023 05:44:29 GMT+0300 (GMT+03:00)
```

Figure 20: gas usage

collectTicketPrize:

```
Transaction: 0x2aaca1a4671524ea1261e54e3473e22b6160b8b266c0f28076e8818b16c2fd6c
Gas usage: 37820
Block number: 38
Block time: Thu Jun 15 2023 05:44:31 GMT+0300 (GMT+03:00)
```

Figure 21: gas usage

withdrawEther:

```
Transaction: 0xb8591303e67d3afc89c8f991c4df08d55de1d9fca2e86664de8e125cf2f65e4f
Gas usage: 34137
Block number: 45
Block time: Thu Jun 15 2023 05:44:31 GMT+0300 (GMT+03:00)
```

Figure 22: gas usage

Since our smart contract does not have any parts that are dependent to the total number of addresses in the lottery, these gas examples apply to all. (There may be slight differences, but not in a degree to cause block gas expiration.)

5 Test Results (Example for 5 accounts)

5.1 Creation of Accounts

To start the test, we run the ganache command with the desired number of accounts. In default, it creates desired number of accounts with 1000 ETH balance each. In this part of the test, we are unlocking the accounts in the contract. The number of the accounts is determined by using a variable and we tested it with several numbers (10, 100, 200, 500 etc.). As the result of this test function, we received this kind of output:

```
Available Accounts
==================
(0) 0xC9d8f763317E87920Dc7Af45251d1de17EE73B79 (1000 ETH)
(1) 0xB0B5f43103B57322c105f45fef78e8709BD356D2 (1000 ETH)
(2) 0x48479BAd4348D506ff03b7111aD449D50E3E28b3 (1000 ETH)
(3) 0xc81117c9eC2b0EEF5f0F0Eca26e9505460A6A88f (1000 ETH)
(4) 0xB90f14049e9014977AF5406D8495AB8f1A7f36b3 (1000 ETH)
```

Figure 23: Example accounts

5.2 Deposit Ether

To join the lottery, users must first deposit ether to the lottery contract to be used for buying tickets. Thus, we call depositEther() function for all accounts. Example below shows each user deposits 2 ETH to the lottery contract.

```
Adress:  0xC9d8f763317E87920Dc7Af45251d1de17EE73B79 Balance:  2000000000000000000
Adress:  0xB0B5f43103B57322c105f45fef78e8709BD356D2 Balance:  2000000000000000000
Adress:  0x48479BAd4348D506ff03b7111aD449D50E3E28b3 Balance:  2000000000000000000
Adress:  0xc81117c9eC2b0EEF5f0F0Eca26e9505460A6A88f Balance:  2000000000000000000
Adress:  0xB90f14049e9014977AF5406D8495AB8f1A7f36b3 Balance:  2000000000000000000
```

Figure 24: Deposit ether test

5.3 Buy a new ticket

In our contract, bought tickets are collected in a list. So we should observe the sold tickets together. While we storing these tickets, we also keep some information with them such as ticket number and the random number. These collection of tickets can be observed in the image below which represents a test output:

```
{
  adress: '0xC9d8f763317E87920Dc7Af45251d1de17EE73B79',
  random_number: '4009589249',
  ticket_no: '1'
},
{
  adress: '0xC9d8f763317E87920Dc7Af45251d1de17EE73B79',
  random_number: '4167090441',
  ticket_no: '2'
},
{
  adress: '0xC9d8f763317E87920Dc7Af45251d1de17EE73B79',
  random_number: '1227190145',
  ticket_no: '3'
},
{
  adress: '0xB0B5f43103B57322c105f45fef78e8709BD356D2',
  random_number: '685584516',
  ticket_no: '4'
},
```

Figure 25: Sold Tickets

As it can be seen from the figure 25, we have several tickets which has been bought from different accounts. Consecutive numbers have been assigned to them for keeping track of the ticket numbers.

5.4 Purchasing and Revealing Stages

In a lottery slot, first week is for purchasing and the second week is for revealing stage. So we assigned different status to tickets so that these stages can differ from each other. Our assignment of status is as below:

- status 0 –> ticket is purchased
- status 1 –> ticket is revealed

For testing this flow, at first we printed out the tickets which are in the purchasing state:

