Rock Paper Scissors on Ethereum

Introduction

We are going to implement the popular Rock-Paper-Scissors (RPS) game on the blockchain. In case, you don't know about this game, just search on youtube or watch this video: https://www.youtube.com/watch?v=2dsHuU10udY

Paper beats Rock since it can cover the rock
Rock beats Scissors since it can smash the scissors
Scissors beats Paper since it can cut the paper
Draw if both pick the same

Hence, there is a circular winning structure. While this is just a game, it highlights important concepts which are used in various applications on the Blockchain like DeFi, Bidding, Trading, Turn-based actions, and more.

The basic idea is that you can play RPS in person since the actions can be synchronized. Moreover, if someone cheats, you can have a third party validate and judge. Now there are multiple apps allowing people to play this or any similar game online but you are then trusting a third party - the app provider.

This project is as much about understanding what's going on, as it is about making relevant changes to complete the game. Your goal should be to reach a basic familiarity with Solidity while understanding the following two concepts.

There are two core concepts to learn here which apply to various domains in blockchain:

1. Proof without knowledge: How do you prove that you picked something without having to tell the other person what you picked?
    a. Let's say A and B are playing a game of RPS. One simple way for A to start is to just store its choice on the blockchain, against B. But then B could read A's choice first as the data is ultimately open on the public blockchain.
    b. An improvement would be to store the hash (let's say SHA256) of the choice. Although this is an improvement, B can still pre-calculate hashes of Rock, Paper, and Scissors, and match the hashes against what A added.
    c. The solution to provide proof without knowledge is for A to take a hash of their choice and a random string that they can generate without sharing.This way B cannot find out about A's choice. The game would then wait until both do this. After this, both will have to call another function to upload their actual choice and the random string. The contract would verify that the hash matches to ensure that both gave the correct hash earlier.

d. This forms the basis of a lot of interaction online which requires valid proof without knowledge between various entities, on blockchain and even in general.

2. How can you block someone from changing their choice arbitrarily (cheat) without the involvement of a third party as a 'judge'
   This is accomplished by storing the choices, first in hashed format and then raw, on the blockchain. The blockchain's internal decentralized and immutable characteristics automatically take care of disallowing cheating by any party without the need for trust.

Housekeeping points

- This is a minimal example and may not follow some standard practices
- We'll leave out some obvious tasks like allowing multiple games between two players, ensuring different blocks between proof and choice storage, etc.

Program Structure and Organization

RPSServer contract stores various games, each between two opponents. Each game is an instance of **RPSGame** contract. You'll deploy the RPSServer contract which will automatically bring in code of RPSGame. You can say that RPSServer is composing various RPSGame instances instead of using inheritance.

Every game will have an initiator and a responder. The nested mapping in RPSServer is from initiator to a mapping of responder to the game instance -
mapping(address => mapping(address => RPSGame))

Any action in the game could be done only by the initiator or responder, by presenting the address of the other party. RPSServer acts as the communication channel for external accounts and uses the core logic in RPSGame instance to actually store and play the game.

The process for a single game flows something like this:

1. The initiator A calls *initiateGame* with responder address and hash proof of its choice. **The hash has to be in a particular format (see information in Program instructions below).** This creates a new RPSGame instance and sets appropriate information. It also sets the game state to be RPSGameState.INITIATED.
2. The responder B calls *respond* with initiator address and hash proof of its choice. This sets the game state to be RPSGameState.RESPONDED
3. Both A and B now call *addInitiatorChoice* and *addResponderChoice* respectively to record their choices and random strings for validation. Choices should be a number: 1 for Rock, 2 for Paper, 3 for Scissors.

