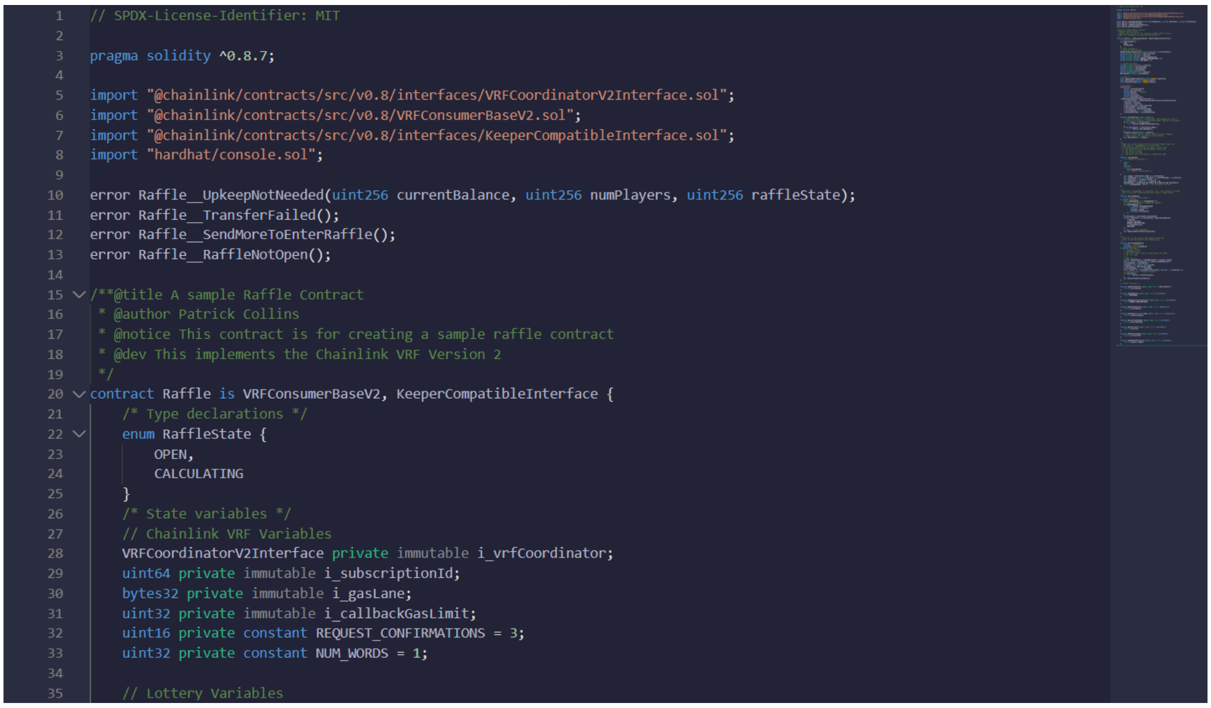
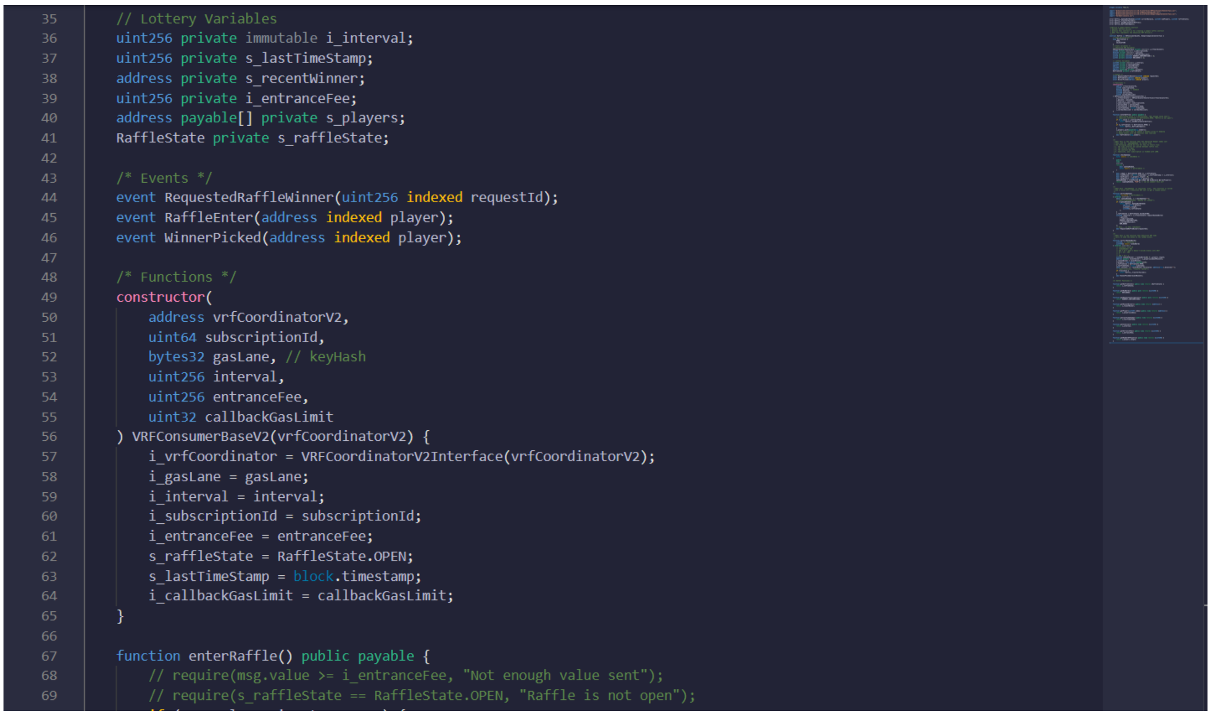
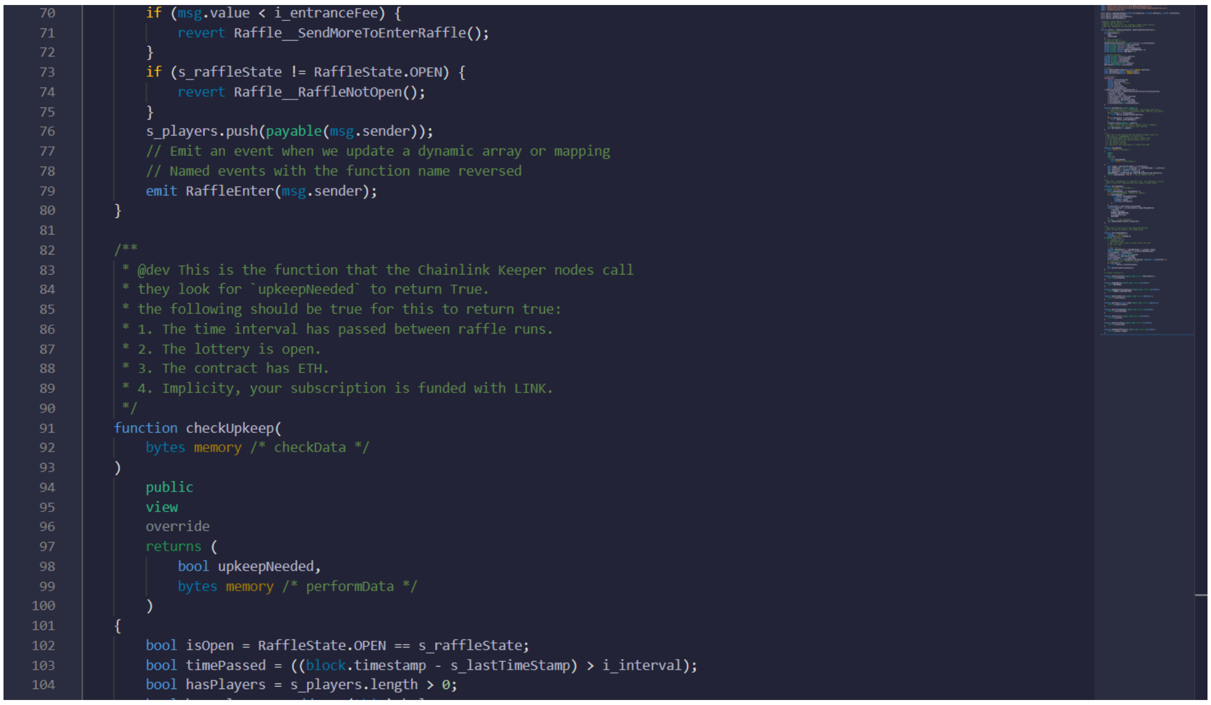
Hardhat Smart Contract Lottery



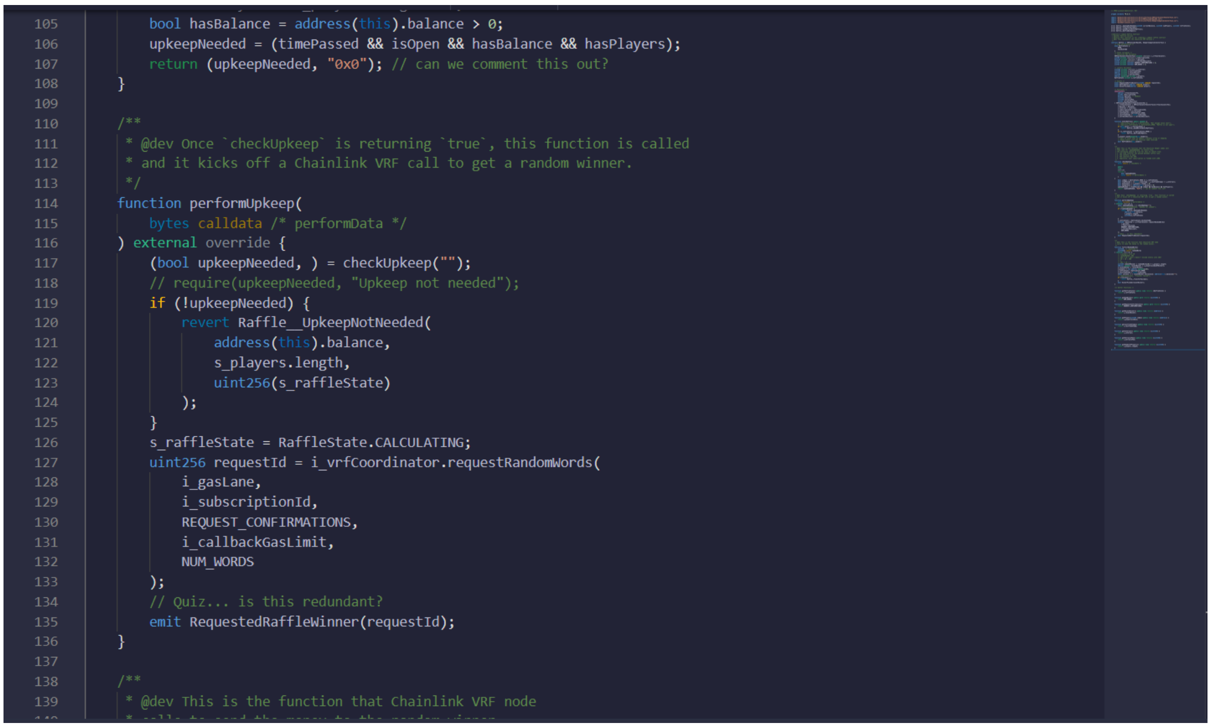
```solidity
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

import "@chainlink/contracts/src/v0.8/interfaces/VRFCoordinatorV2Interface.sol";
import "@chainlink/contracts/src/v0.8/VRFConsumerBaseV2.sol";
import "@chainlink/contracts/src/v0.8/interfaces/KeeperCompatibleInterface.sol";
import "hardhat/console.sol";

error Raffle__UpkeepNotNeeded(uint256 currentBalance, uint256 numPlayers, uint256 raffleState);
error Raffle__TransferFailed();
error Raffle__SendMoreToEnterRaffle();
