**Testing and refactoring the performUpkeep**

**Testing and refactoring the performUpkeep**

Let's give some love to performUpkeep, starting with some tests.

Starting light, open the RaffleTest.t.sol and paste the following:

function testPerformUpkeepCanOnlyRunIfCheckUpkeepIsTrue() public {

// Arrange

vm.prank(PLAYER);

raffle.enterRaffle{value: entranceFee}();

vm.warp(block.timestamp + interval + 1);

vm.roll(block.number + 1);

// Act / Assert

// It doesnt revert

raffle.performUpkeep("");

}

We prank the PLAYER address, then use it to call enterRaffle with the correct entranceFee. We use the warp and roll to set block.timestamp into the future. Lastly, we call performUpkeep.

As you've figured out, we are not running any asserts here. But that is ok because if performUpkeep had a reason to fail, then it would have reverted and our forge test would have caught it.

Run the test using: forge test --mt testPerformUpkeepCanOnlyRunIfCheckUpkeepIsTrue

It passes, amazing!

Keep going! Let's test if performUpkeep reverts in case checkUpkeep is false:

function testPerformUpkeepRevertsIfCheckUpkeepIsFalse() public {

// Arrange

uint256 currentBalance = 0;

uint256 numPlayers = 0;

Raffle.RaffleState rState = raffle.getRaffleState();

// Act / Assert

vm.expectRevert(

abi.encodeWithSelector(

Raffle.Raffle\_\_UpkeepNotNeeded.selector,

currentBalance,

numPlayers,

rState

)

);

raffle.performUpkeep("");

}

This can be understood easier if we start from the end. We want to call performUpkeep and we expect it to revert. For that, we use the vm.expectRevert to indicate that we expect the next call to revert. If we access [this link](https://book.getfoundry.sh/cheatcodes/expect-revert) we can see that in case we use a custom error with parameters we can specify them as follows:

vm.expectRevert(

abi.encodeWithSelector(CustomError.selector, 1, 2)

);

In our case the custom error has 3 parameters:

error Raffle\_\_UpkeepNotNeeded(

uint256 currentBalance,

uint256 numPlayers,

uint256 raffleState

);

First parameter: Raffle.Raffle\_\_UpkeepNotNeeded.selector; Second parameter: currentBalance; Third parameter: numPlayers; Fourth parameter: raffleState;

Out of all of them, the only one available is the first. We define a currentBalance and numPlayers and assign them both 0. To get the raffleState we can use the getRaffleState view function.

Run the test using: forge test --mt testPerformUpkeepRevertsIfCheckUpkeepIsFalse

Everything passes, great!

I know some concepts haven't been explained. I'm referring to encodeWithSelector and the general concept of function selectors. These will be introduced in the next sections.

Great work! Now let's further explore events.