FH JOANNEUM
University of Applied Sciences

Ethereum Lottery

using Ethereum Smart Contracts

Master Thesis
submitted in conformity with the requirements for the degree of
Master of Science in Engineering (MSc)
Master's degree programme **IT & Mobile Security**

FH JOANNEUM (University of Applied Sciences), Kapfenberg

supervisor: Gerhard Seuchter
submitted by: Georg Markowitsch
personal identifier: <1700000017>

August 2018

Formal declaration

I hereby declare that the present master's thesis was composed by myself and that the work contained herein is my own. I also confirm that I have only used the specified resources. All formulations and concepts taken verbatim or in substance from printed or unprinted material or from the Internet have been cited according to the rules of good scientific practice and indicated by footnotes or other exact references to the original source.

The present thesis has not been submitted to another university for the award of an academic degree in this form. This thesis has been submitted in printed and electronic form. I hereby confirm that the content of the digital version is the same as in the printed version.

I understand that the provision of incorrect information may have legal consequences.

Ehrenhausen a.d.W., August 2018

Georg Markowitsch

Abstract

Although Cryptocurrencies are gaining more and more importance in the modern world, it is mostly as a simple digital currency. However, with most of the digital coins you can do much more than simply pay for something. Ethereum is a perfect example for this.

Ethereum has an implementation which called Smart Contracts and they are just that. Smart Contracts allow the creator to define rules and conditions to distribute coins directly in the Blockchain of the Cryptocurrency since a Smart Contract is simply custom Code which gets executed if User interacts with it.

To explore the possibilities of Smart Contracts a demo /acDApp was created in the form of a Lottery. Especially with Lotteries there is always the problem that a trustworthy person or organization has to hold the money until the drawings and later pay the winners correctly.

With a Smart Contract all this becomes unnecessary because the Contract itself holds the coins on the Blockchain until the drawing and then every user can withdraw their winnings theirselfs. Moreover, the Smart Contract itself is also public and the Source Code can be read by anybody with basic porgramming skills, to check under which conditions the coins go from wallet A to wallet B.

Contents

List of Figures **ii**

1 Introduction **1**

2 Ethereum Lottery Rules **2**

3 Ethereum Lottery Rules **3**

4 Ethereum Lottery Rules **4**

5 Implementation **5**
5.1 Smart Contract with Solidity 5
5.2 Frontend with Truffle Framework 13
5.3 User Interface 14
5.4 Tests with Truffle Framework 15

6 Setup **19**
6.1 Local Test Setup 19

7 Ethereum Lottery Rules **21**

8 Ethereum Lottery Rules **22**

9 Ethereum Lottery Rules **23**

10 Conclusion and Outlook **24**

References **26**

List of Figures

5.1 Remix IDE 6
5.2 EtherLotto UI 14

Chapter 1

Introduction

Ethereum is a cryptocurrency similar to Bitcoin. Like Bitcoin, it has a blockchain to guarantee that transactions from one user to another are secure and safe. The blockchain holds all transactions in a linear, time-stamped series of bundled transactions known as blocks and the calculation of each block is depending on the one before. (mining) Essentially a blockchain is like a public ledger for recording transactions which is freely shared, continually updated and most important: under no central control.

The Ethereum cryptocurrency coin is called Ether. To make transactions on teh Ethereum Blockchain a certain fee is required to be paid. This fee is called Gas and can be set to a maximum amount before confirming and sending a transaction by the User.

Ethereum has its own implementation of a blockchain which expands the capabilities with Smart Contracts.

A Smart Contract is basically code written into the blockchain. Therefore it is public and everyone can read the source code and know the specifics of the contract, while the individuals involved are anonymous.

Some time after the Smart Contract is formed between the two parties a triggering event occurs. This can be pretty much anything, for example a expiration date. Now the contract executes itself according to the coded terms and enforces automatically all obligations. This automatic enforcing is the key difference to traditional contracts.

Chapter 2

Ethereum Lottery Rules

Following are the rules and conditions for the Lottery:

- Every user has to bid a fixed amount of EtherLotto. (0.2 ETH)
- Users can choose number to bid on between 1 and 999.
- After a predefined period of time, a random number will be generated.
- If any user submitted the generated number with his bid, he wins and can withdraw the whole pot.
- If there are two or more winners, the pot gets split equally.
- If there are no winners, the pot will be carried and can be won during the next round.

