Tic Tac Toe Smart contract using Solidity

By

Dr. Aparna Kumari

Aparna.kumari@nirmauni.ac.in

Smart Contract Implementation

- **Truffle** is a framework for developing these Ethereum applications (dApps, or distributed apps). Installed via npm, it'll create a local test blockchain and manage your contract deployment to the blockchain.
- **Web3.js** is a JavaScript library that provides an interface for reading from and sending data to the contract once its deployed.
- You include it in your Truffle application and it'll get included with your HTML and other assets. It's also available in test mode, so you can write unit tests in **JavaScript** that modify and assert the state of your contract.

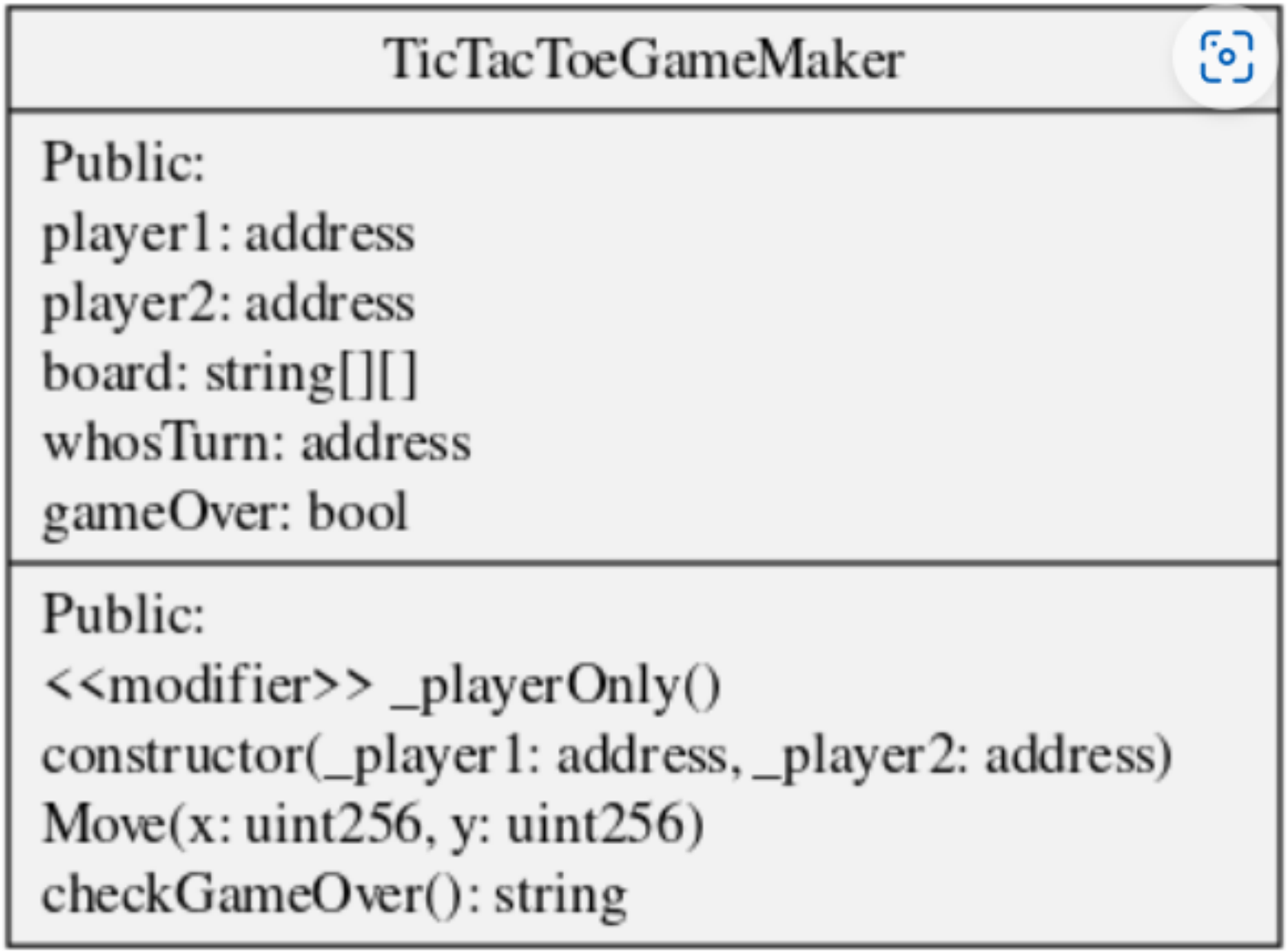
The Rules of the Game

- Player one (the host) makes the first move, followed by player two (the challenger).

