Homework 4

Programming an ERC721 Token Contract

108321033 吳明騰

108321056 唐泳烽

108321062 許恩慈

Introducing our ERC721 Contract

Introduction: When we generate ERC721 NFT, we use the URIStorage.sol and Ownable.sol frameworks from OpenZeppelin ERC721 (In figure 1). The reason why we use the former solidity file URIStorage.sol, is that in addition to the basic functions of ERC721.sol, it also has a hash table to store the NFT ID and its corresponding URLs in IPFS. The latter solidity file Ownable.sol is used because it will focus more on the owner of the smart contract, that is, we would store the address which is the first EOA that generates these NFTs, which can improve the security of these NFTs.

```
contract TimNFT is ERC721URIStorage, Ownable { … }
```

Figure 1: The the signature of our smart contract class

mint(): This function mint() in figure 2 is designed by us. In addition to using the function _mint() provided by ERC721.sol of OpenZeppelin ERC721, we also use the function _setTokenURI() provided by URIStorage.sol to record each NFT id Its corresponding URLs in IPFS.

```
function mint(
  address _to,
  uint256 _tokenId,
  string calldata _uri
 ) external onlyOwner {
  _mint(_to, _tokenId);
  _setTokenURI(_tokenId, _uri);
 }
```

Figure 2: The function mint() we design

Using Sepolia

In Figure 3, we mint our own NFT on the Sepolia testnet and we import it to Metamask. However, opensea does not support Sepolia testnet, so we don't have any screenshot from OpenSea.

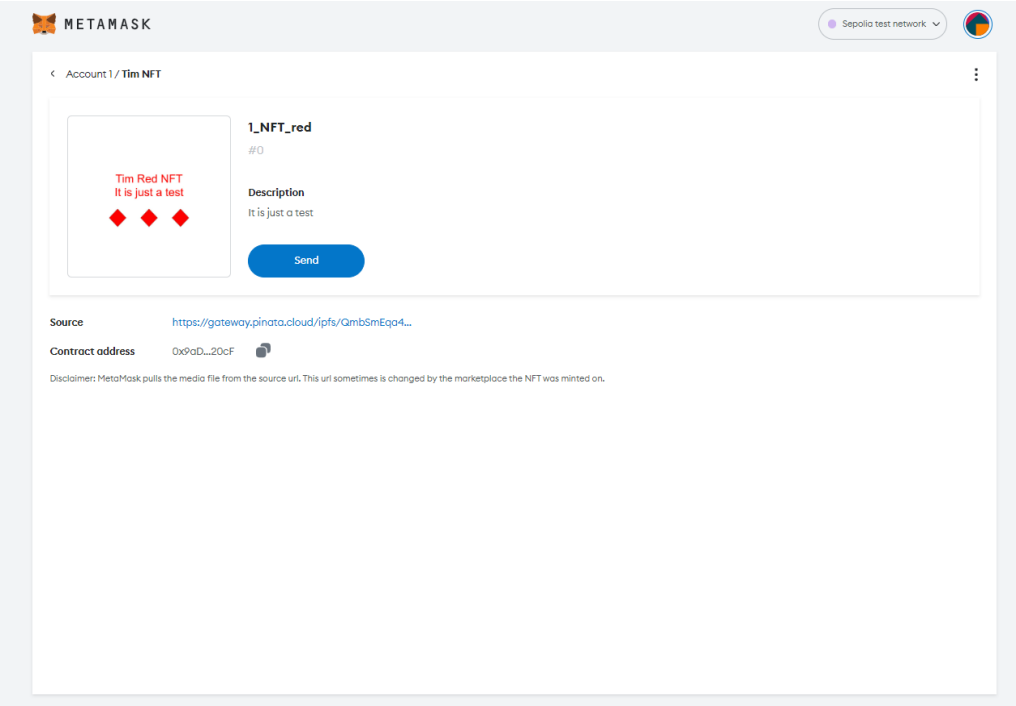

Figure 3: The screen of our metamask.

Function Description

Introduction

In this chapter, we will discuss the function _mint(), ownerOf(), approve(), SafeTransferFrom(), transferFrom(), setApprovalForAll(0 and isApprovedForAll(). The reason why we just list three functions is that we want to show the differences between ERC20 and ERC721.In addition to simply introducing the usage of these three functions, we also design a pseudocode to experiment with these functions.

```
_mint(address _from , uint256 _tokenId,string calldata _uri)
```

Mint a new NFT from the owner address (_from), give it an id (_tokenid) to distinguish different NFT tokens, and introduce a URI(_uri) for describing the details of the NFT token.

```
ownerOf(uint256 _tokenId)
```

Use id(_tokenid) to see who the NFT owner is.

```
approve(address _spender, uint256 _tokenId)
```

Give permission to an address (_spender), and transfer the NFT corresponding to the Id(_tokenid) to another account.

```
SafeTransferFrom(address _from, address _to, uint256 _tokenId)
transferFrom(address _from, address _to, uint256 _tokenId)
```

Use to transfer an NFT token Id(_tokenid) from an address (_from) to another address(_to).

safeTransferFrom() vs TransferFrom()

safeTransferFrom() has more security checks and protection measures to prevent the tokens in the contract from being lost or tampered with. During the transfer process, the contract will call the onERC721Received() function on the (_to) address to check the received tokens Process and return a value indicating whether the token was successfully received, if the reception fails, the transfer operation will be canceled. TransferFrom() will not perform any processing on the received tokens.

```
setApprovalForAll(address _spender, 1)
isApprovedForAll(address _from, address _to)
```

setApprovedForAll：Approve(1) or remove(0) operator as an address(_spender) for the caller.
isApprovedForAll：Returns if the operator(_to) is allowed to manage all of the assets of the owner(_from).

