Implementing Smart Contracts for Online Gambling on the Computecoin Network

Richard Samoilenko
Columbia University
New York, NY
rs4094@columbia.edu

***Abstract*—The emergence and explosion of blockchain technology has made it easier than ever for consumers to buy, sell, and trade virtual assets on their blockchain of choice. This allows for the implementation of decentralized applications (DAPP's) on blockchains for executing certain actions based on any transactions sent by a user to its respective smart contract. One of the applications of these smart contracts is to create nodes that facilitate gambling in games played by other nodes in the network (i.e. smart contracts are the respective games such as blackjack, roulette, etc.). This concept is examined more closely in this work, and a smart contract for playing a simple game of roulette is proposed. The contract is deployed and tested on the Computecoin (CCN) network. The definition of the contract, its novelty and implications, as well as some example interactions and test cases are demonstrated. A discussion about the validity of smart contract technology as a means for facilitating online gambling can be viewed as well.**

I. Introduction

Cryptocurrency and blockchain have surged in popularity over the past decade, introducing sweeping fundamental changes to how the average consumer values currency, digital assets, and various industries. The implementation of smart contracts on Ethereum blockchains, which are programs stored in nodes on the blockchain that can be executed by users initiating transactions, have revolutionized the way that we exchange and trade goods for services online. Smart contracts provide users with the opportunity to develop efficient, automated, transparent, and secure applications for their needs, otherwise known as decentralized applications (DAPP).

The possible use cases of DAPP's vary widely, and one can see a distribution of these applications, the number of active users, and the volume of transactions executed on these applications in Figure 1. One can observe that of all the different sectors encompassing the DAPP ecosystem, the gambling sector has the most active users and transactions performed. The reasoning for this is quite simple in that the ability to gamble with one's money at the comfort of their own home has made an already successful and lucrative market skyrocket in popularity. As with all things, this comes with its drawbacks, as a lack of checks and balances of a user's age, mental health, and financial situation makes the barrier to entry for these applications even lower than previously imaginable. The sector of gambling via DAPP's can be broken down further into several types of games, as can be seen in Figure 2. It is important to mention that the subsectors of betting and casino games only increase over the years. The obvious popularity of these games and the need for fair, robust, and safeguarded gambling DAPP's are some of the motivating factors for this work.

In this work, a preliminary implementation of a gambling DAPP is proposed. The smart contract itself is that of a Roulette Wheel where a user can place bets on the color that the roulette ball lands on within the wheel. Players that correctly guess the outcome are rewarded by receiving their original bet a payout proportional to the rarity of the color (in traditional roulette, there is one green, 18 black, and 18 red tiles). The specific methods associated with this contract will be explored in further detail in subsequent sections of this work. This smart contract is deployed on the Huygens_dev development blockchain which supports transactions and Web3 applications using Computecoin (CCN). Additionally, some example interactions and test use cases for this smart contract are explored. This report concludes with a wholistic discussion on the practicality, feasibility, and morality of using DAPP's for gambling.

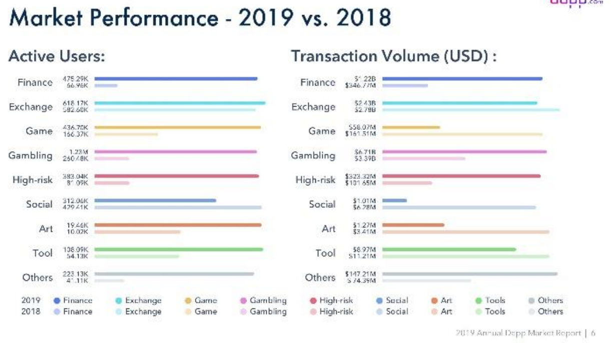


Figure 1: Distribution of DAPP's over different sectors including finance, exchanges, and gambling

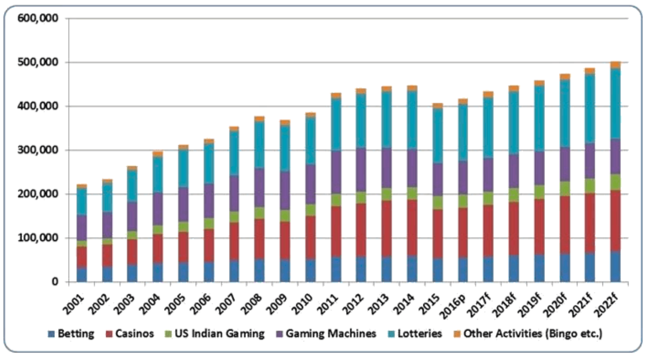


Figure 2: Breakdown of different types of blockchain games showing the rise in popularity of betting and casino games (gambling)