```
Account:  0xC9d8f763317E87920Dc7Af45251d1de17EE73B79 Lottery No:  1 Ticket No:  3 Status:  0
Account:  0xC9d8f763317E87920Dc7Af45251d1de17EE73B79 Lottery No:  1 Ticket No:  2 Status:  0
Account:  0xB0B5f43103B57322c105f45fef78e8709BD356D2 Lottery No:  1 Ticket No:  6 Status:  0
Account:  0xB0B5f43103B57322c105f45fef78e8709BD356D2 Lottery No:  1 Ticket No:  5 Status:  0
Account:  0x48479BAd4348D506ff03b7111aD449D50E3E28b3 Lottery No:  1 Ticket No:  9 Status:  0
Account:  0x48479BAd4348D506ff03b7111aD449D50E3E28b3 Lottery No:  1 Ticket No:  8 Status:  0
Account:  0xc81117c9eC2b0EEF5f0F0Eca26e9505460A6A88f Lottery No:  1 Ticket No:  12 Status:  0
Account:  0xc81117c9eC2b0EEF5f0F0Eca26e9505460A6A88f Lottery No:  1 Ticket No:  11 Status:  0
Account:  0xB90f14049e9014977AF5406D8495AB8f1A7f36b3 Lottery No:  1 Ticket No:  15 Status:  0
Account:  0xB90f14049e9014977AF5406D8495AB8f1A7f36b3 Lottery No:  1 Ticket No:  14 Status:  0
```

Figure 26: Accounts and tickets in the purchasing stage

We can observe the accounts and corresponding ticket numbers in the figure 26. Since the tickets are still in purchasing stage, all of them has status 0. Now, we increased the time by 1 week so that we can simulate the reveal stage. And we revealed the tickets as a result we received this output:

```
Time increased successfully.
Account: 0xC9d8f763317E87920Dc7Af45251d1de17EE73B79 Lottery No: 1 Ticket No: 3 Status: 1
Account: 0xB0B5f43103B57322c105f45fef78e8709BD356D2 Lottery No: 1 Ticket No: 6 Status: 1
Account: 0x48479BAd4348D506ff03b7111aD449D50E3E28b3 Lottery No: 1 Ticket No: 9 Status: 1
Account: 0xc81117c9eC2b0EEF5f0F0Eca26e9505460A6A88f Lottery No: 1 Ticket No: 12 Status: 1
Account: 0xB90f14049e9014977AF5406D8495AB8f1A7f36b3 Lottery No: 1 Ticket No: 15 Status: 1
```

Figure 27: Accounts and tickets in the reveal stage

We only show last tickets of each user which has been revealed for this test and as you can also see from figure 26, their status became 1.

5.5 Ending the Lottery Session and Determining Winners

After the revealing stage has been finished, the winning tickets should be determined and prizes should be distributed. To close the revealing stage, the time should be increased again. By doing that, we close the lottery and determining winners:

```
Time increased successfully.
Total Lottery Money Collected: 66000000000000000
Winner ticket number: 1 Prize: 8250000000000000
Winner ticket number: 4 Prize: 4125000000000000
Winner ticket number: 13 Prize: 4125000000000000
```

Figure 28: Finishing the lottery

As you can see from the above figure, the winning ticket numbers have been announced and corresponding prizes as well based on ticket type (full, half, quarter) and winning (1^{st}, 2^{nd}, 3^{rd}) order.

After the winning accounts have been determined, the corresponding prizes are transferred to the balances of these accounts. You can see the difference between balances of accounts before and after the lottery as below:

