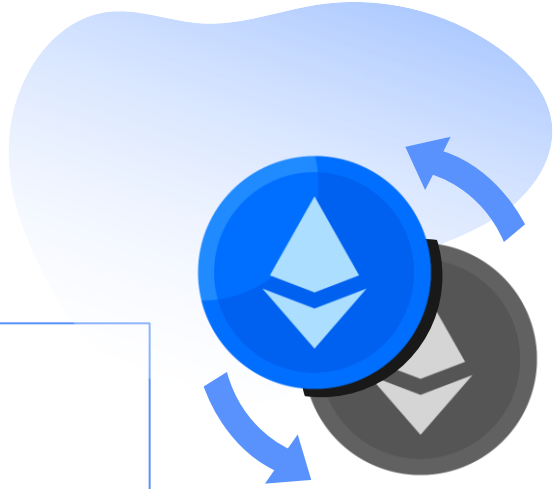
Oracle Casino

[SKKRYPTO] 11기 김여리, 서준, 홍석무 | 10기 강나현

목차

01

프로그램 소개



O’sino

Oracle 기술을 활용한 배팅 게임

02

개발 일정

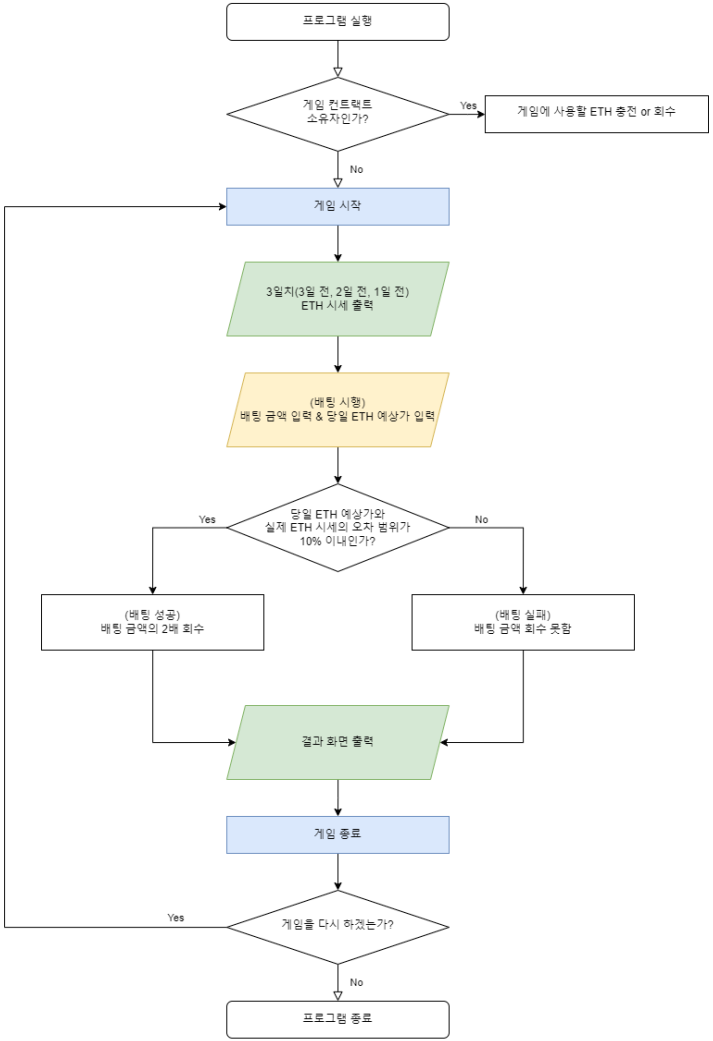
개발 일정

	1차 회의	2차 회의	중간 점검	세션
김여리	컨트랙트 작성 (API / 게임)	컨트랙트 확인	발표 자료 제작	
서준	컨트랙트 작성 (게임)	코드 합쳐서 컨트랙트 완성	콘솔 출력 구현	
홍석무	컨트랙트 작성 (API)	컨트랙트 확인	프론트엔드 구현	

