Bettings on EURO2020 Playoffs

Submitted by:
Andrea Pozzoli Stefano Fedeli

Course held by: Prof. Rana, Plebani, Bruschi

Deib
Politecnico di Milano

Contents

Chapter 1

Introduction

This document aims to describe the work done for the course Blockchain and Distributed Ledger Technologies: Principles, Applications and Research Challenges presented throughout May 2021. Due to the pandemic, 2021 also correspond to the time UEFA Euro 2020 competition was played across the entire Europe attracting many supporters from all the globe. Every four year in fact, this competition takes place to elect the best national team in the whole Europe. Last time Portugal was the final winner but everything can be said for this year. Millions of people are therefore betting on the outcome of the event trying to guess the winner and gain some money. At this point in time, bets are managed by a few companies and people who bets must hope the receipt they get will be accepted by the cashier they will encounter at the time of the cashback. Online bets are not very different as the players must trust the algorithm that is running on the company's servers. In this scenario, a blockchain application that leverages smart contracts could bring more transparency in the process just described and much more possibilities in the world of bets. The following section will go into the details of the solution by offering an overview of the process 2, a details overview of the code 4, a security analysis 5, and a final discussion to underline limits and assumptions made during the development 6.

Chapter 2

Process

2.1 Original Process

Betting is one of the most lucrative and old industry in the world. The main actor are bookmakers, organization or even a single person that accepts and pays off bets on sporting and other events at agreed-upon odds. Digitalization has brought a different way to interact with the bookmakers but the process has not changed much. The interaction start with the bookmaker fixing the odds on the decided event and then everyone can decide to bet on the likelihood of the event. At this point the bookmaker release a receipt to the client. The process then concludes itself when the event is completed and the outcome is known to the public. At this point in case of victory the owner of the receipt can ask for its prize. Making a bet means to subscribe an agreement between a client and a bookmaker that in case of a certain outcome would eventually pay a fixed sum to the owner of the receipt stating the right event outcome.

We can model the process very easily using BPMN as can be seen in Figure 2.1

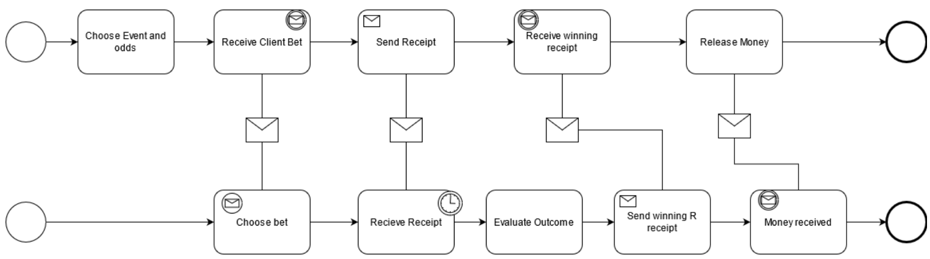

Figure 2.1: Betting BPMN Process

2.2 Blockchain System[1]

The process is clearly a multi party application that involves both client and the bookmaker giving room for a blockchain system to be considered. However, in the process of betting, this is a strong central authority, the bookmaker. When the trusted authority experiences problems, all the users accessing the services from it would be affected and no more bets or money can be accessed. The presence of a central authority it is not

beneficial for a blockchain system, but the ability of decentralize all operations gives us the opportunity to think forward the system to implement.
In this process, transparency is not strictly required but would be not problematic to share previous transaction publicly. Pseudo-anonimization would be enough to protect casual player and avoid money laundry. If some data is strictly confidential it could be easily encrypted and moved on the chain.
To avoid scams, transaction history is required. Having the whole transaction history would be beneficial to verify the good behavior of the parties involved. Without any doubt, in such process immutability is needed to ensure bet are not changed during the event.
The whole system might need high performance depending on the specific type of bet. However, the most typical way of football betting it does not require real time responses and thus high performances are not required. Table 2.1

TABLE 2.1: Feasibility Analysis

Requirement	**Need**
Multiparty	Required
Trusted authority	Decentralised
Centralised operation	Not Required
Data transparency/confidentiality	Trasparent
Data integrity	Required
Data immutability	Required
High performance	Not Required

All the consideration above, bring us to the conclusion that the betting process, as described by Figure 2.1, has good reasons to be transformed using a public distributed ledger such as Ethereum. After the analysis, Ethereum is the only mainstream blockchain project that offers all the capabilities to decentralize our central authority in charge of receiving bets and release money in case of winning.

