# 541.https://stackoverflow.com/questions/70246295/how-can-i-implement-chainlink-vrf-with-giving-reward-to-a-random-owner-of-my-nft

**T:**How can I implement chainlink vrf with giving reward to a RANDOM owner of my nft when someone mints it?

**Q:**I am creating a nft collection and I want the random owner to receive a percentage of the mint price with each mint. But since I need to wait for the VRF response, can't figure out how to implement randomness function and function that will send the percentage to the vrf response (random owner).  
  
pragma solidity >=0.7.0 <0.9.0;import "@chainlink/contracts/src/v0.8/VRFConsumerBase.sol";contract Sofb is ERC721Enumerable, Ownable, VRFConsumerBase {using Strings for uint256;string baseURI;string public baseExtension = ".json";uint256 public cost = 0.015 ether;uint256 public maxSupply = 7070;uint256 public tokenCounter;bool public paused = false;bool public revealed = false;string public notRevealedUri;bytes32 internal keyHash;uint256 internal fee;uint256 public randomResult = 0;address payable giftAddress = payable(msg.sender);uint256 giftValue = 0;mapping(bytes32 => uint256) public requestIdToRandomNumber;mapping(bytes32 => address) public requestIdToAddress;mapping(bytes32 => uint256) public requestIdToRequestNumberIndex;uint256 public requestCounter; constructor(string memory \_name, string memory \_symbol, string memory \_initBaseURI, string memory \_initNotRevealedUri, address \_vrfCoordinator, address \_linkToken, bytes32 \_keyHash, uint256 \_fee) VRFConsumerBase(\_vrfCoordinator, \_linkToken) ERC721(\_name, \_symbol) { setBaseURI(\_initBaseURI); setNotRevealedURI(\_initNotRevealedUri); keyHash = \_keyHash; fee = \_fee; }// internalfunction \_baseURI() internal view virtual override returns (string memory) { return baseURI;}// publicfunction getRandomNumber() public returns (bytes32 requestId) { require(LINK.balanceOf(address(this)) >= fee, "Not enough LINK - fill contract with faucet"); requestIdToAddress[requestId] = msg.sender; requestIdToRequestNumberIndex[requestId] = requestCounter; requestCounter += 1; return requestRandomness(keyHash, fee);}function fulfillRandomness(bytes32 requestId, uint256 randomness) internal override { requestIdToRandomNumber[requestId] = randomness; uint256 requestNumber = requestIdToRequestNumberIndex[requestId];}function mint() public payable { uint256 supply = totalSupply(); require(!paused); require(supply + 1 <= maxSupply); require(msg.value >= cost); if (msg.sender != owner()) { require(msg.value >= cost); } if (supply > 0) { require(randomResult > 0); giftAddress = payable(ownerOf(randomResult)); giftValue = ((supply + 1 == 5) || (supply + 1 == 10)) ? address(this).balance \* 1 / 100 : msg.value \* 10 / 100; (bool success, ) = payable(giftAddress).call{value: giftValue}(""); require(success); } \_safeMint(msg.sender, supply + 1); getRandomNumber(); } ...}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

**C1:**How about this? 1. Crete a mapping of owners; 2. Store mapping length as uint variable and increment/decrement it properly; 3. When a user initiates a mint transaction, after all of the require statements, call VRF within the range based on mappingLength variable; 4. Inside the fulfillRandomness create a new token and send the percentage of the mint price to the random owner from the mapping based on the callback number

**C2:**@Andrej, Would you be willing to post that as an answer, perhaps even with a bit of psuedocode?

1 **Answer**

**A1:**you should mint the token and put the mint logic inside the fulfillRandomness function, to access the data you will need i will recommend you to have an array of structs with the data you need, remember that random returns a requestId, use the request id as an index to get the stored data and then mint it, it's on you if you want to delete the data after mint the nft