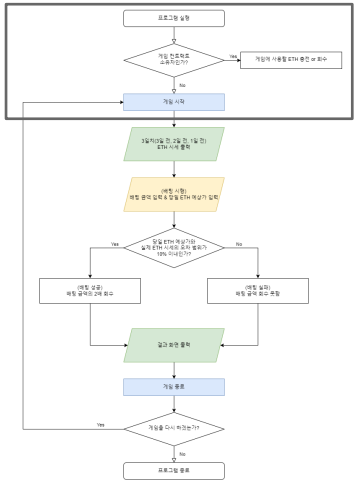
03

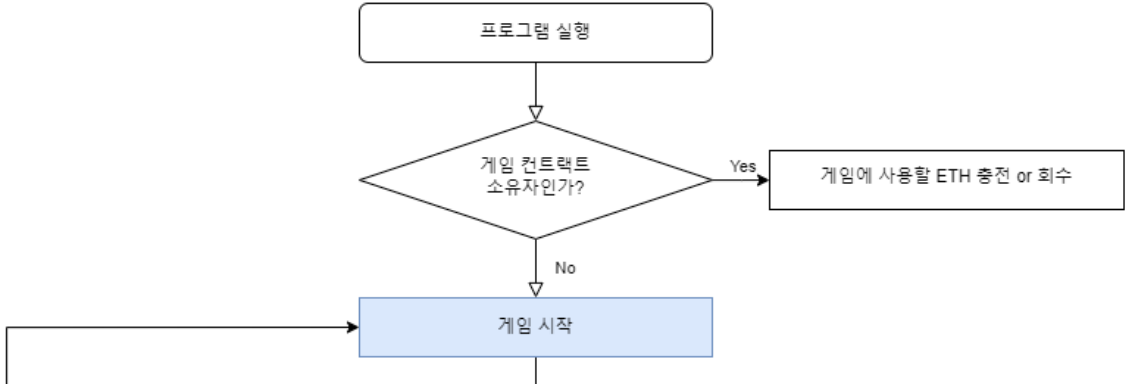
알고리즘

알고리즘

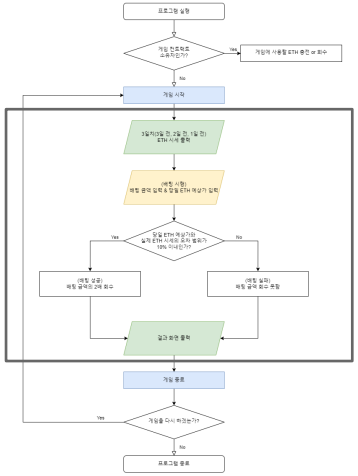

프로그램 실행
게임 컨트랙트 소유자인가?
Yes
게임에 사용할 ETH 충전 or 회수
No
게임 시작
3일치(3일 전, 2일 전, 1일 전) ETH 시세 출력
(배팅 시행) 배팅 금액 입력 & 당일 ETH 예상가 입력
당일 ETH 예상가와 실제 ETH 시세의 오차 범위가 10% 이내인가?
Yes
No
(배팅 성공) 배팅 금액의 2배 회수
(배팅 실패) 배팅 금액 회수 못함
결과 화면 출력
게임 종료
게임을 다시 하겠는가?
Yes
No
프로그램 종료

