Batch: BCT-3	

Batch: BCT-3

Roll Nos.:

16010121200 – Asmi Takle

16010121210 – Bhavya Verma

16010121221 – Varrshinie Aravindan

Experiment No. 10

Title: Mini Project

Aim: Implementing a Lottery System using a Smart Contract on Remix IDE.

Objective:

To create a decentralized lottery system using a smart contract in Solidity, developed on Remix IDE. This contract will allow a transparent and secure way for players to participate in a lottery, with a fair winner selection and reward distribution using the Ethereum blockchain.

Expected Outcome of Experiment:

CO	Outcome
1	Build your own Blockchain businesses with acquired knowledge.
2	Learn Solidity language & Multiple Technology-based developments.
3	Apply the algorithm and techniques used in Blockchain.
4	Grasp the in-depth understanding of Blockchain, Smart Contracts & how it works.
5	Describe the methods of mining.

Books/ Journals/ Websites referred:

- https://youtu.be/ZZFktsWnqfw

Abstract:

This smart contract, implemented on Remix IDE using Solidity, introduces a decentralized lottery system. The contract is designed to be transparent and secure, utilizing the Ethereum blockchain for fairness and immutability. The key entities involved are the manager, responsible for initiating and overseeing the lottery, the players who participate by contributing 1 ether each, and the winner who claims the accumulated funds.

The contract allows three, or more than three players to join the lottery by invoking the participate function, ensuring a payment of 1 ether only. The getBalance function enables the manager to check the current balance of the lottery pool. To determine the winner, the pickWinner function utilizes a pseudo-random number generated either manually during development/testing or by leveraging an off-chain oracle in a production environment.

The contract ensures that the manager has exclusive control over critical functions, maintaining the integrity of the lottery. The winner is selected from the pool of participants when there are at least three players, and the entire prize pool is transferred to the winner's address. After each draw, the players' array is reset for subsequent lotteries.

This implementation serves as a practical demonstration of a decentralized lottery system on the Ethereum blockchain, emphasizing transparency, fairness, and security.

Implementation Details:

Code:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;
contract Lottery{
    //entities - manager, players, winner
    address public manager;
    address payable[] public players;
    address payable public winner;

    constructor(){
        manager = msg.sender;
    }

    function participate() public payable{
        require(msg.value==1 ether, "Please pay 1 ether only!");
        players.push(payable(msg.sender));
    }

    function getBalance() public view returns (uint){
        require(manager==msg.sender, "You are not the manager!");
        return address(this).balance;
    }

    // Simulate the process of obtaining a random number manually during
development/testing
    uint private manualRandomNumber;

    function setManualRandomNumber(uint _manualRandomNumber) external {
        require(msg.sender == manager, "You are not the manager!");
        manualRandomNumber = _manualRandomNumber;
    }

    function getRandomNumber() internal view returns (uint) {
        // Use the manually set random number during development/testing
        if (manualRandomNumber != 0) {
            return manualRandomNumber;
        }

        // In production, we should use an off-chain oracle or service
to get a random number
        // For demonstration purposes, we return a deterministic value
        return  uint(keccak256(abi.encodePacked(blockhash(block.number
- 1), block.timestamp)));
```

```
    }

    function pickWinner() public {
        require(msg.sender==manager, "You are not the manager!");
        require(players.length>=3, "There are less than 3 players!");