II. Related Work

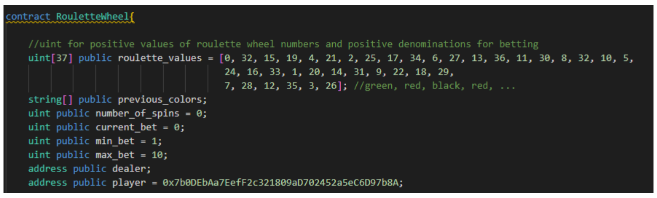
As it can be seen from Figure 2, there are several categories of gambling DAPP games that a user can choose to interact with. Scholten et al. compiled a study of the various types of Ethereum crypto-games. The type of DAPP that most commonly comes to mind when thinking about gambling is an iGaming DAPP, which is any form of gambling found in traditional casinos that can be done online (i.e. online poker, roulette, etc.). Other kinds of games, such as Gacha games that allow a user to exchange currency for random virtual items in games, have been argued to still promote gambling among adolescents. In their extensive study of these games, the authors concluded that, even in the seemingly innocent crypto-games that incorporate some kind of in-game virtual assets or cryptocurrency, the most popular of these games still contain features that can be classified both legally and psychologically as gambling.

Due to the transparency of DAPP's, some researchers and psychologists have used gambling DAPP's as an opportunity to obtain publicly available betting data from the Ethereum blockchain to study the behavior and psychology of gamblers. Meng et al. did just that in their study on gambling behavior and risk attitudes obtained from casino blockchain data [4]. The authors were able to analyze their data from a dice rolling game through the lens of probability theory to capture risk preferences of specific cohorts of people in both financial risk and other decision-making scenarios (while preserving patient privacy due to the pseudonymous nature of the blockchain).

III. Roulette Wheel DAPP methodology

This section outlines the structure and specific methods incorporated in our proposed roulette wheel smart contract *RouletteWheel.sol*. All code seen in this report can be found in the GitHub link in the References section of this report [5].

The RouletteWheel Contract object can be seen defined in Figure 3. The values associated with the wheel are defined in the order that they appear on the wheel. Additionally, attributes specific to the rules of the game are defined so that the player is limited in actions that they can take. For instance, a player must make a minimum bet of 1 CNN and can only make a maximum bet of 10 CCN. Since we are assuming a single player is at the roulette table, we define the address of the player to be the address of the Ale wallet that has been created.

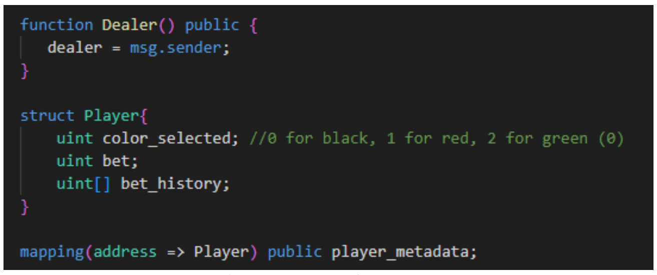


```
contract RouletteWheel{

    //uint for positive values of roulette wheel numbers and positive denominations for betting
    uint[37] public roulette_values = [0, 32, 15, 19, 4, 21, 2, 25, 17, 34, 6, 27, 13, 36, 11, 30, 8, 32, 10, 5,
                                       24, 16, 33, 1, 20, 14, 31, 9, 22, 18, 29,
                                       7, 28, 12, 35, 3, 26]; //green, red, black, red, ...
    string[] public previous_colors;
    uint public number_of_spins = 0;
    uint public current_bet = 0;
    uint public min_bet = 1;
    uint public max_bet = 10;
    address public dealer;
    address public player = 0x7b0DEbAa7EefF2c321809aD702452a5eC6D97b8A;
```

Figure 3: RouletteWheel Contract object and its associated attributes

In Figure 4, one can see the attributes that the contracts should store about both the player and the dealer (or the house/casino). These attributes can be thought as the metadata associated with the player, and they include the color that the player has selected, the player's wager for the current spin, and a history of bets that the user has made in the past. For the dealer, we should store the address associated with the contract so that the player can transfer their bet to the contract when executing the method for placing a bet.



