Challenge Task 2018

Implementation of a Decentralized Application Tic Tac Toe

Departements of Informatics - Communication Systems Group, Chair

Lucas Pelloni, leginumeber
Severin Wullschleger leginumber
Andreas Schaufelbühl, 12-918-843

University of Zurich[UZH]

CSG
Department of Informatics
Communication Systems Group

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

This years Challenge Task (CT) is to implement a Decentralized Application (DApp) running in the Ethereum blockchain. The goal of the application is a playable Tic-Tac-Toe [1] game, which also includes a betting system, all embedded in a SC.

Chapter 2 gives an overview and short explanation of the technologies we use in order to implement the CT.
In Chapter 3 we show the actual implementation of the game. It starts by explaining and showing our project structure. Also we give walk-through of the different processes of playing a game and betting on games.
The problems and challenges occurred within our project are discussed in Chapter 4. Additionally we also describe our open task and goals for the future concerning this project.

[1] https://en.wikipedia.org/wiki/Tic-tac-toe

Chapter 2

TECHNOLOGIES

With Solidity [1] we implement the SC which will run on the Ethereum blockchain platform. For our front-end we choose using React [2] , which is a JavaScript library for building user interfaces. The interaction of the front-end application with our SC is provided through Web3.js [3] and MetaMask [4] . To speed up the testing and development we use Ganache [5] to run our local Etherum blockchain In the following section we describe the different technologies and its use in our project more in detail.

2.1 Solidity

2.2 Web3.js

2.3 MetaMask

2.4 Ganache

[1] https://github.com/ethereum/solidity

[2] https://reactjs.org/

[3] https://web3js.readthedocs.io/en/1.0/

[4] https://metamask.io/

[5] http://truffleframework.com/ganache/

Chapter 3

IMPLEMENTATION OF THE GAME

3.1 The Smart Contract

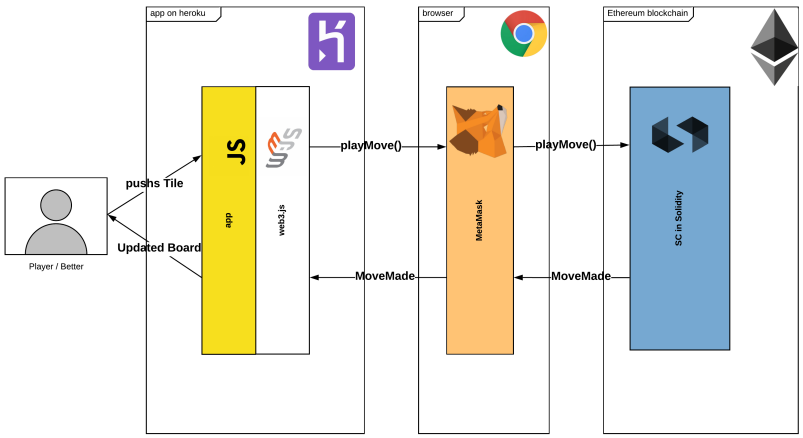

Figure 3.1: Project Structure with Technologies

Figure 3.1 shows the project structure and the interaction between the different systems by the example of an Player choosing a tile on the board. The web-application is running on the Heroku Platform [1] . Through the browser and MetaMask a User can get verified by its Ethereum-account and pay the requested amount of gas in order to run functionalities on the Ethereum SC. The SC itself runs on an blockchain, which can be either a private or the Ropsten Testnet [2] .

[1] https://www.heroku.com/

[2] https://ropsten.etherscan.io/

The SC firstly checks if the move is valid. Secondly it looks for a winner and changes the game state if so. After that it returns a move confirmation to the user.

