CmpE 483 Project 2
Visual Implementation of Decenteralized Lottery

Korhan Çağın Geboloğlu - 2012400075
Salih Sevgican -2013400219
Ahmet Buğrahan Taşdan - 2013400156
Group Name:ZAUM

Spring 2018

Contents

1 Introduction

1.1 Intro

The aim of this project is to demonstrate how a smart contract in Solidity[2] can be used by any ordinary user who do not have a strong background in technology . In this course, how a smart contract can be implemented and how can we build a frontend for our contract have been covered and simply in this project, we need to code what we have learnt in this class. To be able to finish the project, we need to find a way to combine a design and our first project code. The former project was related to decentralization of autonomous lottery. To satisfy the requirements of this project, we need to implement a frontend that helps us to test our environment. We need to focus on the solving defects in first project project, design a solid web based user interface and make the connection between contract and frontend. However, before starting deep into the details of how we handled the project, to remember what is the requirements in this project would be great idea to consider.

1.2 Definition

An autonomous decentralized lottery based on smart contract which distributes specified types of tickets for a given time interval and retrieves different types of winners for this given time range and after then this, repeats the same idea next timeline. There are some requirements that must be satisfied to finish the project. This is the former project definition and this project still dependson this. In this project, we need to implement a system that helps user to play with our contract and by doing this, we build visual test environment. On the other hand, we need to use some other tools. The most significant one is Brave[3] which is a web browser with many available resources for blockchain-based tools. The second one is a third party application that help the user to use Etherium-based blockchain system easily, which is called as Metamask[4]. Finally, we should be able to test our system on a network called as Ropsten Test Network[5].

1.3 Checklist

What project expect us to do can be seperated in three part and totally 8 main specialities should be added to that system. What they are;

- Contract Part

  - Solving Defects

- Visualization Part
  - Building a User Interface
  - Connecting with contract
  - Connecting with Metamask

- Testing Part
  - Controlling contract with Ropsten

2 Contract Part

Contract Part is the base implementation part of the project and in this part, we need to build a structure for this project. We need to build a smart contract that allows users gets one or more tickets which will be used in a lottery. The user will send a number that they bet on. Our contract will calculate a big winner and some small winners. After this process, the winners gets the money and if their accounts eligable to withdraw the money, they can collect their money.
To be able satisfy this scenerio, we divided the coding part into four main part and one helper part. The helper part is just a bunch of functions that retrieves information about the current status of the contract. We also have some internal class level variables that helps the system to save the information about current status and the details of the buyers. The part that we classified will be elaborated more in the upcoming sections one by one.

2.1 Ticket Declaration

The ticket structure is the milestone of the project because the transaction resulted with creating a ticket depends on the how many gas sent by the buyer and with this ticket buyer will join the lottery to win the prize. One ticket consists of four different elements. All of them stores one significant feature of the ticket.
The first item of the ticket is tip and it is the declaration of what kind of ticket this ticket is. This information helps the system to calculate how big the prize will be because there are three different types of ticket and the reward is diferantiated by this element.
The next item is related to who sent this gas to get the ticket. It basically stores the sender's hash address in the ticket. It will speed up the validation process of ticket and check whether this ticket is winner or not.
N is used in the revealing phase. It is sent by the buyer and used for the correctness of ticket.
The last item is the declaration of whether this ticket validated or not.

2.2 Purchasing

We start with controlling with the incoming gas size. If it acceptable, we proceed to create ticket. After creating the ticket, we store the ticket in the contract due to use that information at the revealing phase. Our algoritm can be described like below.

```
1  take(val) from sender
2  get(tickets) from system
3  if msg.value!=8 finney and
4  msg.value!=4 finney and
5  msg.value!=2 finney then
6  |  revert();
7  else
8  |  ticket ⟵ createTicket()
9  |  if msg.value==8 finney then
10 |  |  ticket.tip ⟵ 1
11 |  end
12 |  if msg.value==4 finney then
13 |  |  ticket.tip ⟵ 2
14 |  end
15 |  if msg.value==2 finney then
16 |  |  ticket.tip ⟵ 3
17 |  end
18 |  ticket.ticket_hash ⟵ val
19 |  tickets.push(ticket)
20 end
```

2.3 Revealing

This function takes ticket number as parameter. If an address has bought more than one ticket, than they have to reveal their numbers by order. For example if an address has sent a hash code(purchase) by using Ticket number 12 and 24, then it has to reveal its tickets by sending first 12 then 24. Reveal function also checks whether it is the last block of the lottery or not, if it is, then it maps winner numbers address for reimbursment according to Ticket type. If it is not the last block then it checks number with the hash code which is send in purchase function, if it checks out then winner number is XOR'ed with the given parameter. If it is not a valid ticket then winner number is not XOR'ed with the parameter.