        uint r = getRandomNumber();
        uint index = r%players.length;
        winner = players[index];
        winner.transfer(getBalance());
        players = new address payable[](0); // This will initialize the
players array to zero
    }
}
```

1. **Enlist all the Steps followed and various options explored**

   Step 1: Setting up the Contract -

   - Used Remix IDE for Solidity development.
   - Chose SPDX-License-Identifier as GPL-3.0.
   - Defined the contract named "Lottery" with necessary state variables.

   Step 2: Entities and Initialization -

   - Identified key entities: manager, players, winner.
   - Initialized the manager in the constructor.

   Step 3: Player Participation -

   - Implemented the participate function for players to join by sending 1 ether.

   Step 4: Balance Inquiry -

   - Created getBalance function for the manager to check the lottery balance.

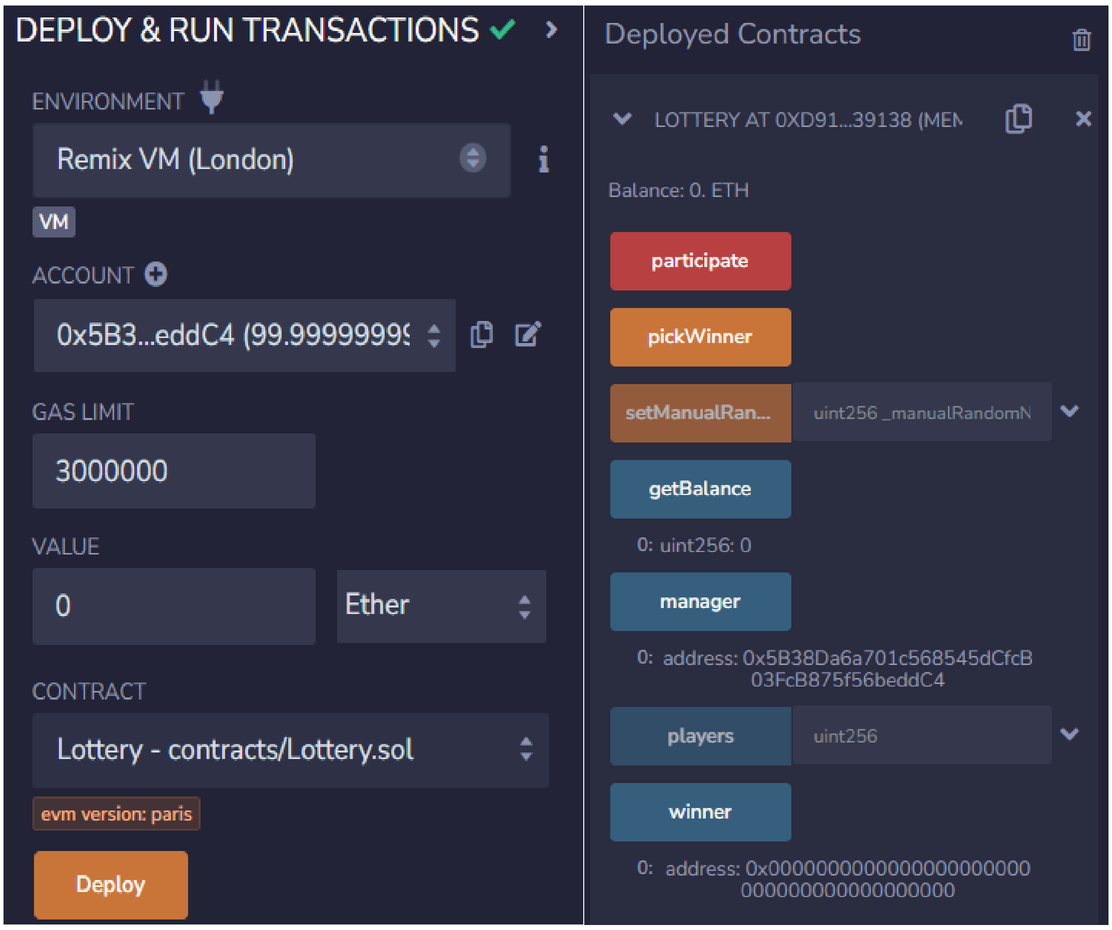
   Step 5: Random Number Generation -

   - Developed a method getRandomNumber for obtaining a random number.
   - Included a provision for manual random number setting during development/testing.

   Step 6: Winner Selection and Payout -

   - Implemented pickWinner function for manager to select a winner.
   - Transferred the entire balance to the winner and reset the players array.

Creating the contract and deploying using the Remix VM:


DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Remix VM (London)

ACCOUNT: 0x5B3...eddC4 (99.99999999

GAS LIMIT: 3000000

VALUE: 0 Ether

CONTRACT: Lottery - contracts/Lottery.sol

evm version: paris

Deployed Contracts

LOTTERY AT 0XD91...39138 (MEM

Balance: 0. ETH

getBalance: 0: uint256: 0

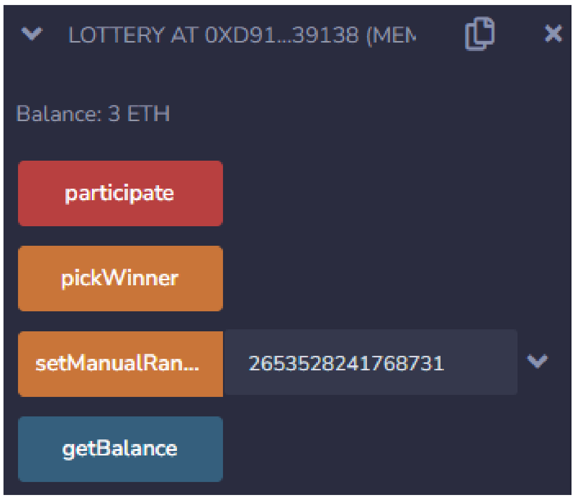
manager: 0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

winner: 0: address: 0x0000000000000000000000000000000000000000

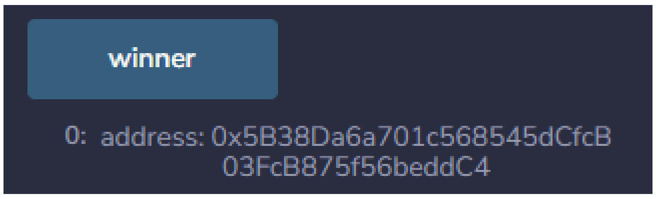
Now 3 or more than 3 players can pay 1 ether each to participate in the lottery.

After this, the manager of the system

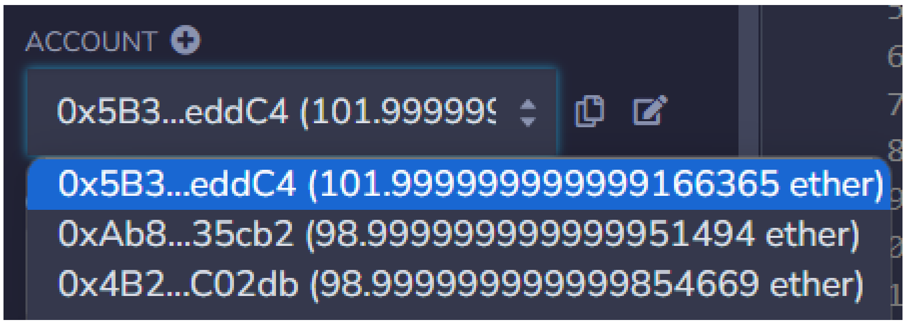
(user 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4) will call the pickWinner function to pick a winner among the players, after entering a random number.



After the manager calls the pickWinner function, the result will be reflected in the users' balances. The selection of this winner is always completely random.