error Raffle__RaffleNotOpen();

/**@title A sample Raffle Contract
 * @author Patrick Collins
 * @notice This contract is for creating a sample raffle contract
 * @dev This implements the Chainlink VRF Version 2
 */
contract Raffle is VRFConsumerBaseV2, KeeperCompatibleInterface {
    /* Type declarations */
    enum RaffleState {
        OPEN,
        CALCULATING
    }
    /* State variables */
    // Chainlink VRF Variables
    VRFCoordinatorV2Interface private immutable i_vrfCoordinator;
    uint64 private immutable i_subscriptionId;
    bytes32 private immutable i_gasLane;
    uint32 private immutable i_callbackGasLimit;
    uint16 private constant REQUEST_CONFIRMATIONS = 3;
    uint32 private constant NUM_WORDS = 1;

    // Lottery Variables
```



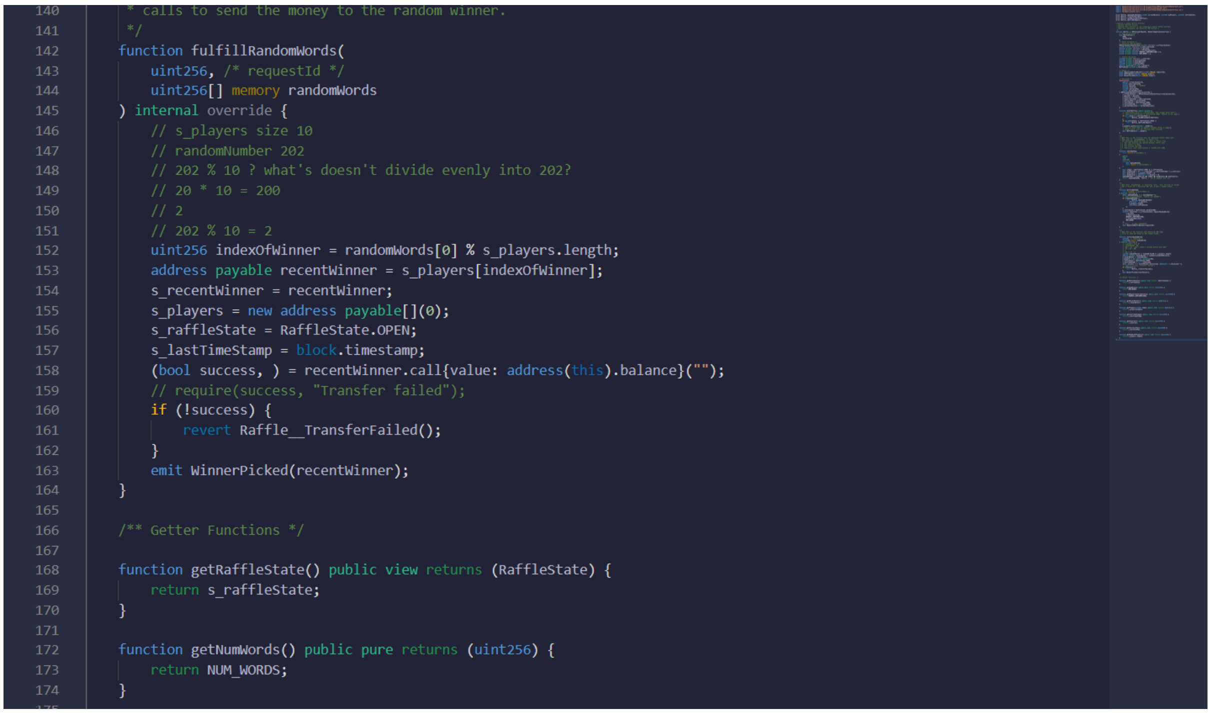
```solidity
    // Lottery Variables
    uint256 private immutable i_interval;
    uint256 private s_lastTimeStamp;
    address private s_recentWinner;
    uint256 private i_entranceFee;
    address payable[] private s_players;
    RaffleState private s_raffleState;

    /* Events */
    event RequestedRaffleWinner(uint256 indexed requestId);
    event RaffleEnter(address indexed player);
    event WinnerPicked(address indexed player);

    /* Functions */
    constructor(
        address vrfCoordinatorV2,