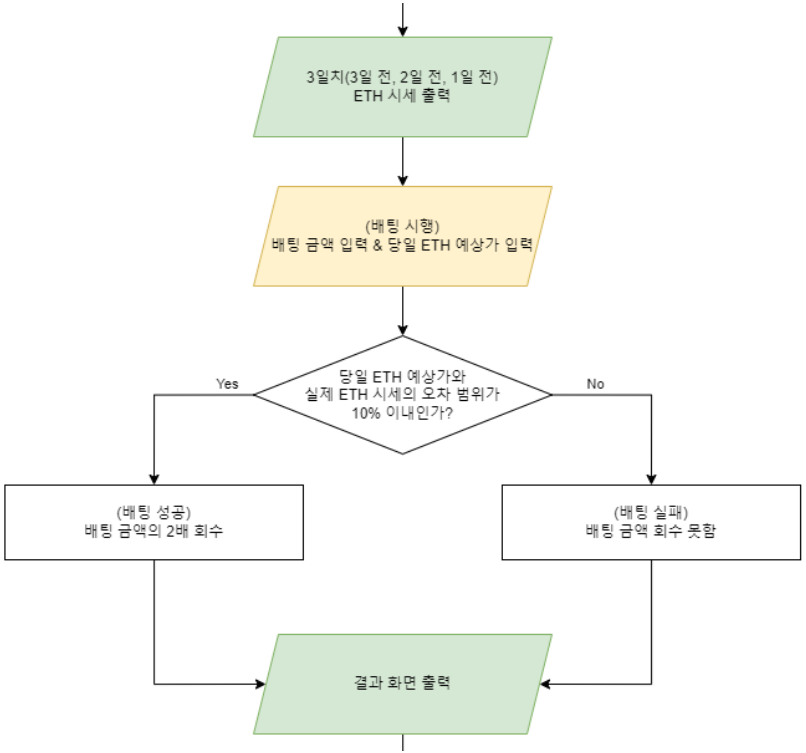
알고리즘


프로그램 실행
게임 컨트랙트 소유자인가?
Yes
게임에 사용할 ETH 충전 or 회수
No
게임 시작
3일차(3일 전, 2일 전, 1일 전) ETH 시세 출력
(예측 시점) 예측 결과 입력 & 입찰 ETH 개상가 입력
당일 ETH 개상가와 실제 ETH 시세의 오차 범위가 10% 이내인가?
Yes
No
(예측 성공) 예측 결과의 2배 회수
(예측 실패) 예측 결과 회수 취소
결과 화면 출력
게임 종료
게임을 다시 하겠는가?
Yes
No
프로그램 종료

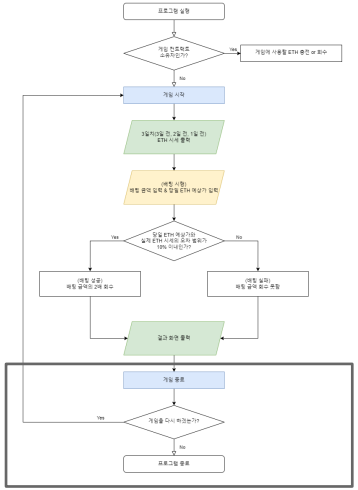

프로그램 실행
게임 컨트랙트 소유자인가?
Yes
게임에 사용할 ETH 충전 or 회수
No
게임 시작

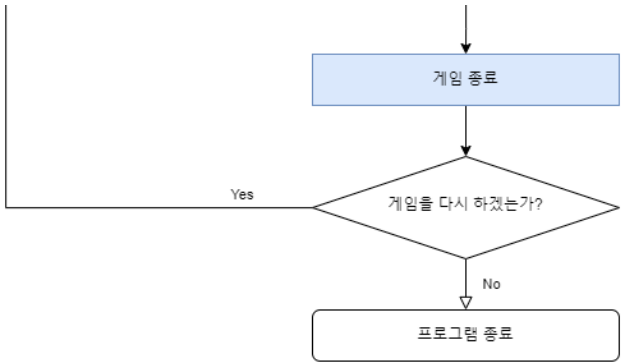
알고리즘


프로그램 실행
게임 한도횟수를 소모하였는가?
Yes
게임에 사용할 ETH 충전 or 회수
No
게임 시작
3일치(3일 전, 2일 전, 1일 전) ETH 시세 출력
(배팅 시행) 배팅 금액 입력 & 당일 ETH 예상가 입력
당일 ETH 예상가와 실제 ETH 시세의 오차 범위가 10% 이내인가?
Yes
No
(배팅 성공) 배팅 금액의 2배 회수
(배팅 실패) 배팅 금액 회수 못함
결과 화면 출력
게임 종료
게임을 다시 하겠는가?
Yes
No
프로그램 종료


3일치(3일 전, 2일 전, 1일 전)
ETH 시세 출력
(배팅 시행)
배팅 금액 입력 & 당일 ETH 예상가 입력
당일 ETH 예상가와
실제 ETH 시세의 오차 범위가
10% 이내인가?
Yes
No
(배팅 성공)
배팅 금액의 2배 회수
(배팅 실패)
배팅 금액 회수 못함
결과 화면 출력

알고리즘


프로그램 실행
게임 전적파일 소유하십니까?
Yes
게임에 사용될 ETH 받으셨나요?
No
게임 시작
3일차(3일 전, 2일 전, 1일 전) ETH 시세 출력
(예측 시점) 매일 종목 입력 & 입력 ETH 개상가 입력
당일 ETH 개상가와 실제 ETH 시세의 오차 범위가 10% 이내인가?
Yes
No
(예측 성공) 매일 얻어지는 2배 획득
(예측 실패) 매일 얻어 획득 못함
결과 화면 출력
게임 종료
게임을 다시 하겠는가?
Yes
No
프로그램 종료