Chapter 3

Ethereum Lottery Rules

Following are the rules and conditions for the Lottery:

- Every user has to bid a fixed amount of EtherLotto. (0.2 ETH)
- Users can choose number to bid on between 1 and 999.
- After a predefined period of time, a random number will be generated.
- If any user submitted the generated number with his bid, he wins and can withdraw the whole pot.
- If there are two or more winners, the pot gets split equally.
- If there are no winners, the pot will be carried and can be won during the next round.

Chapter 4

Ethereum Lottery Rules

Following are the rules and conditions for the Lottery:

- Every user has to bid a fixed amount of EtherLotto. (0.2 ETH)
- Users can choose number to bid on between 1 and 999.
- After a predefined period of time, a random number will be generated.
- If any user submitted the generated number with his bid, he wins and can withdraw the whole pot.
- If there are two or more winners, the pot gets split equally.
- If there are no winners, the pot will be carried and can be won during the next round.

Chapter 5

Implementation

This Chapter provides infromation about Solidity, the programming language used for the EtherLotto Smart Contract, the frontend implementation as well as the tools and frameworks used during development.

5.1 Smart Contract with Solidity

The implementation of the Ethereum Smart Contract can be done in various programming languages. However, the most common and well documented one is Solidity and was therefore chosen for this project work.

Solidity is a contract-oriented, high-level language for implementing smart contracts. It was influenced by C++, Python and JavaScript and is designed to target the Ethereum Virtual Machine (EVM).

Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.

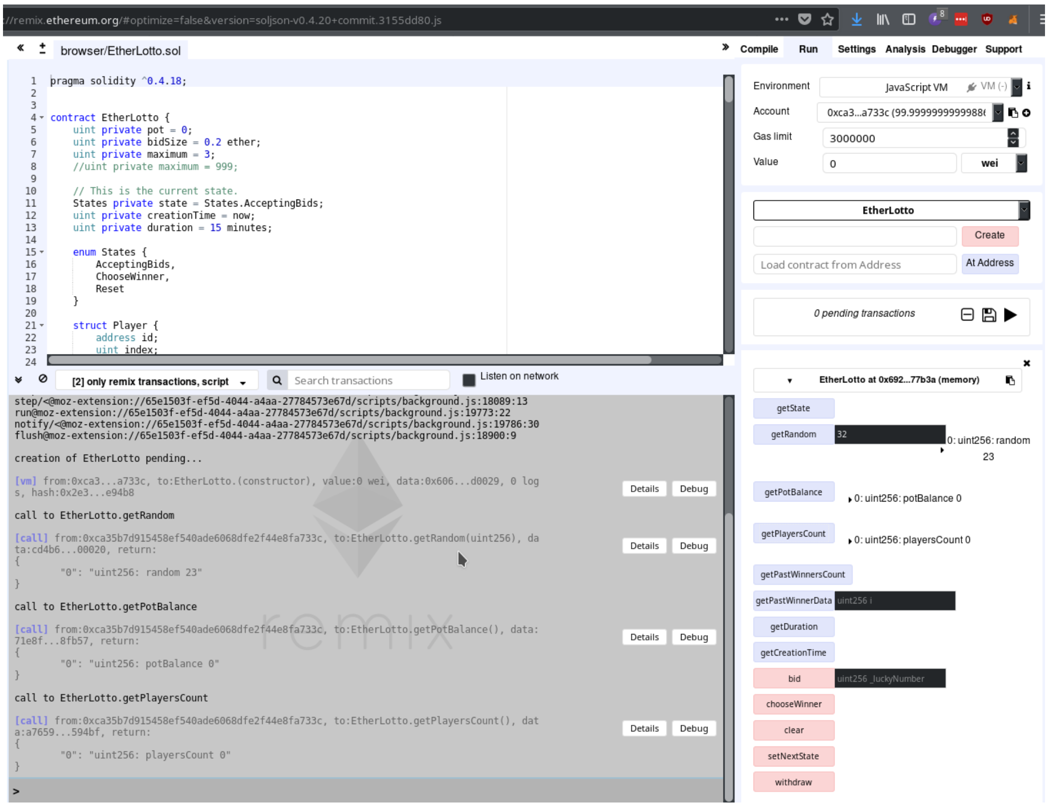
A simple Example in Solidity:

```
pragma solidity ^0.4.0; // defines the solidity version

contract SimpleStorage { // = collection of functions and data
    uint storedData; // unsigned integer variable

    // public setter
    function set(uint x) public {
        storedData = x;
    }

    // public getter
    function get() public constant returns (uint) {
        return storedData;
    }
}
```

