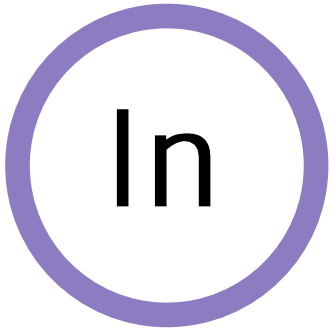
이더리움 DApp

Solidity 언어 실습



Solidity

04 Crypto Zombie(4)

Crypto Zombie Lesson4

Lesson1에선..

ZombieFactory 생성
- 모든 좀비를 기록하도록 state설정
- 좀비 생성 함수
- 각 좀비는 DNA를 통해 랜덤한 외모 설정

Lesson2에선..

- **게임기능**
  - 새로운 좀비 생성 기능
  - 좀비가 다른 좀비를 공격하고 먹을 수 있도록

Lesson3에선..

- 컨트랙트 생성 테크닉
- 의존성 관리하기
- 컨트랙트 보안
- 가스 최적화 및 modifier
- external view와 반복문

Lesson4에선..

좀비 전투시스템
- payable 제어자와 withdraw(출금)
- 난수 생성의 어려움과 외부함수접근(Oracle – 외부에서 데이터를 받아오는 안전한 방법)
- 로직구조개선
- 다양한 business logic 구현

https://cryptozombies.io/ko/lesson/4/chapter/1

04 Crypto Zombie(4)

Chapter1

목표: 제어자(접근제어자, 상태제어자, 함수제어자)의 이해와 payable제어자

- 접근제어자(private, public, internal, external)
- 상태제어자(view, public)
- 함수제어자(modifier – onlyOwner, aboveLevel 등)

→ 제어자들은 한 번에 서로 같이 사용될 수 있음

```
function test() external view onlyOwner anotherModifier { /* ... */ }
```

payable

특별한 제어자!:

- 이더를 지급받을 수 있는 제어자
- 이더를 지급 받으려면 payable 제어자가 필요.

```
contract OnlineStore {
    function buySomething() external payable {
        // 함수 실행에 0.001이더가 보내졌는지 확실히 하기 위해 확인:
        require(msg.value == 0.001 ether);
        // 보내졌다면, 함수를 호출한 자에게 디지털 아이템을 전달하기 위한 내용
        transferThing(msg.sender);
    }
}
```

- 위 예제는 온라인스토어 예제로서, 물건을 살 때 이더를 보내고 해당 물건을 구입자에게 전달하기 위한 컨트랙트
- 특별한 단위 (ether, wei 등 존재)
- 금액은 msg.value로 접근 가능함
- 만약 payable이 없는데 이더를 보내면 error 발생!

실습

ether를 전달받아 levelUp을 시킬 수 있도록 구성하자! (ZombieHelper.sol)

1. uint 타입의 levelUpFee 변수를 정의: 0.001 ether
2. levelUp (uint _zombieId)이라는 함수를 생성. 외부에서 호출 받을 예정! (private vs internal vs external vs public)
3. 이더를 받을 예정 → 제어자 OO? 적용
4. require로 요청 금액이 맞는지 확인
5. zombie의 levelUp

04 Crypto Zombie(4)

Chapter2

목표: 출금과 transfer 함수의 이해 (this.balance,

```
contract OnlineStore {
    function buySomething() external payable {
        // 함수 실행에 0.001이더가 보내졌는지 확실히 하기 위해 확인:
        require(msg.value == 0.001 ether);
        // 보내졌다면, 함수를 호출한 자에게 디지털 아이템을 전달하기 위한 내용
        transferThing(msg.sender);
    }
}
```

- 위 예처럼 이더를 보내면, 해당 CA계좌에 이더리움이 갇히게 된다.
  → 출금할 수 있는 방법이 없기때문.

➔ 우리는 출금할 수 있는 함수를 구현해주어야 한다.

```
uint itemFee = 0.001 ether;
msg.sender.transfer(msg.value - itemFee);
```

주의: 0.4version

예시: 아이템을 초과하여 돈을 지불했을때, 거스름돈을 돌려주는 코드

실습

withdraw 함수 만들기(ZombieHelper.sol)

1. withdraw함수를 정의하여라 → 잔고 전액 인출하는 함수
2. 이더리움 가격이 변화하기 때문에, levelUpFee를 설정시키는 setLevelUpFee(uint _fee) 함수 생성 → 오직 owner만 호출 가능하도록

```
// 0.5버젼
function withdraw() external onlyOwner {
    address payable owner = address(uint160(owner()));
    owner.transfer(address(this).balance);
}
                    0.4버전에서는 this.balance
// 0.5버젼
function withdraw2() external onlyOwner {
    address payable owner = address(uint160(owner()));
    owner.transfer(1 ether);
}

// 0.6버전
function withdraw() external onlyOwner {
    address payable owner = payable(owner());
    owner.transfer(address(this).balance);
}
```

with Ownable

transfer 함수와 send함수

- address로 ether를 전송하겠다.
- 위 예제에서 onlyOwner modifier 확인
- transfer함수의 인자는 금액!
- **특별한 변수는 address(this).balance(현재 컨트랙트의 잔액)**

