Challenge Task 2018

Implementation of a Decentralized Application Tic Tac Toe

Departements of Informatics - Communication Systems Group, Chair

Lucas Pelloni, 13-722-038
Severin Wullschleger leginumber
Andreas Schaufelbühl, 12-918-843

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. 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] . Furthermore, the application provides also the possibility to interact with a localhost provider using Ganache [5] as Ethereum node 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

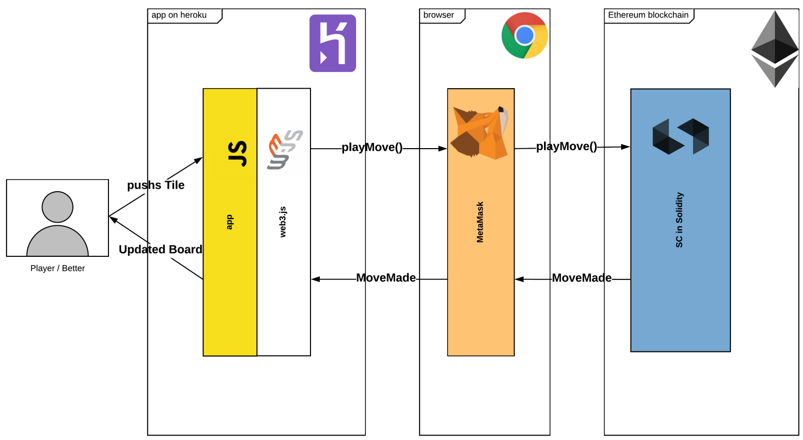

Figure 3.1: Project Structure with Technologies

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:

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

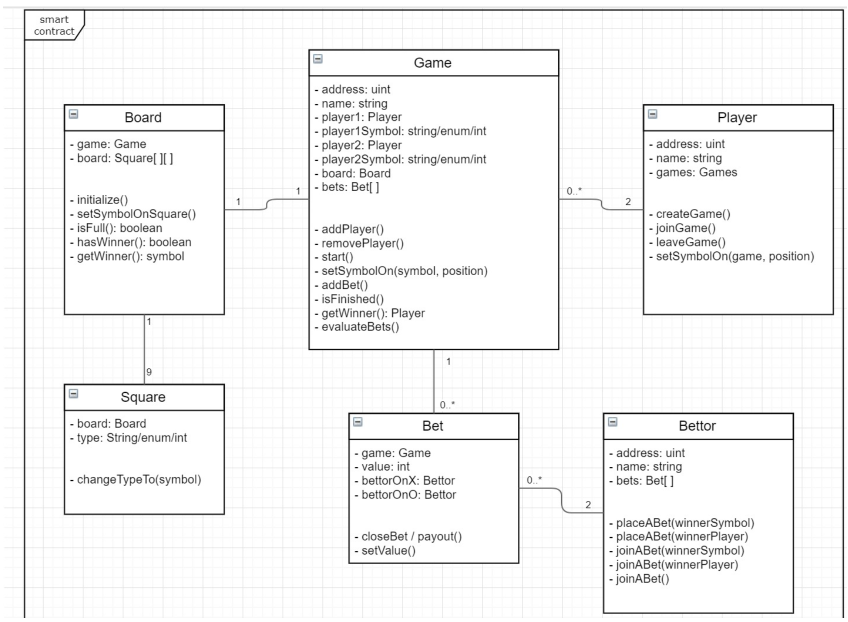

Figure 3.2: Class-Diagram of the SC

3.1.1 Play Move Function

```
event MoveMade(bool success, uint gameId, GameState state, uint x, uint y, string symbol);
function playMove(uint gameId, uint x, uint y) public {
 Game storage game = games[gameId];

 require(game.state >= GameState.X_HAS_TURN, "The game is not started yet.");
 require(game.state < GameState.WINNER_X, "The game is already finished.");

 game.moveCounter += 1;

 if (game.state == GameState.X_HAS_TURN) {
  require(game.playerXAddr == msg.sender
    game.moveCounter == boardSize * boardSize// last move made automatically
   , "Sender not equal player X");
  require(game.board[y][x] == SquareState.EMPTY
  ,"Move not possible because the square is not empty.");

  game.board[y][x] = SquareState.X;
  game.state = GameState.O_HAS_TURN;
  checkForWinner(x, y, gameId, game.playerXAddr);

  emit MoveMade(true, gameId, game.state, x, y, "X");
 }
 else {
  require(game.playerOAddr == msg.sender
    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}
```

Listing 3.1: Play Move Function on the Smart Contract

The code of the playMove function is shown in Listing 3.1. This 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).

