2022 Fourth International Conference on Blockchain Computing and Applications (BCCA) | 978-1-6654-9958-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/BCCA55292.2022.9921994

Secure Decentralized Online Gaming with Lending Functionalities

Katharina Alefs, Florian Hartl, Luke Newman, Banu Özdeveci, Wisnu Uriawan
Department of Computer Science
INSA Lyon
69621 Villeurbanne, France
E-mail: katharina.alefs@insa-lyon.fr, florian.hartl@insa-lyon.fr, luke.newman@insa-lyon.fr, banu.ozdeveci@insa-lyon.fr, wisnu.uriawan@insa-lyon.fr

***Abstract*—We present a decentralized online gaming platform implemented as a Decentralized Application (DApp) on the Ethereum blockchain. The gaming platform enables secure gaming, where the account balances and the stakes of the players are secured by a smart contract. Moreover, the fair enforcement of the game rules and the deposit of the winnings of the players and the gaming platform into their accounts are guaranteed by the smart contract. The gaming platform proposes lending functionalities that allow players to securely borrow tokens from the gaming platform in order to participate in the games.**

***Index Terms*—Gaming, lending, tokens, blockchain, smart contracts, Ethereum, trust, decentralization.**

I. Introduction

Online gaming comprising of probability games is a popular application. Examples of probability games include dice roll, slots, roulette, and blackjack. These games are also called games of chance. In this paper, we present a gaming platform implemented as a Decentralized Application (DApp) on Ethereum. The gaming platform also provides lending functionalities. After the initialization of the gaming platform by the owner, users are able to interact with the main smart contract and the various functionalities embedded within it. We propose to implement a token system, which means that users must buy tokens to partake in the functionalities within the game. The smart contract offers a wide range of use cases, such as buying tokens (in the game currency), playing different games, withdrawing tokens to convert back to Ether, and lending tokens to the users.

The advantage of implementing the application as a DApp with smart contracts is that it is decentralized. This decentralization implies that no external third parties can control the application or intervene with the smart contract. As opposed to existing centralized online gaming websites, a third party is not able to declare new unfair rules, fees, or any other parameters to take advantage of users. The only interactions with our gaming smart contract are between individual users and the gaming platform owner.

The gaming scenario is very suitable for applying smart contracts since both parties can trust in secure and reliable transactions and a fair game is established. Upon interaction between a user account and our smart contract, there is a guarantee that the user can withdraw whenever they wish after either winning or losing in the game, which also respects the fundamentals of a smart contract and decentralization. In addition, users who interact with the game smart contract, remain anonymous since the only given identifier of a user is their account ID.

Our application is built using Remix, which we use to write, compile, debug and deploy solidity files. For front-end interaction we use React (javascript) to provide a visual UX for the smart contract. In order to compile our solidity files into code that could be run on our client-side application (React), we used Hardhat. To enable interactions with our smart contract on the blockchain, we use Ether.js. On top of all this, we use Metamask to help with managing accounts on the Ethereum blockchain.

The remainder of the paper is structured as follows: Section II discusses related work. Section III describes probability games. Section IV presents the use cases of our gaming platform. Section V presents the design of our gaming platform DApp. Section VI discusses implementation details and lists the source code of some of the main functions of the smart contract. Section VII concludes the paper.

II. Related Work

In this section, we look at some of the existing blockchain-based online gaming platforms. To the best of our knowledge, none of these platforms provide lending functionalities at the moment for fungible tokens. In contrast, our gaming platform proposes the possibility of loans to the players.

FUNToken.io [1] is an Ethereum-based gaming platform that offers a large user community. They plan to supplement on-chain transactions with side-chain transactions for lower cost and lower latency. The Atari Token [2] has good integration capabilities with other decentralized applications. Atari Token provides services to individuals as well as businesses, supports multiple platforms, offers liquidity guarantees, and easy payments.

