University of Zurich^UZH

# Challenge Task: Decentralized Lottery Application

**Blockchain and Overlay Networks**

*Danijel Dordevic, Ratanak Hy, Claude Muller, Lucas Thorbecke*
*Student IDs: ..., 11-717-790, 11-067-675, 12-933-917*

Date of Submission: May 22, 2019

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

# Contents

# Chapter 1

# Introduction

To get hands on experience with decentralized applications a Lottery system was implemented. The requirements were the following.

- The lottery system must run entirely within one or more Smart Contracts (SC).
- For determining a winner a random number generator oracle must be implemented an run in a separate SC than the lottery. Though, the random numbers can be generated either on the blockchain or externally.
- The lottery SC must automatically pay the prize to the winner, divide the prize in case of multiple winners, or accumulate the prize for the next lottery round in case of no winners.

The lottery was realized on the Ethereum blockchain with the Solidity programming language. Furthermore, a front-end for convenient interaction with the lottery was implemented using JavaScript and React[1].

The source code of the implementation as well as the sources for this report can be found at `https://github.com/ddanijel/bcoln-ct`.

[1]`https://reactjs.org/`

# Chapter 2

# Design and Implementation

## 2.1 Design

The exact mechanisms of a lottery can be implemented in many ways. The following describes the design choices made in this work.

The most important concept is that there is not only a single lottery contract but rather one contract for each round of the lottery. One round consists of players betting and the picking of a winner. After a winner has been chosen the specific lottery is closed and a new lottery is deployed. The lotteries are controlled via a central lottery factory that spawns new lotteries, knows of the current active lottery, and is the interface by which the lottery issuer can manage the lottery flow and the players can set there bets. The lottery factory is also the owner of the bets made by players. The lottery instance themselves do not have command over the bets. If there is no winner in a lottery round then all bets made in that round are kept in the lottery factory and reused in the next round. The issuer of the lottery factory cannot withdraw the balance accumulated on the lottery factory.

The advantage of the separation of factory and lotteries is that each lottery round is retrievable by its own address and the factory can be queried for the list of all past lotteries. This seems more convenient than the case in which only one lottery is reused for each round. We can avoid having to scan through the blockchain to retrieve the transaction history of the lottery and reconstruct its intermediary states.

Other design decisions are:

- A player can bet multiple times on the same or different numbers in a single lottery round.
- The lottery issuer sets a ticket price which is the minimum price a player has to pay for his bet. The system does not prevent the player to pay more, but he does not get any benefits from it.
- The issuer of the lottery factory sets the number range by providing a maximum number that can be bet.

The interactions of the lottery manager and the players with the lottery systems are displayed Figure 2.1. All interactions are happening via the `LotteryFactory`. The lottery factory forwards the calls to the current lottery. The users do not have to know or keep track of the concrete lottery instances.