Implementation

The figure 4 is the pseudocode we designed, and then we explain the meaning of each line of code one by one. We also can read the description below with the flow chart in figure 5.

- Line 1~3: We mint three different NFTs to Alice.
- Line 4: We use the function ownerOf() to check who is the owner of the NFT with index 0 and its output is ‘Alice’ in line 5.
- Line 6: Alice approves Bob can transfer NFT with index 0 to another address.
- Line 7: Because of the mechanics of the function SafeTransferFrom(), Bob can not transfer the NFT not belonging to himself to another address, otherwise this transaction will show the error message in line 8.
- Line 9: different from the function SafeTransferFrom(), Bob can use the function transferFrom() to transfer NFT approved by Alice to another address, even if the NFT doesn’t belong to him.
- Line 10: We can find that NFT with index 0 belongs to Bob not Alice now, so it outputs Bob in line 11.
- Line 12: Alice uses the function setApprovalForAll() to approve Cindy to transfer all NFTs belonging to Alice to another address.
- Line 13: We can use the function isApprovalForAll() to check whether Cindy can transfer the all NFTs belonging to Alice to another address and this function would output ‘True’ in line 14.
- Line 15: Before Cindy uses the function transferForm(), the owner of the NFT with index 1 is Alice in line 16.
- Line 17: Cindy uses the function transferFrom() to transfer NFT with index 1 from Alice to Bob.
- Line 18: After using the function transferForm(), now the NFT with index 1 belongs to Cindy not Alice in line 19.

```
> mint( _AliceAddr, 0, NFT_0_uri )
> mint(_AliceAddr, 1, NFT_1_uri )
> mint( _AliceAddr, 2, NFT_2_uri )
>  Show ownerOf(0)
 ==> Alice
 > Alice.approve(_BobAddr, 0)
 > Bob.SafeTransferFrom(_AliceAddr, _BobAddr, 0)
 ==> error
 > Bob.transferFrom(_AliceAddr, _BobAddr, 0)
  > Show ownerOf(0)
 ==> Bob
 > Alice.setApprovalForAll(_CindyAddr, 1)
 > Show isApprovedForAll(_AliceAddr, _CindyAddr)
 ==> True
 > Show ownerOf(1)
 ==> Alice
 > Cindy.transferFrom(_AliceAddr, _CindyAddr, 1)
 > Show ownerOf(1)
 ==> Cindy
```

Figure 4: The pseudocode of our implementation.