        uint64 subscriptionId,
        bytes32 gasLane, // keyHash
        uint256 interval,
        uint256 entranceFee,
        uint32 callbackGasLimit
    ) VRFConsumerBaseV2(vrfCoordinatorV2) {
        i_vrfCoordinator = VRFCoordinatorV2Interface(vrfCoordinatorV2);
        i_gasLane = gasLane;
        i_interval = interval;
        i_subscriptionId = subscriptionId;
        i_entranceFee = entranceFee;
        s_raffleState = RaffleState.OPEN;
        s_lastTimeStamp = block.timestamp;
        i_callbackGasLimit = callbackGasLimit;
    }

    function enterRaffle() public payable {
        // require(msg.value >= i_entranceFee, "Not enough value sent");
        // require(s_raffleState == RaffleState.OPEN, "Raffle is not open");
```



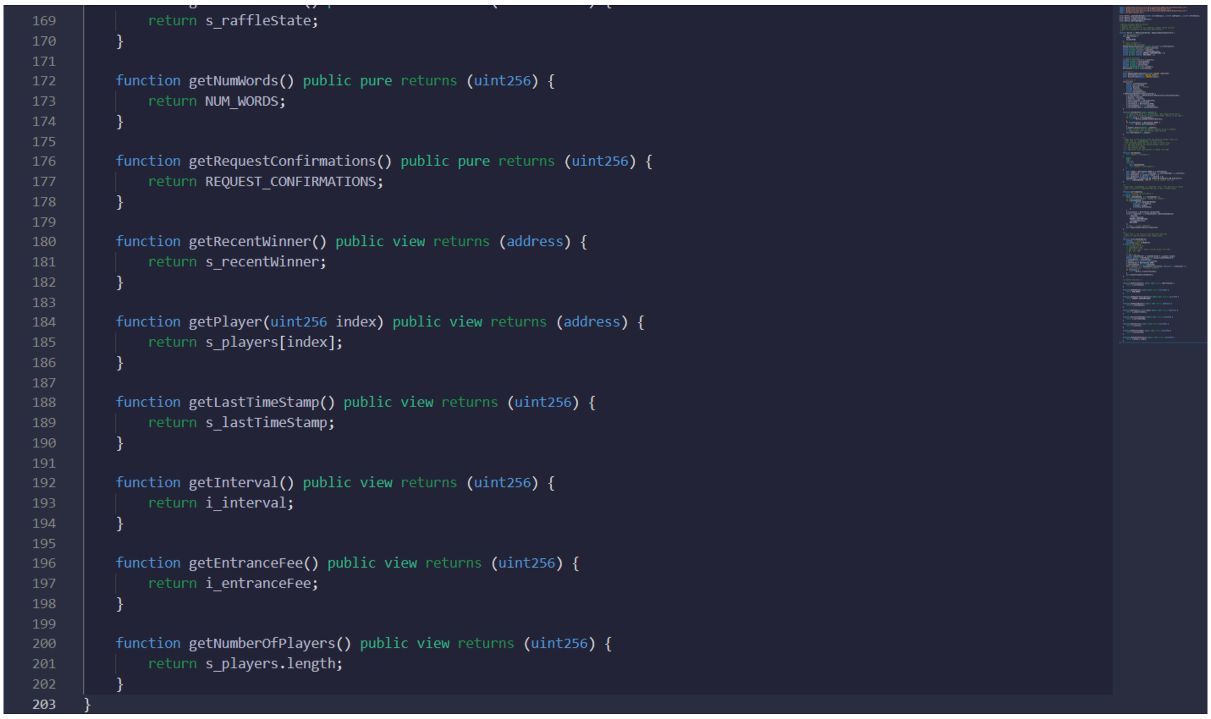
```solidity
        if (msg.value < i_entranceFee) {
            revert Raffle__SendMoreToEnterRaffle();
        }
        if (s_raffleState != RaffleState.OPEN) {
            revert Raffle__RaffleNotOpen();
        }
        s_players.push(payable(msg.sender));
        // Emit an event when we update a dynamic array or mapping
        // Named events with the function name reversed
        emit RaffleEnter(msg.sender);
    }

    /**
     * @dev This is the function that the Chainlink Keeper nodes call
     * they look for `upkeepNeeded` to return True.
