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 13-715-081
Andreas Schaufelbühl, 12-918-843

TABLE OF CONTENTS

1 Introduction 1

2 Technologies 2

3 Implementation of the game 4

3.1 The Smart Contract 4

3.1.1 Play Move Function 6

3.1.2 Check for Winner 8

3.1.3 Payout 10

4 Discussion 11

4.1 Challenges and Problems 11

4.2 Future work 11

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 Smart Contract (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 implementation of the game. It starts by giving an overview of the smart contract. After we show chosen functions in the code and discuss their logic in detail
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.

Domain: http://www.tictactoe.bet
Source Code: https://github.com/lucaspelloni2/dapp-tic-tac-toe

[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.

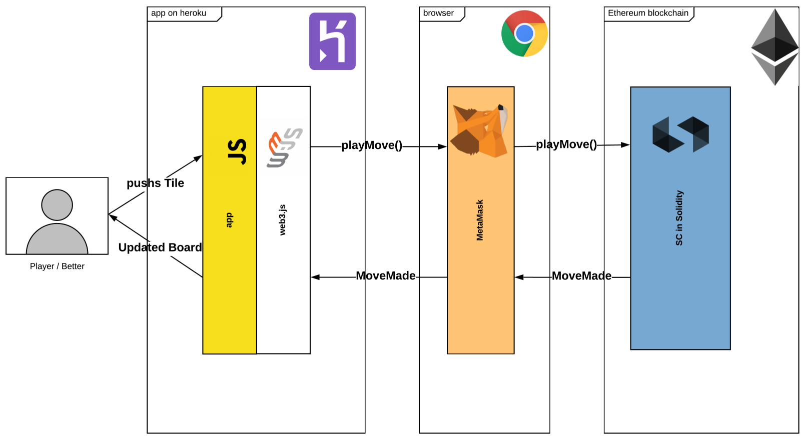

Figure 2.1: The interconnection of the different technologies

[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/

Figure 2.1 shows the project structure and the interaction between these different systems by the example of an Player choosing a tile on the board. The web-application is running on the Heroku Platform [6] . 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 [7] .

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

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

Chapter 3

IMPLEMENTATION OF THE GAME

3.1 The Smart Contract

The diagram in Figure3.1 shows a detail structure explaining the modelling of our SC.

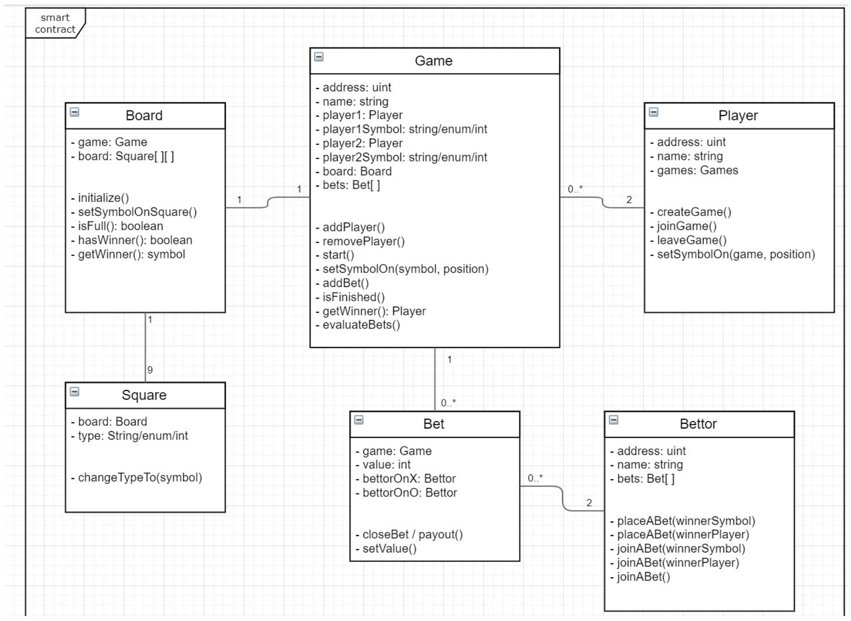

Figure 3.1: Structure of the SC

The SC can hold multiple games, which always contain one board and two players. Every time a user opens up a new Game, he automatically gets assigned as one of the playing party. Now the Game state is automatically set to 'WAITING_FOR_X'. As soon as a second player chooses to enter a open game, the game-owner can start the game, which changes the state to 'X_HAS_TURN' as the game has now started and the guest player has to do his first move.