CoinPoker [3] is a blockchain technology-based platform that uses USDT stable coin as the main in-game currency and the CHP cryptocurrency for bonuses. It offers instant and secure transactions using USDT, ETH, BTC, and CHP tokens. The platform allows anonymity with no KYC checks. The BetU platforms [4] include BetU Verse, EarnU, and BetU. It provides a virtual reality platform, prediction games (sports and esports), and betting platform. BETU tokens are utilized for all winnings, betting rewards, incentives, staking, burning, whale holder benefits, purchasing of NFTs, and governance of the platforms.

LOTTO (The Immutable World Lottery) is a decentralized and cross-border lottery game. The LOTTO [5] lottery runs continuously without a third party. The protocol produces a provably-fair winner using an oracle service. Dotmoovs [6] is a peer-to-peer game playing platform based on an artificial intelligence system. The Moov token is an asset that supports all transactions including buying, selling, and renting.

Exeedme [7] allows gamers to engage and interact with their favorite games. Exeedme provides an earning environment to several gaming communities with one token to fuel the entire platform. Wagerr [8] is a permissionless blockchain-based platform that offers sports betting. Wagerr is supported by Oracles and provides transparency. Wagerr cryptocurrency is designed with a deflationary mechanism that helps the coin retain value under various conditions to protect from market drops and guard price to equilibrium over time.

III. Probability Games

The outcome of a probability game [9], [10] depends on the likeliness of certain events to occur. Probability games may also be called games of chance. In this section, we describe some of the probability games that could be played on our gaming platform.

A. Roll the Dice

In this game, the player enters a guess and a stake. She loses or wins 6 times the stake.

B. Slot machine

The player enters the stake. The result is three digits. If two digits are the same, *Reward* = 3 × *stake*. If three digits are the same, *Reward* = 9 × *stake*.

C. Roulette

The player enters the stake and guesses a number and a color. If the color is correct, *Reward* = 2 × *stake*. If the number is also correct, *Reward* = 36 × *stake*.

D. Blackjack

- *Goal:* Get more points than the dealer, but at most 21 points.
- Draw 2 cards for yourself and 1 for the dealer.
- *Simplification:* Ace is 11 points and unlimited cards.
- Draw as many cards as you want (one after another).
- If you have more than 21 points, you lose directly.
- The dealer draws cards until they have at least 16 points.
- If the dealer has more than 21 points, you win directly.
- If you have more points than the dealer, you win.
- *Reward:* Twice your stake

IV. Use Cases

The use case diagram in Fig. 1 demonstrates the different use cases of the application and who out of the gaming platform owner and user can interact with each use case. The order in which each use case should follow the other is the order in which the use cases are presented in the diagram (top to bottom).

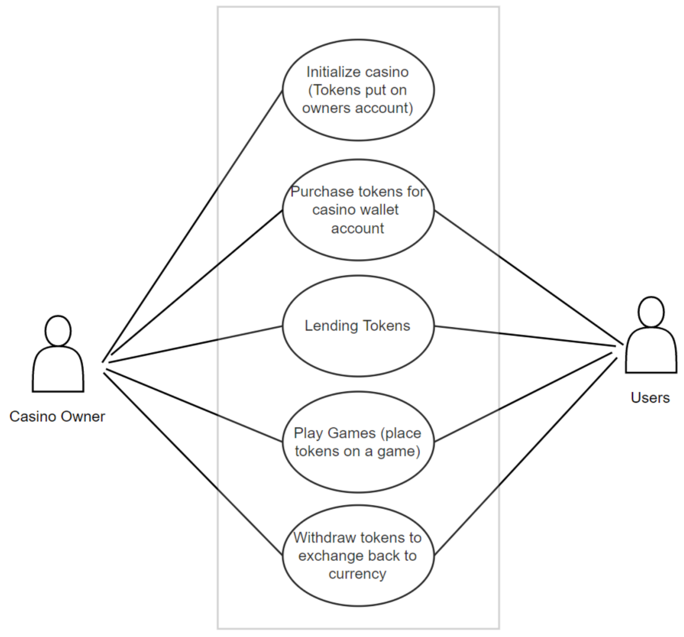
Initialize casino (Tokens put on owners account)
Purchase tokens for casino wallet account
Lending Tokens
Play Games (place tokens on a game)
Withdraw tokens to exchange back to currency
Casino Owner
Users

