Sobre o Schelling

Dessa vez eu pesquisei um pouco mais para calcular o SHA256. Acontece que dá pra gerar o mesmo SHA256 do Solidity usando o module `ethers`. Portanto, não tem mais o problema de gerar uma hash por meio de uma `pure function` no contrato inteligente e correr o risco de que a informação que gera esse hash seja vista.

Também pensei em fazer o Schelling como se fosse um Kleros simples. O contrato é construído como um "julgamento" de um "caso" e os votantes votam sim e não dependendo do que está descrito no caso. Por exemplo, o deploy do contrato é feito com a pergunta "O contratante da empresa ACME deveria pagar o seguro?". Os participantes então votam sim ou não para esse caso. Fiz de um jeito que cabe ao owner do contrato decidir até quando podem votar.

Código do Schelling:

```
//SPDX-License-Identifier: Unlicense
pragma solidity ^0.8.0;
import "./CommitLib.sol";
import "hardhat/console.sol";

contract Schelling {
    using CommitLib for CommitLib.CommitType;
    uint256 private prize;
    address private owner;
    string private votingCase;
    uint256 private yesVoters = 0;
    uint256 private noVoters = 0;
    enum PossibleVotes {
        yes,
        no
    }

    enum RevealingState {
        waiting,
        canReveal,
        finished
    }

    RevealingState private state;
    PossibleVotes private majority;

    struct Participant {
        CommitLib.CommitType sc;
    }

    mapping(address => Participant) private participants;

    constructor(string memory _votingCase) payable {
        owner = msg.sender;
        prize = msg.value;
        state = RevealingState.waiting;
```

```
        votingCase = _votingCase;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "you are not the owner");
        _;
    }

    // coletar os votos
    function commit(bytes32 h) public returns (bool) {
        require(state == RevealingState.waiting, "it's not time to commit");
        participants[msg.sender].sc.commit(h);
        return true;
    }

    // owner coloca estado como "canReveal"
    function setRevealingState() public onlyOwner {
        state = RevealingState.canReveal;
    }

    // owner coloca o estado como "finished"
    function setFinishedState() public onlyOwner {
        state = RevealingState.finished;
    }

    // particiapntes revelam os votos
    function reveal(string memory nonce, uint256 val) public {
        participants[msg.sender].sc.reveal(nonce, val);
        if (participants[msg.sender].sc.value == 1) {
            noVoters += 1;
        }
        if (participants[msg.sender].sc.value == 0) {
            yesVoters += 1;
        }
    }

    //função que checa quem é a maioria
    function setMajority() public returns (PossibleVotes) {
        require(state == RevealingState.finished, "it isn't finished");
        require(yesVoters != noVoters, "tie");
        if (yesVoters > noVoters) {
            majority = PossibleVotes.yes;
            return PossibleVotes.yes;
        } else {
            majority = PossibleVotes.no;
            return PossibleVotes.no;
        }
    }

    // função view para saber o resultado
    function getMajority() public view returns (PossibleVotes) {
        require(yesVoters != noVoters, "tie");
        if (yesVoters > noVoters) {
            return PossibleVotes.yes;
```

```
        } else {
            return PossibleVotes.no;
        }
    }

    // usuários podem pegar seus fundos por essa função.
    // A distribuição poderia ser feita por meio de um array, mas isso custaria
muito gás.
    // então cada um que votou tem que pedir sua recompensa para não ficar caro
em gás.
    function claimReward() public {
        require(state == RevealingState.finished, "it isn't finished");
        require(
            participants[msg.sender].sc.verified == true,
            "commit is not verified"
        );
        require(
            participants[msg.sender].sc.value == uint256(majority),
            "vote is not same as majority"
        );
        if (majority == PossibleVotes.yes) {
            uint256 sendValue = prize / yesVoters;
            payable(msg.sender).transfer(sendValue);
        }

        if (majority == PossibleVotes.no) {
            uint256 sendValue = prize / yesVoters;
            payable(msg.sender).transfer(sendValue);
        }
    }

    // lucro(?) O dinheiro vai ser dividido igualmente entre participantes,
então vai sobrar um poquinho
    // vou fazer o pouquinho que sobrar voltar pro owner

    // função auxiliar para ver o valor commitado
    function seeCommit() public view returns (bytes32) {
        return participants[msg.sender].sc.showCommit();
    }

    // ESSA FUNÇÃO ABAIXO NÃO VAI SER USADA PARA GERAR O COMMIT PORQUE NÃO É
100% GARANTIA QUE O
    // COMMIT VAI SER MANTIDO EM SEGREDO
    // eu só deixei ela aqui para testar se eu conseguia gerar o mesh hash
dessa função em javacript
    function generateHash(string memory nonce, uint256 val)
        public
        pure
        returns (bytes32)
    {
        return sha256(abi.encodePacked(nonce, val));
    }

    function getYesVoters() public view returns (uint256) {
```

```
        return yesVoters;
    }

    function getNoVoters() public view returns (uint256) {
        return noVoters;
    }

    // função auxiliar para mostrar o estado atual
    function showCurrentState() public view returns (RevealingState) {
        return state;
    }
}
```

Esse contrato usa o `CommitLib` que tem pequenas modificações em relação ao `SimpleCommit`:

```
//SPDX-License-Identifier: Unlicense
pragma solidity ^0.8.0;

library CommitLib {
    enum CommitStatesType {
        Waiting,
        Revealed
    }

    struct CommitType {
        bytes32 commited;
        uint256 value;
        bool verified;
        CommitStatesType myState;
    }

    function commit(CommitType storage c, bytes32 h) public {
        c.commited = h;
        c.verified = false;
        c.myState = CommitStatesType.Waiting;
    }

    function reveal(
        CommitType storage c,
        string memory nonce,
        uint256 val
    ) public {
        require(
            c.myState == CommitStatesType.Waiting,
            "commit was already revealed"
        );
        bytes32 ver = sha256(abi.encodePacked(nonce, val));
        c.myState = CommitStatesType.Revealed;
        if (ver == c.commited) {
            c.verified = true;
            c.value = val;
```

```
        } else {
            revert("value does not match commit");
        }
    }

    function isRevealed(CommitType storage c) public view returns (bool) {
        return c.verified;
    }

    function showCommit(CommitType storage c) public view returns (bytes32) {
        return c.commited;
    }

    function getValue(CommitType storage c) public view returns (uint256) {
        require(
            c.myState == CommitStatesType.Revealed,
            "commit not revealed yet"
        );
        require(c.verified == true, "commit does not match");
        return c.value;
    }
}
```