```
0: address: 0x5B38Da6a701c568545dCfcB
03FcB875f56beddC4
```



```
0x5B3...eddC4 (101.99999
0x5B3...eddC4 (101.999999999999166365 ether)
0xAb8...35cb2 (98.999999999999951494 ether)
0x4B2...C02db (98.999999999999854669 ether)
```

2. **Explain your program logic, classes and methods used.**

Entities:

- manager: The address that initiates and oversees the lottery.
- players: An array of payable addresses representing participants.
- winner: The address of the player selected as the winner.

Functions:

- participate: Allows players to join by sending 1 ether.
- getBalance: Returns the current balance of the lottery pool.

- setManualRandomNumber: Allows the manager to set a manual random number for testing.
- getRandomNumber: Generates a random number, considering manual setting during development.
- pickWinner: Manager-exclusive function to select a winner and transfer the prize.

3. **Explain the Importance of the approach followed by you**

Transparency and Security:

- The contract ensures transparency by allowing participants to verify the fairness of the lottery.
- Security is maintained by restricting critical functions to the manager.

Decentralization:

- Leveraging the Ethereum blockchain ensures decentralization, eliminating the need for a trusted third party.
- Immutability of the blockchain prevents tampering with lottery outcomes.

Randomness and Testing:

- The contract accommodates both manual random number setting for testing and a production-ready approach using an off-chain oracle.
- This flexibility is crucial for development, testing, and future production use.

User Participation and Payouts:

- The participate function facilitates user engagement by allowing them to easily join the lottery.
- Payouts are automated and transparent, promoting trust in the lottery system.

Conclusion:

From this experiment, we learnt the importance of well-designed smart contracts for trustworthy decentralized applications, such as randomized lottery systems.