Decentralized Lottery DApp Report

By Sammy Guo

This document details the Decentralized Lottery DApp, its functionality, and the testing and interaction process using MetaMask and Remix on the Sepolia Ethereum test network.

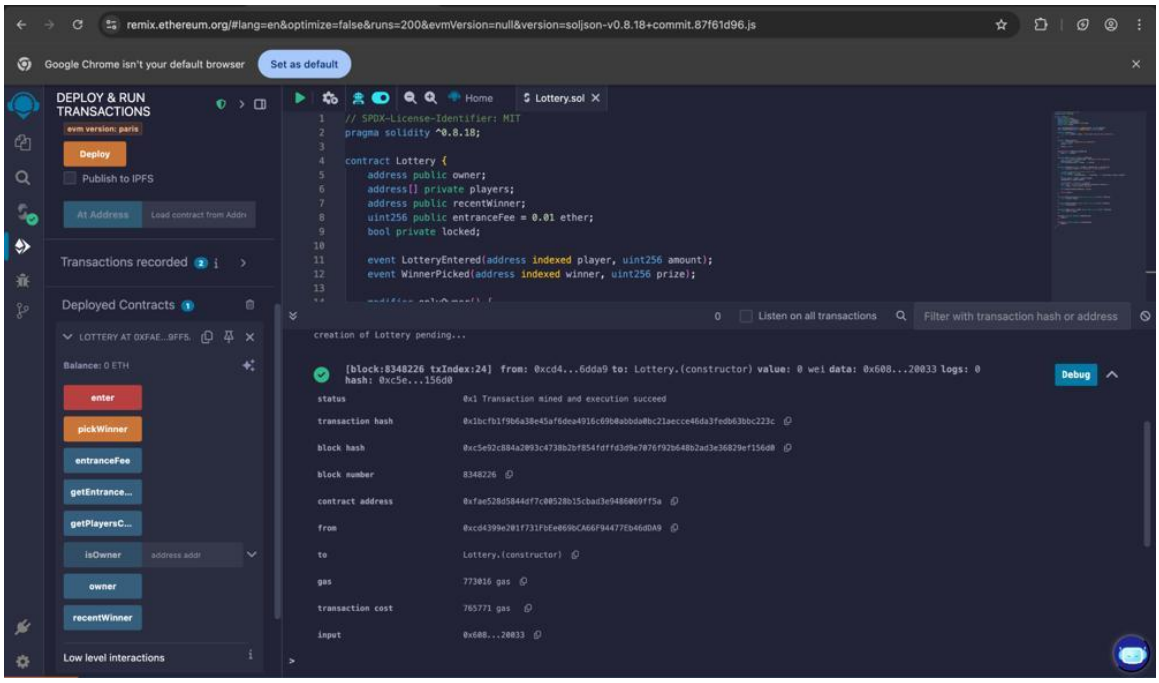
Application Functionality

The Decentralized Lottery allows users to participate by paying 0.01 ETH. Once entered, the contract keeps track of players and allows the contract owner to pick a winner. The winner receives the total balance of the contract, and the participant list is reset. The contract includes reentrancy protection and access control to ensure secure operation.

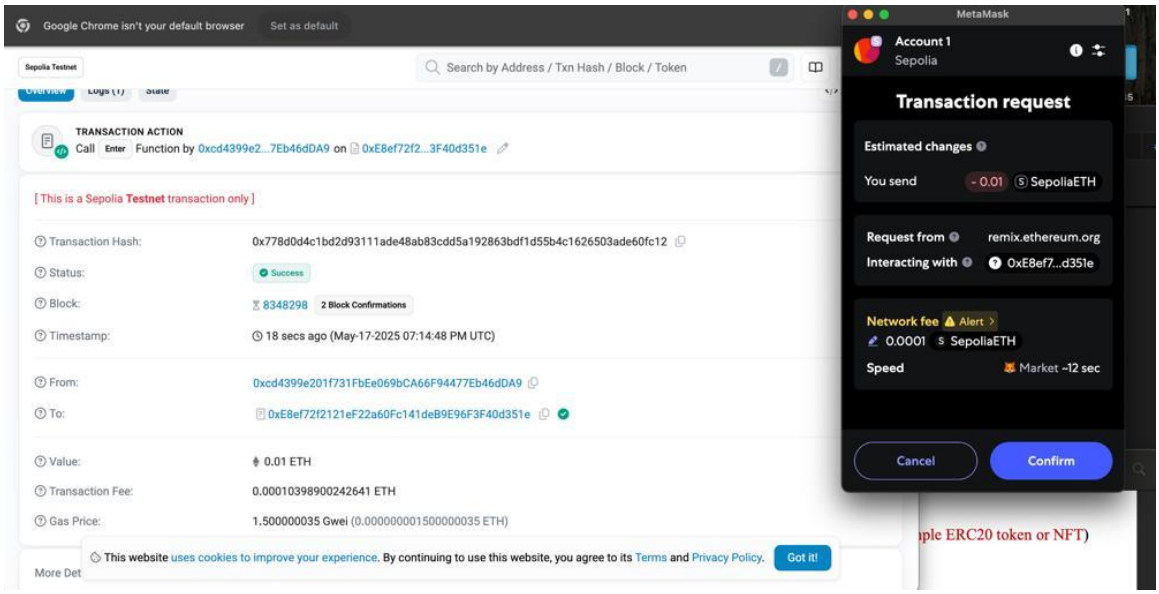
While the smart contract enforces that only the owner can pick the winner, ownership was not manually assigned. This was intentional to allow the professor and teaching assistants to interact with the contract and demonstrate its features, including sending and receiving Sepolia ETH, without restriction.

1. Remix Deployment and Initialization

The contract was first deployed on Remix using the Sepolia network. Below is the screenshot showing successful deployment and function visibility. The deployment account becomes the default owner.

