P2P Systems and Blockchains
Final Project - Academic year 2022/2023
Hit and Sunk!
Playing Battleship on Ethereum

1. Project Description

2 giocatore che vedono solo la propria board, questo influenza le scelte che essi faranno.

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.

fase piazzamento, piazza k navi, di dimensione costante in una matrice

The game is divided into two phases: during the *placement phase*, each player places k ships of varying lengths and of constant width on their board, a *n X n matrix* which represents a coarse discretization of the ocean. After the first phase, the game proceeds to the *shooting phase*, which consists of players taking turns and making guesses about the location of the ships on the opponent's board (referred to as *launching a torpedo*). The guess consists of telling the opponents the coordinates *[i,j]* of a zone of the board. If any of the opponent's ships are at that location, the opponent replies "*Hit!*", otherwise "*Miss!*". Once all the squares that the ship occupies have been hit, the ship is considered "*sunk*".

The *termination of the game* happens when one of the players has sunk all of their opponent's ships and that player wins the game. Fig. 1 presents a Battleship game, with the view of the two players.

A battleship game may be played on the Internet and is normally hosted on a third-party centralized server. However, in a centralized server scenario, the players must rely on the information coming from the server, since it acts as a mediator. If the server is not trusted, it may decide to send the wrong information to the players, and this is particularly critical if the game involves a reward for the winner, since an untrusted server may tamper with the player's actions in order not to transfer the money to the winner.

Tamper significa modificare qualche inforamzione dell'utente per vantaggi personali, senza che nessuno possa provare il contrario.

The project requires the implementation of the Battleship game on the Ethereum blockchain, so that the properties of the blockchain, i.e. tamper-freeness, the possibility of auditing the fair rules of the game, encoded in smart contracts, instant and secure reward payments, and secure rewarding implementation, can be exploited. Furthermore, implementing the game on a permissionless blockchain implies that participating in the game cannot be prevented from anyone, since there is no censoring authority.

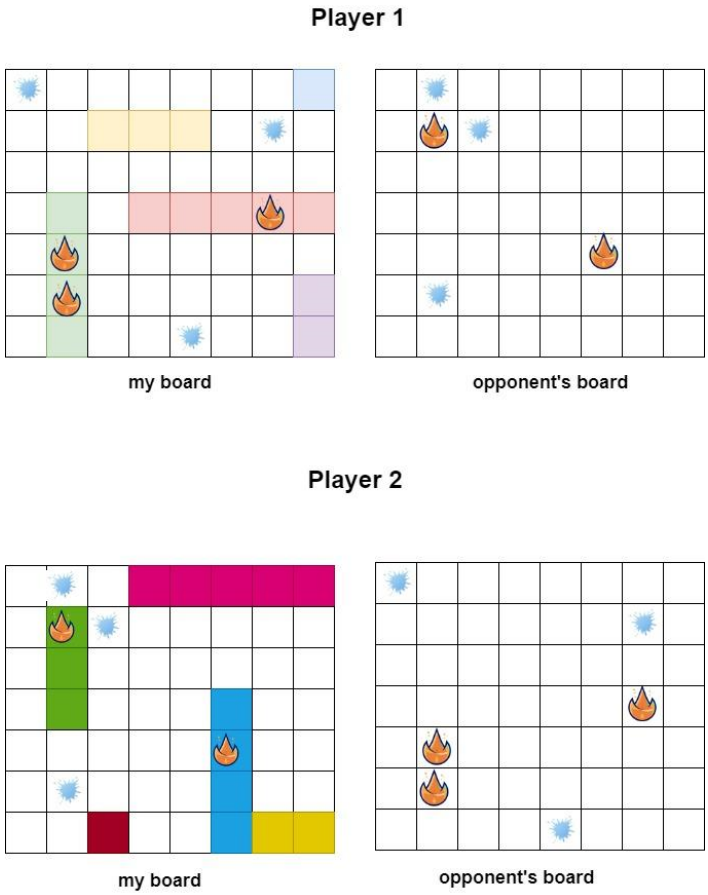

Figure 1: the player's views

2. Hit and sunk: implementation on the blockchain

Since the Blockchain maintains a public ledger, a player cannot publish the content of its board on the ledger, otherwise, an adversary can read the Blockchain and always hit the public locations of their adversary. On the other side, the blockchain is exploited:

1. to guarantee that the initial choice of the ship locations made in the placement phase is not modified by a cheater after the placement phase; board of users can't be published in the blockchain
2. to guarantee that the initial placement of the ships is valid;
3. to guarantee, at each turn, that a player does not cheat about the exact outcome of the previous torpedo shot made by their adversary (i.e. hit or miss);
4. to implement a rewarding system for winning players, through some tokens or cryptocurrency stored in the smart contract balance;
5. to define a penalty mechanism for cheating players and for players taking no action at a particular turn, so freezing the deposit of money in the smart contract.

Placing the battleships Calcolo del merkle tree delle posizioni, dopodiché salvarsi la merkle tree root nel contratto stesso, cosi chiunque veda il contratto può testare che la posizione delle battleship non è stata modificata, alla fine quando la battleship è stata mostrata.

To fulfill requirement 1., a player must place the ships on the board and then they must commit the board on the blockchain, without exposing the location value of the ships. The commit consists in computing a *Merkle tree* and storing in the smart contract the root of said Merkle tree.

Assume player P is playing with an N x N board, with N^2 cells in total, and places k ships on the board. To keep it simple, consider N equal to a power of 2. Consider a row-by-row linearization of the cells of the 2-dimensional matrix. For each cell of the board, a leaf node L_i, $i \in [1, N^2]$, of the Merkle tree is created, which stores the value $v_i = H(s_i || r_i)$, where s_i is the state of the i-th cell of the board and r_i is a random salt, different for each cell. The state of the cell may be 1, denoting the presence of a ship in that cell of the board, or 0, denoting the empty cell. Figure 1 shows the Merkle tree for a 2 X 2 matrix. done by the frontend

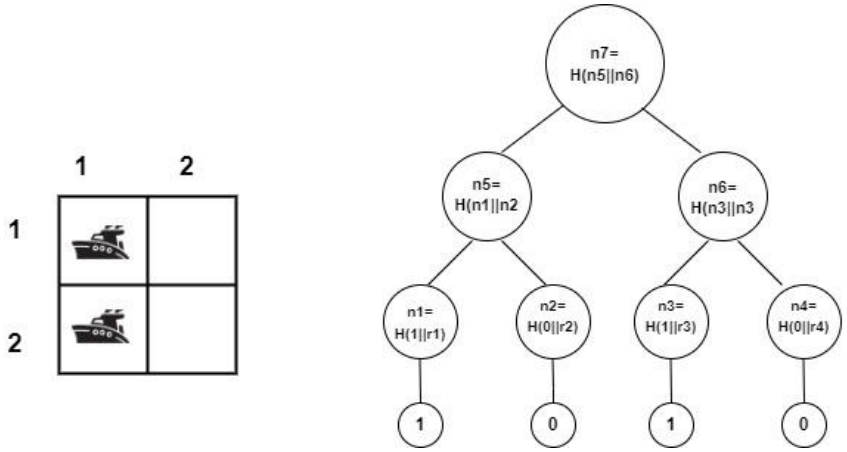

Figure 2: Merkle Tree - Committing Board status

Launching torpedoes

The second phase of the game consists of each player taking turns launching a torpedo targeted to a coordinate of their opponent's board. Each player is then notified of whether their shot has hit or missed. Let us suppose it is Player 1 turn that selects the coordinates to shoot and sends them to the smart contract. The smart contract stores the target coordinate for Player 2 to read. Player 2 reads the target coordinate, and checks if the shot has hit a ship. Then, the Player generates a *Merkle proof* demonstrating that their secret board configuration satisfies the hit/miss result they claim and that their board configuration aligns with the Merkle Root that is stored in the smart contract. The smart contract then receives this information from Player 2 and runs the verification method on the proof. If the verification returns true, it stores the shot result in its game state, along with all of the shot results already stored from the previous turns. Player 1 will be able to view the result of their shot from the smart contract.

parte un evento al secondo giocatore.

