**Cmpe483 BULOT Lottery Smart Contract Project1**

Hande Sirikci

Nursima Celik

Tahir Kaan Ögel

Contents:

• Installation Instructions and Test Explanation

• Project Description - Implemented Functions

• Gas Usage Estimation

• Installation Instructions:

To Install and run and test of our project you need to follow the following steps.

• Install geth on your device.

• Start your local blockchain with geth using the following command inside your terminal:

geth --datadir test-chain-dir --rpc --dev --rpccorsdomain "https://remix.ethereum.org,http://remix.ethereum.org" --allow-insecure-unlock

• Change directory then to test-chain-dir and open your geth console to run javaScript commands with the following command:

geth attach geth.ipc

• Now open remix inside your browser( don’t forget to switch your environment in remix to web3 provider to connect with your prior started local chain) and start deploying the smart contracts of the project ( erc.sol and bulot.sol) by pasting their content into files inside remix and then first compile and then deploy the them. Make sure to deploy the erc20.sol contract first and pass it’s contracts address over for deploying the bulot.sol contract. Also don’t forget to switch your environment in remix to web3 provider to connect with your prior started local chain.

• Now open the test-script “test-bulot.js” in a texteditor and pass the resulting contract addresses from remix inside the designated variables “bulotAddress” and “erc20Contract” . Also make sure that the test-file and the abi-Files of the contracts are inside the same directory where the geth console is running.

• You can then run the updated test-script via the following command in the prior opened geth console:

loadScript(“test-bulot.js”)

Test-Description

The test-scirpt is simulating a whole lottery circle with multiple user. The number of users can be set dynamically (ACCOUNT\_NUM). The default value is 10. The Test-Script is first connecting with the deployed contracts. After that it creates accounts till the wished “ACCOUNT\_NUM” is reached. These around will then be equipped with tokens by the contracts creator account. After getting the tokens every user will buy a ticket with a random generated input number. The amount of money collected in the lottery is finalized after the purchase phase, since the money paid to those who do not count their tickets as valid is not returned. For this reason, we print the amount of money collected in the lottery at this stage. After reaching an interval function(*var revealCheck = setInterval(function () {*) that is executed after a static time interval the user will check if the reveal stage of the lottery they have participated at is reached and if so they will reveal their numbers to be able to win with their tickets. After again reaching an interval function(*var withdrawCheck = setInterval(function () {*) with is withdrawing the prizes it is checked if the withdraw stage is reached and if it is reached every user is checking if he/her has won and if so is initiating the withdraw of the prize.

• Project Description:

This solidity code is a lottery smart contract. Participants should buy a ticket with 10 tokens in order to participate. Each token is equal to 1TL. TL-TOKENS exchange is done using EIP20 Contact. There are two periods of the lottery: submission and reveal. In submission period users buy ticket, and in the reveal stage they need to reveal their random numbers with tickets to confirm the hashed number and the hash function of the actual number is equal. According to the reveals, by XOR ing all valid numbers we conclude a number which is going to be the winner. The other winners are determined by hashing the number and getting the modulo of it and take this element in valid tickets. Each winner takes half of the token prize of the previous winner and the modulo with two. The first winner takes approximately half of the total tokens that is collected via ticket prices. Total number of winners is equal to logarithm base two of the total money collected in lottery.

bulot.sol:

• This contract has to be interact with the ERC20 so one has to import the ERC by writing : import "./erc20.sol";

• This contract needs to be compiled with a compiler whoose version is greater than 0.4.21 and should not be newer.

• Structs:

struct Ticket {

uint ticket\_no;

bytes32 hash\_rnd\_number;

address owner;

bool withdrawn;

}

Ticket: This struct created in order to hold ticket object.

• ticket\_no= It is a unique number for each ticket which specifies the ticket number.

• hash\_rnd\_number = This is the hash function of the random number which is given with the ticket.

• Owner = Address of the owner of the ticket

• Withdrawn= bool variable that holds whether the prize of the ticket is delivered or not

struct Lottery {

uint money\_collected;

uint winnerNumber;

Ticket[] tickets;

mapping(uint => Ticket) validTickets;

uint numOfValidTickets;

mapping(address => uint[]) usersTicketNos;

}

Ticket: This struct created in order to hold lottery object.

• money\_collected= total money collected for this lottery

• winnerNumber= Random number given by the winner of the lottery

• tickets= Participating tickets for his lotttery

• validTickets= mapping that holds the valid tickets which are revealed successfully in reveal stage

• numOfValidTickets: Number of valid tickets in this lottery

• start: start time of the lottery

• usersTicketNos: mapping that maps user addresses to the ticket numbers

• Global Variables

uint stagePeriod = 2 weeks;

• In our case, every lottery last two weeks

mapping(uint => Lottery) lotteries;

• This is a mapping maps numbers of lotteries to lottery object

Uint unusedVar = 1;

• Lottery number calculation did not work without the declaration of this variable. Unfortunately no reasonable information why contract behaves this way

uint start;

• Creation time stamp of the contract in order to determine stages and order of the lottery

address public contractaddr;

• Address of the ERC contract to create connection this will be necessary

function() public {

revert();

}

• This function prevents illegal payments in fallback function. It is implemented in order to provide security

constructor(address conaddr) {

contractaddr = conaddr;

start = now;

}

• Constructor called when the contract is created. Sets the timestamp and takes the EPC address to connect.