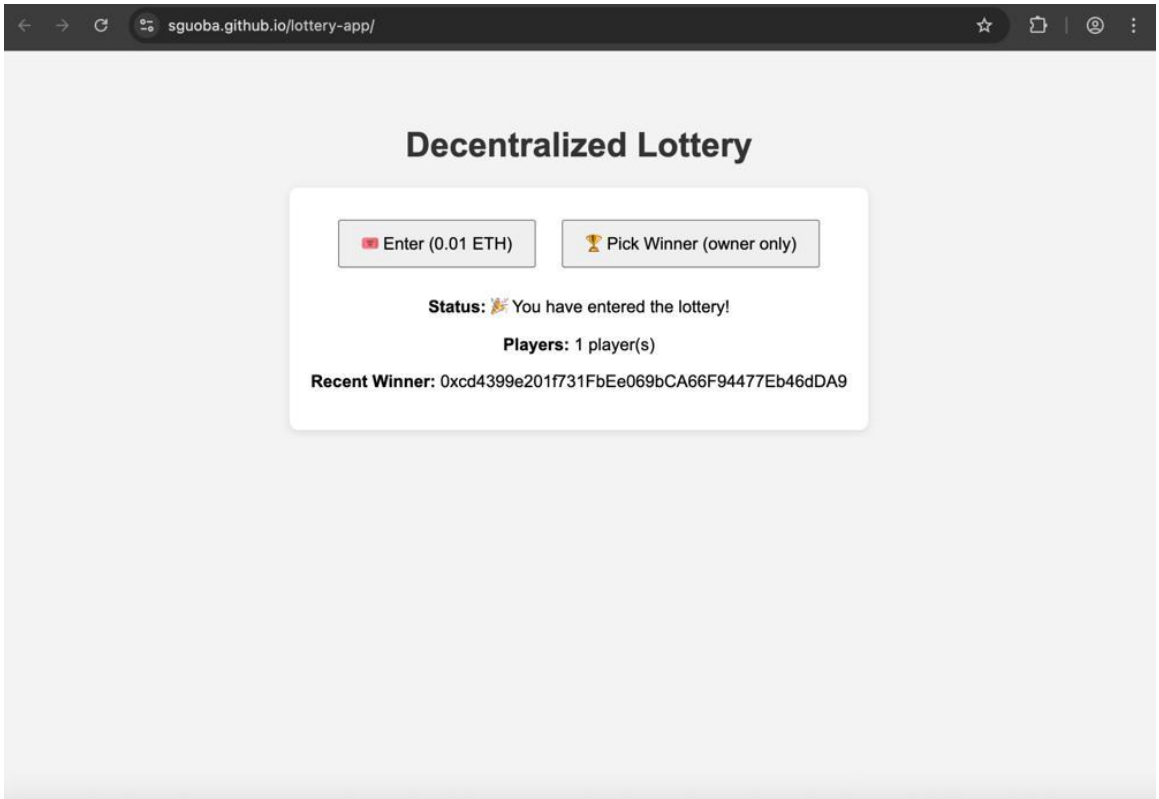


Logs from Remix confirm that the constructor set the initial state correctly:



2. Entering the Lottery (Single Participant)

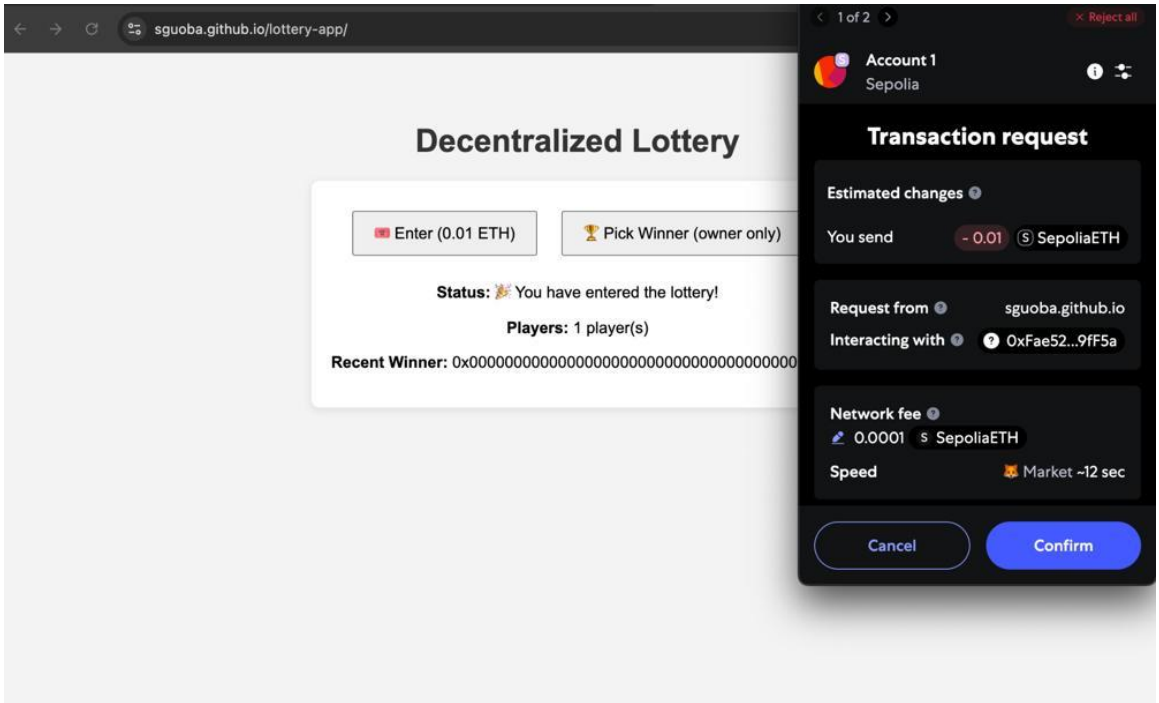
When one user enters the lottery, the system records their entry and updates the UI accordingly.



MetaMask shows the transaction details for entering with 0.01 ETH.