게임 종료
게임을 다시 하겠는가?
Yes
No
프로그램 종료

04

컨트랙트 구성

컨트랙트 구성

01

DataConsumerV3.sol

Chain link로부터 ETH 시세
데이터를 가져오는 컨트랙트

02

OracleGame.sol

Oracle Casiono를
실행하는 컨트랙트

05

코드

https://github.com/giirafe/oracle_toyproject

코드

01

DataConsumerV3.sol

Chain link로부터 ETH 시세
데이터를 가져오는 컨트랙트

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;

import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

contract DataConsumerV3 {
    AggregatorV3Interface internal dataFeed;

    constructor() {    639518 gas 614600 gas
        dataFeed = AggregatorV3Interface(
            0x694AA1769357215DE4FAC081bf1f309aDC325306 // * Network: Sepolia, * Data Feed: ETH/USD
        );
    }

    event ConsoleLog(
        string console
    );

    // uint80 public ROUNDID = 0;

    // ETH/USD 데이터 피드의 최신 데이터의 roundID 저장
    function getLatestData() public view returns (uint80,uint256) {    infinite gas
        // prettier-ignore
        (
            uint80 roundID,
            int answer,
            /*uint startedAt*/,
            /*uint timeStamp*/,
            /*uint80 answeredInRound*/
        ) = dataFeed.latestRoundData();
        uint256 DECIMALS =  10 ** (dataFeed.decimals());
        uint256 answerUnsigned = uint256(answer);
        uint256 answerDecimalApplied = answerUnsigned / DECIMALS;
        return (roundID, answerDecimalApplied);
    }
```

```
// roundID를 입력하면 ETH/USD를 출력하는 함수
function GETROUNDDATA(uint80 _roundID) public view returns (uint256) {    infinite gas
    (
        ,
        int256 answer,
        ,
        ,
    ) = dataFeed.getRoundData(_roundID);
    uint256 DECIMALS =  10 ** (dataFeed.decimals());
    uint256 answerUnsigned = uint256(answer);
    uint256 answerDecimalApplied = answerUnsigned / DECIMALS;
    return answerDecimalApplied;
}

 // 이전 데이터 출력
function getFormerPrice() public view returns (uint256, uint256, uint256) {    infinite gas
    (
        uint80 roundID,
    ) = getLatestData();
    return (GETROUNDDATA(roundID-3), GETROUNDDATA(roundID-2), GETROUNDDATA(roundID-1));
}
```

```
    // 예상가 입력, 결과 출력
    function inputExpectedPrice(uint256 _userExpect) public returns(uint256, bool) {
            infinite gas
        (
            ,
            uint256 currentRoundAssetPrice
        ) = getLatestData();
        uint256 positiveTenPercentFigure = (currentRoundAssetPrice + (currentRoundAssetPrice*10)/100);
        uint256 negativeTenPercentFigure = (currentRoundAssetPrice - (currentRoundAssetPrice*10)/100);
        // int256 errorFigure = int256(currentRoundAssetPrice - _userExpect);
        emit ConsoleLog("tenPercentFigure Set");
        bool win;

        if(_userExpect >= positiveTenPercentFigure) {
            win = false;
        }
        else if(_userExpect <= negativeTenPercentFigure) {
            win = false;
        }
        else{
            win = true;
        }

