

# Dice or Die

블록체인 3조

윤서호

이병헌

정재영

조현흠



## **Table of Contents**

- 1. What is Dice or Die**
- 2. Why on-chain – pros and cons**
- 3. Components**
- 4. Code Design**
- 5. How to play**

# 1. What is Dice or Die?

---

## Description

이더리움 기반으로 스마트 컨트랙트

주사위를 굴려 나오는 숫자에 따라 베팅 금액 이상의 수익을 거두거나, 손실을 입는다

## Betting Rule

각 주사위 수에 따른 수익률

$1 = -100\%$  /  $2 = -50\%$  /  $3 \ \& \ 4 = 0\%$  /  $5 = 50\%$  /  $6 = 100\%$

유저의 기대 수익률:  $0\%$ (수수료 포함 시  $-5\%$ )

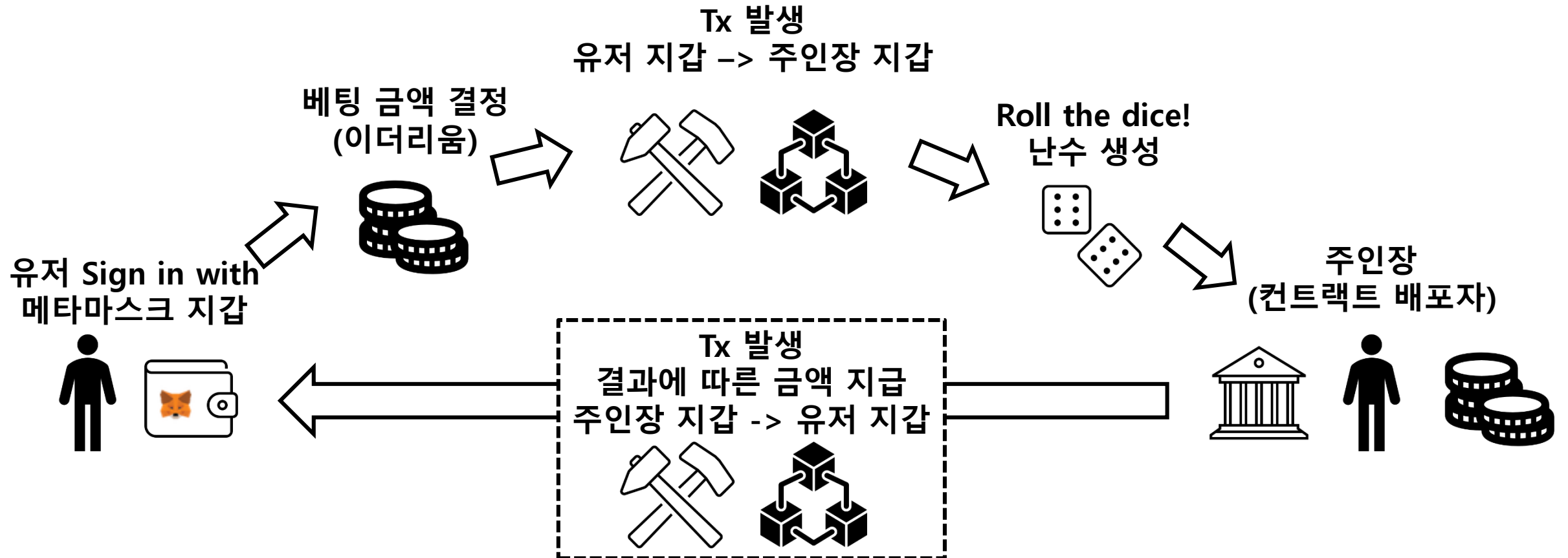
따라서 contract account의 이더리움 pool이 고갈되지않고 영속적으로 운영 가능하다.

## Metamask와 Balance 연동

Metamask로 Sign in, 연동하여 게임이 끝날 때마다 컨트랙트에 의해 자동적으로 정산한다.

# 1. What is Dice or Die?

## Framework



## 2. Why on-chain – pros

---

### Advantage

1. 베팅된 금액이 중개인이 아닌 컨트랙트 계좌에 보관되므로 정산이 자동화  
-> 중개인 신뢰 문제 해결
2. modifier: 코드를 공개하더라도 다른 이는 함수를 실행할 수 없음  
-> 보안의 문제 해결
3. 수수료 비즈니스 모델 자동화  
-> 트랜잭션 생성마다 수수료 수입
4. 베팅 게임에 대한 엔트리를 낮춤(법정화폐가 아닌 가상화폐 이용)  
-> 차후 비즈니스로 활용 시(기대 수익률을 0% 아래로 낮출 시 수익 발생) 더 큰 수익 가능

## 2. Why on-chain – cons

---

### Disadvantage

1. 트랜잭션 배포와 기록에 시간이 걸려 효율성이 떨어진다  
-> 주사위 베팅 후 금액 정산까지 딜레이가 길
2. 트랜잭션 시마다 발생하는 수수료