Fig. 1. Use Cases

1) **Initialize the gaming platform:** The owner must first initialize the gaming platform by investing some amount of Ether for there to be a "gaming platform bank". The users can win tokens from the bank or pay tokens to the bank based on the result of their game. Once this is done, the gaming platform is ready to be interacted with by the users.
2) **Purchase tokens:** For this use case, a user can buy tokens using their existing account balance from the Ethereum network. Once a user purchases tokens, an account balance and user ID is generated within the gaming platform and the user is ready to play the games. The gaming platform owner can also top-up on tokens if they wish.

3) **Lend tokens:** Token lending is an additional use case implemented where a user can request a loan from the gaming platform owner. Upon approval of this loan, the owner transfers the amount desired and a record of this loan (along with interest) is recorded within the contract.
4) **Play games:** Once the user has the tokens, they can then play any of the aforementioned games and based on the result of the game, the user either pays the bank the stake they proposed or receives their winnings. The odds of winning each game are different hence the returns based on the outcome varies from game to game.
5) **Withdraw tokens:** A user can decide to withdraw their tokens from the gaming platform at any stage. When a user withdraws, their tokens get transferred back into Ether and put back onto their Ethereum account balance.

V. Gaming DApp Design

We design a Decentralized Application (DApp) for playing probability games. As discussed before, the game options include Roll the Dice, Slot Machine, Roulette and Blackjack. We also include the lending use case, which allows for a novel functionality. As discussed in Section II, to the best of our knowledge, this functionality is currently not offered for fungible tokens by existing platforms.

This section serves to describe our design choices for modeling the use cases. To provide a deeper understanding of our ideas, we present several diagrams to explain processes and entities within our DApp. Looking at the actors involved in probability games, we first model users/players that can earn or lose tokens by placing bets on an outcome of a game. We add another type of user, a representative of the gaming platform, who can interact with the smart contract and increases the complexity of our application. A justification of having the gaming platform as its own actor is that it is needed for the lending use case, where it actively has to approve a loan.

To establish a fair game within the contract, we make sure to collect the stakes from both parties upfront before the game starts and distribute them according to the rules of the game played. This way, we ensure that once a player and the casino agreed on the conditions and started playing the game, both actors will definitely receive their share once they win. Additionally, we assume that there is general law enforcement outside of the contract that guarantees that users will pay back their debt since our contract does not inherently provide this functionality for lending tokens.

We choose to implement a "gaming platform wallet", an account with which users can manage their spending and losses within the gaming platform. This wallet feature was chosen on the basis that there is a commonly used withdrawal pattern in Solidity programming, a practice that ensures to never let a third user initiate payments to other users. For this reason, instead of distributing token to players' accounts, only they can initiate a payment to themselves by withdrawing tokens from their gaming platform balance. Gaming platforms generally work with their own currency, as for example gaming chips, this therefore provides another reason for tokenizing Ether and having a gaming platform account. In addition, it simplifies lending to work with a local currency, since it is much safer for the gaming platform to lend out chips rather than actual tokens that might be spent outside of their institution.

Regarding the implementation of our application within smart contracts, we initially considered creating a modular environment in which requesting a loan and playing a game would both be part of their own contracts, as both situations work under different conditions. Additionally, we planned to have a different contract for each game type. However, for the current prototype, we decided to focus on the functionalities of our application, and to implement one big contract instead. For future projects we would choose to not work with a monolithic approach as comprehensibility and maintainability can suffer. A representation of the smart contract and its variables and functions can be seen in the class diagram in Fig. 2 (here we model smart contracts as classes).