Chapter 3

Technical Implementation

As mentioned in the previous section, the work opted for a public permissionless blockchain able to deploy smart contracts to be able to decentralize all the bookmaker operation. Given the requirements, Ethereum blockchain was the best choice and therefore the project was developed upon it. The code was written in Solidity and compiled with version 0.4.26 on the RemixIDE [1]. The smart contract is acting as both as a bookmaker and a decentralized exchange. Changing behavior depending on the time in the real world. For this project, the decision was to concentrate on the most relevant event of the football season which corresponds to EURO2020. The contract is deployed at the address 0xDc26F4B5Ad700D4D4c4dD5E9b9b19Fb9bacC8Ce4, on the Ropsten Ethereum Testnet.

3.1 System Logic

The idea of the project is to build an application that releases national team tokens in a first phase and then, in a second phase, owners of the tokens would be able to redeem their coins in exchange of money. The first phase goes on until the beginning of EURO2020 playoffs and then the second phase starts at the end of the tournament.Very simple activity of a traditional bookmaker but decentralized and simplified.
Interesting enough, aside phase one and phase two, there is another possibility for the owners of a token which is the one of accessing the dentralized exchange of national team tokens.

3.1.1 Phase One

Phase One is the period of time that goes between the deployment of the smart contract and June 26th. In this period of time people are able to purchase national team token from the contract or from other people's offers. In this way, people are betting on one or multiple national team they prefer, with the promise that the contract will release a certain amount of money based on the final placement of the team in the competition.

[1] https://remix.ethereum.org/

3.1.2 Phase Two

This phase goes live on July 12th and make available the prizes to all the addresses that shows ownership of tokens. Users can asks the contract to convert their national team token to the relative price.

3.1.3 Decentralized Exchange

The decentralized exchange is a feature that does not usually appear in the traditional process of betting but it became possible once the operations are decentralized. The smart contract always offers the possibility to sell and buy national team tokens between people at the price they like to most. This DEX capabilities are inspired by 1st generation DEX with onchain order books [2].

[2] https://github.com/daifoundation/maker-otc

Chapter 4

Code Implementation

In this chapter, an analysis of the code implementetion is provided, in particular focusing on the most important functions. The code has been developed through Remix Ethereum. The entire code can be found in A. The structure of the code follows the ERC20 standard.

4.1 place_bet

The function "place_bet" is used to obtain an amount of tokens of a specific team. The function is defined as payable because it requires a transaction of Ethereum in order to complete the order of tokens. The order can be placed before the beginning of the playoff of EURO202, if there are enough tokens of the selected national team. After the transaction, the tokens are added to the buyer's wallet and they are no more available to be ordered.

4.2 oracle

The oracle function is used to set the results of the competition after the end of the final match and assigns a payback value to each team's token.

4.3 collect_bet

After the end of the competition, people that own tokens can collect the prize assigned to the tokens of each team.

4.4 sell

This function is needed to create an offer for selling tokens to other players. The owner of a token can decide to sell it fixing a price and an amount of token. A fee is added to the price and then the offer is published.

4.5 buy

A person can decide to buy tokens from another player by paying the price of a published offer. If the amount of money offered is greater than the base price and the amount of tokens is lower or equal than the available tokens in the offer, a transaction is made.

Chapter 5

Security Issues

When working with a distributed ledger that is immutable such as the blockchain, security assume a central role in the picture. When the code is deployed in fact, nobody would be able to change the code and attacks have all the time to figure out ways to hack the system. Given the amount of time available for completing the project, security was not a major concern as the focused fall into other issues. However, a simple analysis was carried out while developing the application. Focusing on the main attacks known in the blockchain ecosystem, here below is possible to read the analysis.

- **Arithmetic Underflow and Overflow** As simple as it might sound, this issue were not considered in the deployment as from Solidity 0.8.0 those checks are automatically embedded. SafeMath library was not considered for this project.
- **Unexpected Ether** This attacks tries to hack the smart contract by augmenting the budget of the contract. However, in the code we deployed there are no checks on the account balance and therefore this attacks is not really possible.
- **Default Visibility** Choosing the wrong visibility for the function would allow hacks to the contract but during the development everything was chosen carefully and double checked.
- **Reentrancy** Reentrancy make use of fallback function to hack the contract. However to contrast this attack threat, all the instruction that are actually sending money are always the last in each method. As the state is already changed when money is send out, reentrancy is made much more difficult.
- **Entropy Illusion** As there is no randomness in the contract, this attack is not an issue.
- **Denial Of Service** Unfortunately, the contract is very susceptible to this king of attack. There are few bugs that an attacker could leverage to create a Denial Of Service. While most of them are very expensive attacks, an attack can easily create hundreds of token offers and DoS the contract.

Chapter 6

Discussion

The project aimed to show a possible blockchain systems, build on Ethereum's smart contract, able to transform and decentralize the betting process analyzed in Section 2. The results are interesting and confirm the analysis carried out, giving the impression it would be possible to shift betting on this kind of platforms. However, the solution proposed is not flawless. Several limitations are present, making almost impossible to deploy the contract on the mainnet.