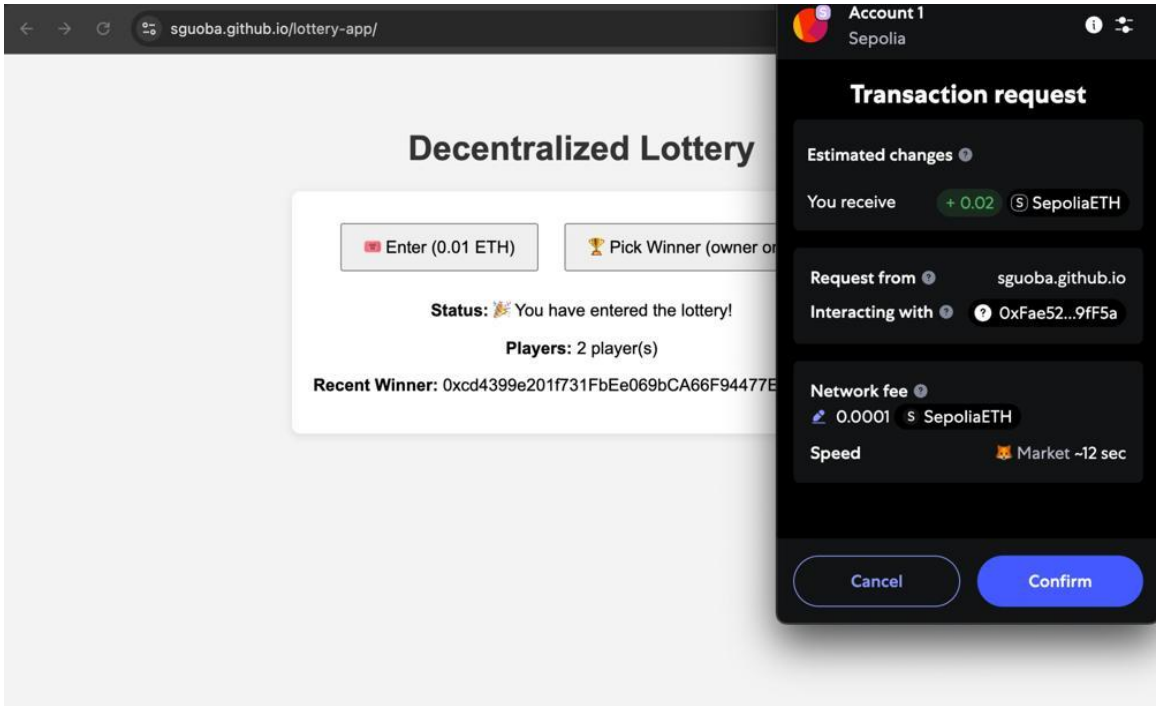
3. Multiple Participants Enter

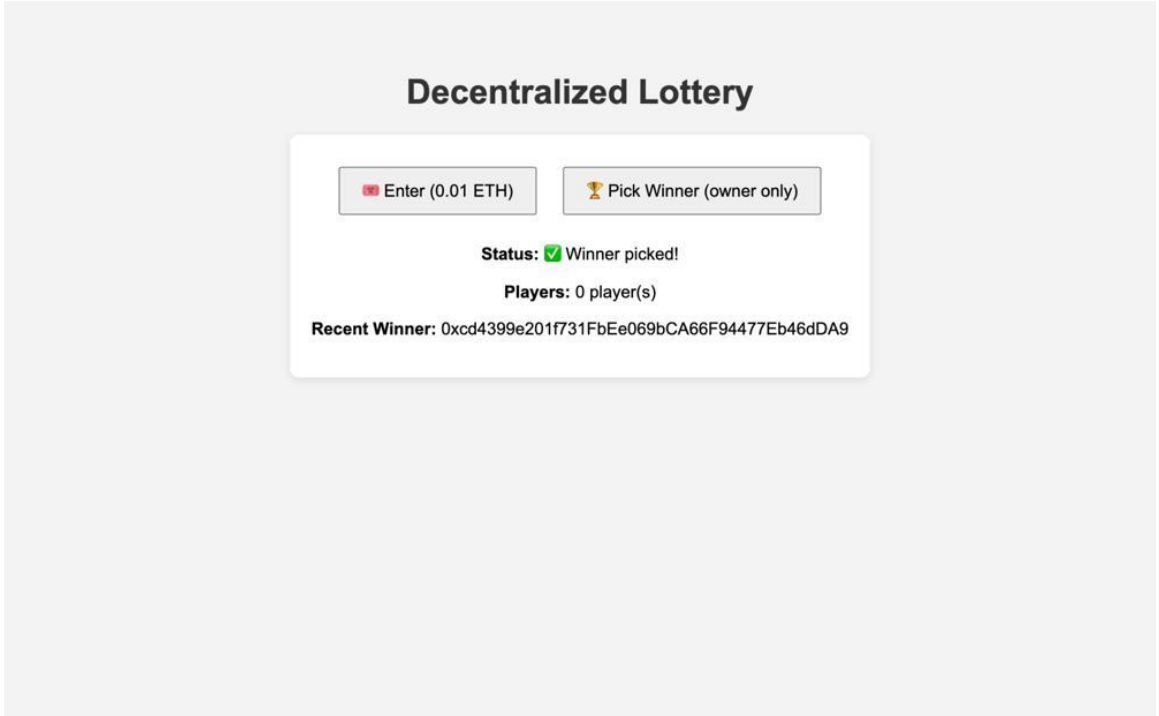
A second participant enters the lottery, increasing the total players to two.