Player 1 attacca -> invia coordinate a smart contract -> smart contract avvia coordinate a Player 2.
Player 2 genera Merkle Proof per quella coordinata, dicendo che ha missato oppure no.
Player 1 attacca -> se l'invio va a buon fine allora deve aspettare il proprio turno -> si mette in ascolto su un evento.

Declaring a winner

Players take turns shooting until one player manages to sink all of their opponent's ships. The smart contract can easily check this win condition after every turn by counting the total number of hit coordinates for every player and checking if it equals the total number of ship coordinates of the game. Once this happens, the winner sends their board to the smart contract that checks that the position of the ships on the board is valid (the ships do not intersect,...), then the smart contract updates the state of the game board to display the winner and can transfer the prize.

Penalty mechanism

At any point, a player may accuse the other player of having left: if the opponent is taking too long to make a move and it is currently their turn, for example, the player may accuse them of leaving. This may be implemented by notifying the contract that you accuse the opponent of leaving the game. This accusation may trigger an event that must be addressed by the opponent within a time limit of *5 blocks*. If the time runs out, the accuser may claim the whole reward, and the game is concluded. The opponent may make a guess, within the *5 blocks delay*, to indicate they are still playing.

3. Implementation Details

- Develop a smart contract that is able to manage a set of games simultaneously. A user can create a new game by invoking a function of the smart contract which adds the new game to a list or join a preexisting game if no one else has joined that game (battleship is a 2-player game).
- When the user creates the game, the smart contract returns a unique identifier ID of that game.
- There are two options for a user joining a previously created game:
    - The game is chosen by the smart contract, at random;
    - The creator of the game can share its ID with a friend, who can use it to join the newly-created game.
- Once the second user has joined the game, the smart contract emits an event to advise the players that the game can start. At this point, the two parties make a joint decision on the amount of value to commit to the game (both players contribute the same agreed value). The deposited value is considered a factor that enforces the players to play by the rules. In fact, any attempt to cheat in the game will result in a punishment of crediting the deposited value to the opponent.
- The smart contract stores all state information about all games, such as the address of the creator of the game, the current phase of the game, and the hashed board configurations committed on the blockchain. merkle root

- The type and size of ships must obey a predefined set of rules depending on the board size N. It is admitted to represent each ship as a rectangle of width 1 and variable height (i.e. battleships of sizes 1 × 5, 1 × 4, etc.), even if more complex ship configurations are possible.
- The main task of the front-end code will be the creation of the Merkle proofs and the interaction with the smart contract. The implementation of a graphical representation of the boards is not mandatory. It is possible to exploit a text-based representation of the board.

4. Submission

The project must be developed individually. The material to be submitted for evaluation is:

- The project implementing the Battleship game: smart contracts and front-end code.
- An evaluation of the cost of the gas of the functions provided by the smart contract. Provide also an evaluation of the total cost of the smart contracts for a game played on a board 8 X 8, assuming that each player misses all the guesses
- An analysis of the potential vulnerabilities of the contracts.
- A pdf report containing:
    - the main decisions made during the implementation of the smart contracts and of the front end;
    - a user manual with the instructions to set it up and try it;
    - the instructions to execute the demo. The demo can be either automatic or manual (in the latter case, provide a detailed description of the steps to perform).

The report and the code must be submitted electronically, through Moodle. The project will be discussed around a week after its submission. The oral exam will consist of a discussion of the project, the presentation of a short demo made on the candidate's personal laptop, and an assessment of the topics presented in the course not covered by the project. Do not hesitate to contact us (laura.ricci@unipi.it, r.almeida@studenti.unipi.it) by e-mail, we will fix a meeting on Teams or a physical meeting.