2.4 Withdrawing

In the revealing phase if a ticket owner gains a right to withdraw, reimbursement map in the code has the address and the amount of gas that needs to be sent. If sender's ticket number has won the lottery then withdraw function returns required amount of money.

3 Visualization Part

4 Testing Part

4.1 Testing with JavaScript VM

The project mostly tested by giving inputs manually via JavaScript VM which is a feature of Remix-Solidity editor. We'll explain one of our test cases.
Since we cannot give thousands of inputs, we've limited our purchase and reveal periods to 4 blocks. A lottery time will take 8 blocks at total.
JavaScript VM has five different accounts at default with 100 ethers of balance. We've used 4 accounts and prepared random numbers and their hashes calculated by contract's getHashValue function. This function only calclulates a hash by using keccak256 function of solidity.

address : 0xca35b7d915458ef540ade6068dfe2f44e8fa733c
Ticket num:2
hash: 0x825cd35b19e019914b602821314f167512f39929f15d522c0e63f90f5b00be43
address : 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c
Ticket num:3
hash : 0x5633db22326de9f1587f5badbf225727243bf43db44d438c1bae8ab9cb3b5af8
address : 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c
Ticket num:4
hash : 0x39d68e7f9d4d49c4737b7b7e4b42e4e45ddb689f506ff3699943909d3418326b
address : 0x583031d1113ad414f02576bd6afabfb302140225
Ticket num:6
hash : 0x46e0d76af42eef38e0360a7e5303ef14565a51eabb891042f5429ae7bf432188
Winner is Ticket number : 3
— Test 2 —
address : 0xca35b7d915458ef540ade6068dfe2f44e8fa733c
Ticket num:0
hash : 0xdea76eae5e144f255069f6348a9c319d0aac96c61ff4f1ca03a283a0ff44921c
address : 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c
Ticket num : 3
hash : 0x5633db22326de9f1587f5badbf225727243bf43db44d438c1bae8ab9cb3b5af8
Winner is again Ticket number : 3

4.2 Testing with Code

In this part, we have to prove what kinds of features our contract can offer to users.We coded a script written in Javascript[1]. To be able to demonstrate the full scale of capabilities of our project, We divide testing part into four different sub-group.

4.2.1 Create Different accounts

The contract is meaningless without users. That's why, we need to add new users that can play with our lottery contract. User are capable to buy three different types of tickets according to the gas they sent. We created four different accounts in order to use in our testbed. We also add the private informations of the accounts into the test code due to be able use in it.

4.2.2 Purchase the tickets

The accounts can buy the tickets in different amounts and buys more than one ticket for turn. We wrote those in different orders to be able to see different results.

4.2.3 Sending the money

After the revealing results, we are just checking the accounts in order to display the account's balances change.

4.2.4 Distributability of Prize

The system can run for different times and accounts can win the prize according to what they sent to system.

5 Assessment and Conclusion

In this project, Solidity is one of best options to implement that project can be coded on. Although it is suitable to use in the project, the concept that we are getting familiar with is hard to process and understand. However, the best practise we can get is playing with these tools. That's why what we tried to do in this procect is helpful.
We are successfully implamented the purchasing tickets by different users in different types and can be sold to one user more than ones for one turn. Our revealing algorithm is working perfectly by itself. The conflict between revealing and upcoming turns purchasing regretfully are not responding the user perfectly.
On the other hand, our contract are not distributing the second and third winner but it can be added to the system easily. We wish to keep the system fast as much as we can. This distribution cost a lot of gas that's why, even though we are capable to add into system, we choose not to.
The overall results of the project is better than our expectations. The consecutive lotteries work perfectly. However, the conflicts between the revealing and purchasing cause withdrawing the gases from common pools for the lottaries.

Bibliography

[1] JavaScript,
`https://developer.mozilla.org/bm/docs/Web/JavaScript/`

[2] Solidity Language version.0.4.23,
`https://solidity.readthedocs.io/en/v0.4.23/`

[3] Brave Browser,
`https://brave.com`

[4] Metamask ,
`https://metamask.io`

[5] Ropsten ,
`https://ropsten.etherscan.io`