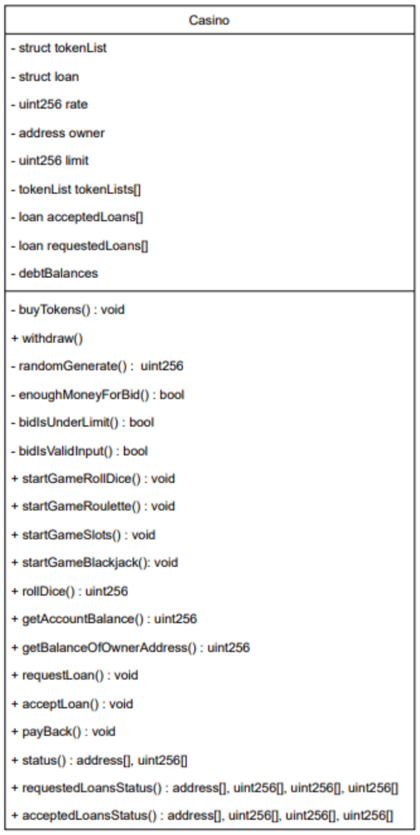
Casino
- struct tokenList
- struct loan
- uint256 rate
- address owner
- uint256 limit
- tokenList tokenLists[]
- loan acceptedLoans[]
- loan requestedLoans[]
- debtBalances
- buyTokens() : void
+ withdraw()
- randomGenerate() : uint256
- enoughMoneyForBid() : bool
- bidIsUnderLimit() : bool
- bidIsValidInput() : bool
+ startGameRollDice() : void
+ startGameRoulette() : void
+ startGameSlots() : void
+ startGameBlackjack(): void
+ rollDice() : uint256
+ getAccountBalance() : uint256
+ getBalanceOfOwnerAddress() : uint256
+ requestLoan() : void
+ acceptLoan() : void
+ payBack() : void
+ status() : address[], uint256[]
+ requestedLoansStatus() : address[], uint256[], uint256[], uint256[]
+ acceptedLoansStatus() : address[], uint256[], uint256[], uint256[]

Fig. 2. Class Diagram: Gaming Platform

We also modeled a hypothetical class diagram for the Blackjack game to demonstrate how such a smart contract could be implemented. The diagram is shown in Fig. 3.

To develop a deeper understanding of our design choices, we also include two sequence diagrams illustrating the roll dice game (Fig. 4) and the lending use case (Fig. 5).

Finally, we conclude this section with a state diagram that covers all the states that a user can find themselves in in terms

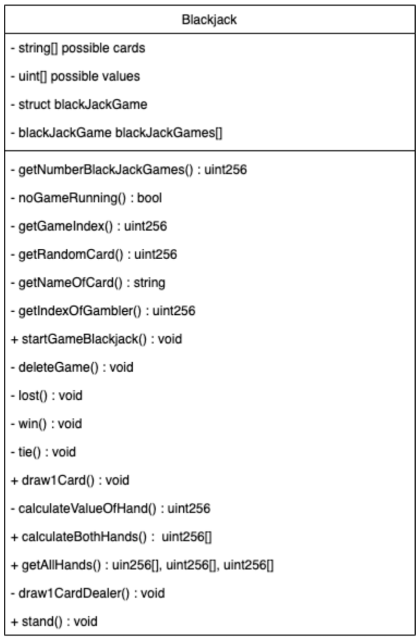
Blackjack
- string[] possible cards
- uint[] possible values
- struct blackJackGame
- blackJackGame blackJackGames[]
- getNumberBlackJackGames() : uint256
- noGameRunning() : bool
- getGameIndex() : uint256
- getRandomCard() : uint256
- getNameOfCard() : string
- getIndexOfGambler() : uint256
+ startGameBlackjack() : void
- deleteGame() : void
- lost() : void
- win() : void
- tie() : void
+ draw1Card() : void
- calculateValueOfHand() : uint256
+ calculateBothHands() : uint256[]
+ getAllHands() : uin256[], uint256[], uint256[]
- draw1CardDealer() : void
+ stand() : void