Source: *Solidity*

5.1.1 Solidity Remix Integrated Development Environment (IDE)

For ease of use during development of the EtherLotto Smart Contract, the Remix IDE for Solidity was used. The Remix IDE is a Web IDE designed for Solidity Decentralized Application (DApp) developers. It offers syntax highlighting, static analysis and debugging tools via a JavaScript EVM running in the browser without any installation required by the developer.



```
pragma solidity ^0.4.18;


contract EtherLotto {
    uint private pot = 0;
    uint private bidSize = 0.2 ether;
    uint private maximum = 3;
    //uint private maximum = 999;

    // This is the current state.
    States private state = States.AcceptingBids;
    uint private creationTime = now;
    uint private duration = 15 minutes;

    enum States {
        AcceptingBids,
        ChooseWinner,
        Reset
    }

    struct Player {
        address id;
        uint index;
```

```
step/<@moz-extension://65e1503f-ef5d-4044-a4aa-27784573e67d/scripts/background.js:18089:13
run@moz-extension://65e1503f-ef5d-4044-a4aa-27784573e67d/scripts/background.js:19773:22
notify/<@moz-extension://65e1503f-ef5d-4044-a4aa-27784573e67d/scripts/background.js:19786:30
flush@moz-extension://65e1503f-ef5d-4044-a4aa-27784573e67d/scripts/background.js:18900:9

creation of EtherLotto pending...

[vm] from:0xca3...a733c, to:EtherLotto.(constructor), value:0 wei, data:0x606...d0029, 0 log
s, hash:0x2e3...e94b8

call to EtherLotto.getRandom

[call] from:0xca35b7d915458ef540ade6068dfe2f44e8fa733c, to:EtherLotto.getRandom(uint256), da
ta:cd4b6...00020, return:
{
        "0": "uint256: random 23"
}

call to EtherLotto.getPotBalance

[call] from:0xca35b7d915458ef540ade6068dfe2f44e8fa733c, to:EtherLotto.getPotBalance(), data:
71e8f...8fb57, return:
{
        "0": "uint256: potBalance 0"
}

call to EtherLotto.getPlayersCount

[call] from:0xca35b7d915458ef540ade6068dfe2f44e8fa733c, to:EtherLotto.getPlayersCount(), dat
a:a7659...594bf, return:
{
        "0": "uint256: playersCount 0"
}

>
```

Environment: JavaScript VM VM (-)
Account: 0xca3...a733c (99.999999999988(
Gas limit: 3000000
Value: 0 wei

EtherLotto

0 pending transactions

EtherLotto at 0x692...77b3a (memory)

getRandom: 32 — 0: uint256: random 23
getPotBalance — 0: uint256: potBalance 0
getPlayersCount — 0: uint256: playersCount 0
getPastWinnerData: uint256 i
bid: uint256 _luckyNumber

Figure 5.1: Remix IDE. Contract is running in a JavaScript EVM, all functions can be called and debugged.

5.1.2 Lottery States

A lottery usually goes through several states. From the start of the lottery, where bids are accepted, to the drawing of the winner and payout of the sum won.

Therefore in Solidity the following states where defined using Enums:

```
// This is the current state.
States private state = States.AcceptingBids;

    enum States {
    AcceptingBids,
    ChooseWinner,
    Reset
}
```

An Enum is a way in Solidity to create a user-defined type. They are explicitly convertible to and from all integer types but implicit conversion is not allowed. The explicit conversions check the value ranges at runtime and a failure causes an exception. Enums needs at least one member.