4. Picking the Winner

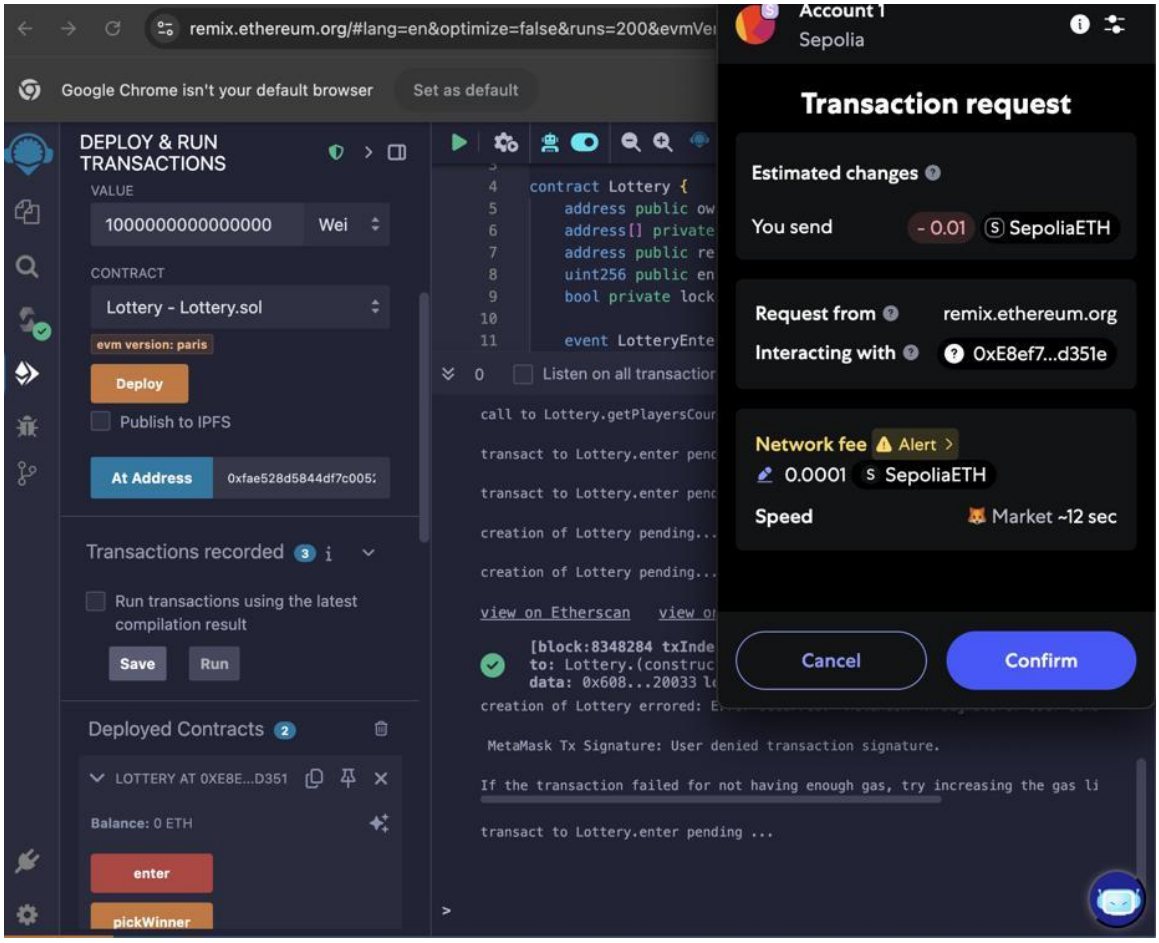
Only the owner is permitted to pick a winner. Once the button is clicked, MetaMask prompts for confirmation. Upon success, the winner receives the full prize, and the player list resets to 0.

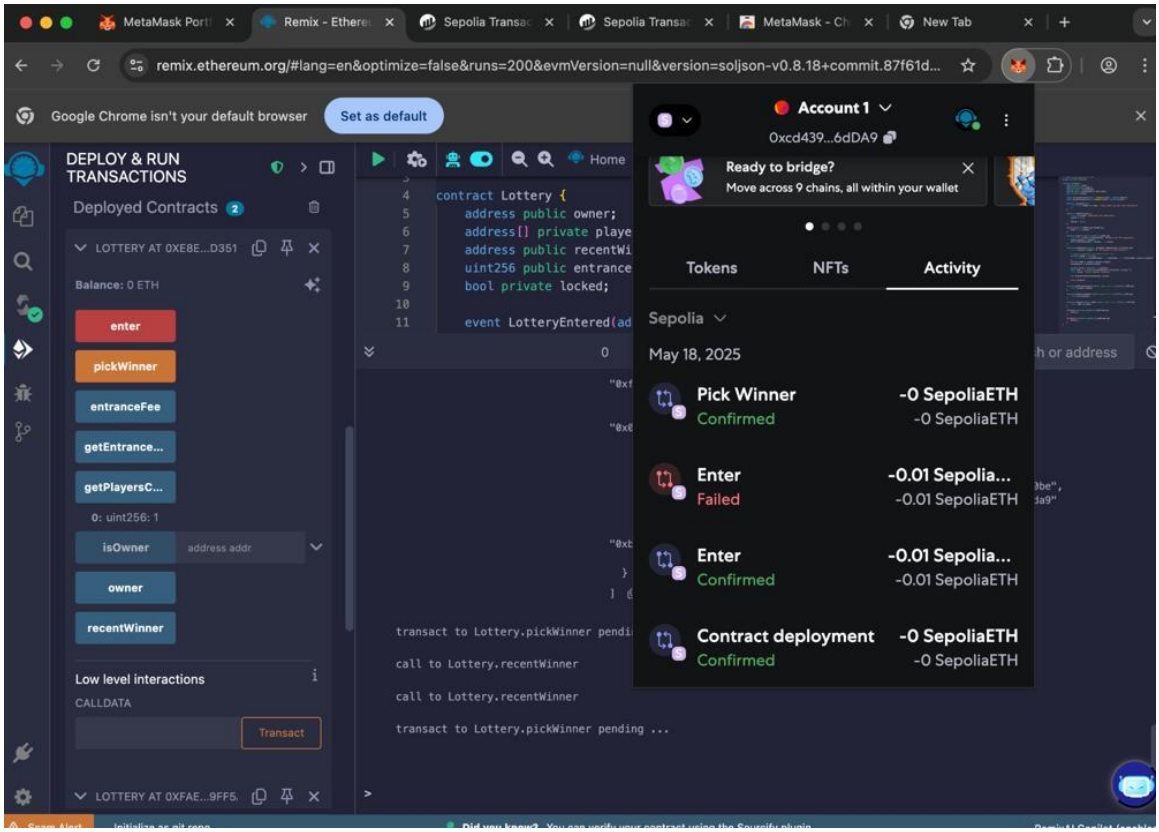

sguoba.github.io/lottery-app/
Decentralized Lottery
Enter (0.01 ETH)
Pick Winner (owner or
Status: You have entered the lottery!
Players: 2 player(s)
Recent Winner: 0xcd4399e201f731FbEe069bCA66F94477E
Account 1
Sepolia
Transaction request
Estimated changes
You receive
+ 0.02
SepoliaETH
Request from
sguoba.github.io
Interacting with
0xFae52...9fF5a
Network fee
0.0001
SepoliaETH
Speed
Market ~12 sec
Cancel
Confirm


Decentralized Lottery
Enter (0.01 ETH)
Pick Winner (owner only)
Status: Winner picked!
Players: 0 player(s)
Recent Winner: 0xcd4399e201f731FbEe069bCA66F94477Eb46dDA9

5. Transaction Confirmation and Activity

Transaction details are logged on Sepolia Testnet and confirmed via MetaMask activity.





6. Edge Case: Null Winner

If pickWinner is called prematurely or data fails to reset, the winner may be zero-address.