3.1.2 Check for Winner

```
function checkForWinner(uint x, uint y, uint gameId, address
   currentPlayer) private {
 Game storage game = games[gameId];

 //is winning already possible?
 if (game.moveCounter < 2 * boardSize - 1)
  return;
 }

 SquareState symbol = game.board[y][x];

 //check column
 for (uint i = 0; i < boardSize; i++) {
  if (game.board[i][x] != symbol) {
   break;
  }
  else if (i == (boardSize - 1)) {
   game.winnerAddr = currentPlayer;
   game.state = getGameState(symbol);
   payoutBets(game.gameId);
   return;
  }
 }

 //check row
 for (i = 0; i < boardSize; i++) {
  if (game.board[y][i] != symbol) {
   break;
  }
```

```
  else if (i == (boardSize - 1)) {
   game.winnerAddr = currentPlayer;
   game.state = getGameState(symbol);
   payoutBets(game.gameId);
   return;
  }
 }

 //check diagonal: (x-y) 0-0, 1-1, 2-2
 if (x == y) {
  for (i = 0; i < boardSize; i++) {
   if (game.board[i][i] != symbol) {
    break;
   }
   else if (i == (boardSize - 1)) {
    game.winnerAddr = currentPlayer;
    game.state = getGameState(symbol);
    payoutBets(game.gameId);
    return;
   }
  }
 }

 // check antidiagonal: (x-y) 2-0, 1-1, 0-2
 if (x + y == (boardSize - 1)) {
  for (i = 0; i < boardSize; i++) {
   if (game.board[i][boardSize - 1 - i] != symbol) {
    break;
   }
   else if (i == (boardSize - 1)) {
    game.winnerAddr = currentPlayer;
```

```
60    game.state = getGameState(symbol);
61    payoutBets(game.gameId);
62    return;
63   }
64  }
65 }
66 //check for draw
67 if (game.moveCounter == boardSize * boardSize) {
68  game.state = GameState.DRAW;
69  payoutBets(game.gameId);
70 }
71}
```
Listing 3.2: Check for Winner Function on the Smart Contract

The logic for the evaluating the winner function is shown in Listing 3.2. The function stop if the number of moves done are not enough to possibly have a winner (line 5). After it goes through all possible combination of winning and checks if there are the same symbol a line (line 12, 25, 38 and 53). If it does not found any winning row it checks at last for draw (line 67). Is there a winning line found or the game is a draw, the payout function gets called (line 19, 32, 46, 61 and 69).

3.1.3 Payout

```
function payoutBets(uint gameId) internal {
 for (uint i = 0; i < openBetIds.length; i++)
  Bet storage iBet = bets[openBetIds[i]];

  if (iBet.gameId == gameId) {
   if (iBet.state == BetState.FIXED) {

    // bettorOnX wins
    if (games[gameId].state == GameState.WINNER_X) {
     (iBet.bettorOnXAddr).transfer(SafeMath.mul(2, iBet.value));

    // bettorOnO wins
    } else if (games[gameId].state == GameState.WINNER_O) {
     (iBet.bettorOnOAddr).transfer(SafeMath.mul(2, iBet.value));

    // draw
    } else {
     (iBet.bettorOnOAddr).transfer(iBet.value);
     (iBet.bettorOnXAddr).transfer(iBet.value);
    }
    iBet.state = BetState.PAYEDOUT;
   }

   // handle not fixed bets: transfer value back to owner
   if (iBet.state == BetState.MISSING_O_BETTOR) {
      (iBet.bettorOnXAddr).transfer(iBet.value);
      iBet.state = BetState.WITHDRAWN;
   }
   if (iBet.state == BetState.MISSING_X_BETTOR) {
```

```
30    (iBet.bettorOn0Addr).transfer(iBet.value);
31    iBet.state = BetState.WITHDRAWN;
32   }
33  }
34 }
35}
```

Listing 3.3: Payout Function on the Smart Contract

If there is a winner or a draw the tokens reserved for the particular game get paid out with the payout method describe in Listing 3.3. As it can have multiple bets on one game the function runs through all bets checking if they are referencing on the finished game (line 2-5). SafeMath is used for the payout to assure the amount is correct and prevent overflow (line 10 and 14). When a bettor has not found an opponent to go along with his bet, the function pays back the value to the origin bettor (line 25-32)

Chapter 4

DISCUSION

4.1 Challenges and Problems

4.2 Future work

APPENDIX A

RAW DATA