4. The game on the last of the above two calls automatically does the following:
    a. Validates that hashes match raw data for both A and B
    b. Changes the game state to RPSGameState.WIN or RPSGameState.DRAW if one or both validations fail respectively. It can also update the comment with relevant information about what happened.
    c. If both validations pass, apply the actual game logic to figure out if it's a draw or win, and assign the address of A or B to the winner field. It can also update the comment with relevant information about what happened.
    d. In case of a DRAW in either point b or point c, the winner should be set to address(0)
5. Both A and B can call *getInitiatorResult* and *getResponderResult* respectively to check the result.
6. For all of these interactions, corresponding functions in RPSGame are called.

Problem Statement

The program structure is already set and there are specific methods that you are expected to implement or complete. Please also read the comments in the code, especially in the methods to be implemented

1. Complete the __validateAndExecute function in RPSGame. The initial part of calculating hash and comparing it with stored values is already there. You need to do the following:
    a. If at least one attempt is invalid, complete the game accordingly. DRAW if both are invalid, WIN if one is invalid. Winner address should be address(0) in case of DRAW
    b. If both attempts are valid, compare choices based on game logic and assign DRAW or WIN accordingly. Winner address should be address(0) in case of DRAW
2. Implement relevant checks. You can directly use *require* in the functions or write specific modifiers and use them
    a. Add checks for ensuring that zero address (address(0)) is not passed in all appropriate functions in RPSServer. Also, ensure that the caller is not passing its own address as the opponent.
    b. Add checks in addInitiatorChoice and addResponderChoice in RPSServer, to ensure that the choice is between 1-3.
    c. Add checks in addResponse, addInitiatorChoice, addResponderChoice, and getResult in RPSGame to ensure that there are at the right game state to take those actions.

Evaluation Rubric

Total Project Points: **240**

- Basic compilation without errors (**10%**) : **24 Points**
- Correctness:
    - Problem statement - 1.a (**30%**) : **72 Points**
    - Problem statement - 1.b (**30%**) : **72 Points**
    - Problem statement - 2.a (**10%**) : **24 Points**
    - Problem statement - 2.b (**10%**) : **24 Points**
    - Problem statement - 2.c (**10%**) : **24 Points**

Program Instructions

1. Go to https://remix.ethereum.org, create a new file in the contracts folder, name it RPSServer.sol and load up the initial code given in the RPSServer.sol file, given along with this doc.
2. Your goal is to modify this file to accomplish solving Problems 1 and 2.
3. Use 0.8.6 solidity compiler version as already specified in the file.
4. Deploy the RPSServer contract.
5. You can use the local 'Javascript VM' itself to try this out. There are multiple accounts already available so you can use them to test the game.
6. Please don't change existing functions unless needed to solve the problems.
7. You might need to concatenate strings to generate appropriate game comments. This is one way to do that in Solidity:
   string(abi.encodePacked(string1, string2, string3))

8. **Please note the details in this point carefully.** To generate the initiator and responder hashes to enter while calling the appropriate functions, you can use the following python code separately:

   import hashlib
   choice_str = <choice> + '-' + <random string>
   '0x' + hashlib.sha256(choice_str.encode('utf-8')).hexdigest()

   Examples:

   choice_str = 'Rock' + '-' + 'foobar'
   '0x' + hashlib.sha256(choice_str.encode('utf-8')).hexdigest()
   '0x4acf6c068d6a4e38a89657b7aab229d82339aab96739e59b3ab1c525a9c1dae1'

```
choice_str = 'Scissors' + '-' + 'barfoo'
'0x' + hashlib.sha256(choice_str.encode('utf-8')).hexdigest()
'0x241372170e4fd374c50021b5d36bd9ccf0ba6052da75ea326a2afdc33c80f88c'
```

Choice (case-sensitive) should be one of - ‘Rock’, ‘Paper’, Scissors’

Please note that you should take the output, without the quotes but with the 0x, to enter as input in the solidity function. Example:
0x4acf6c068d6a4e38a89657b7aab229d82339aab96739e59b3ab1c525a9c1dae1

References

1. https://remix.ethereum.org
2. https://www.youtube.com/watch?v=2dsHuU10udY
3. https://docs.soliditylang.org/en/v0.8.6/