Based on these Enums and the variable state, it is possible to determine the current state of the Lottery and make only specific function available.

To transition between these states, custom function modifiers can be created. These modifiers not only handle the transition between states, but also restrict access to functions not allowed in the current state.

```
// modifier to restrict access
modifier atState(States _state) {
    // throws exception if condition not met
    require(state == _state);

    // code from the function being modified is inserted here
    _;
}

// function so set the next state
function nextState() internal {
    // increment state by 1
    state = States(uint(state) + 1);
}

// modifier to trasition to the next state after function execution
modifier transitionNext() {
    // code from the function being modified is inserted here
    _;

    nextState();
}

// transistion modifier with time condition
modifier timedTransitions() {
    if (state == States.AcceptingBids && now >= creationTime +
        duration)
         nextState();

    // code from the function being modified is inserted here
    _;
}
```

A modifier can then simply be added to a function declaration.

```
// only possible to call when state = Reset
function clear() public atState(States.Reset) {
    playerIndex = 0;
    players.length = 0;
    winnerIndex = 0;
    winners.length = 0;
    state = States.AcceptingBids;
    creationTime = now;
}
```

5.1.3 Player and Winner data struct

To save players, determine winners and in turn also save them, custom data types where necessary.

In Solidity this can be realized via Structs and is somewhat similar to Structs in C/C++.

```
struct Player {
    address id; // Ethereum address
    uint index; // player index
    uint luckyNumber; // number the player bet on
}

// array of players
Player[] private players;
uint private playerIndex = 0;

// array of winners in the current round
```

```
Player[] private winners;
uint private winnerIndex = 0;


struct PastWinner {
    Player player; // the player who won
    uint amount; // the amount won
    uint winningTime; // the time the player won
}

// array of all-time winners
PastWinner[] private pastWinners;
```

5.1.4 Events

Events allow the convenient usage of the EVM logging facilities, which in turn can be used to trigger JavaScript callbacks in the user interface of a DApp, which listen for these events.

However, in this Project Work Events are simply used to log relevent information.

```
// event declaration
event NewBidRecieved(address id, uint index, uint amount, uint luckyNumber);
event CalculatedNewRandom(uint random);
event StateChanged(States state);


function nextState() internal {
    state = States(uint(state) + 1);
    // trigger event
    StateChanged(state);
}
```

5.1.5 Placing a bid

The following function can be called to place a new bid:

```
function bid(uint _luckyNumber) public payable timedTransitions atState(States.AcceptingBids) {
    // check correct amount sent
    require(msg.value == bidSize);
    // check numbe rin range
    require(_luckyNumber != 0 && _luckyNumber < (maximum + 1));

    // trigger event
    NewBidRecieved(msg.sender, playerIndex, msg.value, _luckyNumber);

    // add amount sent to pot size
    pot += msg.value;
    // add new player to array
    insertPlayer(Player(msg.sender, playerIndex, _luckyNumber));
}
```

The modifier payable indicates that this function can recieve ether, timedTransitions and atState are explained in Section 5.1.2.

5.1.6 Random Generation

A random Generation in a deterministic Blockchain like the one Ethereum uses is impossible. Therefore the random number is calculated from the bblockhash of the next block generated after the state of the changed to ChooseWinner. In this state no additional bids are accepted. This method of calculating a pseudo-random number is potentially manipulateable, because a miner controlling about 50% the Ethereum hash calculation could influence the next blockhash by invalidating the calculated block if he chooses. Since this magnitude of manipulation is extremely resource heavy, it is no feasable to use it for such small amounts of Ether used in this demo /acDApp.

Function to get the pseudo-random number:

```
function getRandom(uint _maximum) public view returns (uint random) {
    return uint(block.blockhash(block.number-1)) % _maximum + 1;
}
```