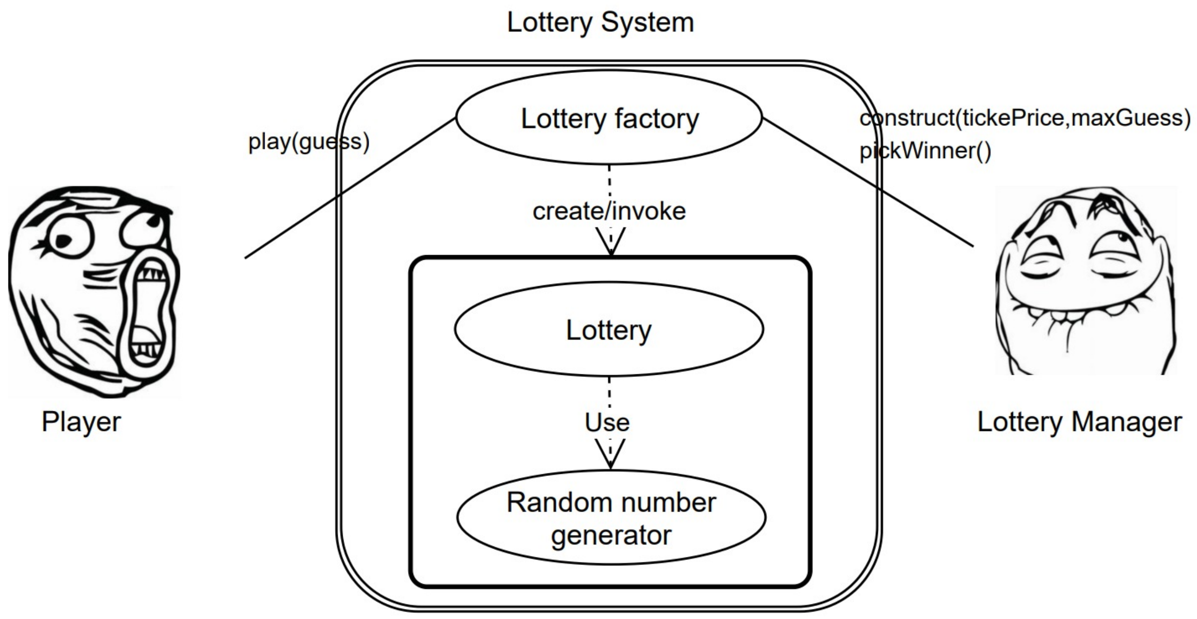

Figure 2.1: Use case diagram of the lottery system.

In the following sections the details of the implementation are explained hand in hand with the actual contract code. Figure 2.2 show the existing classes/contracts which are part of the lottery system.

## 2.2 Lottery Factory construction

Once created, the `LotteryFactory` remembers the `manager` (the address of the lottery issuer) for later checks. The ticket price and the maximum guess number are chosen by the issuer, i.e. provided to the constructor, and are fixed for all future lotteries. The lottery factory creates its own `RandomNumberOracle` instance which will be called by all future lotteries when a number has to be drawn. Finally, a first `Lottery` instance is created which becomes the current lottery.

```
contract LotteryFactory {

    address manager;
    uint ticketPrice;
    uint maxGuessNumber;
    address[] allLotteries;
    Lottery currentLottery = Lottery(address(0x0));
    RandomNumberOracle randomNumberGenera-
tor = RandomNumberOracle(address(0x0));

  constructor(uint _ticketPrice, uint _maxGuessNumber) public {
      manager = msg.sender;
```

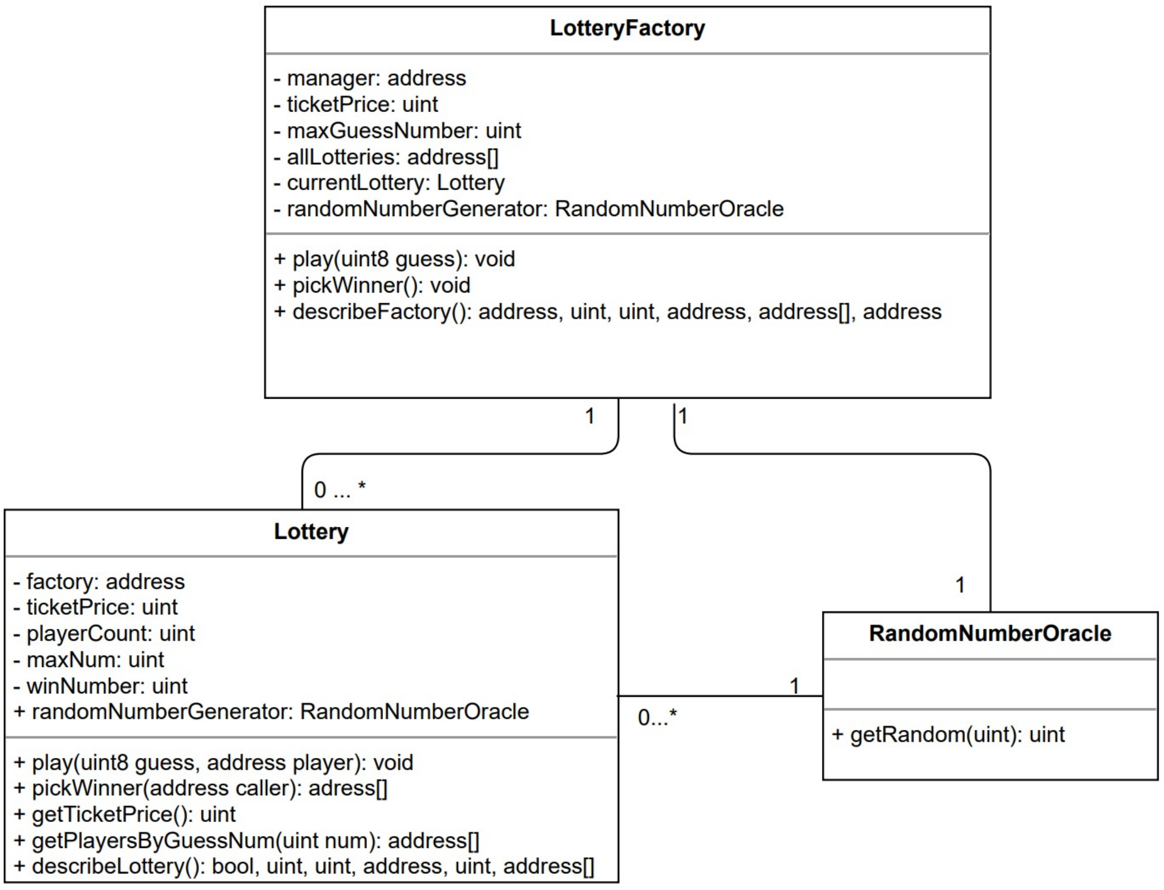
LotteryFactory
- manager: address
- ticketPrice: uint
- maxGuessNumber: uint
- allLotteries: address[]
- currentLottery: Lottery
- randomNumberGenerator: RandomNumberOracle
+ play(uint8 guess): void
+ pickWinner(): void
+ describeFactory(): address, uint, uint, address, address[], address
1
1
0 ... *
1
Lottery
- factory: address
- ticketPrice: uint
- playerCount: uint
- maxNum: uint
- winNumber: uint
+ randomNumberGenerator: RandomNumberOracle
+ play(uint8 guess, address player): void
+ pickWinner(address caller): adress[]
+ getTicketPrice(): uint
+ getPlayersByGuessNum(uint num): address[]
+ describeLottery(): bool, uint, uint, address, uint, address[]
1
0...*
RandomNumberOracle
+ getRandom(uint): uint

