# 485.https://stackoverflow.com/questions/70745661/lock-transfer-of-erc721-nft

**T:**Lock transfer of erc721 NFT

**Q:**Hi all I am building a blockchain-based game around an NFT project and am looking to understand if it's possible to implement the following.  
  
Have a method on the NFT contract that when called can locks the transfer of all minted NFT's for a period. A bit like a game of tag than when your tag the contract all the (NFT / players) cant (move /transfer)  
  
I assume I would need overide the transfer method then do a boolean check.Something like  
  
 function safeTransferFrom( address from, address to, uint256 tokenId ) public virtual override { if(!isLocked){ safeTransferFrom(from, to, tokenId, "");} }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Will this work as I expect and is there any issues with this and would override the transfer method especially around security etc.  
  
Sorry for such a broad question  
  
Thanks

1 **Answer**

**A1:**From the context, it seems like you're using the OpenZeppelin implementation. If that's the case, you can override their \_beforeTokenTransfer() function, which is effectively called from all functions performing token transfer (as there is multiple of them).  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
pragma solidity ^0.8;import "@openzeppelin/contracts/token/ERC721/ERC721.sol";contract MyCollection is ERC721 { constructor() ERC721("MyCollection", "MyC") { \_mint(msg.sender, 1); } // needs to be unlocked for the `\_mint()` function in constructor bool locked = false; function setLocked(bool \_locked) external { locked = \_locked; } function \_beforeTokenTransfer( address from, address to, uint256 tokenId ) internal override { require(!locked, "Cannot transfer - currently locked"); }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
If you're using other or custom implementation, or chose to only override the publicly visible functions, make sure that there is no other public or external function allowing the transfer without the lock mechanism.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Also few notes to your current code:  
  
 ● You probably won't need the virtual keyword. It allows the function to be further overridden. Unless you're expecting to override this function as well, you can safely remove the keyword.  
  
 ● You're calling the parent function with the same name but different argument set. If you wanted to call the parent function with the same name and the same argument set (assuming it's defined), you'd need to use the super keyword (same as in Java): super.safeTransferFrom(from, to, tokenId);

**C1:**Do this only after sold out because you won't be able to mint anymore.