### How to deal with

- 1-1. (수익성 측면) 대기 시간에 광고 삽입을 통한 추가적 수익 창출 가능
- 1-2. (사용자 경험 측면) 대기 시간에 긴장감을 극적으로 끌고 갈 수 있는 시각 효과 삽입
2. 해당 수수료에 추가 마진을 붙여 수익 모델 형성

# 3. Components

---

## Front end

'Figma' 로 디자인 → 'React' 로 웹 프론트엔드 구성

## Back end

'Solidity' 로 diceBet 스마트 컨트랙트 작성 후 배포

'Truffle' 로 contract를 compile, test, deploy

'Web3.js' 로 contract를 call

'Ganache' 사용해서 local에서 배포 및 테스트

# 4. Code Design

---

- `remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.6.12+commit.27d51765.js`



# 4. Code Design

---

```
pragma solidity >=0.4.21 <0.7.0;

contract RollDice {
    uint256 constant MAX_CASE = 6; //for dice
    uint256 constant MIN_BET = 0.01 ether;
    uint256 constant MAX_BET = 10 ether;
    uint256 constant HOUSE_FEE_PERCENT = 5;
    uint256 constant HOUSE_MIN_FEE = 0.005 ether;

    address payable public owner; //=> account address data type
    uint256 public lockedInBets;

    struct Bet {
        uint256 amount;
        address payable gambler;
        uint256 winningAmount;
        uint256 diceNum;
    }
}
```

## 4. Code Design

---

```
mapping(address => Bet) bets;

event Reveal(address indexed gambler, uint256 diceNum, uint256 amount);
event Payment(address indexed beneficiary, uint256 amount);
event FailedPayment(address indexed beneficiary, uint256 amount);

constructor() public {
    owner = msg.sender;
}

modifier onlyOwner {
    require(msg.sender == owner, "Only owner can call this function.");
    _;
}
```

## 4. Code Design

---

```
function withdrawFunds(address payable beneficiary, uint256 withdrawAmount)
    external
    onlyOwner
{
    require(
        withdrawAmount + lockedInBets <= address(this).balance,
        "larger than balance"
    );
    sendFunds(beneficiary, withdrawAmount);
}

function sendFunds(address payable beneficiary, uint256 amount) private {
    if (beneficiary.send(amount)) {
        emit Payment(beneficiary, amount);
    } else {
        emit FailedPayment(beneficiary, amount);
    }
}
```

## 4. Code Design

---

```
function kill() external onlyOwner {  
    require(  
        lockedInBets == 0,  
        "All bets should be processed before self-destruct."  
    );  
    selfdestruct(owner);  
}
```

## 4. Code Design

---

```
function getWinningAmount(uint256 amount)
    private
    returns (uint256, uint256)
{
    Bet storage bet = bets[msg.sender];
    bet.gambler = msg.sender;

    uint256 housefee = (amount * HOUSE_FEE_PERCENT) / 100;
    uint256 reward;

    if (housefee < HOUSE_MIN_FEE) {
        housefee = HOUSE_MIN_FEE;
    }

    uint256 random = uint256(keccak256(
        abi.encodePacked(
            block.timestamp
        )
    ));
```

# 4. Code Design

---

```
uint256 tt = random % 6;

if (tt == 0) {
    reward = 0;
}

if (tt == 1) {
    reward = amount / 2;
}

if (tt == 2 || tt == 3) {
    reward = amount;
}

if (tt == 4) {
    reward = (amount * 3) / 2;
}

if (tt == 5) {
    reward = amount * 2;
}

uint256 winningAmount = (reward > housefee) ? reward - housefee : reward;
uint256 diceNum = tt + 1;
bet.diceNum = diceNum;
bet.winningAmount = winningAmount;
return (winningAmount, diceNum);
}
```

# 4. Code Design

---

```
function revealResult() external payable {
    uint256 amount = msg.value;
    require(
        amount >= MIN_BET && amount <= MAX_BET,
        "Amount is out of range."
    );
    Bet storage bet = bets[msg.sender];
    // bet.gambler = msg.sender;
    address payable gambler = bet.gambler;

    require(amount > 0, "Bet should be in an 'active' state.");

    (uint256 winningAmount, uint256 diceNum) = getWinningAmount(amount);

    emit Reveal(bet.gambler, bet.diceNum, bet.winningAmount);
    if (winningAmount > 0) {
        sendFunds(gambler, winningAmount);
    }
    lockedInBets -= winningAmount;
    clearBet(msg.sender);
}
```

# 4. Code Design

---

```
function clearBet(address player) private {
    Bet storage bet = bets[player];

    bet.amount = 0;
    bet.winningAmount = 0;
    bet.gambler = address(0);
}

function refundBet() external {
    Bet storage bet = bets[msg.sender];
    uint256 amount = bet.amount;
    address payable gambler = bet.gambler;
    require(amount > 0, "Bet should be in an 'active' state.");
    uint256 possibleWinningAmount;
    possibleWinningAmount = bet.winningAmount;

    lockedInBets -= possibleWinningAmount;
    clearBet(msg.sender);
    sendFunds(gambler, amount);
}

function checkHouseFund() public view onlyOwner returns (uint256) {
    return address(this).balance;
}
```



# 5. How to play

---

- 링크