Playing Battleship on Ethereum

OVERVIEW

The Battleship board game is played by two players who cannot see each others' board until the end of the game. The game operates on hidden information, and that hidden state influences each action taken by the players. The game is played in two phases, placement phase and attack phase. **The winner is the one that destroys all the ships of the other player.**

IMPLEMENTATION

The game is divided in different phases:

Creation/Join of a game: a user creates a game by choosing the **board dimension, amount of eth that is willing to play and the number of ships to place on the board.** A user can join a game **randomly or by having a game Id.** In the contract we store a data structure that contains information about all created and running games. It's a list of structs (STRUCT GAME).

Agree on an amount of eth: after a player joins a created game, there's a phase where these two players need to agree upon a certain amount. **When the game is created, the creator decides a certain amount that the opponent can accept or advise for another amount and so on and so forth.** After agreeing a certain amount of eth, this amount **needs to be sent to the contract.**

Placement phase: this phase involves placing "K" ships, a number chosen by the creator of the game, on the board. After placing the ships a **merkle root is calculated and sent to the contract that will store it. Check "Merkle root calculation".**

Attack Phase: this phase involves choosing a cell from the board and attacking those coordinates in the opponent board. The opponent will receive and event that someone is attacking him and needs to **answer with the result of the attack and give the merkle proof of it.** The contract has the job to check if the merkle proof is correct, **so if it generates a wrong merkle root, the game ends declaring the winner as who sent the attack.**

Payment Phase: the game can end in different ways:

1) one of the players has won the game

2) one of the players has failed the merkle proof
3) one of the players didn't play for more than 5 blocks (Timeout).

MERKLE PROOF AND ROOT CALCULATION

The hash function i used is keccak256.

The merkle root is calculated by doing the **hash of every cell + a random value** for the first level, then going up and composing the lower levels using the **XOR operator.** I used the XOR operator rather than using a function like abi.encodePacked, that is used to compose two bytes32, because the XOR **ignores the order of the operands and this helped in doing the merkle proof.**

The merkle proof is done by the contract, it checks if the combination of the hashes generates the merkle root already stored in the blockchain.

GAS COST FOR EACH FUNCTION

gas price = 0.00000002 eth

CREATE GAME:GAS USED: **144797** - TOTAL PRICE = 0.00289594 eth

JOIN GAME RANDOM: GAS USED: **90415** - TOTAL PRICE = 0.0018083 eth

JOIN GAME ID: GAS USED: **64370** - TOTAL PRICE= 0.0012874 eth

DECIDE AMOUNT (AmountCommit): GAS USED: **75033**- TOTAL PRICE= 0.00150066 eth

ACCEPT AMOUNT (AcceptedAmount): GAS USED: **58563**- TOTAL PRICE= 0.00117126 eth

SEND ETH TO CONTRACT (SendEther): GAS USED: **51737** - TOTAL PRICE= 0.00103474 eth

ATTACK OPPONENT (AttackOpponent): GAS USED: **36734** - TOTAL PRICE = 0.00103474 eth

MERKLE PROOF ATTACK (MerkleProofAttack): GAS USED: **41678**- TOTAL PRICE = 0.00103474 eth

All these values are taken from the ganache tool.

GAS COST FOR 8x8 BOARD WITH 1 SHIP

attack opponent * 64 = 36734 * 64 = **2350976** = TOTAL PRICE = 0.04701952 eth

merkle proof * 63 = 41678 * 63 = **2625714** = TOTAL PRICE = 0.05251428 eth

VULNERABILITIES

A function that can be exploited is the **accusation function**, **a player can "notify" the opponent 5 times in a row without giving it the time to respond, in this way he can win the bet amount.**

To fix this we can save the **latest block number** in which the notify transaction was seen, if the **latest block number** is equal to the **new block + 1 - 2 - 3** then the notify function is **reverted**. This can work in a production environment where blocks are mined quickly.

Another vulnerability is the infinite waiting that can be achieved during the creation of the game, this can be fixed by making the notify option available since the creation of the game.

PROBLEMS WITH EVENTS

I have some problems with certain events received twice, in this link: https://github.com/web3/web3.js/issues/398 there's an explanation of why this happens. "can be chain reorganizations because your private network is small. Therefore they happen more often. This means that one peer is mining a block and creating the event, then another one is doing the same block and your node suddenly accepts this new block as valid and makes the other a uncle, so it fires the event again."

MAKE SURE TO REJECT THE SECOND TRANSACTION THAT POPS UP IN METAMASK DURING THE MERKLE PROOF, YOU CAN SEE THIS AT MINUTE 2:02 IN THE DEMO VIDEO.

INSTRUCTIONS

1) **truffle migrate – deploy the contract**
2) **npm install – install dependecies**
3) **npm run dev – run the application**