Test your contract (Beta)

Test Result:

Using contract tester version 0.7.6

(1/3) 🧬 Compiling contract: Lottery_21197014_1747561121.sol

[✅PASS]Compilation

(2/3) 🤖 Generating test case for: Lottery_21197014_1747561121.sol

[✅PASS]Read contract

[ℹ️INFO] Contract file already exists in destination, skipping adjust.

[⚠️WARNING]No </think> tag found in input

[✅PASS]Test case generation

[✅PASS]Write test contract file

(3/3) 🧪 Running tests in Lottery_21197014_1747561121.t.sol

(🧪Attempt 1/3)

[DEBUG]STDOUT

Compiling 1 files with Solc 0.8.28

Solc 0.8.28 finished in 731.45ms

Compiler run successful with warnings:

Warning (2018): Function state mutability can be restricted to view

```
 --> test/Lottery_21197014_1747561121.t.sol:44:5:
  |
44 |     function test_owner_is_correct() public {
  |     ^ (Relevant source part starts here and spans across multiple lines).
```

Ran 11 tests for test/Lottery_21197014_1747561121.t.sol:LotteryTest

[PASS] test_enter_insufficientFee() (gas: 20330)

Logs:

  Lottery owner: 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496

  Initial Lottery balance: 0 ether

Traces:

  [20330] LotteryTest::test_enter_insufficientFee()

    ├─ [0] VM::prank(0x0000000000000000000000000000000000001001)

    │   └─ ← [Return]

    ├─ [0] VM::expectRevert(custom error 0xf28dceb3: Minimum 0.01 ETH required)

    │   └─ ← [Return]

    ├─ [2594] Lottery::enter{value: 9000000000000000}()

    │   └─ ← [Revert] revert: Minimum 0.01 ETH required

    └─ ← [Stop]

[PASS] test_enter_success_via_enter() (gas: 72432)

Logs:

  Lottery owner: 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496

  Initial Lottery balance: 0 ether

  User 0x0000000000000000000000000000000000001001 has successfully entered the lottery with 10000000000000000 wei

Traces:

  [72432] LotteryTest::test_enter_success_via_enter()

    ├─ [0] VM::prank(0x0000000000000000000000000000000000001001)

    │   └─ ← [Return]

    ├─ [2433] Lottery::getEntranceFee() [staticcall]

│ └─ ← [Return] 10000000000000000 [1e16]

├─ [46255] Lottery::enter{value: 10000000000000000}()

│ ├─ emit LotteryEntered(player: LotteryTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], amount: 10000000000000000 [1e16])

│ └─ ← [Stop]

├─ [484] Lottery::getPlayersCount() [staticcall]

│ └─ ← [Return] 1

├─ [0] VM::assertEq(1, 1, "Players count should be 1 after one entry") [staticcall]

│ └─ ← [Return]

├─ [0] console::log("User %s has successfully entered the lottery with %d wei", 0x0000000000000000000000000000000000001001, 10000000000000000 [1e16]) [staticcall]

│ └─ ← [Stop]

└─ ← [Stop]

[PASS] test_enter_via_receive() (gas: 72390)

Logs:

Lottery owner: 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496

Initial Lottery balance: 0 ether

User 0x0000000000000000000000000000000000001001 entered via receive/fallback with 10000000000000000 wei

Traces:

[72390] LotteryTest::test_enter_via_receive()

├─ [0] VM::prank(0x0000000000000000000000000000000000001001)

│ └─ ← [Return]

