# Assignment 3: Build a Simple NFT Contract

## ERC721 Minting Dapp & Play-2-Earn Game

- Goerli Network
- Contract Address: 0xaA143964680239Ab244036c77713333Cbd223131

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

import '@openzeppelin/contracts/token/ERC721/ERC721.sol';
import '@openzeppelin/contracts/access/Ownable.sol';

contract NFTTalentsCollection is ERC721, Ownable {
    uint256 public mintPrice = 0.02 ether; // ************************* Cost to Mint
    uint256 public totalSupply; // ------------------------------------ Current Number of Minted NFTs
    uint256 public maxSupply; // ************************************** Current Supply Cap
    uint256 public votes; // ------------------------------------------ Number of Votes
    uint256 roundNumber = 0; // ************************************** Number of Iterations of Winner Selections
    address public winner; // ----------------------------------------- Winner of the Previous Round
    string _baseTokenURI; // ***************************************** Metadata Hash
    mapping(address => bool) public voterStatus; // ------------------- True = Voted; False = Has not Voted
    mapping(address => uint256) public mintedWallets; // ************** Number of Minted NFTs per Wallet


    constructor(string memory baseURI) payable ERC721('NFT Talents', 'TLNT') {
        //baseURI = ('ipfs://QmVX3T4Gj2hDaJ6y6oE6wpqxvZ9RjZLkhH2bmxF1qaM2hd');
        _baseTokenURI = baseURI;
        // initial NFT supply cap
        maxSupply = 10;
    }

    function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
        require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");
        return
            _baseTokenURI;
    }

    function selectRandomWinner() internal {
        // PseudoRandom Number Generator
        uint randomHash = uint(keccak256(abi.encodePacked(
            votes, // ********************************************** Number of Votes
            maxSupply, // ------------------------------------------ Current Supply Cap
            ownerOf(totalSupply), // ************************************ Address of Newest NFT Owner
            winner, // --------------------------------------------- Winner of the Previous Round
            roundNumber, // ****************************************** Number of Iterations of Winner Selections
            msg.sender))); // ------------------------------------------ Caller of the Function
        winner = ownerOf(1 + (randomHash % totalSupply));
    }

    function increaseMaxSupply() external {
        // If at least 50% of the maximum number of NFT holders vote to increase the
        // maximum supply, then any owner of an NFT Talents token can activate an
        // "_increaseMaxSupply()" which increases the maxSupply by 50%
        require(balanceOf(msg.sender) > 0);
        require(votes >= (maxSupply / 2), 'Not enough votes');
        // select pseudo random winner
        selectRandomWinner();
        maxSupply = maxSupply * 3 / 2;
        roundNumber++;
    }

    function addVote() external {
        // adds vote to increase maxSupply
        require(balanceOf(msg.sender) > 0);
        require(voterStatus[msg.sender] == false, 'Address has already voted');
        voterStatus[msg.sender] = true;
        votes++;
    }

    function removeVote() external {
        require(balanceOf(msg.sender) > 0);
        require(voterStatus[msg.sender] == true, 'You cannot remove your vote because you have not voted');
        voterStatus[msg.sender] = false;
        votes--;
    }

    function mint() external payable {
        require(mintedWallets[msg.sender] < 1, 'exceeds max per wallet');
        require(msg.value == mintPrice, 'wrong value');
        require(maxSupply > totalSupply, 'sold out');

        mintedWallets[msg.sender]++;
        totalSupply++;
        uint256 tokenId = totalSupply;
        _safeMint(msg.sender, tokenId);
    }

    // Remove Vote from Previous NFT Holder when NFT is Transferred to a New Account
    function _beforeTokenTransfer(address from, address to, uint256 tokenId) internal override(ERC721){
        super._beforeTokenTransfer(from,to,tokenId);
        require(balanceOf(to) == 0,"Recipient already owns an NFT Talents Token");
        if (voterStatus[from] == true) {
           voterStatus[from] = false;
           votes--;
        }
    }