Figure 2.2: Class diagram of the lottery system.

```
        ticketPrice = _ticketPrice;
        maxGuessNumber = _maxGuessNumber;
        randomNumberGenerator = new RandomNumberOracle();
        currentLottery = new Lottery(_ticketPrice, address(this),
            address(randomNumberGenerator), maxGuessNumber);
        allLotteries.push(address(currentLottery));
    }
```

## 2.3 Playing the lottery

For depositing a bet a player calls the `LotteryFactory.play()` method. The `LotteryFactory` does basic checks for the validity of the bet and calls the currently active `Lottery`'s `play()` method. The amount of Ether sent with the bet are not transferred to the `Lottery` but are kept in the `LotteryFactory`. The `Lottery` only keeps track of the player address and his guess.

```
contract LotteryFactory {
    ...
    function play(uint8 guess) public payable {
        require(currentLottery != Lottery(address(0x0)),
            "There is no lottery running.");
```

```
      require(msg.value >= currentLottery.getTicketPrice(),
          "You have to send enough money.");
      currentLottery.play(guess, msg.sender);
  }
  ...
}

contract Lottery {
  ...
  function play(uint8 guess, address player) public {
      require(closed == false, "The lottery is closed.");
      require(guess <= maxNum, "Guess guess number not valid.");
      playersByNumber[guess].push(player);
      playerCount++;
  }
  ...
}
```

## 2.4 Picking a winner

The `LotteryFactory.pickWinner()` method can only be used by the issuer/manager of the `LotteryFactory`. The `LotteryFactory` forwards the call to the current `Lottery` which retrieves a random number from the `RandomNumberOracle` and returns the players that have bet on that number. The factory then distributes its current balance equally between the winners. If there are no winners, the balance of the factory remains untouched. The current `Lottery` is marked as closed and a new `Lottery` is created which becomes the new current lottery.

```
contract LotteryFactory {
  ...
    function pickWinner() public {
        require(msg.sender == manager, "You are not authorized.");
        address[] memory winners =
            currentLottery.pickWinner(address(this));
        if (winners.length != 0) {
            uint prize = address(this).balance / winners.length;
            for (uint i = 0; i < winners.length; i++) {
                winners[i].transfer(prize);
            }
        }
        currentLottery = new Lottery(ticketPrice, address(this),
            address(randomNumberGenerator), maxGuessNumber);
        allLotteries.push(address(currentLottery));
    }
    ...
}

contract Lottery {
  ...
    function pickWinner(address caller) public returns (address[]) {
      require(caller == factory,
          "You are not authorized to call this method.");
      require(closed == false, "The lottery is closed.");
      winNumber = randomNumberGenerator.getRandom(maxNum);
      closed = true;
      return (playersByNumber[winNumber]);
  ...
}
```

# Chapter 3

# Random Number Generator

# Chapter 4

# Frontend