CMPE 483 HW1 – Autonomous Decentralized Lottery on Ethereum Blockchain

Abdullah Furkan İlısu - 2013400201
Alim Türkmen - 2014400165
Giray Eryılmaz – 2014400045

Our project implements an **Autonomous Decentralized Lottery** as instructed in the description of homework1. Each lottery has two parts. Each part continues for 20.000 blocks. There always is a lottery running (a new lottery starts immediately previous one's purchase state finishes) and there are always 2 lotteries running at the same time except for the very first 20.000 blocks where there is no reveal state, there is only purchase stage. In other words except for the first half of the very first lottery there are always two states happening concurrently: the (n-1)th lottery's reveal state and the nth lottery's purchase state. All these and more implemented according to the description.

How we handle time (number of blocks passing):
Since there are no built-in mechanisms for alerting time (number of blocks passing) such as timed interrupts, we use another approach for keeping the system up to date.

What we do is: when any function (for instance revealTicket or buyFullTicket) is called by users, firstly the last "version" of the system and the current block number are compared to see if a new period has started. If a new period has started then `update` function is called and all variables about the status of the system are updated to current state.

For example say a customer uses buyFullTicket function to purchase a ticket when the 3rd lottery has just started. Firstly the block number is checked and compared with last updated state. Since the new lottery round has just started it is seen that the system variables are out dated. Update function is called and the current lottery number is set to 3, since reveal state of 1st lottery ends at the same time lottery3 starts, winners of lottery1 are picked using random numbers provided by users to lottery1 and prizes are recorded in a map of user – balance (address – uint map) in the storage so that winners can withdraw their winnings whenever they want (the contract does not transfer the rewards because that would cost gas. Users have to withdraw money themselves). The excess prize is passed on to the next lottery.
One note: a boolean is kept to check if the current lottery is the very first period because in the very first period, unlike the other ones, there is no reveal state at the same time the purchase state of 1st lottery.

How we implement commitment - reveal:
We receive hashes of random numbers while user is purchasing tickets. Then during the reveal stage of that lottery the user must use revealTicket function to provide the real random numbers he/she previously committed to. Then the revealed numbers are hashed and the resulting hashes are compared with previously given hashes, if they do not match then the chance of winning is lost, the numbers provided are ignored and not used for picking winners, no refund is given. If a user do not reveal then again he/she can not join the lottery and no refund is given.

Keeping track of tickets:
We use ticket structs for keeping random numbers - hashes, the ticket owner's address, ticket type and a boolean for verification.

We also make sure that no two full tickets are bought with same random numbers and at most 2 half tickets or at most 4 quarter tickets can be bought with same random numbers. We do this by keeping track of used random numbers as 'mapping(string => uint[3]) used_randoms along with ticket structs. Keys are actually hashes converted to string.

How the winners are determined:
Winners are determined in the update function, which is called when a function is called by users iff the system is out dated. The successfully verified (revealed) tickets' random numbers are XORed to get winning numbers.

The program works because we always keep track of time and update variables before doing anything else and hold necessary data in the storage so that it is not lost.

Also we use revert() when ever we need to cancel an operation, for instance while purchasing a ticket if enough money is not sent then we revert() changes that are made.