5.1.7 Choosing a Winner

The following snippet chooses all winners for a lottery round and lets them withdraw their share of the pot.

```
function chooseWinner() public timedTransitions atState(States.ChooseWinner) {
    // get random number
    uint winningNumber = getRandom(maximum);

    // trigger event
    CalculatedNewRandom(winningNumber);

    // get winners and insert them into winners array
    getWinners(winningNumber);

    if (winners.length > 0) {
        // set the amount each winner can withdraw
        setPendingWithdrawals();
    }

    // transition to Reset state
    nextState();
    clear();
}

function getWinners(uint winningNumber) internal {
    for (uint i = 0; i < playerIndex; i++) {
        if (players[i].luckyNumber == winningNumber) {
            insertWinner(players[i]);
        }
    }
}

function setPendingWithdrawals() internal {
    uint potPerWinner = pot / winners.length;
    pot = 0;

    for (uint i = 0; i < winnerIndex; i++) {
        pendingWithdrawals[winners[i].id] += potPerWinner;
        pastWinners.push(PastWinner(winners[i], potPerWinner, now));
    }
}
```

5.1.8 Withdrawing Ether

The recommended method of sending funds after an effect is using the withdrawal pattern. Although the most intuitive method of sending Ether, as a result of an effect, is a direct send call, this is not recommended as it introduces a potential security risk.

```
function withdraw() public {
    uint amount = pendingWithdrawals[msg.sender];
    // Remember to zero the pending refund before
    // sending to prevent re-entrancy attacks
    pendingWithdrawals[msg.sender] = 0;
    msg.sender.transfer(amount);
}
```

5.2 Frontend with Truffle Framework

Truffle is a JavaScript development framework for Ethereum. It provides built-in Smart Contract compilation, linking, deployment and binary management via an interactive console called the truffle environment or deployment scripts. In addition to that it also provides an environment for automated contract testing using either Solidity or JavaScript.

More information on truffle and all its features can be found in the *Truffle Documentation* and *Truffle Tutorials*.

Source: *Truffle*

Using truffle it is relatively easy to build a frontend for any type of interaction with a Smart Contract deployed on the Blockchain. Since the Smart Contract is also compiled and deployed with truffle, there are no extra steps necessary to give the frontend implementation the necessary artifact files. Artifact files provide the interface to communicate with a deployed Smart Contract and its functions.

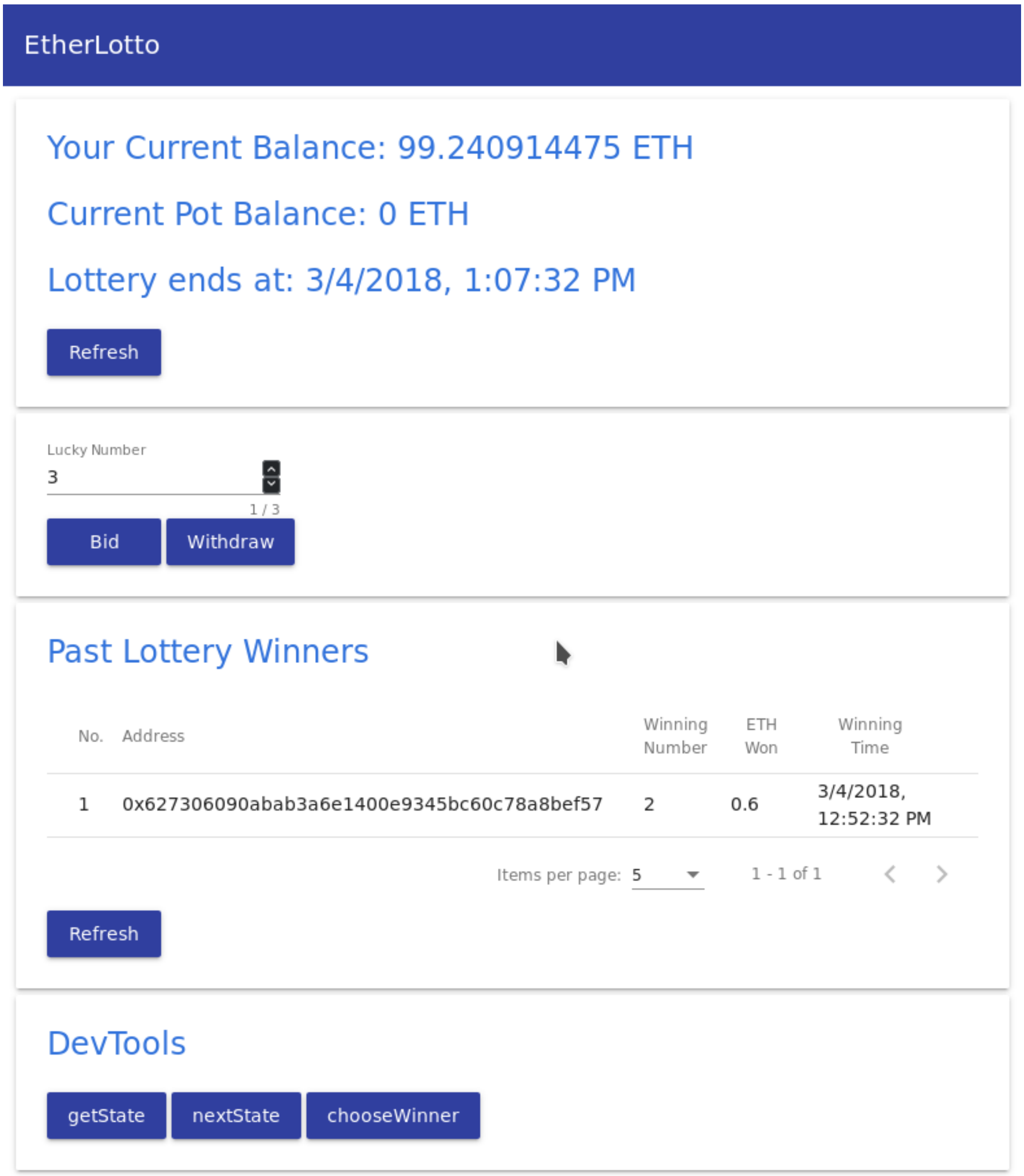
All that is left to do before communcation is possible, is to spcify a Provider. The Provider holds information on which Ethereum network to use and how to communicate with it without running a full Ethereum node which is usually needed to make transaction on the Ethereum Blockchain. At the time of writing, the easiest way to provide this is a broweser addon called /citetitlemetamask. /citetitlemetamask includes a secure identity vault, providing a user interface to manage your identities on different sites and sign blockchain transactions.

