Tic Tac Toe using Ethereum Smart Contracts

Timo Hegnauer Lenz Baumann Cyrill Halter

2018
May

This report documents the implementation of the *Tic Tac Toe* game logic by means of an *Ethereum Smart Contract* written in the *Solidity* programming language. The smart contract should be compilable with the *SOLC* compiler version `>= 0.4.21`.

1 Functionality

The smart contract supports the following functionality:

1. The full *Tic Tac Toe* game logic is implemented entirely within the smart contract and can be accessed either via the *GETH* console or a custom web interface.

2. The smart contract is able to store and operate multiple games at the same time.

3. A betting system was implemented where each player has to pay a wager of 5 ether in order to join a game. The winner is awarded the entire pot of 10 ether. In case of a tie the pot is split up equally between the two players.

2 The Game Struct

```
struct Game
{
    address opponent;
    bool isHostsTurn;
    uint turnNr;
    mapping(uint => mapping(uint => uint)) board;
}
```

```
mapping (address => Game) games;
```

3 Hosting and Joining a New Game

```
function hostNewGame() payable rightAmountPaid public
{
    clearBoard(msg.sender);
    Game storage g = games[msg.sender];
    emit Log("successfully hosted Game!");
}

modifier rightAmountPaid {
    if(msg.value != pot){
        emit Error("You need to make a transaction of 5 eth...");
    }else{
        _;
    }
}

function joinExistingGame(address host) payable rightAmountPaid public
{
    Game storage g = games[host];
    if(g.opponent == 0 && msg.sender != host)
    {
        g.opponent = msg.sender;
    }
    emit Log("successfully joined Game!");
}
```

4 Playing a Move

```
function play(address host, uint row, uint column) public{
    Game storage g = games[host];
    uint player;
    if(msg.sender == host){
        emit Log("executing move for host");
        player = 1;
    }
    else if(msg.sender == g.opponent){
        emit Log("executing move for opponent");
        player = 2;
    } else{
```

```
            emit Error("You are not part of this game");
            return;
        }
        if((g.isHostsTurn && player != 1) || (!g.isHostsTurn && player == 1)){
            emit Error("Its not your turn! Wait for your opponent to play");
            return;
        }else{
            if(row >= 0 && row < 3 && column >= 0 && column < 3
                    && g.board[row][column] == 0)
            {
                g.board[row][column] = player;
                g.turnNr ++;

                if(youWon(host))
                {
                    if(player == 1){
                        host.transfer(10 ether);
                        emit GameOver("host");
                    }else{
                        g.opponent.transfer(10 ether);
                        emit GameOver("opponent");
                    }
                    g.isHostsTurn = !g.isHostsTurn;
                    return;
                }

                if(isTie(host))
                {
                    host.transfer(5 ether);
                    g.opponent.transfer(5 ether);
                    g.isHostsTurn = !g.isHostsTurn;
                    emit GameOver("tie");
                    return;
                }

                emit Log("move successfully applied");
                g.isHostsTurn = !g.isHostsTurn;
                return;

            } else {
                emit Error("Your choice of field was not valid");
            }
        }
    }
```

```
function youWon(address host) public view returns (bool didYouWin)
{
    Game storage g = games[host];
    for (uint i; i < 3; i++){
        if(g.board[i][0] != 0 && g.board[i][0] == g.board[i][1] &&
                g.board[i][1] == g.board[i][2] ){
            return true;
        }
        if(g.board[0][i] != 0 && g.board[0][i] == g.board[1][i] &&
                g.board[1][i] == g.board[2][i]){
            return true;
        }
    }
    if(g.board[0][0] != 0 && g.board[0][0] == g.board[1][1] &&
            g.board[1][1] == g.board[2][2]){
        return true;
    }
    if(g.board[2][0] != 0 && g.board[2][0] == g.board[1][1] &&
            g.board[1][1] == g.board[0][2]){
        return true;
    }
    return false;
}

function isTie(address host) internal view returns (bool isItATie)
{
    Game storage g = games[host];
    if(g.turnNr > 8){
        return true;
    }
}
```

5 Some Helper Functions

```
function clearBoard(address host) internal
{
    Game storage g = games[host];
    delete g.board[0][0];
    delete g.board[0][1];
    delete g.board[0][2];
    delete  g.board[1][0];
    delete g.board[1][1];
    delete g.board[1][2];
    delete g.board[2][0];
```

```
    delete  g.board[2][1];
    delete g.board[2][2];
    delete g.opponent;
    delete g.isHostsTurn;
    delete g.turnNr;
}

function printBoard(address host) public view returns (bool _isHostsTurn,
        uint board1, uint board2, uint board3)
{
    Game storage g = games[host];
    board1 = (999000 + 100 * (g.board[0][0])) + (10 * (g.board[0][1])) +
            (g.board[0][2]);
    board2 = (999000 + 100 * (g.board[1][0])) + (10 * (g.board[1][1])) +
            (g.board[1][2]);
    board3 = (999000 + 100 * (g.board[2][0])) + (10 * (g.board[2][1])) +
            (g.board[2][2]);
    _isHostsTurn = g.isHostsTurn;
}
```