├─ [2433] Lottery::getEntranceFee() [staticcall]

```
    │   └─ ← [Return] 10000000000000000 [1e16]

    ├─ [46149] Lottery::receive{value: 10000000000000000}()

    │   ├─ emit LotteryEntered(player: LotteryTest:
[0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], amount: 10000000000000000
[1e16])

    │   └─ ← [Stop]

    ├─ [484] Lottery::getPlayersCount() [staticcall]

    │   └─ ← [Return] 1

    ├─ [0] VM::assertEq(1, 1, "Players count should be 1 after receiving ETH") [staticcall]

    │   └─ ← [Return]

    ├─ [0] console::log("User %s entered via receive/fallback with %d wei",
0x0000000000000000000000000000000000001001, 10000000000000000 [1e16])
[staticcall]

    │   └─ ← [Stop]

    └─ ← [Stop]


[PASS] test_fallback_receive_entry() (gas: 72280)

Logs:

  Lottery owner: 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496

  Initial Lottery balance: 0 ether

  Fallback/receive entry test successful for user:
0x0000000000000000000000000000000000001001


Traces:

  [72280] LotteryTest::test_fallback_receive_entry()

    ├─ [2433] Lottery::getEntranceFee() [staticcall]

    │   └─ ← [Return] 10000000000000000 [1e16]

    ├─ [0] VM::prank(0x0000000000000000000000000000000000001001)
```

```
    │   └─ ← [Return]
    ├─ [46150] Lottery::fallback{value: 10000000000000000}(0x3078)
    │   ├─ emit LotteryEntered(player:
0x0000000000000000000000000000000000001001, amount: 10000000000000000
[1e16])
    │   └─ ← [Stop]
    ├─ [484] Lottery::getPlayersCount() [staticcall]
    │   └─ ← [Return] 1
    ├─ [0] VM::assertEq(1, 1, "Player entry via fallback/receive did not register") [staticcall]
    │   └─ ← [Return]
    ├─ [0] console::log("Fallback/receive entry test successful for user: %s",
0x0000000000000000000000000000000000001001) [staticcall]
    │   └─ ← [Stop]
    └─ ← [Stop]


[PASS] test_multiple_entries() (gas: 106365)
Logs:
  Lottery owner: 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496
  Initial Lottery balance: 0 ether
  Multiple entries successful: 2 players


Traces:
  [106365] LotteryTest::test_multiple_entries()
    ├─ [2433] Lottery::getEntranceFee() [staticcall]
    │   └─ ← [Return] 10000000000000000 [1e16]
    ├─ [0] VM::prank(0x0000000000000000000000000000000000001001)
    │   └─ ← [Return]
```

```
    ├─ [46255] Lottery::enter{value: 10000000000000000}()
    │   ├─ emit LotteryEntered(player:
0x0000000000000000000000000000000000001001, amount: 10000000000000000
[1e16])
    │   └─ ← [Stop]
    ├─ [0] VM::prank(0x0000000000000000000000000000000000001002)
    │   └─ ← [Return]
    ├─ [24355] Lottery::enter{value: 10000000000000000}()
    │   ├─ emit LotteryEntered(player:
0x0000000000000000000000000000000000001002, amount: 10000000000000000
[1e16])
    │   └─ ← [Stop]
    ├─ [484] Lottery::getPlayersCount() [staticcall]
    │   └─ ← [Return] 2
    ├─ [0] VM::assertEq(2, 2, "Players count should be 2 after two entries") [staticcall]
    │   └─ ← [Return]
    ├─ [0] console::log("Multiple entries successful: %d players", 2) [staticcall]
    │   └─ ← [Stop]
    └─ ← [Stop]


[PASS] test_no_reentrancy_lock_reset() (gas: 166441)
Logs:
  Lottery owner: 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496
  Initial Lottery balance: 0 ether
  Reentrancy guard test passed across multiple lottery cycles.


Traces:
  [213317] LotteryTest::test_no_reentrancy_lock_reset()
```

```
    ├─ [2433] Lottery::getEntranceFee() [staticcall]
    │   └─ ← [Return] 10000000000000000 [1e16]
    ├─ [0] VM::prank(0x0000000000000000000000000000000000001001)
    │   └─ ← [Return]
    ├─ [46255] Lottery::enter{value: 10000000000000000}()
    │   ├─ emit LotteryEntered(player:
0x0000000000000000000000000000000000001001, amount: 10000000000000000
[1e16])
    │   └─ ← [Stop]
    ├─ [58091] Lottery::pickWinner()
    │   ├─ [0] 0x0000000000000000000000000000000000001001::fallback{value:
10000000000000000}()
    │   │   └─ ← [Stop]
    │   ├─ emit WinnerPicked(winner:
0x0000000000000000000000000000000000001001, prize: 10000000000000000 [1e16])
    │   └─ ← [Stop]
    ├─ [0] VM::prank(0x0000000000000000000000000000000000001002)
    │   └─ ← [Return]
    ├─ [42255] Lottery::enter{value: 10000000000000000}()
    │   ├─ emit LotteryEntered(player:
0x0000000000000000000000000000000000001002, amount: 10000000000000000
[1e16])
    │   └─ ← [Stop]
    ├─ [32191] Lottery::pickWinner()
    │   ├─ [0] 0x0000000000000000000000000000000000001002::fallback{value:
10000000000000000}()
    │   │   └─ ← [Stop]
    │   ├─ emit WinnerPicked(winner:
0x0000000000000000000000000000000000001002, prize: 10000000000000000 [1e16])
```

```
    │   └─ ← [Stop]

    ├─ [0] console::log("Reentrancy guard test passed across multiple lottery cycles.")
[staticcall]

    │   └─ ← [Stop]

    └─ ← [Stop]


[PASS] test_owner_is_correct() (gas: 21275)

Logs:

  Lottery owner: 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496

  Initial Lottery balance: 0 ether


Traces:

  [21275] LotteryTest::test_owner_is_correct()

    ├─ [2529] Lottery::owner() [staticcall]

    │   └─ ← [Return] LotteryTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496]

    ├─ [0] VM::assertEq(LotteryTest:
[0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], LotteryTest:
[0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], "Owner is not set correctly in
constructor") [staticcall]

    │   └─ ← [Return]

    ├─ [822] Lottery::isOwner(LotteryTest:
[0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496]) [staticcall]

    │   └─ ← [Return] true

    ├─ [0] VM::assertTrue(true, "Owner check failed for owner address") [staticcall]

    │   └─ ← [Return]

    ├─ [822] Lottery::isOwner(0x0000000000000000000000000000000000001001)
[staticcall]

    │   └─ ← [Return] false
```

```
    ├─ [0] VM::assertFalse(false, "isOwner returned true for non-owner") [staticcall]
    │   └─ ← [Return]
    └─ ← [Stop]

[PASS] test_pickWinner_noPlayers() (gas: 35410)
Logs:
  Lottery owner: 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496
  Initial Lottery balance: 0 ether

Traces:
  [35410] LotteryTest::test_pickWinner_noPlayers()
    ├─ [0] VM::expectRevert(custom error 0xf28dceb3:  No players in the lottery)
    │   └─ ← [Return]
    ├─ [27113] Lottery::pickWinner()
    │   └─ ← [Revert] revert: No players in the lottery
    └─ ← [Stop]

[PASS] test_pickWinner_notOwner() (gas: 70974)
Logs:
  Lottery owner: 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496
  Initial Lottery balance: 0 ether

Traces:
  [70974] LotteryTest::test_pickWinner_notOwner()
    ├─ [2433] Lottery::getEntranceFee() [staticcall]
    │   └─ ← [Return] 10000000000000000 [1e16]
```

```
    ├─ [0] VM::prank(0x0000000000000000000000000000000000001001)
    │   └─ ← [Return]
    ├─ [46255] Lottery::enter{value: 10000000000000000}()
    │   ├─ emit LotteryEntered(player:
0x0000000000000000000000000000000000001001, amount: 10000000000000000
[1e16])
    │   └─ ← [Stop]
    ├─ [0] VM::prank(0x0000000000000000000000000000000000001001)
    │   └─ ← [Return]
    ├─ [0] VM::expectRevert(custom error 0xf28dceb3:  "!Only owner can call this function)
    │   └─ ← [Return]
    ├─ [2680] Lottery::pickWinner()
    │   └─ ← [Revert] revert: !Only owner can call this function
    └─ ← [Stop]


[PASS] test_pickWinner_success_multiplePlayers() (gas: 131916)
Logs:
  Lottery owner: 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496
  Initial Lottery balance: 0 ether
  Multiple-player lottery cycle successful. Winner:
0x0000000000000000000000000000000000001001, Prize: 20000000000000000 wei


Traces:
  [170161] LotteryTest::test_pickWinner_success_multiplePlayers()
    ├─ [2433] Lottery::getEntranceFee() [staticcall]
    │   └─ ← [Return] 10000000000000000 [1e16]
    ├─ [0] VM::prank(0x0000000000000000000000000000000000001001)
```

│ └─ ← [Return]

├─ [46255] Lottery::enter{value: 10000000000000000}()

│ ├─ emit LotteryEntered(player: 0x0000000000000000000000000000000000001001, amount: 10000000000000000 [1e16])

│ └─ ← [Stop]

├─ [0] VM::prank(0x0000000000000000000000000000000000001002)

│ └─ ← [Return]