The following shows an Example of how to call a Smart Contract function with JavaScript using Promises for safe execution and error handling.

```
withdraw = () => {
    this.EtherLotto
        .deployed()
        .then(instance => {
        return instance.withdraw({from: this.account});
    }).then(() => {
        console.log('Withdraw complete!');
        this.getBalance();
    }).catch(function(e) {
        console.log(e);
        this.setStatus('Error withdrawing coins; see log.');
    });
};
```

5.3 User Interface

The User Interface for this demo DApp was kept as simple as possible providing a Material Design approach using *Angular Material*.



EtherLotto

Your Current Balance: 99.240914475 ETH

Current Pot Balance: 0 ETH

Lottery ends at: 3/4/2018, 1:07:32 PM

Lucky Number: 3

1 / 3

Past Lottery Winners

No.	Address	Winning Number	ETH Won	Winning Time
1	0x627306090abab3a6e1400e9345bc60c78a8bef57	2	0.6	3/4/2018, 12:52:32 PM

Items per page: 5 1 - 1 of 1

DevTools

Figure 5.2: EtherLotto Material Design User Interface

5.4 Tests with Truffle Framework

As already mentioned in Section Frontend with Truffle Framework, truffle comes standard with automated contract testing framework using either Solidity or JavaScript.

Truffle provides a clean room environment when running test files. When running tests against *Ganache*[1] or Truffle Develop, Truffle will use advanced snapshotting features to ensure test files don't share their state with each other.

5.4.1 Solidity Tests

Solidity tests are included as a seperate test suite per test-contract during execution, have direct access to deployed contracts and can potentially import andy contract dependency. In addition to that, the assertion library can be changed at any time and tests can be run against any Ethereum client. More information can be found in the *Truffle Documentation*.