- The first player to complete a row or diagonal of either X’s or O’s wins the game.

- If no player completes a row or diagonal of either X’s or O’s, the game is a draw

Basic Structure



Implementation

```
    // GameCreated signals that `creator` created a new game with this `gameId`.
    GameCreated(uint256 gameId, address creator);

    // PlayerJoinedGame signals that `player` joined the game with the id `gameId`.
    // That player has the player number `playerNumber` in that game.
    PlayerJoinedGame(uint256 gameId, address player, uint8 playerNumber);

    // PlayerMadeMove signals that `player` filled in the board of the game with
    // the id `gameId`. She did so at the coordinates `xCoordinate`, `yCoordinate`.
    PlayerMadeMove(uint256 gameId, address player, uint xCoordinate, uint
yCoordinate);

    // GameOver signals that the game with the id `gameId` is over.
    // The winner is indicated by `winner`. No more moves are allowed in this game.
    GameOver(uint256 gameId, Winners winner);
```

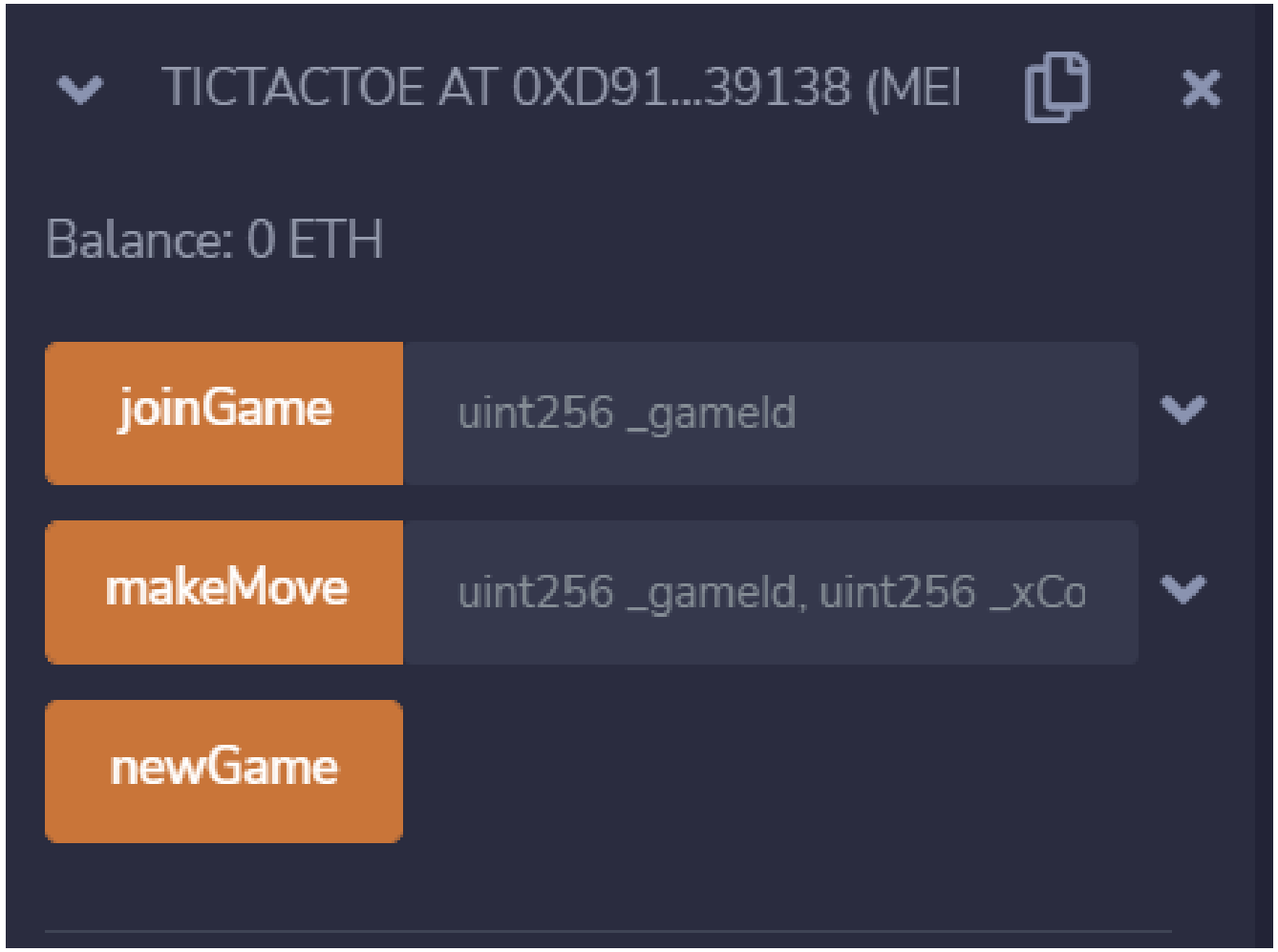
Implementation

```
// newGame creates a new game and returns the new game's `gameId`.
// The `gameId` is required in subsequent calls to identify the game.
function newGame() public returns (uint256 gameId) {
    Game memory game;
    game.playerTurn = Players.PlayerOne;

    nrOfGames++;
    games[nrOfGames] = game;

    emit GameCreated(nrOfGames, msg.sender);

    return nrOfGames;
}
```

