Lottery Docs

Upside Academy 1기 - nullorm (정효영)

▼ 목차

Variables / Arrays / Mappings

```
mapping(uint16 => mapping(address => uint8)) public buyLott
eryNum;
// 매핑[lotteryNum][address]가 구매를 했으면 1, 당첨됐으면 2를 저
장
mapping(uint256 => address) public buyRoster; // buy를 한 ad
dress를 모두 저장

uint256 public totalBuyLotteryNum = 0; // lottery를 buy한 사
람의 수
uint256 public numOfWinner = 0; // winner의 수
uint256 public Deadline; // buy를 할 수 있는 deadline
uint16 public winningNumber; // winning number

bool public isClaim; // claim phase에서 draw를 하지 않도록 하기
위한 변수
bool public isInit; // initialize를 위한 변수
```

Constructor

```
constructor() {
    Deadline = block.timestamp + 24 hours;
    isInit = true;
}
```

deadline을 deploy한 시점에서 24 hour를 더해주고, isInit을 true로 설정.

Functions

Helper Function - initialize()

rollover를 위한 소스코드.

```
function initialize() internal {
    Deadline = block.timestamp + 24 hours;
    isInit = true;
    isClaim = false;
    totalBuyLotteryNum = 0;
}
```

다음번 lottery를 시작하고자 할 때, initialize를 실행하면

- 다시 deadline이 연장되고,
- initialize되며,
- claim phase에서 벗어나고,
- lottery 구매량이 초기화된다.

buy()

```
function buy(uint16 _lotteryNum) public payable {
    if (isInit == false)
        initialize();
    require(block.timestamp < Deadline, "!!!Times out!!!");
    require(msg.value == 0.1 ether, "!!!Insufficient Fund
s!!!");
    require(buyLotteryNum[_lotteryNum][msg.sender] == 0,
"!!Already Bought!!!");
```

```
    buyRoster[totalBuyLotteryNum] = msg.sender;
    buyLotteryNum[_lotteryNum][msg.sender] = 1; // buy 표시
    totalBuyLotteryNum++;
}
```

lottery를 구매하는 함수. `_lotteryNum`을 입력으로 받아 해당 숫자에 해당하는 lottery를 구매한다.

- 만약, initialize가 false라면, `initialize()` 를 한 번 수행하고 시작한다.
- Deadline을 지났으면 buy를 할 수 없다
- lottery 가격은 0.1 ether로 고정
- buyLotteryNum 매핑을 검사해서 이미 lottery를 구매한 적이 있는지 검사한다.
- 다음, buyRoster함수에 msg.sender를 저장하고 (draw시에 필요한 매핑임)
- 구매했다는 뜻인 1을 buyLotteryNum에 저장한다.
- 구매량이 늘어났으므로, totalBuyLotteryNum을 증가시켜준다.

draw()

```
function draw() public {
    require(isClaim == false, "!!!Not Draw Phase!!!");
    require(block.timestamp >= Deadline, "!!!Not Draw Ye
t!!!");
    uint16 winNum = winningNumber = uint16(uint256(keccak25
6(abi.encode(block.timestamp, block.number))));
    for(uint256 i = 0; i < totalBuyLotteryNum; i++)
    {
        address buyer = buyRoster[i];
        if (buyLotteryNum[winNum][buyer] == 1)
        {
            buyLotteryNum[winNum][buyer] = 2;
            numOfWinner++;
        }
        else
        {
            buyLotteryNum[winNum][buyer] = 0;
```

```
        }
    }
    isInit = false;
}
```

- claim phase에는 draw를 할 수 없기 때문에, isClaim변수가 false상태여야 한다.
- block.timestamp가 Deadline을 넘어 draw phase가 되어야 한다.
- block.timestamp와 block.number를 keccak한 값을 통해 유사난수를 생성해 winnginNumber에 저장해주었다.
- lottery 총 구매량만큼 for문을 돌면서, buyRoster에 들어가있는 lottery구매자들의 number들을 검사한다.
- 검사 결과, 숫자를 맞췄다면 buyLotteryNum의 값을 2로 바꿔주고, Winner의 수를 올려준다.
- 틀렸다면, buyLotteryNum의 값을 0으로 바꿔준다.
- isInit을 False로 만들어준다.

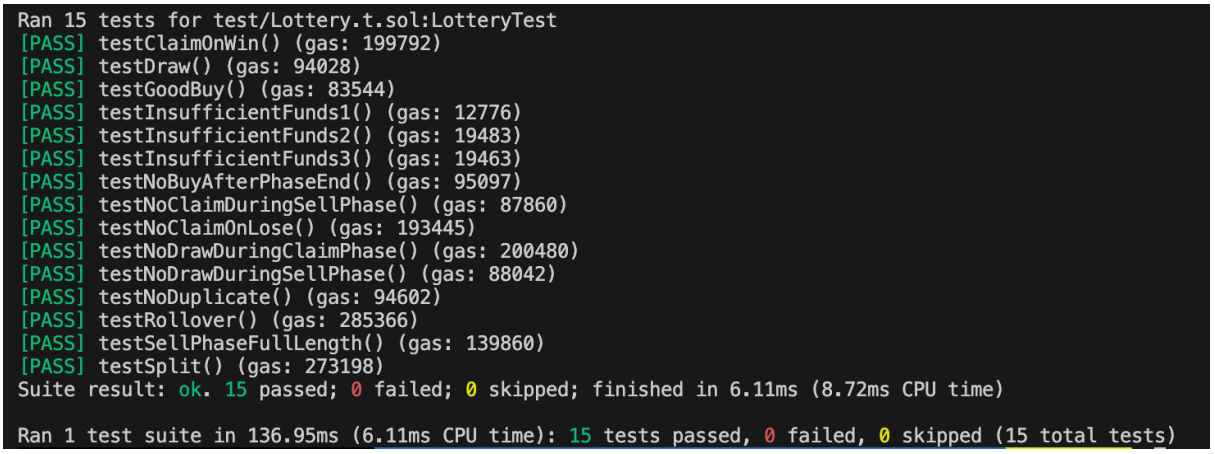
Claim()

```
function claim() public {
    require(block.timestamp >= Deadline, "!!!Claim Not Ye
t!!!");
    isClaim = true;
    uint256 prize;

    if (numOfWinner > 0)
        prize = address(this).balance / numOfWinner;

    if (buyLotteryNum[winningNumber][msg.sender] == 2)
    {
        buyLotteryNum[winningNumber][msg.sender] = 0;
        (bool suc, ) = payable(msg.sender).call{value: priz
e}("");
        require(suc, "!!!claim failed!!!");
    }

    if (numOfWinner > 0)
```

```
            numOfWinner--;
    }
```

- require문을 통해 block.timestamp가 Deadline을 넘도록 하여, buy phase에서 실행할 수 없도록 한다.
- isClaim을 true로 만들어, claim phase라는 것을 표시
- winner가 0일 때, balance를 winner로 나누게 되면 zero divide 에러가 뜨기 때문에, 0 이상인지 검사한 후, balance를 winner로 나누어 prize에 저장한다.
- msg.sender가 당첨자라면(buyLotteryNum == 2) 0으로 만들어주고, prize만큼 winner에게 전송해준다.

Test



test를 통과하였다.