├─ [24355] Lottery::enter{value: 10000000000000000}()

│ ├─ emit LotteryEntered(player: 0x0000000000000000000000000000000000001002, amount: 10000000000000000 [1e16])

│ └─ ← [Stop]

├─ [58246] Lottery::pickWinner()

│ ├─ [0] 0x0000000000000000000000000000000000001001::fallback{value: 20000000000000000}()

│ │ └─ ← [Stop]

│ ├─ emit WinnerPicked(winner: 0x0000000000000000000000000000000000001001, prize: 20000000000000000 [2e16])

│ └─ ← [Stop]

├─ [553] Lottery::recentWinner() [staticcall]

│ └─ ← [Return] 0x0000000000000000000000000000000000001001

├─ [0] VM::assertTrue(true, "Winner must be either user1 or user2") [staticcall]

│ └─ ← [Return]

├─ [0] VM::assertEq(10010000000000000000 [1.001e19], 10010000000000000000 [1.001e19], "User1 prize amount mismatch") [staticcall]

│ └─ ← [Return]

├─ [484] Lottery::getPlayersCount() [staticcall]

│ └─ ← [Return] 0

├─ [0] VM::assertEq(0, 0, "Players list was not reset after picking a winner") [staticcall]

│ └─ ← [Return]

├─ [0] console::log("Multiple-player lottery cycle successful. Winner: %s, Prize: %d wei", 0x0000000000000000000000000000000000001001, 20000000000000000 [2e16]) [staticcall]

│ └─ ← [Stop]

└─ ← [Stop]

[PASS] test_pickWinner_success_singlePlayer() (gas: 104167)

Logs:

Lottery owner: 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496

Initial Lottery balance: 0 ether

Single-player lottery cycle successful. Winner: 0x0000000000000000000000000000000000001001, Prize: 10000000000000000 wei

Traces:

[135474] LotteryTest::test_pickWinner_success_singlePlayer()

├─ [2433] Lottery::getEntranceFee() [staticcall]

│ └─ ← [Return] 10000000000000000 [1e16]

├─ [0] VM::prank(0x0000000000000000000000000000000000001001)

│ └─ ← [Return]

├─ [46255] Lottery::enter{value: 10000000000000000}()

│ ├─ emit LotteryEntered(player: 0x0000000000000000000000000000000000001001, amount: 10000000000000000 [1e16])

│ └─ ← [Stop]

├─ [58091] Lottery::pickWinner()

│ ├─ [0] 0x0000000000000000000000000000000000001001::fallback{value: 10000000000000000}()

│ │ └─ ← [Stop]

│ ├─ emit WinnerPicked(winner: 0x0000000000000000000000000000000000001001, prize: 10000000000000000 [1e16])

│ └─ ← [Stop]

├─ [553] Lottery::recentWinner() [staticcall]

│ └─ ← [Return] 0x0000000000000000000000000000000000001001

├─ [0] VM::assertEq(0x0000000000000000000000000000000000001001, 0x0000000000000000000000000000000000001001, "The recentWinner should be user1") [staticcall]

│ └─ ← [Return]

├─ [0] VM::assertEq(10000000000000000000 [1e19], 10000000000000000000 [1e19], "Winner did not receive the correct prize amount") [staticcall]

│ └─ ← [Return]

├─ [484] Lottery::getPlayersCount() [staticcall]

│ └─ ← [Return] 0

├─ [0] VM::assertEq(0, 0, "Players list was not reset after picking a winner") [staticcall]

│ └─ ← [Return]

├─ [0] console::log("Single-player lottery cycle successful. Winner: %s, Prize: %d wei", 0x0000000000000000000000000000000000001001, 10000000000000000 [1e16]) [staticcall]

│ └─ ← [Stop]

└─ ← [Stop]

Suite result: ok. 11 passed; 0 failed; 0 skipped; finished in 1.64ms (752.92µs CPU time)

Ran 1 test suite in 15.68ms (1.64ms CPU time): 11 tests passed, 0 failed, 0 skipped (11 total tests)

🧹Start Cleaning

[✅PASS]Moved test file to 'finished'

Generated Test Cases:

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.18;


import {Test, console} from "forge-std/Test.sol";

// Import the Lottery contract from the provided file path

import {Lottery} from "../src/Lottery_21197014_1747561121.sol";


// The test contract for the Lottery smart contract.

// This contract covers the core business flows:

// 1. Players entering the lottery via the enter() function, and the fallback/receive functions.

// 2. Validating the entrance fee requirement.

// 3. Picking a winner through the pickWinner() function, including access control and
proper funds distribution.

// 4. Verifying the state reset after picking a winner.

// 5. General helper functions: getPlayersCount, getEntranceFee and isOwner().

// 6. Demonstrating potential reentrancy safe behavior with the noReentrancy modifier.

contract LotteryTest is Test {

   Lottery public lottery;

   address public owner;

   address public user1;

   address public user2;
```

```
// setUp function to deploy the Lottery contract and setup test accounts.
function setUp() public {
    // The deploying address (this contract) will be the owner.
    lottery = new Lottery();
    owner = address(this);

    // Define two test user addresses
    user1 = address(0x1001);
    user2 = address(0x1002);

    // Fund test user addresses with ample ETH to perform lottery entries.
    vm.deal(user1, 10 ether);
    vm.deal(user2, 10 ether);

    // Log the owner and initial contract balance.
    console.log("Lottery owner: %s", owner);
    console.log("Initial Lottery balance: %d ether", address(lottery).balance / 1 ether);
}

// Test that the contract owner is set correctly upon deployment,
// and the isOwner() function behaves as expected.
function test_owner_is_correct() public {
    assertEq(lottery.owner(), owner, "Owner is not set correctly in constructor");
    assertTrue(lottery.isOwner(owner), "Owner check failed for owner address");
    assertFalse(lottery.isOwner(user1), "isOwner returned true for non-owner");
```

```
    }

    // Test that entering the lottery with an insufficient fee reverts.
    function test_enter_insufficientFee() public {
        vm.prank(user1);
        // Expect revert due to insufficient entrance fee: must be at least 0.01 ETH.
        vm.expectRevert("Minimum 0.01 ETH required");
        lottery.enter{value: 0.009 ether}();
    }

    // Test a successful entry using the enter() function.
    function test_enter_success_via_enter() public {
        vm.prank(user1);
        uint256 fee = lottery.getEntranceFee();
        lottery.enter{value: fee}();
        uint256 playersCount = lottery.getPlayersCount();
        assertEq(playersCount, 1, "Players count should be 1 after one entry");