```
Before Balance of Address: 0xC9d8f763317E87920Dc7Af45251d1de17EE73B79 1980000000000000000
After Balance of Address: 0xC9d8f763317E87920Dc7Af45251d1de17EE73B79 1988250000000000000
Before Balance of Address: 0xB0B5f43103B57322c105f45fef78e8709BD356D2 1992000000000000000
After Balance of Address: 0xB0B5f43103B57322c105f45fef78e8709BD356D2 1996125000000000000
Before Balance of Address: 0xB90f14049e9014977AF5406D8495AB8f1A7f36b3 1992000000000000000
After Balance of Address: 0xB90f14049e9014977AF5406D8495AB8f1A7f36b3 1996125000000000000
```

Figure 29: Balances after prizes

Lastly, we have checked **withdrawEther** function. To do that we simulated to withdraw 1ETH from balances and this is what we obtained:

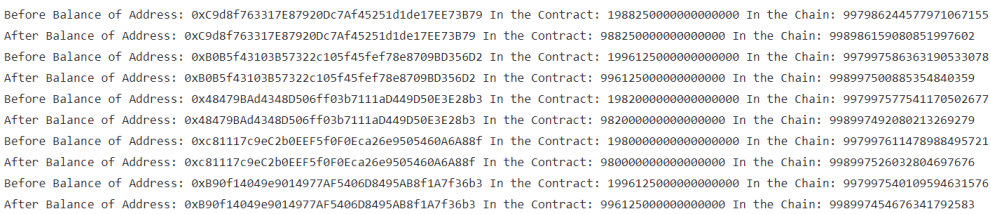

Figure 30: Balances after withdrawing ether

In this image, we can observe the amount of money in the balance within the contract and in the chain which means wallet of the user. When we withdraw 1ETH from the balance, the amount of money in the contract is reduced and the amount of money in the users wallet is increased.

Note: This example is representing the test on 5 accounts, test results for higher number of accounts could be found in txt files which is attached to the submission.

6 Task Achievement Table

Task Achievement Table	Yes	Partially	No
I have prepared documentation with at least 6 pages.	x		
I have provided average gas usages for the interface functions.	x		
I have provided comments in my code.	x		
I have developed test scripts, performed tests and submitted test scripts as well as documented test results.	x		
I have developed smart contract Solidity code and submitted it.	x		
Function depositEther is implemented and works.	x		
Function withdrawEther is implemented and works.	x		
Function buyTicket is implemented and works.	x		
Function collectTicketRefund is implemented and works.	x		
Function revealRndNumber is implemented and works.	x		
Function getLastOwnedTicketNo(uint lottery_no) is implemented and works.	x		
Function getIthOwnedTicketNo is implemented and works.	x		
Function checkIfTicketWon is implemented and works.	x		
Function collectTicketPrize is implemented and works.	x		
Function getIthWinningTicket is implemented and works.	x		
Function getLotteryNo is implemented and works.	x		
Function getTotalLotteryMoneyCollected(uint lottery_no) is implemented and works.	x		
Function depositTL is implemented and works.	x		
Function withdrawTL is implemented and works.	x		
Function buyTicket is implemented and works.	x		
Function collectTicketRefund is implemented and works.	x		
Function revealRndNumber is implemented and works.	x		
Function getLastOwnedTicketNo is implemented and works.	x		
Function getIthOwnedTicketNo is implemented and works.	x		
Function checkIfTicketWon is implemented and works.	x		
I have tested my smart contract with 5 addresses and documented the results of these tests.	x		
I have tested my smart contract with 10 addresses and documented the results of these tests.	x		
I have tested my smart contract with 100 addresses and documented the results of these tests.	x		
I have tested my smart contract with 200 addresses and documented the results of these tests.	x		
I have tested my smart contract with more than 200 addresses and documented the results of these tests.	x		

Figure 31: Task Achievement Table