6.1 Limitations

The main limitation is the absence of a decentralized Oracle for the results of the event. Unfortunately there are not yet established projects able to simply provide data coming from the off-chain world. Given the short amount of time available the only option was to use a Centralized Oracle that corresponds to the owner of the contract. Another limitation concerns the DEX capabilities that leverage a 1st generation schema and therefore an onchain order book instead of liquidity pools. This brings on board all the problems that those type of exchanges have shown in the past such as wasting fees and low volumes. However, given the nature of the application all those problem are smoothed out. As no static analysis as been carried out on the code, it might be possible to have created bugs that lead to hack the system and lose money.

6.2 Future Work

The future work should concentrate on removing as much as possible the list of limitation and fully decentralize the process of betting. As few security holes have been evaluated it would be highly important to addess all the security issue before releasing the code on the mainnet.

Appendix A

Appendix A

```
/*
Implements EIP20 token standard: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
.*/


pragma solidity ^0.4.21;


contract EURO2020 {

    uint256 constant private MAX_UINT256 = 2**256 - 1;

    uint256 constant private PRICE_PER_UNIT = 0.03 ether;
    uint256 constant private MAX_AVALIABLE = 10;
    uint256 constant private NUM_TEAMS = 16;
    uint256 public time_start_playoff;
    uint256 public time_end_tournament;
    address private owner;
    mapping (string => uint256) private TEAM_MAP;
    mapping (uint8 => uint256) private LEVELS;
    mapping (string => uint256) private PAYBACK;

    uint public last_offer_id;
    uint256 constant private our_fee = 0.005 ether;
    bool locked;
    mapping (uint => OfferInfo) private offers_map;
    struct OfferInfo {
        uint256  price;
        uint256  amount;
        string   pair;
        address  owner;
        uint64   timestamp;
    }

```

```
    mapping (address => uint256) public balances;
    mapping (address => mapping (string => uint256)) private bets;
    /*
    NOTE:
    The following variables are OPTIONAL vanities. One does not have to include them.
    They allow one to customise the token contract & in no way influences the core functionality.
    Some wallets/interfaces might not even bother to look at this information.
    */
    string public name = "Project Blockchain Polimi";    //fancy name: eg Simon Bucks
    uint8 public decimals = 0;                //How many decimals to show.
    string public symbol = "EURO2020";               //An identifier: eg SBX
    uint256 public totalSupply;

    constructor() public {
        totalSupply = MAX_AVALIABLE*NUM_TEAMS;
        TEAM_MAP['ITA'] = MAX_AVALIABLE;
        TEAM_MAP['WAL'] = MAX_AVALIABLE;
        TEAM_MAP['SWI'] = MAX_AVALIABLE;
        TEAM_MAP['BEL'] = MAX_AVALIABLE;
        TEAM_MAP['DEN'] = MAX_AVALIABLE;
        TEAM_MAP['NED'] = MAX_AVALIABLE;
        TEAM_MAP['AUS'] = MAX_AVALIABLE;
        TEAM_MAP['UKR'] = MAX_AVALIABLE;
        TEAM_MAP['CRO'] = MAX_AVALIABLE;
        TEAM_MAP['RCZ'] = MAX_AVALIABLE;
        TEAM_MAP['SWE'] = MAX_AVALIABLE;
        TEAM_MAP['SPA'] = MAX_AVALIABLE;
        TEAM_MAP['POR'] = MAX_AVALIABLE;
        TEAM_MAP['FRA'] = MAX_AVALIABLE;
        TEAM_MAP['GER'] = MAX_AVALIABLE;
        TEAM_MAP['ENG'] = MAX_AVALIABLE;

        LEVELS[0] = 1;
        LEVELS[1] = 2;
        LEVELS[2] = 4;
        LEVELS[3] = 16;
        LEVELS[4] = 50;

        time_end_tournament =  now + 18 days;
        time_start_playoff = now + 3 days;
        owner = msg.sender;
        balances[owner] = totalSupply;
    }

    /*
```

```
    * FUNCTIONs FOR ERC-20 COMPATIBILITY
    */
    function transfer(address _to, uint256 _value) public returns (bool success) {
        require(balances[msg.sender] >= _value);
        balances[msg.sender] -= _value;
        balances[_to] += _value;
        return true;
    }

    function balanceOf(address _owner) public view returns (uint256 balance) {
        return balances[_owner];
    }


    /*
    * BOOKMAKER FUNCTIONs
    */
    function place_bet(string _team, uint256 _amount) public payable returns (bool success) {
        require(now <= time_start_playoff);
        require(msg.value >= _amount * PRICE_PER_UNIT);
        require(TEAM_MAP[_team] >= _amount);
        TEAM_MAP[_team] -= _amount;
        bets[msg.sender][_team] += _amount;
        balances[msg.sender] += _amount;
        balances[owner] -= _amount;
        return true;
    }

    function collect_dust() public returns(bool success) {
        // instead of checking if the owner has totalSupply number of coins, it is possible to put a time condition
        require(msg.sender == owner);
        require(balances[msg.sender] == totalSupply);
        owner.transfer(address(this).balance);
        return true;
    }

    function oracle(string _team, uint8 _level) public returns (bool success) {
        require(msg.sender == owner);
        if (TEAM_MAP[_team] == MAX_AVALIABLE){
            PAYBACK[_team] = 0;
        } else {
            PAYBACK[_team] = (address(this).balance * LEVELS[_level] / 100) / (MAX_AVALIABLE - TEAM_MAP[_team]);
        }
        return true;
    }
```

```
    function collect_bet(string _team) public returns (bool success) {
        require(now >= time_end_tournament );
        require(bets[msg.sender][_team] >= 1);
        uint256 amount = bets[msg.sender][_team];
        bets[msg.sender][_team] = 0 ;
        balances[owner] += amount;
        balances[msg.sender] -= amount;
        msg.sender.transfer(PAYBACK[_team] * amount);
        return true;

    }



    /*
    * DEX FUNCTIONs
    */
    modifier can_buy(uint id) {
        require(offer_isActive(id));
        _;
    }

    modifier can_cancel(uint id) {
        require(offer_isActive(id));
        require(offer_getOwner(id) == msg.sender);
        _;
    }
    modifier synchronized {
        require(!locked);
        locked = true;
        _;
        locked = false;
    }

    function offer_isActive(uint id) public view returns (bool active) {
        return offers_map[id].timestamp > 0;
    }
    function offer_getOwner(uint id) public view returns (address own) {
        return offers_map[id].owner;
    }

    function getOfferInfo(uint id) public view returns (string, uint256, uint256) {
      OfferInfo memory offer = offers_map[id];
      return (offer.pair, offer.price, offer.amount);
    }