• Conaddr : address of the EIP20 network. EIP20 network should be deployed before this bulot contract and it has be given as a parameter to deploy this contract.

• Lottery number calculator Functions:

• getCurrentLotteryNo: This function returns lottery number by measuring how much

time passed since the starting of the lottery and divides the passed time to the week time.

• 1 week passed -> returns 0

• 4 weeks passed -> returns 3

**• Function buyTicket(bytes32 hash\_rnd\_number) public**

This function is called when the user wants to buy a ticket. It takes the hashed random number as parameter and requires the transaction of 10 tokens from user. This function operates in purchase stage.

• It enables the connection with ERC20

• Tries to transfer 10 tokens from ERC20. User has to allow it before calling this function.

• Ticket object is created and filled the necessary fields of the object.

•

**• function revealRndNumber(uint ticketno, uint rnd\_number) public**

This function is called when the user reveal his/her random chosen number. It takes the initially inserted random number and the ticket number as parameters. The function makes bought tickets valid by revealing their random number through the user. °°The function does:

• It checks if the lottery is in the right stage.This function operates in reveal stage.

• Then takes the passed random number parameter and calculates the hash of it.

• This Hash is then again compared to the hash-value the ticket was initially created with.

• If hash is valid the ticket will be push into a array of valid tickets and a counter will be increased which tracks the number of valid tickets per lottery

**• function getLastBoughtTicketNo(uint lottery\_no) public view returns(uint)**

This function is called when the user want to know his/her last bought ticket number. It takes the lottery number as a parameter. The function does:

• It takes the number of the total bought tickets of the user and takes the last one.

• It then takes the ticket with this number and returns it

**• function getHash(uint rnd\_number) public view returns(bytes32)**

This function is called when the user wants calculate his/her hash of the chosen random number to buy a ticket. It takes a uint number and return a bytes32 variable.

**• function getIthBoughtTicketNo(uint i,uint lottery\_no) public view returns(uint)**

This function is called when the user want to know a specific bought ticket number. It takes the lottery number and an index as parameters. The function does:

• Directly reaches for the specific ticket in the array of the bought tickets of the user.

**• function checkIfTicketWon(uint lottery\_no, uint ticket\_no) public view returns (uint amount)**

This function is to check if a ticket won in a lottery. It takes the lottery number and the ticket number as parameters. It returns the amount a ticket has won (if lost it will return 0) The function does:

• It checks the current lottery number with the passed one to compare and only proceed if passed lottery is already over.

• It takes the prior saved total amount of money collected in this lottery and passes this into the logarithm function to calculate the wins.

• The wins are then iterated over till the earlier passed ticket number is reached to check the prize.

• The resulting prize will then be returned and in case the ticket didn’t win a 0 will be returned.

**• function withdrawTicketPrize(uint lottery\_no, uint ticket\_no) public**

This function is called when the user wants to withdraw his/her won prize a specific bought ticket number. It takes the lottery number and the ticket number as parameters. It initiates the transfer of the prize to the users account. The function does:

• It checks the current lottery number with the passed one to compare and only proceeds if passed lottery is already over.

• It takes the prior passed ticket number and uses the earlier described function checkIfTicketWon.

• If ticket has won and prize was not already withdrawn the contract sends the prize to the user.

**• function getIthWinningTicket(uint i, uint lottery\_no) public view returns (uint ticket\_no,uint amount)**

This function can return a specific ticket that won in a lottery by his index. It takes the lottery number and an index as parameters. It return a ticket number and it’s prize.The function does:

• It checks the current lottery number with the passed one to compare and only proceed if passed lottery is already over.

• Again uses the total amount of Money collected in the lottery to call the logarithm function to get the wins.

• It then again iterates over all wins until it reaches the specific ticket of the passed index.

• It then returns the ticket number and prize of this ticket.

**• function getCurrentLotteryNo() public view returns (uint lottery\_no)**

This function returns the lottery number of the lottery that is currently in purchase Stage. It does it by comparing the current time with the time the first lottery started.

**• function getMoneyCollected(uint lottery\_no) public view returns (uint amount)**

This function returns the total amount of money that was collected in a specific lottery. It takes the lottery number as a parameter.

• Gas Usage Estimation :

Function

Gas-Usage Estimation

buyTicket(bytes32)

"infinite"

checkIfTicketWon(uint256,uint256)

"infinite"

getCurrentLotteryNo()

444

getHash(uint256)":

388

getIthBoughtTicketNo(uint256,uint256)"

1171

getLastBoughtTicketNo(uint256)":

1410

getIthWinningTicket(uint256,uint256)

"infinite"

getMoneyCollected(uint256)":

377

revealRndNumber(uint256,uint256)":

"infinite"

withdrawTicketPrize(uint256,uint256)

"infinite"

Unfortunately, our Research on our Estimated Gas Usage resulted many “infinite” results. Our guess is that the usage of arrays including the use of array.push() leads to this result. So the compiler simply doesn’t know how big a array could become. The other Estimation reasonably easy to follow. *getCurrentLotteryNo, getMoneyCollected* and *getHash* simply convert numbers or do simple calculations while *getIthBoughtTicketNo* and *getLastBoughtTicketNo* s interations and loops include which increase the gas costs extremely. Also saving variable lead to high gas costs.