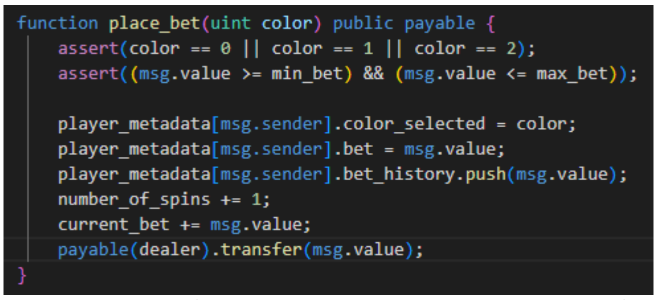
```
function Dealer() public {
    dealer = msg.sender;
}

struct Player{
     uint color_selected; //0 for black, 1 for red, 2 for green (0)
     uint bet;
     uint[] bet_history;
}

mapping(address => Player) public player_metadata;
```

Figure 4: Attributes and functions specific to the dealer (contract address) and the player

The primary interaction between the player wallet and the contract is done when the player places their bet on the roulette wheel, as seen from the function defined in Figure 5. The contract receives a block from the player address containing the place_bet method and color argument, as well as the amount of CCN sent (the bet). The method checks if the color selected is one of the three options and if the bet is within the allowed limits. Afterwards, the player's metadata is updated and a transfer is initiated to the dealer for the amount that the player has staked. The player metadata is used in the final method for determining whether they should be payed out from the result of the roulette wheel spin.



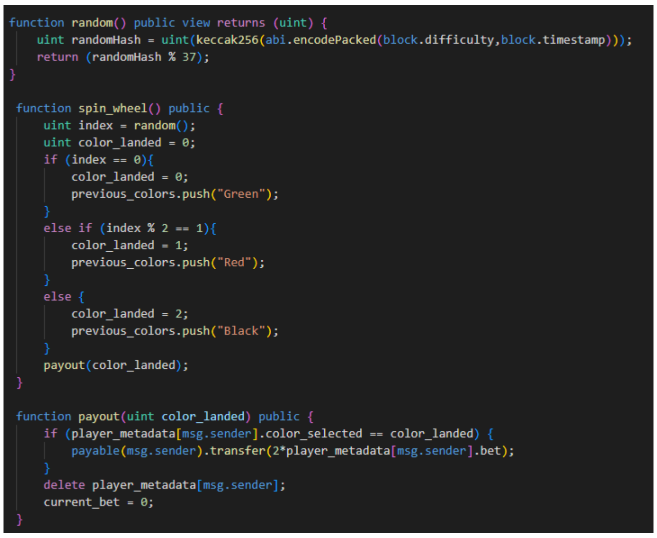
```
function place_bet(uint color) public payable {
    assert(color == 0 || color == 1 || color == 2);
    assert((msg.value >= min_bet) && (msg.value <= max_bet));

    player_metadata[msg.sender].color_selected = color;
    player_metadata[msg.sender].bet = msg.value;
    player_metadata[msg.sender].bet_history.push(msg.value);
    number_of_spins += 1;
    current_bet += msg.value;
    payable(dealer).transfer(msg.value);
}
```

Figure 5: place_bet() function that allows a user to select a color and define the value of the transaction as the bet placed on that color

Finally, a few additional methods were defined for carrying out the act of spinning the roulette wheel and paying out players if they are winners. Since Solidity does not support pure random number generation, a pseudo-random approach was used by generating a hash value for an arbitrary block. To obtain an index value on the wheel, the hash value modulo 37 is used.

When the player places their bet, the contract calls a method to spin the wheel, determining what color the ball has landed on from the pseudorandom number generated previously. Then, the payout() function is called to check if the player has selected the same color as the one that was observed from the spin. If so, then the user is payed out for their win and certain attributes (such as the current bet) are reset to 0.



```
function random() public view returns (uint) {
    uint randomHash = uint(keccak256(abi.encodePacked(block.difficulty,block.timestamp)));
    return (randomHash % 37);
}

 function spin_wheel() public {
     uint index = random();
     uint color_landed = 0;
     if (index == 0){
         color_landed = 0;
         previous_colors.push("Green");
     }
     else if (index % 2 == 1){
         color_landed = 1;
         previous_colors.push("Red");
     }
     else {
         color_landed = 2;
         previous_colors.push("Black");
     }
     payout(color_landed);
 }

 function payout(uint color_landed) public {
     if (player_metadata[msg.sender].color_selected == color_landed) {
         payable(msg.sender).transfer(2*player_metadata[msg.sender].bet);
     }
     delete player_metadata[msg.sender];
     current_bet = 0;
 }
```