Fig. 3. Class Diagram: Blackjack

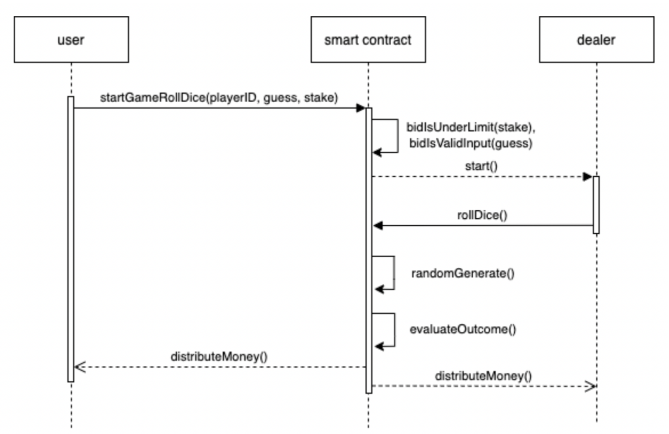

Fig. 4. Sequence Diagram: Roll the Dice

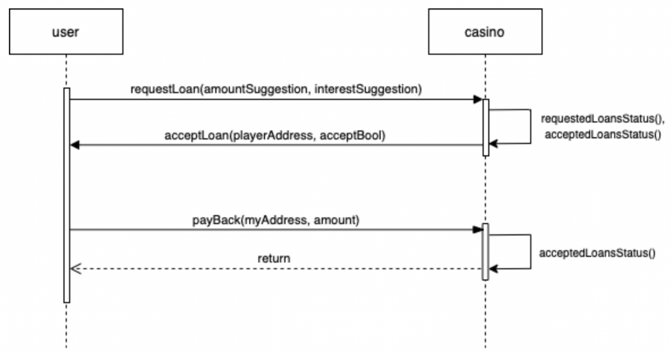
user
casino
requestLoan(amountSuggestion, interestSuggestion)
requestedLoansStatus(),
acceptedLoansStatus()
acceptLoan(playerAddress, acceptBool)
payBack(myAddress, amount)
acceptedLoansStatus()
return

Fig. 5. Sequence Diagram: Lend Tokens

of their financial stability (with respect to tokens).

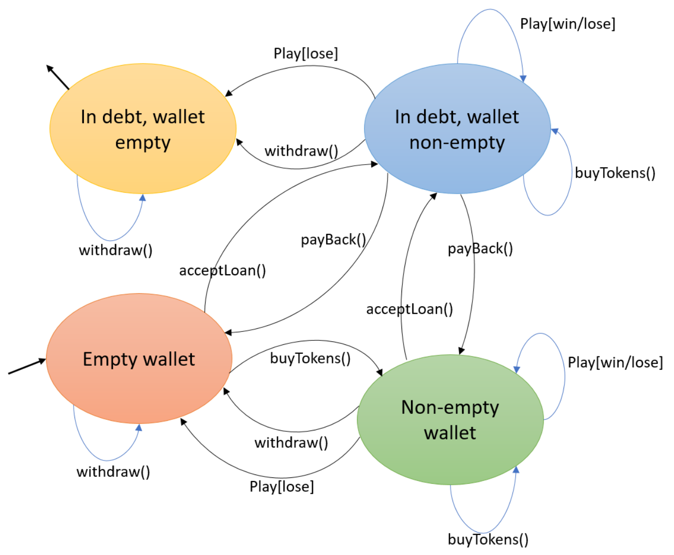

Fig. 6. State Diagram

VI. Implementation

The front end aspect of our project provides a graphical user interface for displaying the functionalities of our Ethereum smart contract. For front-end interaction, we use React [11], [12] to provide a visual interface for the smart contract. React is a free and open-source front-end JavaScript library for building user interfaces using UI components. We ran our client-side application (React) in conjunction with Solidity [13] using Hardhat [14]. Similar to tools such as Ganache [15] and Truffle [16], Hardhat is used to develop an Ethereum environment and framework for full stack development. To enable interactions with our smart contract on the blockchain, we use Ether.js [17]. Similar to Web3.js [18], Ether.js is a javascript web client library, which we use to build our javascript frontend and interact with the Ethereum blockchain. MetaMask [19] is a tool, which we used that allows users to store and manage account keys, broadcast transactions, send and receive Ether, and securely connect to decentralized applications through a web browser.