        return (currentRoundAssetPrice, win);
    }
}
```

코드

02

OracleGame.sol

Oracle Casino를
실행하는 컨트랙트

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.3;

import "./DataConsumerV3.sol"; // 외부가격 가져오는 ChainLink Contract Import

contract OracleGame {

    address payable public owner;
    uint256 constant oracleFee = 0.01 ether;

    // 각 회차는 gameId를 key로 저장되며 새로운 회차가 진행될 시 gameId를 증가시켜준다.
    // 각 회차마다 사용자는 하나의 Bet을 생성할 수 있고 이를 Double Mapping을 통해 관리한다.
    mapping(uint256 => mapping(address => Bet)) public betsHistory;

    event GameResult(
        address indexed gamePlayer,
        uint256 currentAssetPrice,
        bool win,
        uint256 winnings
    );

    event PreviousPrices(
        uint256 threeDaysBefore,
        uint256 twoDaysBefore,
        uint256 oneDayBefore
    );

    event ConsoleLog(
        string consoleMsg,
        uint256 value
    );

    uint256 public gameId; // gameId의 Increment를 통해 한 회차의 베팅 종료시 다음 회차의 베팅으로 넘어가게끔 한다
    uint public assetPrice; // 현재는 임의로 assetPrice를 설정(추후에 ChainLink를 통해 1일전,2일전,3일전,현재의 assetPrice를 가져온다)
    bool public bettingOpen; // Betting 활성화 상태
```

```
// ChainLink Oracle 객체
DataConsumerV3 internal ChainLinkOracle;

struct Bet {
    uint amount;
    uint prediction;
    bool exists;
}

// 배포한 DataConsumerV3.sol 주소를 처음 deploy시 인자로 전달하여 ChainLinkOracle 객체 설정
constructor(address _ChainLinkOracleAddress) {
    owner = payable(msg.sender);
    bettingOpen = true;
    gameId = 0;
    ChainLinkOracle = DataConsumerV3(_ChainLinkOracleAddress);
}

// 한 차례의 게임이 끝났을시(다음날로 넘어갔을시) 그 다음 회차의 게임을 위해 gameId++
function gameInitialization() public {    infinite gas 956400 gas
    gameId++; // GameId Increment
    bettingOpen = true; // Betting 활성화
    // 실제 Chain Link 연결시 어떤 asset으로 Betting을 열지 _asset input을 통해 설정한다.

}

// 7.21 기존 Emit을 이용하여 자산의 가격을 Return 했으나 gas fee 절약을 위해 return 사용
function getThreeDaysPrices() public view returns(uint256,uint256,uint256) {    48838 gas
    (
        uint256 threeDaysBefore,
        uint256 twoDaysBefore,
        uint256 oneDayBefore
    ) = ChainLinkOracle.getFormerPrice();

    return (threeDaysBefore,twoDaysBefore,oneDayBefore);
}
```

```
// 현재의 gameId를 key로 유저는 Betting을 시행한다
function placeBet(uint _prediction) public payable {
    Bet storage betInstance =  betsHistory[gameId][msg.sender]; // betsHistory에서 gameId와 msg.sender의 정보를 통해 Bet structure 객체를 가져온다.
    require(bettingOpen, "Betting is closed");
    require(!betInstance.exists, "User already placed a bet");
    require(msg.value > 0, "User should bet more than 0");

    // 사용자가 보낸 ETH양 만큼 베팅 금액 설정
    betInstance.amount = msg.value;    75512 gas
    betInstance.prediction = _prediction;
    betInstance.exists = true;
}
```

```
// 사용자가 현재 회차에 자신의 Bet이 존재시 해당 Bet에 대한 결과를 확인하는 함수
function redeemResult() public {
    Bet storage betInstance =  betsHistory[gameId][msg.sender];
    address payable payableMsgSender = payable(msg.sender);
    require(betInstance.exists, "User's bet doesn't exist");
    uint256 winnings;
    uint256 currentAssetPrice;
    bool win;

    (
        currentAssetPrice,
        win
    ) = ChainLinkOracle.inputExpectedPrice(betInstance.prediction);

    // emit ConsoleLog("Input Expected Price Done", 1);

    if(win){
        winnings = betInstance.amount * 2;
        emit ConsoleLog("Winning Amount", winnings);    infinite gas
        require(address(this).balance > winnings,"This Service Smart Contract doesn't have enough Balance to pay user");
        payableMsgSender.transfer(winnings);
        betInstance.exists = false;
        emit GameResult(msg.sender, currentAssetPrice, win, winnings);
    } else {
        winnings = 0;
        betInstance.exists = false;
        emit GameResult(msg.sender, currentAssetPrice, win, winnings);
    }
}
```

```
// ChainLink 가격 조회 수수료 위해 필요한 Eth 수급
function fillInEth() public payable {
    require(msg.sender == owner, "Only the owner can fill in ETH for this Smart Contract");
    require(msg.value > 0,"You should fill in ETH larger than 0");
}

// 서비스 owner가 해당 Smart Contract에 저장된 ETH를 모두 출금하는 function
function withdrawAllEth() public {
    require(msg.sender == owner, "Only the owner can withdraw");
    owner.transfer(address(this).balance);
}
```

06

참고 자료

참고 자료

Chainlink Overview
| Chainlink Documentation

https://docs.chain.link/getting-started/conceptual-overview

Chainlink Any API Documentation
| Chainlink Documentation

https://docs.chain.link/any-api/introduction

Price Feed Contract Addresses
| Chainlink Documentation

https://docs.chain.link/data-feeds/price-feeds/addresses

Smart Contract Kit Chainlink
| Github

https://github.com/smartcontractkit/chainlink/blob/develop/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol

Thanks!