Figure 6: The remaining methods in the RouletteWheel contract used for generating random result from roulette spin, updating attributes related to spin result, and paying out the player if he/she wins

While the previous discussion revolved around the specific contract used for creating the DAPP, there is yet to be any over view on the interface that a user interacts with for playing the roulette game. Unfortunately, this is the part of the project that I was unable to make a stride in. Due to the learning curve associated with picking up frameworks like Solidity, JavaScript, and Hardhat, and that this was work was done by a single individual, the only implementation of this game was done through testing the smart contract. More information about this can be found in later sections of the report.

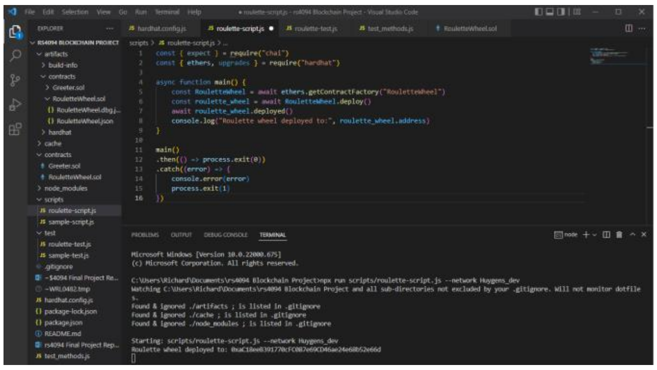
IV. ORIGINALITY/NOVELTY

This brief section includes some overview on the originality and novelty of this type of smart contract. Currently, there are hundreds of Dapps that have been deployed where users can sign up, create a new wallet for use on the DAPP, and deposit cryptocurrency from their personal wallets to the casino wallet. One example of such a DAPP can be seen in Figure 7, which is a website known as bitslot.io. This website allows users to play a variety of games ranging from slots to online table card games. With that being said, the proposed smart contract in this work is not novel by any means, but a well designed gambling DAPP has ramifications when considering user experience and safety with the application. These are concerns that will be addressed in the conclusion of this report.

Figure 7: Example of slot machine iGaming DAPP which is the most classic type of gambling DAPP

V. DAPP TESTING

Without the explicit definition of a front-end for the smart contract of the DAPP, several tests were implemented to check the functionality of the methods within the RouletteWheel smart contract. A screenshot of the script used to deploy the contract can be seen below in Figure 8. This script is standard among most Hardhat scripts that were viewed in the process of writing this contract. The contract is deployed, and the console displays the address of the contract to be used during testing and interaction.



```
const { expect } = require("chai")
const { ethers, upgrades } = require("hardhat")

async function main() {
    const RouletteWheel = await ethers.getContractFactory("RouletteWheel")
    const roulette_wheel = await RouletteWheel.deploy()
    await roulette_wheel.deployed()
    console.log("Roulette wheel deployed to:", roulette_wheel.address)
}

main()
.then(() => process.exit(0))
.catch((error) => {
    console.error(error)
    process.exit(1)
})
```

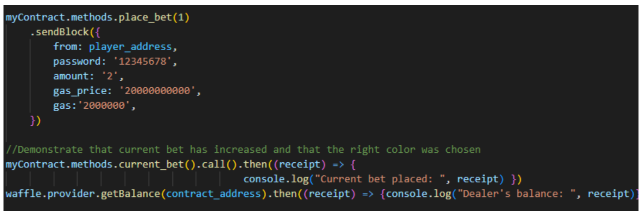
```
Microsoft Windows [Version 10.0.22000.675]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Richard\Documents\rs4094 Blockchain Project>npx run scripts/roulette-script.js --network Huygens_dev
Watching C:\Users\Richard\Documents\rs4094 Blockchain Project and all sub-directories not excluded by your .gitignore. Will not monitor dotfile
s.
Found & ignored ./artifacts ; is listed in .gitignore
Found & ignored ./cache ; is listed in .gitignore
Found & ignored ./node_modules ; is listed in .gitignore

Starting: scripts/roulette-script.js --network Huygens_dev
Roulette wheel deployed to: 0xaC18ee8391770cFC087e69CD46ae24e68b52e66d
```

