QuizGame Docs

Upside Academy 1기 - nullorm (정효영)

▼ 목차

Struct

```
struct Quiz_item {
      uint id;
      string question;
      string answer;
      uint min_bet;
      uint max_bet;
}
```

quiz들에 대한 데이터를 담고있는 구조체이다. 퀴즈의 ID, 문제, 정답, betting 최대/최소금액이 명시되어있다.

Variable / Array / Mappings

```
mapping(address => uint256)[] public bets; // 어떤 문제에 어떤 address가 얼마를 bet?
uint public vault_balance; // quiz contract vault의 balance
// 새롭게 정의한 변수 / 매핑들
mapping(uint256 => Quiz_item) private Quizes; // Quiz_item
```

```
구조체를 담은 매핑
mapping(address => bool)[] private isSolved; // quiz[문제]를 주소가 풀었는지
uint256 private quizNum = 1; // quiz의 총 수, 배포시에 하나가 만들어지므로 초기값은 1
```

Functions

addQuiz

```
function addQuiz(Quiz_item memory q) public {
    require(msg.sender != address(1), "invalid address");
    quizNum++;
    bets.push();
    Quizes[q.id] = q;
}
```

- test에서, 주소(1)이 addQuiz를 할 때, revert시키는 조건이 있어 require에서 처리
- addquiz를 할 때 quiz가 하나 늘어나기 때문에, `quizNum++`
- add를 하기 위해 bets의 memory expansion을 해주고
- expansion된 memory에 Quiz_item q를 넣음

getAnswer

```
function getAnswer(uint quizId) public view returns (string memory) {
    Quiz_item memory quiz = Quizes[quizId];
    return quiz.answer;
}
```

quiz의 정답을 return해줌. test contract에서 채점을 하기 위해 필요한 함수이다.

getQuiz

```
function getQuiz(uint quizId) public view returns (Quiz_ite
m memory) {
    Quiz_item memory quiz = Quizes[quizId];
    quiz.answer = "";
    return quiz;
}
```

quiz의 정답을 마스킹한 채로 quiz를 return해주는 함수. 정답이 포함된 문제를 받게 되면 make sense하지 않기 때문에, 이렇게 하는 것으로 보임

getQuizNum

```
function getQuizNum() public view returns (uint){
    return quizNum;
}
```

지금까지 만들어진 quiz의 총 수를 return하도록 하였음. 하지만, 이게 testcode를 보면 이전 quiz의 id를 리턴하는것인지, 갯수를 리턴하는 것인지는 애매하긴 함.

solveQuiz

quizId와 이에 대한 정답을 함께 제출하여 정답이 맞는지를 검사하는 함수

```
function solveQuiz(uint quizId, string memory ans) public r
eturns (bool) {
    Quiz_item memory quiz = Quizes[quizId];

    if (keccak256(abi.encode(quiz.answer)) == keccak256(ab
i.encode(ans))) {
        isSolved[quizId - 1][msg.sender] = true;
        return true;
    }
    else {
        vault_balance += bets[quizId - 1][msg.sender];
        bets[quizId - 1][msg.sender] = 0;
        return false;
```

```
    }
}
```

- 정답이 맞는지 검사
- 맞다면
  - msg.sender가 quizId - 1 (testcode가 이렇게 되어있음) 를 풀었다는 것을 mapping에 저장 (claim을 위해)
  - true를 return
- 틀렸다면
  - 틀리면, solver가 아닌 quiz contract가 돈을 가져가게 되어있으므로, vault_balance에 bet한 만큼의 ETH를 더해줌
  - solver가 돈을 뺏기므로, bets 매핑의 값을 0으로 만들어줌
  - false를 return

claim

```
function claim() public {
    for(uint256 i = 1; i < quizNum; i++) {
        if (isSolved[i - 1][msg.sender] == true) {
            isSolved[i - 1][msg.sender] = false; // avoid R
e-entrancy
            (bool suc, ) = payable(msg.sender).call{value:
bets[i - 1][msg.sender] * 2}("");
            require(suc, "claim() failed");
            bets[i - 1][msg.sender] = 0;
        }
    }
}
```

매개변수로 아무것도 받지 않기 때문에, 어떤 퀴즈의, 그리고 누가 돈을 받아가야하는지도 알 수 없어 오직 msg.sender 정보만 이용하여 ETH를 claim시켜주어야 한다.

- solveQuiz함수를 통해 solve매핑을 true로 만들어주었는지 (정답을 맞췄는지) 검사
- solve매핑 값을 false로 바꿔줌 (re-entrancy attack 방지)
- solver가 betting한 금액의 두 배 만큼 ETH를 보내줌

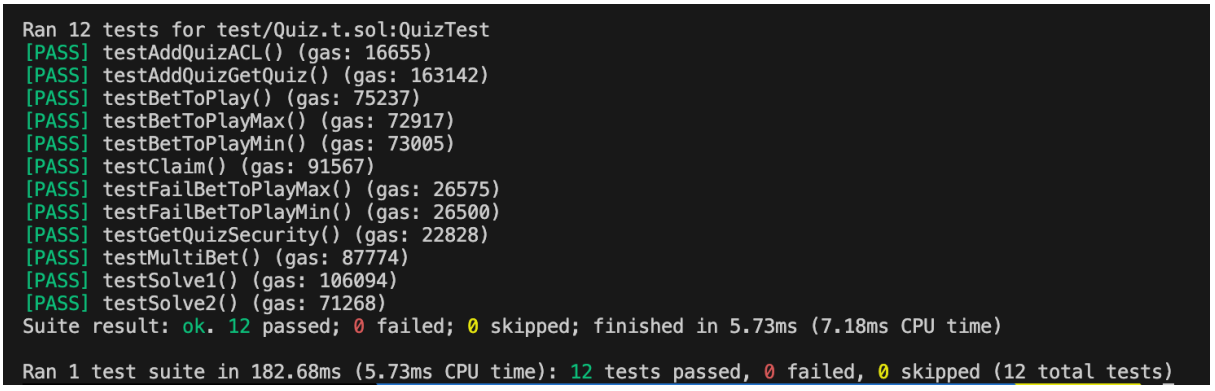
- betting한 금액을 0으로 만들어줌 (보내줬으니까)

Receive

```
receive() external payable {
    vault_balance = payable(address(this)).balance;
}
```

test.sol의 setUp 과정에서 Quiz컨트랙트에 5 ether를 보내는 과정이 있기 때문에, receive함수를 추가

Test



Test를 모두 통과하였습니다.