     * the following should be true for this to return true:
     * 1. The time interval has passed between raffle runs.
     * 2. The lottery is open.
     * 3. The contract has ETH.
     * 4. Implicity, your subscription is funded with LINK.
     */
    function checkUpkeep(
        bytes memory /* checkData */
    )
        public
        view
        override
        returns (
            bool upkeepNeeded,
            bytes memory /* performData */
        )
    {
        bool isOpen = RaffleState.OPEN == s_raffleState;
        bool timePassed = ((block.timestamp - s_lastTimeStamp) > i_interval);
        bool hasPlayers = s_players.length > 0;
```



```solidity
        bool hasBalance = address(this).balance > 0;
        upkeepNeeded = (timePassed && isOpen && hasBalance && hasPlayers);
        return (upkeepNeeded, "0x0"); // can we comment this out?
    }

    /**
     * @dev Once `checkUpkeep` is returning `true`, this function is called
     * and it kicks off a Chainlink VRF call to get a random winner.
     */
    function performUpkeep(
        bytes calldata /* performData */
    ) external override {
        (bool upkeepNeeded, ) = checkUpkeep("");
        // require(upkeepNeeded, "Upkeep not needed");
        if (!upkeepNeeded) {
            revert Raffle__UpkeepNotNeeded(
                address(this).balance,
                s_players.length,
                uint256(s_raffleState)
            );
        }
        s_raffleState = RaffleState.CALCULATING;
        uint256 requestId = i_vrfCoordinator.requestRandomWords(
            i_gasLane,
            i_subscriptionId,
            REQUEST_CONFIRMATIONS,
            i_callbackGasLimit,
            NUM_WORDS
        );
        // Quiz... is this redundant?
        emit RequestedRaffleWinner(requestId);
    }

    /**
     * @dev This is the function that Chainlink VRF node
```



```
     * calls to send the money to the random winner.
     */
    function fulfillRandomWords(
        uint256, /* requestId */
        uint256[] memory randomWords
    ) internal override {
        // s_players size 10
        // randomNumber 202
        // 202 % 10 ? what's doesn't divide evenly into 202?
        // 20 * 10 = 200
        // 2
        // 202 % 10 = 2
        uint256 indexOfWinner = randomWords[0] % s_players.length;
        address payable recentWinner = s_players[indexOfWinner];
        s_recentWinner = recentWinner;
        s_players = new address payable[](0);
        s_raffleState = RaffleState.OPEN;
        s_lastTimeStamp = block.timestamp;
        (bool success, ) = recentWinner.call{value: address(this).balance}("");
        // require(success, "Transfer failed");
        if (!success) {
            revert Raffle__TransferFailed();
        }
        emit WinnerPicked(recentWinner);
    }

    /** Getter Functions */

    function getRaffleState() public view returns (RaffleState) {
        return s_raffleState;
    }

    function getNumWords() public pure returns (uint256) {
        return NUM_WORDS;
    }
```



```
        return s_raffleState;
    }

    function getNumWords() public pure returns (uint256) {
        return NUM_WORDS;
    }
```

```

    function getRequestConfirmations() public pure returns (uint256) {
        return REQUEST_CONFIRMATIONS;
    }

    function getRecentWinner() public view returns (address) {
        return s_recentWinner;
    }

    function getPlayer(uint256 index) public view returns (address) {
        return s_players[index];
    }

    function getLastTimeStamp() public view returns (uint256) {
        return s_lastTimeStamp;
    }

    function getInterval() public view returns (uint256) {
        return i_interval;
    }

    function getEntranceFee() public view returns (uint256) {
        return i_entranceFee;
    }

    function getNumberOfPlayers() public view returns (uint256) {
        return s_players.length;
    }
}
```