- transfer함수는 전송실패시 에러!
  send함수는 전송실패시 false 반환 (에러 처리 필요할때만 사용)

04 Crypto Zombie(4)

Chapter3 & Chapter4 & Chapter5

목표: 공격기능 구현하기

요구사항

- 좀비 중 하나를 고르고, 상대방의 좀비를 공격 대상으로 선택.
- 공격하는 쪽의 좀비: 70%의 승리 확률.
  방어하는 쪽의 좀비: 30%의 승리 확률
- 모든 좀비들(공격, 방어 모두)은 전투 결과에 따라 증가하는 winCount와 lossCount를 가짐.
- 공격하는 좀비 이기면 → winCount증가 + 레벨업 + 새로운 좀비 생성
- 좀비 지면 → lossCount 증가
- 공격하는 좀비는 쿨타임 trigger

실습

ZombieBattle.sol 구현
[요구사항]
1. ZombieHelper 상속
2. 난수 생성함수 만들기(Fake Random) // 승리했는지 안했는지 체크 위해 생성
   a. 컨트랙트에 uint randNonce= 0;
   b. randMod(uint _modulus)함수 생성 internal returns(uint256)
   c. randNonce 증가시키고
   d. return uint(keccak256(abi.encodePacked(now, msg.sender, randNonce))) % _modulus
   0.5버전에서는 해시하기전에 abi.encodePacked사용

실습

ZombieBattle.sol 구현
[요구사항]

1. 승리확률 70으로 생성하기
   (상태변수: uint attackVictoryProbability) 값은 70
2. 함수 attack(uint _zombieId, uint _targetId) external 생성

04 Crypto Zombie(4)

Chapter6 & Chapter7

목표: 공통 로직 구조 개선 → 함수화 (modifier)

- 기능을 구현하다 보면 해당 좀비가 자신의 좀비가 맞는지 검증(아래 빨간박스) 하는 부분이 반복해서 나온다.
- 공격 기능을 추가할 때에도 다음과 같은 검증절차를 항상 거쳐야 할 것!

→ 가독성을 높이기 위해 modifier 등록하자.

```
function changeName(uint256 _zombieId, string calldata _newName)
    external
    aboveLevel(2, _zombieId)
{
    require(msg.sender == zombieToOwner[_zombieId]);
    zombies[_zombieId].name = _newName;
}

function changeDna(uint256 _zombieId, uint256 _newDna)
    external
    aboveLevel(20, _zombieId)
{
    require(msg.sender == zombieToOwner[_zombieId]);
    zombies[_zombieId].dna = _newDna;
}
```

실습

자신의 좀비인지 검증하는 modifier 구현
[요구사항]

1. modifier zombieOwnerOf 구현(해당 좀비가 자신의 좀비인지 검증하는 제어자)
   - 들어갈 인자 생각해보자!
2. feedAndMultiply에 zombieOwnerOf 적용
3. changeName, changeDna에 zombieOwnerOf 적용
4. attack 함수에 zombieOwnerOf 적용

04 Crypto Zombie(4)

Chapter8 & Chapter9

목표: attack함수 구현

- 좀비 중 하나를 고르고, 상대방의 좀비를 공격 대상으로 선택.
- 공격하는 쪽의 좀비: 70%의 승리 확률.
  방어하는 쪽의 좀비: 30%의 승리 확률
- 모든 좀비들(공격, 방어 모두)은 전투 결과에 따라 증가하는 winCount와 lossCount를 가짐.
- 공격하는 좀비 이기면 → winCount증가 + 레벨업 + 새로운 좀비 생성
- 좀비 지면 → lossCount 증가
- 공격하는 좀비는 쿨타임 trigger

실습

ZombieFactory

[요구사항]
1. ZombieFactory에 존재하는 Zombie 구조체 변경
   winCount, lossCount 선언(uint16)
2. → _createZombie 수정

실습

attack 함수 구현
-- 만들어 놓은 난수함수로 승, 패 결정

[요구사항]
1. _zombieId를 이용해 zombie 객체를 배열에서 꺼내어
   Zombie storage myZombie로 저장 (값을 변경하겠다.)
2. _targetId를 이용해 zombie 객체를 배열에서 꺼내어
   Zombie storage enemyZombie로 저장 (값을 변경하겠다.)

3. uint rand를 선언하고 randMod(100)의 return값을 할당
   // 왜 100? 0부터 69까진 공격좀비 승 (승률 70프로) // 70부턴 방어좀비 승
4. rand의 값과 승률(상태변수)과 비교
   a. 우리 좀비가 이기면
      - 우리 좀비의 winCount증가, 레벨업, enemyZombie lossCount증가,
        feedAndMultiply함수 호출 (_species = “zombie”)
   b. 만약 우리 좀비가 지면
      - 우리 좀비의 lossCount증가, enemyZombie winCount 증가
5. 승패와 상관없이 공격좀비의 재사용 대기시간 증가
   → 함수가 존재.