        // Debug log
        console.log("User %s has successfully entered the lottery with %d wei", user1, fee);
    }

    // Test that entering the lottery via the receive() fallback function also registers the
player.
    function test_enter_via_receive() public {
        vm.prank(user1);
```

```
    uint256 fee = lottery.getEntranceFee();
    // Calling the lottery contract with ETH and no data should trigger the receive()
function.
    (bool success, ) = address(lottery).call{value: fee}("");
    require(success, "Calling receive failed");
    uint256 playersCount = lottery.getPlayersCount();
    assertEq(playersCount, 1, "Players count should be 1 after receiving ETH");

    console.log("User %s entered via receive/fallback with %d wei", user1, fee);
  }

  // Test multiple players entering the lottery.
  function test_multiple_entries() public {
    uint256 fee = lottery.getEntranceFee();

    vm.prank(user1);
    lottery.enter{value: fee}();
    vm.prank(user2);
    lottery.enter{value: fee}();

    uint256 playersCount = lottery.getPlayersCount();
    assertEq(playersCount, 2, "Players count should be 2 after two entries");

    console.log("Multiple entries successful: %d players", playersCount);
  }
```

```
// Test that pickWinner() reverts when there are no players in the lottery.
function test_pickWinner_noPlayers() public {
    vm.expectRevert("No players in the lottery");
    lottery.pickWinner();
}

// Test that only the owner can call pickWinner().
function test_pickWinner_notOwner() public {
    uint256 fee = lottery.getEntranceFee();

    // Let user1 enter the lottery.
    vm.prank(user1);
    lottery.enter{value: fee}();

    // Now, try to call pickWinner() from user1 (non-owner). Expected to revert.
    vm.prank(user1);
    vm.expectRevert("!Only owner can call this function");
    lottery.pickWinner();
}

// Test a full lottery cycle with a single player: entry and then picking a winner.
// In this case, the only player should automatically be the winner.
function test_pickWinner_success_singlePlayer() public {
    uint256 fee = lottery.getEntranceFee();

    // user1 enters the lottery.
```

```
    vm.prank(user1);
    lottery.enter{value: fee}();

    // Record the lottery balance which is the prize.
    uint256 lotteryBalance = address(lottery).balance;
    uint256 user1BalanceBefore = user1.balance;

    // Owner picks the winner.
    lottery.pickWinner();

    // The only player should be the winner.
    address winner = lottery.recentWinner();
    assertEq(winner, user1, "The recentWinner should be user1");

    // Check that the winner received the prize.
    uint256 user1BalanceAfter = user1.balance;
    assertEq(user1BalanceAfter, user1BalanceBefore + lotteryBalance, "Winner did not
receive the correct prize amount");

    // After picking the winner, the players array should have been reset.
    uint256 playersCount = lottery.getPlayersCount();
    assertEq(playersCount, 0, "Players list was not reset after picking a winner");

    console.log("Single-player lottery cycle successful. Winner: %s, Prize: %d wei", winner,
lotteryBalance);
  }
```

```
// Test a lottery cycle with multiple players.
// Since randomness is based on block properties and the players count,
// with two players, the winner is deterministic modulo 2.
function test_pickWinner_success_multiplePlayers() public {
    uint256 fee = lottery.getEntranceFee();

    vm.prank(user1);
    lottery.enter{value: fee}();
    vm.prank(user2);
    lottery.enter{value: fee}();

    uint256 lotteryBalance = address(lottery).balance;

    // Record balances of both participants before picking a winner.
    uint256 user1BalanceBefore = user1.balance;
    uint256 user2BalanceBefore = user2.balance;

    // Owner picks the winner.
    lottery.pickWinner();
    address winner = lottery.recentWinner();

    // Validate that the winner is either user1 or user2.
    bool validWinner = (winner == user1) || (winner == user2);
    assertTrue(validWinner, "Winner must be either user1 or user2");

    // Check that the winner received the lottery prize.
```

```
        if (winner == user1) {
            uint256 user1BalanceAfter = user1.balance;
            assertEq(user1BalanceAfter, user1BalanceBefore + lotteryBalance, "User1 prize
amount mismatch");
        } else {
            uint256 user2BalanceAfter = user2.balance;
            assertEq(user2BalanceAfter, user2BalanceBefore + lotteryBalance, "User2 prize
amount mismatch");
        }

        // Ensure the players array is cleared.
        uint256 playersCount = lottery.getPlayersCount();
        assertEq(playersCount, 0, "Players list was not reset after picking a winner");

        console.log("Multiple-player lottery cycle successful. Winner: %s, Prize: %d wei",
winner, lotteryBalance);
    }

    // Test the reentrancy guard indirectly by ensuring consecutive calls of pickWinner() do
not conflict.
    // Since separate lottery cycles are processed without reentrancy issues, multiple valid
calls should succeed.
    // Note: A direct reentrancy simulation would require a custom attacker contract.
    function test_no_reentrancy_lock_reset() public {
        uint256 fee = lottery.getEntranceFee();

        // First lottery cycle with user1.
        vm.prank(user1);
```

```
        lottery.enter{value: fee}();
        lottery.pickWinner();

        // Second lottery cycle with user2.
        vm.prank(user2);
        lottery.enter{value: fee}();
        lottery.pickWinner();

        // If reentrancy had occurred, one of the calls would have reverted.
        // Successful execution of two rounds indicates that the noReentrancy guard resets
properly.
        console.log("Reentrancy guard test passed across multiple lottery cycles.");
    }

    // Fallback and receive functions in the Lottery contract allow direct ETH transfers
    // to also enter the lottery. Test that sending ETH without calldata registers an entry.
    function test_fallback_receive_entry() public {
        uint256 fee = lottery.getEntranceFee();
        vm.prank(user1);
        (bool success, ) = address(lottery).call{value: fee}("0x");
        require(success, "Fallback/receive entry failed");

        uint256 playersCount = lottery.getPlayersCount();
        assertEq(playersCount, 1, "Player entry via fallback/receive did not register");

        console.log("Fallback/receive entry test successful for user: %s", user1);
```

```
    }


    // The contract must be able to receive ETH, so include empty receive and fallback
functions.

    receive() external payable {}

    fallback() external payable {}

}
```