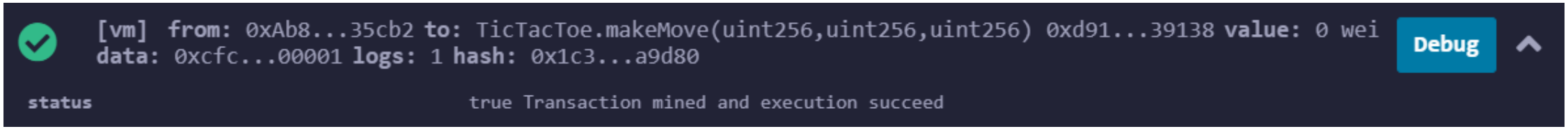
Steps to deploy the Smart Contract

- Go to Remix IDE to Compile and Run our Solidity Code.
- **Step 1** – Copy the given code in Remix IDE Code Section.
- **Step 2** – Under Compile Tab, click **Start to Compile** button.
- **Step 3** – Under Run Tab, click **Deploy** button.
- **Step 4** – Under Run Tab, Select **SimpleContract at 0x...** in drop-down.
- **Step 5** – Click **set** Button after giving input value and the click on the **get** Button to display the result.

Input


TICTACTOE AT 0XD91...39138 (MEI
Balance: 0 ETH
joinGame
uint256 _gameId
makeMove
uint256 _gameId, uint256 _xCo
newGame

Output


[vm] from: 0xAb8...35cb2 to: TicTacToe.makeMove(uint256,uint256,uint256) 0xd91...39138 value: 0 wei
data: 0xcfc...00001 logs: 1 hash: 0x1c3...a9d80
Debug
status
true Transaction mined and execution succeed

Output

status	true Transaction mined and execution succeed
transaction hash	0x1c39c418781e142824f31998503802c083a0fcdbea98a5b0130b790c42ba9d80
from	0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
to	TicTacToe.makeMove(uint256,uint256,uint256) 0xd9145CCE52D386f254917e481eB44e9943F39138
gas	58475 gas
transaction cost	50847 gas
execution cost	50847 gas
input	0xcfc...00001
decoded input	{ "uint256 _gameId": "1", "uint256 _xCoordinate": "2", "uint256 _yCoordinate": "1" }
decoded output	{ "0": "bool: success true", "1": "string: reason "