Figure 8: Screenshot from script used to deploy RouletteWheel.sol

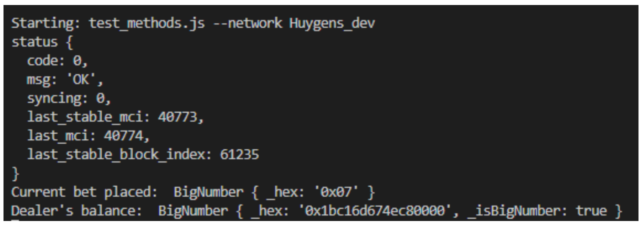
After deploying the contract, two separate scripts were created to test specific methods of the roulette wheel contract. Screenshots of sample code and output console results from executing these methods can be seen below in Figures 9 and 10. Figure 9 shows the player placing a bet on the wheel and sending a block containing their address, password, and bet amount to the address of the contract. Afterwards, the current bet of the player is checked (in the event that multiple bets are made on the same or several colors), as well as the current balance of the dealer (whose address is given by the contract address. One can observe the results of these methods from the console output in Figure 10. The code executed yields an ok message from the perspective of the contract, and one can see the hexadecimal values for the current bet placed and the balance of the dealer. Inside the GitHub repository, one can find more code to execute other functionality such as determining a winner via spinning the wheel and paying out the player if they have won.



```
myContract.methods.place_bet(1)
    .sendBlock({
        from: player_address,
        password: '12345678',
        amount: '2',
        gas_price: '20000000000',
        gas:'2000000',
    })

//Demonstrate that current bet has increased and that the right color was chosen
myContract.methods.current_bet().call().then((receipt) => {
                                        console.log("Current bet placed: ", receipt) })
waffle.provider.getBalance(contract_address).then((receipt) => {console.log("Dealer's balance: ", receipt)
```

Figure 9: Excerpt from



```
Starting: test_methods.js --network Huygens_dev
status {
  code: 0,
  msg: 'OK',
  syncing: 0,
  last_stable_mci: 40773,
  last_mci: 40774,
  last_stable_block_index: 61235
}
Current bet placed:  BigNumber { _hex: '0x07' }
Dealer's balance:  BigNumber { _hex: '0x1bc16d674ec80000', _isBigNumber: true }
```

Figure 10: Console output from executing code in Figure 9

VI. Conclusion

This work demonstrated a simple implementation of a gambling DAPP as a roulette wheel smart contract that, given some front-end implementation, can be one of many features incorporated in a DAPP casino. The current framework of smart contracts deployed within a blockchain makes it feasible to support such a system of online gambling (as evidenced by the number of gambling DAPPS that already exist).

With that being said, there are still a number of things that need to be addressed before society can adopt and standardize the fair use of decentralized applications for online gambling. For instance, the stability of altcoins other than USD stable coins will be necessary to ensure that players are not suffering losses when choosing to play on DAPP's that restrict their choice of cryptocurrency.

Additionally, more safeguards have to be implemented to ensure that the lawful and ethical demographic of individuals are being targeted for these gambling DAPP's. Some kind of location based or pseudo-identity-based measures should be used to verify that a player's wallet belongs to someone that is of age to be gambling in that location.

The final area of concern for these applications is with security. Given that these gambling applications are decentralized, there is ultimately nothing stopping such a force from pulling out all currency stored by a user on the application's wallet, otherwise known as a "rug pull". Some level of centralization may be necessary to achieve security and confidence within the player of the games, and it may also assist in the prior concern related to verifying the age of prospective users.

References

[1] "Dapp.com 2019 annual dapp market report," *By Dapp.com*. [Online]. Available: https://www.dapp.com/article/dapp-com-2019-annual-dapp-market-report. [Accessed: 11-May-2022].

[2] LunarCrush, "The billion dollar industry crypto's been waiting for," *Medium*, 30-Jan-2020. [Online]. Available: https://lunarcrush.medium.com/the-billion-dollar-industry-cryptos-been-waiting-for-48be2ae2c4cd. [Accessed: 11-May-2022].

[3] "Ethereum crypto-games - White Rose University Consortium." [Online]. Available: https://eprints.whiterose.ac.uk/150151/1/fp6475_scholtenA.pdf. [Accessed: 11-May-2022].

[4] J. Meng and F. Fu, "Understanding gambling behaviour and risk attitudes using cryptocurrency-based casino blockchain data," *Royal Society open science*, 21-Oct-2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7657926/. [Accessed: 11-May-2022].

[5] https://github.com/rs4094/rs4094_blockchain_project