

19BCE248
BCT
Practical 10

AIM: Implement Tic-Tac-Toe game

Source Code:

```
pragma solidity ^0.4.20;
contract TicTacToe {

    address player1; address player2; uint turn;
    uint winnerWinnerChickenDinner; uint[] board = new uint[](9);
    //top left corner is 1, then left to right, up to down

    function TicTacToe(address p2) public {
        // at inception the contract takes the two players addresses
        player1 = msg.sender; player2 = p2;
        turn = 1;
    }

    function kill() public {
        if (msg.sender == player1 && winnerWinnerChickenDinner>0) {
            selfdestruct(player1);
        }
    }

    function myTurn() public view returns(bool) {
        //returns true if it's the sender's turn to play return (msg.sender == player1 && turn
        == 1) ||
        (msg.sender == player2 && turn == 2) ;
    }

    function subCheckWin(uint a, uint b, uint c) private view returns (bool){
        return board[a]>0 && board[a]==board[b] && board[b]==board[c];
    }

    function checkWin() private view returns (uint) {

        //function used to check if there is a winning player for (uint i = 0; i < 3; i++) {
        if (subCheckWin(i,3+i,6+i)) { return board[i];
        }
        if (subCheckWin(3*i,3*i+1,3*i+2)) { return board[3*i];
        }
        }
        if (subCheckWin(0,4,8)) {
            return board[0];
        }
    }
}
```

```

}
if (subCheckWin(2,4,6)) { return board[2];
}
if (board[0]+board[1]+board[1]+board[1]+board[1]+board[1]+board[1]
]+board[1]+board[1] == 13){
return 3;
}
return 0;
}

function winner() public view returns (uint) {

//getter for the winner attribute if (winnerWinnerChickenDinner>0) {
return winnerWinnerChickenDinner;
} else {
return 0;
}
}

function validMove(uint a) public view returns (bool) {

or not

//function to assess whether a required move is valid

//used both as an external callable function and

internally to validate moves
//before writing them to the board

return !(winnerWinnerChickenDinner>0 || a<0 || a>8 || board[a]>0) &&
((msg.sender == player1 && turn==1) || (msg.sender == player2 && turn==2));
}

function play(uint a) public {

board

//checks if a move is valid before inserting it to the

if (validMove(a)) { board[a] = turn; if (turn==1) {
turn = 2;
} else {
turn = 1;
}
winnerWinnerChickenDinner = checkWin();
}

```



```

} else {
  revert();
}
}

function displayBoard() public view returns(uint[]) {


  //getter for the board return board;
}
}


```

▼ TICTACTOE AT 0X793...FDEFB (ME)  

Balance: 0 ETH

kill

play 2 


TicTacToe 0xAb8483F64d9C6d1EcF9b84 

displayBo...

0: uint256[]: 1,0,2,0,0,0,0,0



myTurn

0: bool: true

validMove 0 


0: bool: false


winner

▼ TICTACTOE AT 0X793...FDEFB (ME)  

Balance: 0 ETH

kill

play 4 


TicTacToe 0xAb8483F64d9C6d1EcF9b84 

displayBo...

0: uint256[]: 1,0,2,0,1,0,0,0

myTurn


0: bool: true


validMove 5 

0: bool: true

winner

kill

play 8 


TicTacToe 0xAb8483F64d9C6d1EcF9b84 

displayBo...

0: uint256[]: 1,0,2,2,1,0,1,2,1

myTurn

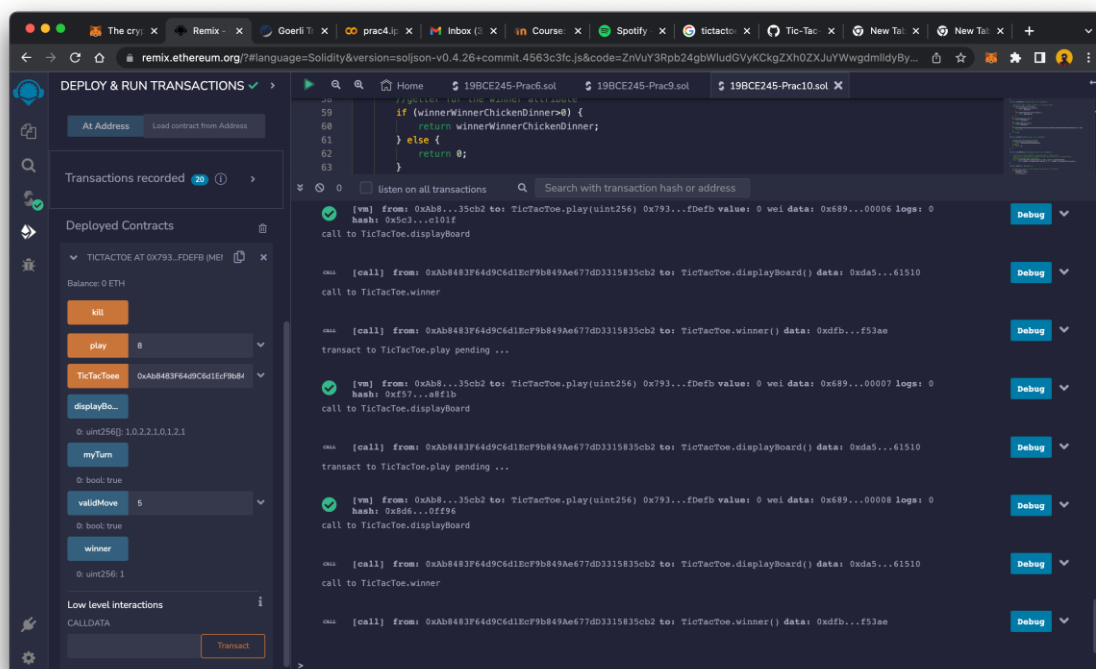
0: bool: true

validMove 5 

0: bool: true

winner

0: uint256: 1



The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar displays the deployed contract 'TICTACTOE AT 0X793...FDEFB (ME)' with its balance (0 ETH) and various buttons: 'kill', 'play' (set to 8), 'TicTacToe' (address: 0xAb8483F64d9C6d1EcF9b84), 'displayBo...', 'myTurn', 'validMove' (set to 5), and 'winner'. The main area shows the Solidity code for the contract, and the bottom console displays a series of transaction logs and debug information, including function calls like 'TicTacToe.play(uint256)', 'TicTacToe.displayBoard()', and 'TicTacToe.winner()'.

Conclusion

From these practical we learned about how we can implement a very basic game from scratch including all basic rules and regulations. If we deploy these concept in games where there is usage of real currency it can be very helpful as it doesn't contains any human intervention.