    function withdraw() public payable {
        uint balance = address(this).balance;
        require(msg.sender == winner);
        require(balance > 0, "No ether left to withdraw");
        (bool success, ) = (msg.sender).call{value: balance}("");
        require(success, "Transfer failed.");
    }
}
```

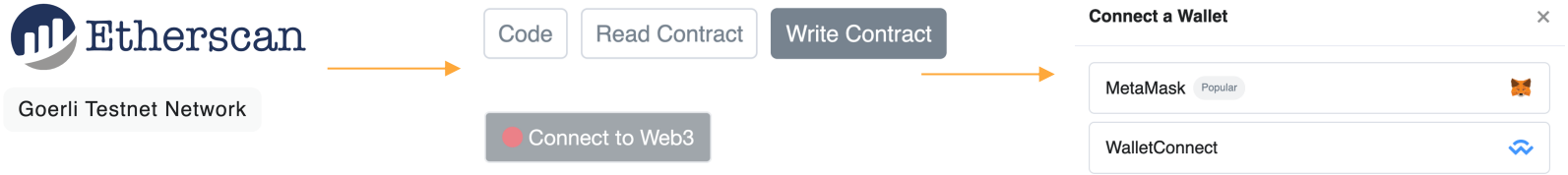
# Completely Decentralized

- This contract was designed to operate completely decentralized
- No special privileges for the contract deployer
- All of the contract logic is programmed into the smart contract, it cannot be edited by the contract deployer after the contract is deployed
  - Usually NFT smart contracts maintain a feature to change the metadata URI at anytime, however that makes it a centralized smart contract
  - On the other hand, if there is a problem with the metadata later on, it will be impossible to fix
- All features of the contract are accessible through Goerli Etherscan
  - Many Dapps consist of elements that assisted by Web2 frontends, which enhance the User Experience; however it does create room for error

"0xaA143964680239Ab244036c77713333Cbd223131" on Goerli Etherscan

Click "Connect to Web3"

Login to MetaMask

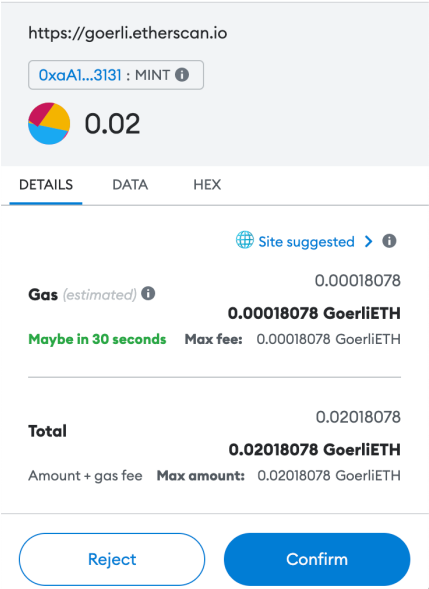


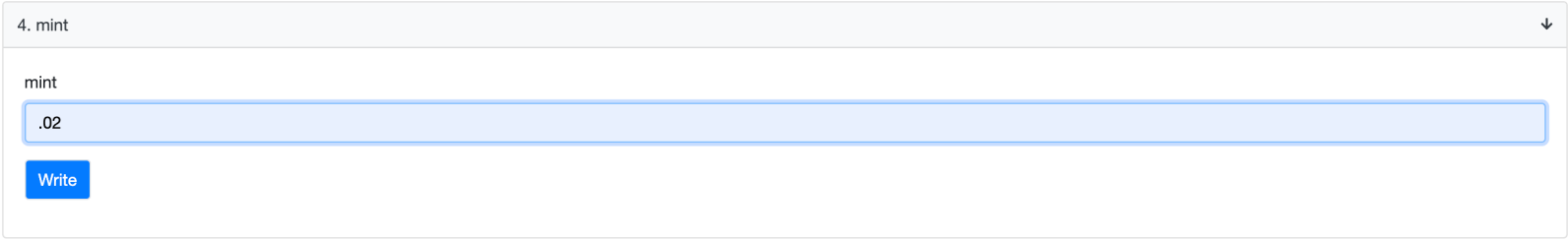
# Mint() Function

- Cost to Mint = **.02 Ether**
- Can be minted by anyone who has not previously minted
- Number of NFTs minted must be less than the maximum

```
function mint() external payable {
    require(mintedWallets[msg.sender] < 1, 'exceeds max per wallet');
    require(msg.value == mintPrice, 'wrong value');
    require(maxSupply > totalSupply, 'sold out');

    mintedWallets[msg.sender]++;
    totalSupply++;
    uint256 tokenId = totalSupply;
    _safeMint(msg.sender, tokenId);
}
```



# Voting Functions

- Only one NFT Talents Token per wallet
- The NFT collection has an initial Maximum of 10 NFTs that can be minted
- Each NFT holder has the ability to vote to increase that maximum or remove their vote at any time
- When at least 50% of the maximum number of NFT holders vote to increase the supply cap, the maximum number of mintable tokens can then be increased by 1.5x

```
    function addVote() external {
        // adds vote to increase maxSupply
        require(balanceOf(msg.sender) > 0);
        require(voterStatus[msg.sender] == false, 'Address has already voted');
        voterStatus[msg.sender] = true;
        votes++;
    }

    function removeVote() external {
        require(balanceOf(msg.sender) > 0);
        require(voterStatus[msg.sender] == true, 'You cannot remove your vote because you have not voted');
        voterStatus[msg.sender] = false;
        votes--;
    }
```

- addVote() & removeVote() can only be called by NFT holders

# Game Functions

- At the same time that the supply cap is increased a random winner is selected!
- The random winner is selected using a *Pseudo Random number generator* using the number Keccak hashing algorithm
- increaseMaxSupply() can only be called by an NFT holder, it **will not** be called automatically when the number of votes rises above 50% of maxSupply
- The winner then can use the withdraw() function to collect their winnings