In this section, we list the Solidity source code of some of the significant functions of the gaming platform smart contract. The complete source code of the prototype is available on GitHub: https://github.com/Newman251/CasinoSmartContract.

A. Buy Tokens

This function allows anyone to buy tokens by putting Ether into the contract. At least 10 tokens have to be bought, and a maximum of 1000 tokens can be bought. If the user already has an account on the gaming platform, the newly bought tokens get added to his account. Otherwise, a new account is created for the user to which the tokens are added.

```
    //Anyone can buy tokens by putting Ether into the contract.
    function buyTokens() external payable{
        require(msg.value > rate*10, "Too low, cost must be greater
than 100");     //At least 10 tokens have to be bought
        require(msg.value <= rate*1000, "Too high, cost must be
less than 10000");  //To prevent fraud at maximum 1000 tokens can
be bought at the same time

        uint256 tokens = (msg.value/rate);

        //check if the user already has a casino account
        bool userFound;
        for (uint i = 0; i < tokenLists.length; i++) {
            //If the user has already a casino account, add the
            //newly bought tokes to his account
                if(tokenLists[i].userId == msg.sender){
                  tokenLists[i].amount += tokens;
                  userFound = true;
            }
            }
        //If all accounts belong to other users, create
        //a new one for the new customer and add their tokens
        if(!userFound){
            tokenList memory newTokenList = tokenList(msg.sender,
tokens);
            tokenLists.push(newTokenList);
        }
    }
```

Fig. 7. Buy Tokens

B. Request Loan

This function allows players to request a loan by proposing an amount and interest rate. In this function, we make sure that the requested amount is not less than 100 and not more than 10000. Moreover, the interval of the proposed interest rate is also verified. The loan proposal requires approval by the gaming platform owner.

```
    // requesting Loan by proposing an amount and interest rate
    function requestLoan (uint256 amountSuggestion, uint256
interestSuggestion) public payable{
        require(amountSuggestion > 100, "You must lend more than
100");
        require(amountSuggestion < 10000, "You cannot lend more
than 10000");
        require(interestSuggestion < 100, "The interest rate needs
to be lower than 100");
        require(interestSuggestion > 0, "The interest rate needs to
be higher than 0");
        require(getBalanceOfOwnerAddress() > amountSuggestion,
"Your requested amount is bigger than what the casino can provide");

        uint256 interest = interestSuggestion;
        uint256 amountLended = amountSuggestion;
        uint256 amountToPayBack = amountSuggestion +
(amountSuggestion * interest)/100;

        bool foundRequest = false;
        for (uint i = 0; i < requestedLoans.length; i++){
            if (msg.sender == requestedLoans[i].account){
                foundRequest = true;
            }
        }
        if (!foundRequest){
        loan memory newLoan = loan(msg.sender, amountLended,
interest, amountToPayBack);
               requestedLoans.push(newLoan);
        } else {
            require(!foundRequest, "You already requested a loan
that hasn't been approved yet");
        }
    }
```

