

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 // One shouldn't use any values from inside the blockchain as randomness
5 // Use something like Chainlink VRF for verifiable randomness
6 // https://docs.chain.link/docs/get-a-random-number/
7
8 contract BadRNG {
9     address payable[] private s_players;
10
11     function enterRaffle() external payable {
12         require(msg.value >= 1000000000000000000);
13         s_players.push(payable(msg.sender));
14     }
15
16     function pickWinner() external {
17         uint256 randomWinnerIndex = uint256(
18             keccak256(abi.encodePacked(block.difficulty, msg.sender))
19         );
20         address winner = s_players[randomWinnerIndex % s_players.length];
21         (bool success, ) = winner.call{value: address(this).balance}("");
22         require(success, "Transfer failed");
23     }
24 }
25
26 // How could you make a contract that exploits this?
27
```

The screenshot shows the Solidity IDE interface. On the left, the 'FILE EXPLORERS' sidebar displays the 'Workspaces' section with a tree view containing 'BadRNG.sol', 'MetamorphicContract.sol', and 'LiquidityPoolAsOracle.sol'. The 'LiquidityPoolAsOracle.sol' file is selected. The main editor area shows the code for the 'LiquidityPoolAsOracle' contract. The code includes imports for 'IERC20' and 'IERC20.sol', a constructor that initializes 's_token1' and 's_token2' addresses, a 'swap' function that checks for token validity and balance, and an 'addLiquidity' function that transfers tokens to the contract.

```

10
11 import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
12 import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
13
14 contract LiquidityPoolAsOracle {
15     address public s_token1;
16     address public s_token2;
17
18     constructor(address token1, address token2) {
19         require(token1 != address(0x0), "Address cannot be 0");
20         require(token2 != address(0x0), "Address cannot be 0");
21         s_token1 = token1;
22         s_token2 = token2;
23     }
24
25     function swap(
26         address from,
27         address to,
28         uint256 amount
29     ) external {
30         require(
31             (from == s_token1 && to == s_token2) || (from == s_token2 && to == s_token1),
32             "Invalid tokens"
33         );
34         require(IERC20(from).balanceOf(msg.sender) >= amount, "Not enough to swap");
35         uint256 swap_amount = getSwapPrice(from, to, amount);
36         bool txFromSuccess = IERC20(from).transferFrom(msg.sender, address(this), amount);
37         require(txFromSuccess, "Failed to transfer from");
38         bool txToSuccess = IERC20(to).transfer(msg.sender, swap_amount);
39         require(txToSuccess, "Failed to transfer to");
40     }
41
42     function addLiquidity(address tokenAddress, uint256 amount) external {
43         bool success = IERC20(tokenAddress).transferFrom(msg.sender, address(this), amount);
44         require(success, "Failed to add liquidity");
45     }
46 }

```

The screenshot shows the VS Code editor interface. On the left, the 'FILE EXPLORERS' sidebar is open, displaying a project structure with three files: 'BadRNG.sol', 'MetamorphicContract.sol', and 'LiquidityPoolAsOracle.sol'. The 'LiquidityPoolAsOracle.sol' file is selected and its content is displayed in the main editor area. The code is written in Solidity and includes functions for token swapping and adding liquidity. The code is as follows:

```
28     uint256 amount
29 } external {
30     require(
31         (from == s_token1 && to == s_token2) || (from == s_token2 && to == s_token1),
32         "Invalid tokens"
33     );
34     require(IERC20(from).balanceOf(msg.sender) >= amount, "Not enough to swap");
35     uint256 swap_amount = getSwapPrice(from, to, amount);
36     bool txFromSuccess = IERC20(from).transferFrom(msg.sender, address(this), amount);
37     require(txFromSuccess, "Failed to transfer from");
38     bool txToSuccess = IERC20(to).transfer(msg.sender, swap_amount);
39     require(txToSuccess, "Failed to transfer to");
40 }
41
42 function addLiquidity(address tokenAddress, uint256 amount) external {
43     bool success = IERC20(tokenAddress).transferFrom(msg.sender, address(this), amount);
44     require(success, "Failed to add liquidity");
45 }
46
47 function getSwapPrice(
48     address from,
49     address to,
50     uint256 amount
51 ) public view returns (uint256) {
52     return ((amount * IERC20(to).balanceOf(address(this))) /
53         IERC20(from).balanceOf(address(this)));
54 }
55 }
56
57 // How could you make a contract that exploits this Dex?
58
```

The screenshot shows the VS Code editor interface. On the left, the 'FILE EXPLORERS' sidebar is open, displaying a project structure with three files: 'BadRNG.sol', 'MetamorphicContract.sol', and 'LiquidityPoolAsOracle.sol'. The 'MetamorphicContract.sol' file is selected and its content is displayed in the main editor area. The code is written in Solidity and includes a contract named 'MetamorphicContract' that is 'Initializable'. The code is as follows:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.7;
3 import "@openzeppelin/contracts/proxy/utils/Initializable.sol";
4
5 contract MetamorphicContract is Initializable {
6     address payable owner;
7
8     function kill() external {
9         require(msg.sender == owner);
10         selfdestruct(owner);
11     }
12 }
13
```