```

```
    function _next_id() internal returns (uint) {
        last_offer_id++;
        return last_offer_id;
    }

    function buy(uint256 id, uint8 quantity) public payable can_buy(id) synchronized returns (bool) {

        OfferInfo memory offer = offers_map[id];
        uint256 to_spend = quantity * offer.price;
        require(bets[offer.owner][offer.pair] >= quantity);
        require(to_spend <= address(this).balance);
        require(to_spend <= msg.value);
        require(quantity > 0);
        require(to_spend > 0);
        require(offer.amount >= quantity);

        bets[offer.owner][offer.pair] -= quantity;
        bets[msg.sender][offer.pair] += quantity;
        balances[offer.owner] -= quantity;
        balances[msg.sender] += quantity;
        offers_map[id].amount -= quantity;
        address(offer.owner).transfer(to_spend - our_fee * quantity);

        if (offers_map[id].amount == 0) {
          delete offers_map[id];
        }

        return true;
    }

    // Cancel an offer. Refunds offer maker.
    function cancel_offer(uint id) public can_cancel(id) synchronized returns (bool success) {
        // read-only offer. Modify an offer by directly accessing offers[id]
        delete offers_map[id];
        return true;
    }

    // Make a new offer. Takes funds from the caller into market escrow.
    function sell(string pair, uint256 price, uint256 amount) public synchronized returns (uint256 id) {
        require(price > 0);
        require(amount > 0);
        require(bets[msg.sender][pair] >= amount);

        OfferInfo memory info;
        info.pair = pair;
        info.price = price + our_fee;
```

```
        info.amount = amount;
        info.owner = msg.sender;
        info.timestamp = uint64(now);
        id = _next_id();
        offers_map[id] = info;

        return id;
    }

    function getNumberTokensOnTeam(address _address, string _team)
    public view returns (uint256){
        return bets[_address][_team];
    }

    function getTeamNames() public pure returns (string){
        return "Teams: ENG, ITA, FRA, GER, WAL, SWI, BEL, DEN, NED,
    AUS, UKR, CRO, RCZ, SWE, SPA, POR";
    }

    function getAvailableTokensOnTeam(string _team) public view
    returns (uint256){
        return TEAM_MAP[_team];
    }

    /*function getPayback(string _team) public view returns (uint256)
    {
        return PAYBACK[_team];
    }*/

    /*function getTeamMap(string _team) public view returns (uint256)
    {
        return TEAM_MAP[_team];
    }*/

    function getPricePerUnit() public pure returns (uint256){
        return PRICE_PER_UNIT;
    }

}
```

References

[1] Sin Kuang Lo, Xiwei Xu, Yin Kia Chiam, and Qinghua Lu. Evaluating suitability of applying blockchain. In *2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 158–161. IEEE, 2017.