Example of a simple Solidity test:

```
pragma solidity ^0.4.2;

import "truffle/Assert.sol";
import "truffle/DeployedAddresses.sol";
import "../contracts/EtherLotto.sol";


contract TestEtherLotto {
    // Truffle will send the TestContract one Ether after deploying the contract.
    uint public initialBalance = 1 ether;

    function testInitialPotBalance() public {
        EtherLotto etherLotto = EtherLotto(DeployedAddresses.EtherLotto());

        uint expextedPot = 0;

        Assert.equal(etherLotto.getPotBalance(), expextedPot, "Initial Pot Balance should always be 0");
    }
}
```

[1] Ganache provides a local Blockchain to run tests, execute commands and inspect states while controllin ghow the chain operates

5.4.2 Javascript Tests

Truffle uses the *MochaJs* unit testing framework and therefore works mostly similiar to a standard Mocha test. The only exception is that instead of `describe()`, the truffle specific `contract()` function is used. This enables all clean room features described in Section Tests with Truffle Framework.

This means that:

- Before each `contract()` function is run, all contracts are re-deployed to provide a clean state.
- The `contract()` function provides a list of test accounts made available by the Ethereum client.

The following code snippet shows a JavaScript test to validate the pot balance after one recieved bid using the NewBidRecieved Event implemented in the EtherLotto Smart Contract.

```
var EtherLotto = artifacts.require("EtherLotto");

contract(’EtherLotto’, function(accounts) {
  it("test contract deployed", function() {
    return EtherLotto.deployed().then(function(instance) {
      assert(instance != undefined, "contract is not correctly
         deployed");
    });
  });

  it("test pot balance after one bid", function() {
    var etherLotto;
    var expectedBid = 2000000000000000000;
    var expectedLuckyNumber = 1;
    var expectedIndex = 0;

    return EtherLotto.deployed().then(function(instance) {
      etherLotto = instance;
      // execute bid function
      return etherLotto.bid(expectedLuckyNumber, {from: accounts[0],
          value: expectedBid});
    }).then(function(result) {

      var recievedBidValue;
      var recievedPlayerAddress;
      var recievedLuckyNumber;
      var recievedIndex;

      // parse result for NewBidRecieved-Event
      for (var i = 0; i < result.logs.length; i++) {
        var log = result.logs[i];

        if (log.event == "NewBidRecieved") {
          recievedPlayerAddress = log.args.id;
          recievedBidValue = log.args.amount.toNumber();
          recievedLuckyNumber = log.args.luckyNumber.toNumber();
          recievedIndex = log.args.index.toNumber();
        }
      }

      assert.equal(recievedPlayerAddress, accounts[0], "address from event and sender address need to match");
      assert.equal(recievedBidValue, expectedBid, "bid value from event and sent bid have to match");
      assert.equal(recievedIndex, expectedIndex, "index should match");
      assert.equal(recievedLuckyNumber, expectedLuckyNumber, "lucky Number should match");

      return etherLotto.getPotBalance();
    }).then(function(balance) {
      // check pot balance
      assert.equal(balance.toNumber(), expectedBid, "wrong value in potBalance, bid went wrong");
    });
  });
});
```

5.4.3 Executing Tests

The easiest way to run Truffle tests is with the built-in Ethereum Client and interactive shell. To start it simply run the following command in the root directory of the truffle project.

```
$ truffle develop
```

This will start a local Ethereum Blockchain and generate accounts with some Ether in their wallet.

Using the interactive shell, the tests can now be run directly.

```
truffle(develop)> test
Using network 'develop'.

Compiling ./contracts/EtherLotto.sol...
Compiling ./test/TestEtherLotto.sol...
Compiling truffle/Assert.sol...
Compiling truffle/DeployedAddresses.sol...


  TestEtherLotto
     testInitialPotBalance (71ms)

  Contract: EtherLotto
     test contract deployed
     test pot balance after one bid (55ms)

  Contract: EtherLotto
     test a full lottery (577ms)


  4 passing (1s)
```

Chapter 6

Setup

This explains how to setup and run the EthereLotto DApp on any Linux/Unix device. Prerequistes:

- A Linux/Unix shell (tested on MacOS x 10.11 and Arch Linux)
- Node package manager *npm*

6.1 Local Test Setup

To install the *Truffle* and *Angular Material* dependencies, load the project and install project dependencies.

```
$ sudo npm install -g truffle
$ sudo npm install -g @angular/cli
$ git clone https://github.com/winterwurzel/EtherLotto.git && cd
    etherlotto_v2
$ npm install
```

To run the DApp, a local Ethereum Testnetwork is needed. While *Truffle* comes with a test network, *Ganache* is the preferred way to provide it.