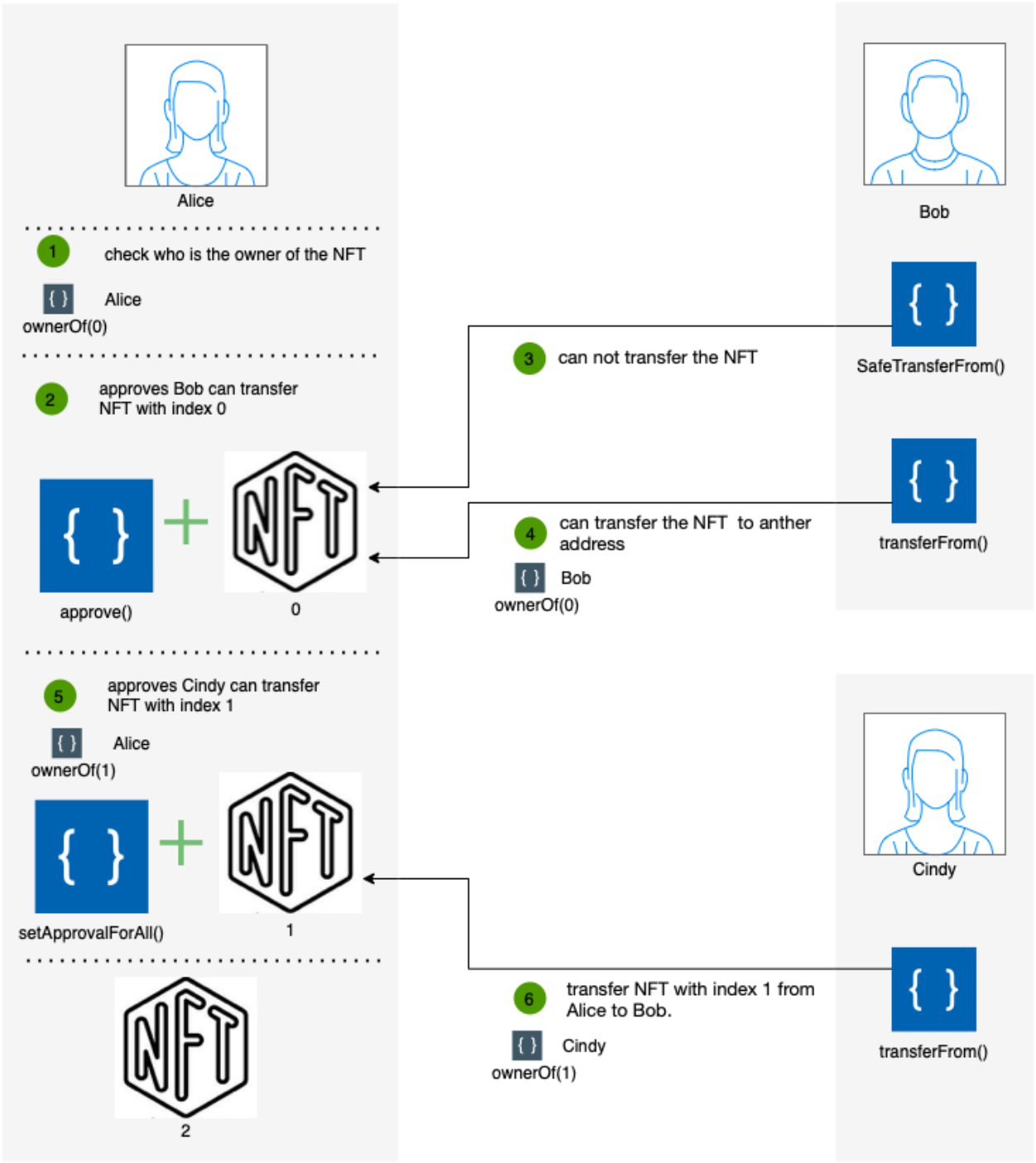

Figure 5: The flowchart of our implementation.

ERC721 vs ERC1155

Introduction

When we were collecting the differences between ERC721 and ERC1155, on the stack exchange website, someone was discussing "Is ERC721 more gas-efficient protocol than ERC1155?" and "Does ERC-1155 contract use less gas to mint tokens?" This issue has attracted our attention, so at the beginning of this chapter we will first discuss the differences between ERC721 and ERC1155, as well as the individual gas consumption when using these two different ways to generate tokens.

Difference

ERC721 launched in 2017, ERC-721 is a revolutionary token standard that kicked off the NFT phenomenon and became the backbone of NFT creation in the Ethereum ecosystem. The ERC-721 token standard has been the cornerstone of billion-dollar NFTs. The token standard provides a standard interface for NFT, and all tokens are unique. ERC-721 allows NFTs to be transferred between accounts and also makes them tradeable with other currencies.

Launched in 2018, ERC-1155 is a multi-token standard that provides a standard interface for smart contracts that manage multiple types of tokens. ERC-1155 improves upon the functionality of ERC-721, increasing efficiency and correcting glaring implementation errors. It can support an unlimited number of tokens in a single contract, supporting fungible, semi-fungible and NFT, which can be converted between the former during its lifetime.

In the two figures 6, 7 below, you can see the function _mint() and function _minBatch() shown. Unlike ERC721, ERC1155 needs to add a new parameter amount, so the same image can be quantified into an amount greater than one in this smart contract. This token we called as semi -fungible NFT

```
ERC1155._mint(address account, uint256 id, uint256 amount, bytes data)
```

Figure 6: Creates amount tokens of token type id, and assigns them to account.

```
ERC1155._mintBatch(address to, uint256[] ids, uint256[] amounts, bytes data)
```

Figure 7: Batch version of _mint. ids and amounts must have the same length.

Experiment

The experiment we designed was to generate 20 different NFTs and compare the gas usage of ERC721 and ERC1155.

ERC 721: The figure below uses Javascript to call the function mint() from ERC721, and uses the for loop to repeatedly execute function mint() 20 times. At this time, 20 different transactions will be generated, and the gas used of each transaction is 52,385, so the total gas used is 52,385 * 20 = 1,047,700

```
for(let i = 0; i < 20; i++){
    await ERC721.mint(accounts[0],i);
}
```

ERC 1155: The figure below uses Javascript to call the function mint() from ERC721. Unlike ERC721 which uses a for loop, ERC1155 only uses the function mintBatch() to generate 20 tokens at a time, so only one transaction is generated, and its gas used is 533,631.

```
const ids = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19];
const amounts = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1];
await ERC1155.mintBatch( accounts[0], ids, amounts, "");
```

Conclusion

From the above experiments, it can be found that if only from the perspective of token generation, if only one token needs to be generated, using ERC721 will save gas. If a large number of tokens are generated, ERC1155 will be the better choice. However, considering that ERC1155 is an extension of ERC721 version, when deploying smart contracts, ERC1155 must spend more Gas and time.In the future, when generating tokens, we still have to use our own task content to choose a suitable framework.