

## 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



**University of  
Zurich**<sup>UZH</sup>





## TABLE OF CONTENTS

	Page
LIST OF TABLES .....	iv
LIST OF FIGURES .....	v
CHAPTER	
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 .....	11
4 Discusion .....	13
4.1 Challenges and Problems .....	13
4.2 Future work .....	13
A Raw Data .....	14
REFERENCES .....	14

## LIST OF TABLES

Table	Page
-------	------

## LIST OF FIGURES

Figure	Page
2.1 The interconnection of the different technologies.....	3
3.1 Class-Diagram of the Smart Contract (SC) .....	4

## 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 <sup>1</sup> 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.

---

<sup>1</sup><https://en.wikipedia.org/wiki/Tic-tac-toe>

## Chapter 2

### TECHNOLOGIES

With Solidity <sup>1</sup> we implement the SC which will run on the Ethereum blockchain. For our front-end we choose using React <sup>2</sup>, which is a JavaScript library for building user interfaces. The interaction of the front-end application with our SC is provided through Web3.js <sup>3</sup> and MetaMask <sup>4</sup>. Furthermore, the application provides also the possibility to interact with a localhost provider using Ganache <sup>5</sup> as Ethereum node.

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 <sup>6</sup>. 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 <sup>7</sup>.

---

<sup>1</sup><https://github.com/ethereum/solidity>

<sup>2</sup><https://reactjs.org/>

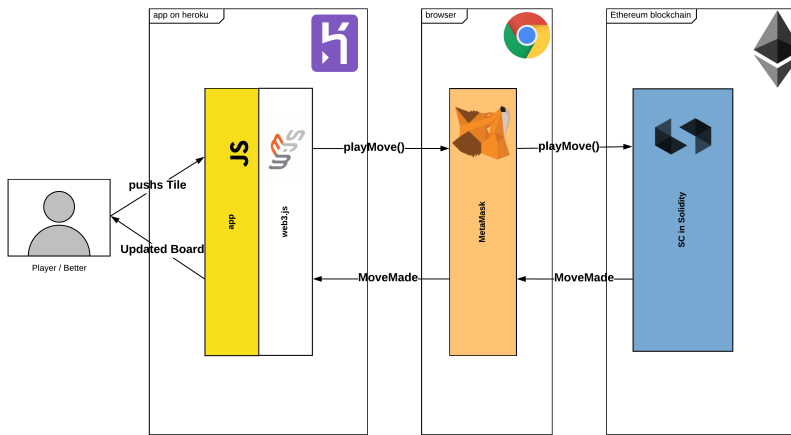
<sup>3</sup><https://web3js.readthedocs.io/en/1.0/>

<sup>4</sup><https://metamask.io/>

<sup>5</sup><http://truffleframework.com/ganache/>

<sup>6</sup><https://www.heroku.com/>

<sup>7</sup><https://ropsten.etherscan.io/>



**Figure 2.1:** The interconnection of the different technologies



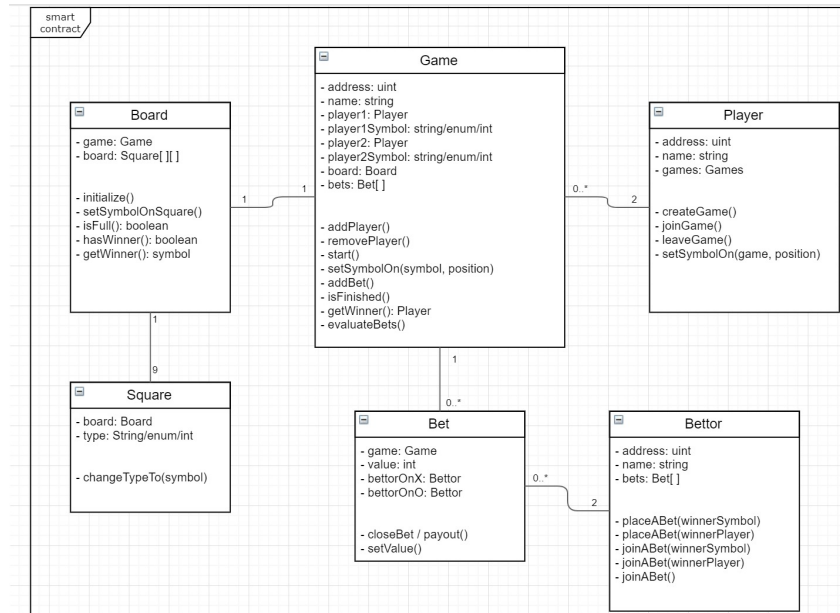
## Chapter 3

### IMPLEMENTATION OF THE GAME

#### 3.1 The Smart Contract

The class-diagram in Figure 3.1 show a detail class diagram explaining the modelling and structure of our SC.

The SC can hold multiple games, which always contain one board and two players.



**Figure 3.1:** Class-Diagram of the SC

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 the game the bet is referencing on it. The user who creates a bet chooses the game and the amount of token he want to bet with and pays it into the SC. Is there an opponent bettor joining the bet is changing to the state '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 0");
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.0;
31     game.state = GameState.X_HAS_TURN;
32     checkForWinner(x, y, gameId, game.player0Addr);
33
34     emit MoveMade(true, gameId, game.state, x, y, "0");
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 a parameter (line 2). With this id the corresponding game object can be called through a mapping (line 3). It adds the player's 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 a confirmation of the move (line 21 and 34). There is also an auto completion for the last move if there is no winner set yet (line 37).

### 3.1.2 Check for Winner

```
1function checkForWinner(uint x, uint y, uint gameId, address
    currentPlayer) private {
2  Game storage game = games[gameId];
3
4  //is winning already possible?
5  if (game.moveCounter < 2 * boardSize - 1)
6    return;
7  }
8
9  SquareState symbol = game.board[y][x];
10
11 //check column
12 for (uint i = 0; i < boardSize; i++) {
13   if (game.board[i][x] != symbol) {
14     break;
15   }
16   else if (i == (boardSize - 1)) {
17     game.winnerAddr = currentPlayer;
18     game.state = getGameState(symbol);
19     payoutBets(game.gameId);
20     return;
21   }
22 }
23
24 //check row
25 for (i = 0; i < boardSize; i++) {
26   if (game.board[y][i] != symbol) {
27     break;
28   }
29 }
```

```

29  else if (i == (boardSize - 1)) {
30      game.winnerAddr = currentPlayer;
31      game.state = getGameState(symbol);
32      payoutBets(game.gameId);
33      return;
34  }
35  }
36
37  //check diagonal: (x-y) 0-0, 1-1, 2-2
38  if (x == y) {
39      for (i = 0; i < boardSize; i++) {
40          if (game.board[i][i] != symbol) {
41              break;
42          }
43          else if (i == (boardSize - 1)) {
44              game.winnerAddr = currentPlayer;
45              game.state = getGameState(symbol);
46              payoutBets(game.gameId);
47              return;
48          }
49      }
50  }
51
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 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

```
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        // bettorOn0 wins
13      } else if (games[gameId].state == GameState.WINNER_0) {
14        (iBet.bettorOn0Addr).transfer(SafeMath.mul(2, iBet.value));
15
16        // draw
17      } else {
18        (iBet.bettorOn0Addr).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_0_BETTOR) {
26      (iBet.bettorOnXAddr).transfer(iBet.value);
27      iBet.state = BetState.WITHDRAWN;
28    }
29    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

### DISCUSSION

#### 4.1 Challenges and Problems

#### 4.2 Future work

APPENDIX A  
RAW DATA