Three functions of the SC we show here, as we consider them complex and interesting:

3.1.1 1.Play Move Function

```
1event MoveMade(bool success, uint gameId, GameState state, uint x,
    uint y, string symbol);
2function playMove(uint gameId, uint x, uint y) public {
3 Game storage game = games[gameId];
4
5 require(game.state >= GameState.X_HAS_TURN, "The game is not
    started yet.");
6 require(game.state < GameState.WINNER_X, "The game is already
    finished.");
7
8 game.moveCounter += 1;
9
10 if (game.state == GameState.X_HAS_TURN) {
11  require(game.playerXAddr == msg.sender
12    game.moveCounter == boardSize * boardSize// last move made
      automatically
13   , "Sender not equal player X");
14  require(game.board[y][x] == SquareState.EMPTY
15  ,"Move not possible because the square is not empty.");
16
17  game.board[y][x] = SquareState.X;
18  game.state = GameState.O_HAS_TURN;
19  checkForWinner(x, y, gameId, game.playerXAddr);
```

```
20
21   emit MoveMade(true, gameId, game.state, x, y, "X");
22  }
23  else {
24   require(game.playerOAddr == msg.sender
25     game.moveCounter == boardSize * boardSize      // last move made
         automatically
26    , "Sender not equal player O");
27   require(game.board[y][x] == SquareState.EMPTY
28    , "Move not possible because the square is not empty.");
29
30   game.board[y][x] = SquareState.O;
31   game.state = GameState.X_HAS_TURN;
32   checkForWinner(x, y, gameId, game.playerOAddr);
33
34   emit MoveMade(true, gameId, game.state, x, y, "O");
35  }
36  if (game.moveCounter == boardSize*boardSize - 1 && game.state <
      GameState.WINNER_X) {
37   doLastMoveAutomatically(game);
38  }
39 }
```

This functions gets called when a player clicks on a tile in order to make his move.The game id is send as an parameter (line 2). With this id the corresponding game object can be called through a mapping (line 3) It adds the players symbol into the specific location within the game-board firstly (line 17 and 30). Afterwards it calls the 'checkForWinner' function (line 19 and 32). The event MoveMade (line 1) is triggered returning the user an confirmation of the move (line 21 and 34). There is also an an auto completion for the last move if there is no winner set yet (line 37).

The class-diagram in Figure3.2 show a detail class diagram explaining the modelling and structure of our SC

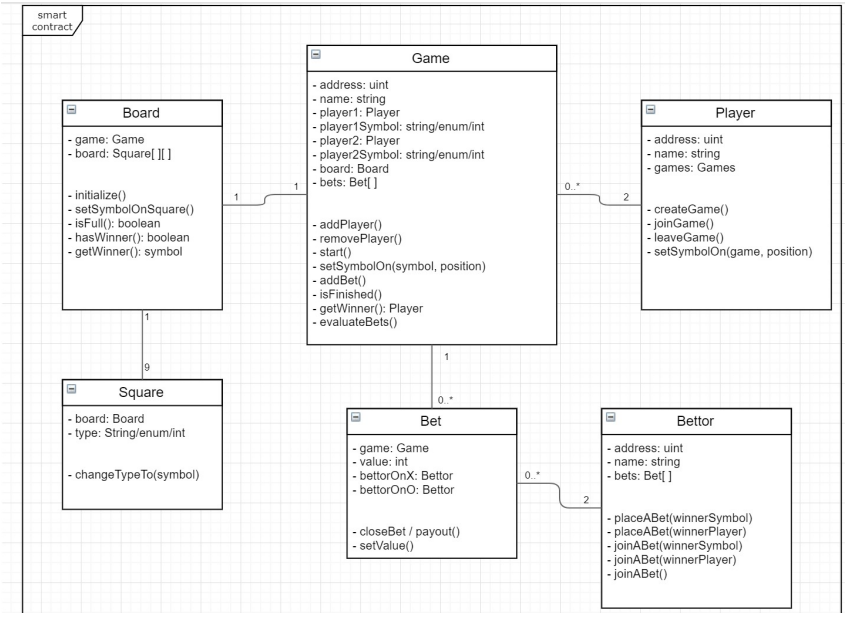

Figure 3.2: Class-Diagram of the SC

3.2 Game Walk-through

Chapter 4

DISCUSION

4.1 Challenges and Problems

4.2 Future work

APPENDIX A

RAW DATA