```
function selectRandomWinner() internal {
    // PseudoRandom Number Generator
    uint randomHash = uint(keccak256(abi.encodePacked(
        votes, // ************************************************** Number of Votes
        maxSupply, // ------------------------------------------- Current Supply Cap
        ownerOf(totalSupply), // ********************************** Address of Newest NFT Owner
        winner, // ---------------------------------------------- Winner of the Previous Round
        roundNumber, // ********************************************* Number of Iterations of Winner Selections
        msg.sender))); // ---------------------------------------- Caller of the Function
    winner = ownerOf(1 + (randomHash % totalSupply));
}

function increaseMaxSupply() external {
    // If at least 50% of the maximum number of NFT holders vote to increase the
    // maximum supply, then any owner of an NFT Talents token can activate an
    // "_increaseMaxSupply()" which increases the maxSupply by 50%
    require(balanceOf(msg.sender) > 0);
    require(votes >= (maxSupply / 2), 'Not enough votes');
    // select pseudo random winner
    selectRandomWinner();
    maxSupply = maxSupply * 3 / 2;
    roundNumber++;
}
```

- It can be dangerous to use a pseudo random number generator because they can be manipulated by Nodes to produced desired numbers
- **However**, Since this contract is only deployed on a testnet there is no real incentive for it to be manipulated
- The variables used in the hashing algorithm are difficult to manipulate for a node, unlike other variables which are normally used

```
function withdraw() public payable {
    uint balance = address(this).balance;
    require(msg.sender == winner);
    require(balance > 0, "No ether left to withdraw");
    (bool success, ) = (msg.sender).call{value: balance}("");
    require(success, "Transfer failed.");
}
```

# Issues with NFT Media

- While building this app I realized some of the downsides to different metadata approaches

1. Gas is very expensive
   - Hard to make creative on-chain solutions when it directly increases the production cost of the smart contract

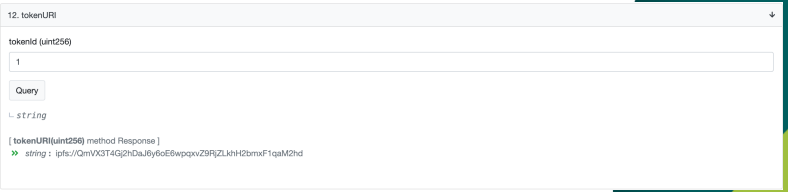
2. Using APIs, Web2 frontends, and Oracles leaves room for error
   - If your frontend is designed to submit data (e.g. Names & Numbers) to the smart contract, it can be intercepted and changed by the user if they decide to communicate directly to the smart contract

3. While there is incredible potential for IPFS it can be difficult to work with:
   1. The cost for a pinning service (e.g. Pinata) is high
   2. Data must be found by its Hash, so you have to add all the data needed in-advance
   3. Very slow compared to centralized storage alternatives
   4. If it gets unpinned it may never be recovered
   5. What incentive do NFT collections have to pay to host their metadata years later

To view on metadata on IPFS:

1. Click on tokenURI
2. Add the tokenId of interest (in this collection they are all the same)
3. Copy the string beginning with Qm ... all the way to the end
4. In a new browser write https://ipfs.io/ipfs/
5. Then paste the copied URI string

*See below*





```
{
  "description": "This completely decentralized NFT collection is made for students of the NFT Talents Program. This NFT is also your ticket to a play-to-earn game! The rules are every time that at least 50% of NFT holders vote to increase the maximum number of NFTs, a random winner is selected who receives the entire sum of the contract's balance. BUT only one NFT can be minted per wallet.",
  "image": "ipfs://QmWttkXqjBqymGrn5rohaYU2UvqYQrj5ibkkvQoUeYMUs3",
  "name": "NFT Talents"
}
```

# NFT Investor Due Diligence

- NOT ALL NFTS ARE CREATED EQUAL

- It is important for NFT investors, artists, developers, and contributors to do research on:
  - The NFT Community
  - The blockchain that the NFTs will be built on
  - Smart Contract security audit
  - Where will the metadata be held
    - Recurring payment for storage?
    - Decentralized vs Centralized
  - Smart Contract centralization vs decentralization
  - Methods for Minting
    - Lazy Mint
    - Whitelist
    - Minting Supply Cap
    - ERC720 vs ERC1155
  - Theft
    - Reentrancy Attack
    - Phishing (Never give out your Private Key!)
    - Read the Smart Contract (Know what your agreeing to)
    - Make sure it follows OpenZeppelin Standards


# ERC721

- **Non-Fungible Token** (AKA One-of-a-kind)
- Only one owner at a time
- Individual Mint
- NFT ≠ NFT

Can't be partitioned

Mint

Transfer

tokenURI

Burn

# ERC1155

- **Semi-Fungible Token** (AKA Multiple Copies)
- Multiple owners of same token
- Bulk Mint (Select Quantity)
- Interchangeable