Fig. 8. Request Loan

C. Accept Loan

The gaming platform owner can accept a loan request by inserting the player's address and a boolean (true = accept) and (false = reject). If the loan is accepted, the smart contract transfers the balance amount to the message sender.

```
    // the casino can accept a Loan request by inserting the
players address and a boolean (true = accept) and (false = reject)
    function acceptLoan (address payable userAdress, bool accept)
public{
        require((accept == false || accept == true), "The value
must be either 0 (not accept) or 1 (accept)");
        require(debtBalances[userAdress] == 0, "Your still in debt.
Please pay for your past loans first");
        if(accept == true){
            for (uint i = 0; i < requestedLoans.length; i++){
                if (requestedLoans[i].account == userAdress){
                    debtBalances[userAdress] +=
requestedLoans[i].amountToPayBack;

                    // transfer the balance amount to the message
sender

userAdress.transfer(requestedLoans[i].amountLended);

                    loan memory newLoan = loan(userAdress,
requestedLoans[i].amountLended, requestedLoans[i].interestRate,
requestedLoans[i].amountToPayBack);
                           acceptedLoans.push(newLoan);

                    if (requestedLoans.length > 1){
                        delete requestedLoans[i];
                        requestedLoans[i] =
requestedLoans[requestedLoans.length-1];
                        delete
requestedLoans[requestedLoans.length-1];
                        requestedLoans.length--;
                    } else {
                        delete requestedLoans[i];
                        requestedLoans.length--;
                    }
                }
            }
        }
    }
```

Fig. 9. Accept Loan

VII. Conclusion

In this paper, we presented a decentralized online gaming platform. The smart contract of the gaming platform provides security for the players as well as the gaming platform itself. This means that all stakeholders are ensured of fairness of the rules of the games, as well as fairness regarding their account balances and winnings. Furthermore, we propose lending functionalities in this gaming platform. This allows players to engage in games using borrowed tokens in addition to playing with their own tokens. We included the implementation details of a prototype of the platform as a Decentralized Application (DApp) on the Ethereum blockchain. The complete source code of the prototype is available on GitHub.

Acknowledgment

Wisnu Uriawan wishes to acknowledge the MORA Scholarship from the Indonesian government and the LIRIS laboratory, UMR 5205 CNRS, INSA de Lyon, which partially supports and funds his PhD work.

References

[1] Funtoken.io. [Online]. Available: https://funtoken.io/about/
[2] Atari token whitepaper. [Online]. Available: https://atarichain.com/documents/atari-white-paper.pdf
[3] Coinpoker whitepaper. cryptocurrency based online poker room. [Online]. Available: https://coinpoker.com/downloads/coinpoker-whitepaper.pdf?v180126
[4] Betu platforms whitepaper. [Online]. Available: https://betu-1.gitbook.io/betu/
[5] Lotto. the immutable world lottery. [Online]. Available: https://www.lotto.finance/
[6] Dotmoovs whitepaper. [Online]. Available: https://docsend.com/view/pjamgzxazgmbwt24
[7] Exeedme whitepaper. a skill-based blockchain economy where developers and organisers create competitive game ecosystems for all gamers. [Online]. Available: https://docs.google.com/presentation/d/1WTOdWQ50lcjd4BPg0_AWJ_i_S2FWsy1UxxR0GZsJtT0/edit#slide=id.ga55150f730_0_7
[8] Wagerr. blockchain secured sportsbook. [Online]. Available: https://wagerr.com/
[9] A. Flynn. What are examples of probability games? [Online]. Available: https://greedhead.net/what-are-examples-of-probability-games/
[10] Probability games. [Online]. Available: https://ablconnect.harvard.edu/book/probability-games
[11] React. [Online]. Available: https://reactjs.org
[12] A. Bhalla, S. Garg, and P. Singh, "Present day web-development using reactjs," *International Research Journal of Engineering and Technology*, 2020. [Online]. Available: www.irjet.net
[13] Solidity. [Online]. Available: https://docs.soliditylang.org/en/latest/
[14] Hardhat. ethereum development environment for professionals. [Online]. Available: https://hardhat.org
[15] Ganache, one click blockchain. [Online]. Available: https://trufflesuite.com/ganache/
[16] Truffle. sweet tools for smart contracts. [Online]. Available: https://trufflesuite.com/
[17] Ethers. ethereum development environment for professionals. [Online]. Available: https://docs.ethers.io/v5/
[18] web3.js. ethereum javascript api. [Online]. Available: https://web3js.readthedocs.io/en/v1.7.3/
[19] Metamask. a crypto wallet & gateway to blockchain apps. [Online]. Available: https://metamask.io/