All Bets contain a game-id pointing to a game which the bet is referencing on it. The user who creates a bet chooses the game and the amount of token he wants to bet with and pays it into the SC. Is there an opponent bettor joining the bet, the state changes to 'FIXED'.
If the game finds an end, the state of it changes to either 'WINNER_X', 'WINNER_O' or 'DRAW', which activates the function 'payout' ,triggering all bets referencing to this particular game. So the bets will change the state to 'PAYOUT' and user will get paid according to their betting.
Three functions of the SC we show here, to discuss in detail the implementation and structure of the SC:

3.1.1 Play Move Function

```
1 event MoveMade(bool success, uint gameId, GameState state, uint x, uint y, string symbol);
2 function 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 }
```

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 sent 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;
   }
  }
 }

```

```
52 // check antidiagonal: (x-y) 2-0, 1-1, 0-2
53 if (x + y == (boardSize - 1)) {
54  for (i = 0; i < boardSize; i++) {
55   if (game.board[i][boardSize - 1 - i] != symbol) {
56    break;
57   }
58   else if (i == (boardSize - 1)) {
59    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 stops if the number of moves played 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 within a line (line 12, 25, 38 and 53). If it does not found any winning line it checks at last for a 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

```
 1 function payoutBets(uint gameId) internal {
 2  for (uint i = 0; i < openBetIds.length; i++)
 3   Bet storage iBet = bets[openBetIds[i]];
 4 
 5   if (iBet.gameId == gameId) {
 6    if (iBet.state == BetState.FIXED) {
 7 
 8     // bettorOnX wins
 9     if (games[gameId].state == GameState.WINNER_X) {
10      (iBet.bettorOnXAddr).transfer(SafeMath.mul(2, iBet.value));
11 
12     // bettorOnO wins
13     } else if (games[gameId].state == GameState.WINNER_O) {
14      (iBet.bettorOnOAddr).transfer(SafeMath.mul(2, iBet.value));
15 
16     // draw
17     } else {
18      (iBet.bettorOnOAddr).transfer(iBet.value);
19      (iBet.bettorOnXAddr).transfer(iBet.value);
20     }
21     iBet.state = BetState.PAYEDOUT;
22    }
23 
24    // handle not fixed bets: transfer value back to owner
25    if (iBet.state == BetState.MISSING_O_BETTOR) {
26       (iBet.bettorOnXAddr).transfer(iBet.value);
27       iBet.state = BetState.WITHDRAWN;
28    }
29    if (iBet.state == BetState.MISSING_X_BETTOR) {
30     (iBet.bettorOnOAddr).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

DISCUSSION

4.1 Challenges and Problems

As the language of solidity is pretty new, the functionalities are quiet limited. Writing the SC we were facing the problem of not being able to transact strings from SC to the front-end and reverse. Yet we could achieve the same result with sending bytes32 data and transform them in the front-end. So we are able to store strings as game name, player name etcetera.
Also the lack of good documentation was sometimes raising up some challenges, although with the help of StackOverFlow [1] most of the problems found their solutions.

4.2 Future work

Our goal is to release our game on the Ethereum main network. A major challenge in order to achieve that is the reduction of the amount of gas required. All the different transaction are not optimized in terms of minimal gas reduction. Also the code in our SC is very long. As a result the gas usage of transaction is high.
To tackle this problem Prof. Thomas Bocek will audit our SC, so we can achieve with his help and guidance an major gas reduction.

Besides some minor GUI improvements a second goal is to get an free https certificate with deploying our game through GitHub Pages [2] . As a third goal we aim to

[1]https://stackoverflow.com/

[2]https://pages.github.com/

submit our DApp to State of the Apps [3] , a platform for open DApp's, so we reach a broader blockchain audience and let them play and bet on our game.

[3]https://www.stateofthedapps.com/