Either start the *Truffle* test network with `truffle develop` or simply run *Ganache* (preferred).

Additionally the Browser Extension *MetaMask* is needed. Install and configure *MetaMask* to use the local test network (and port 7545) and import the first account through the Mnemonic specified in `truffle develop` or *Ganache*. Additional Accounts can be import through their private keys.

Now the DApp can be started and will be available in a browser at http://localhost:4200/.

```
$ cd etherlotto_v2
$ truffle compile
$ truffle migrate
$ ng serve
```

Chapter 7

Ethereum Lottery Rules

Following are the rules and conditions for the Lottery:

- Every user has to bid a fixed amount of EtherLotto. (0.2 ETH)
- Users can choose number to bid on between 1 and 999.
- After a predefined period of time, a random number will be generated.
- If any user submitted the generated number with his bid, he wins and can withdraw the whole pot.
- If there are two or more winners, the pot gets split equally.
- If there are no winners, the pot will be carried and can be won during the next round.

Chapter 8

Ethereum Lottery Rules

Following are the rules and conditions for the Lottery:

- Every user has to bid a fixed amount of EtherLotto. (0.2 ETH)
- Users can choose number to bid on between 1 and 999.
- After a predefined period of time, a random number will be generated.
- If any user submitted the generated number with his bid, he wins and can withdraw the whole pot.
- If there are two or more winners, the pot gets split equally.
- If there are no winners, the pot will be carried and can be won during the next round.

Chapter 9

Ethereum Lottery Rules

Following are the rules and conditions for the Lottery:

- Every user has to bid a fixed amount of EtherLotto. (0.2 ETH)
- Users can choose number to bid on between 1 and 999.
- After a predefined period of time, a random number will be generated.
- If any user submitted the generated number with his bid, he wins and can withdraw the whole pot.
- If there are two or more winners, the pot gets split equally.
- If there are no winners, the pot will be carried and can be won during the next round.

Chapter 10

Conclusion and Outlook

As this Project Work shows, it is possible to build a self-regulating Lottery application which holds all currency bet until a winner can be determined.

Using this approach is not only fully transparent, due to the implementation of the Ethereum Blockchain and Smart Contracts, but also provably fair to everyone who places bids in the Lottery.

Acronyms

EVM	Ethereum Virtual Machine
DApp	Decentralized Application
IDE	Integrated Development Environment

Bibliography

ConsenSys (Feb. 15, 2018a). *Ganache*. Available from: http://truffleframework.com/ganache/ [Feb. 15, 2018] (cit. on pp. 15, 19).

– (Feb. 15, 2018b). *Truffle*. Available from: http://truffleframework.com/ [Feb. 15, 2018] (cit. on pp. 13, 19).

– (Feb. 15, 2018c). *Truffle Documentation*. Available from: http://truffleframework.com/docs/ [Feb. 15, 2018] (cit. on pp. 13, 15).

– (Feb. 15, 2018d). *Truffle Tutorials*. Available from: http://truffleframework.com/tutorials/ [Feb. 15, 2018] (cit. on p. 13).

Ethereum (Feb. 15, 2018). *Solidity*. Available from: http://solidity.readthedocs.io/en/develop/index.html [Feb. 15, 2018] (cit. on p. 6).

Google (Feb. 15, 2018). *Angular Material*. Available from: https://material.angular.io/ [Feb. 15, 2018] (cit. on pp. 14, 19).

kumavis, Dan Finlay, Thomas Huang, Frankie Pangilinan, Kevin Serrano, James Moreau, Christian Jeria, Ethereum DEV Grants, ConsenSys (Feb. 15, 2018). *MetaMask*. Available from: https://metamask.io/ [Feb. 15, 2018] (cit. on p. 19).

MochaJs (Feb. 15, 2018). Available from: https://mochajs.org/ [Feb. 15, 2018] (cit. on p. 16).

npm, Inc. (Feb. 15, 2018). *npm*. Available from: https://www.npmjs.com/ [Feb. 15, 2018] (cit. on p. 19).