P2P Systems and Blockchains
Final Project - Academic year 2023/2024
"Break the hidden code!"

1. Mastermind: description of the game

The goal of the project is to implement the well known game *Mastermind* on the *Ethereum blockchain*.

Mastermind is a two player *code breaking game*, i.e. a game in which the objective of one player, the "*CodeBreaker*" is to figure out the code that the opponent, the "*CodeMaker*", has chosen. The codemaker *secretly selects* a code consisting of an ordered sequence of *N colors*, $CS=(c_1, c_2, \ldots, c_n)$, each chosen from a set of *M possible colors,* with repetitions allowed. The codebreaker then tries to guess the code by repeatedly proposing a sequence of *N* colors, $(g_1, g_2, \ldots, g_n)$. After each guess, the *CodeMaker* notifies the *CodeBreaker* two numbers: *CC*, the number of colors belonging to *CS* in the correct positions, and *NC* the number of colors belonging to *CS*, but not in the correct positions. The *CodeBreaker* continues guessing until they guess the code correctly or until they reach a maximum allowable number of guesses without having correctly identified the secret code.

The *CodeBreaker* and the codemaker jointly decide the *even* number of turns *NT,* of the game. In each turn, one of the players acts as *CodeBreaker* and the other as *CodeMaker*, then they swap their roles. For each turn, the *CodeBreaker* has a maximum fixed number of guesses *(NG)*.

At the end of each turn, the number of guesses that the *CodeBreaker* needed to crack the secret code is the number of points awarded to the other player, the *CodeMaker.* If the *CodeBreaker* does not end up breaking the code, *K* extra points are awarded to the *CodeMaker*. At the end of the agreed number of rounds *NT*, the scores are tallied up and the player with more points wins.

In the real world, the *CodeMaker* places the pegs corresponding to the code they have chosen behind a shield, the codebreaker places its pegs on the other part of the board. There are smaller holes on the right of the board, where the *CodeMaker* places white and black smaller pegs. Each black peg corresponds to a color correctly guessed in the correct position by the codebreaker, while a white peg corresponds to a color correctly guessed, but in the wrong position. For a better understanding, Fig.1, shows the board used in a real game, played in the physical world. Note that the gray small circles on the right are not occupied by any peg.

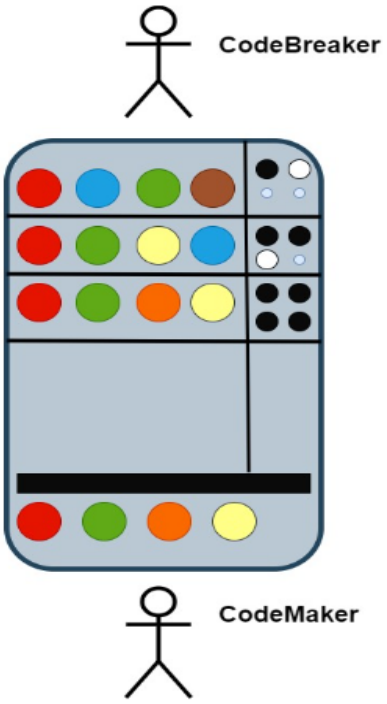

Fig. 1. A Mastermind Board

2. A blockchain based Mastermind

The goal of the project is to implement a blockchain based version of the classical MasterMind game, based on the development of a set of smart contracts written in Solidity and executed on a EVM based blockchain.

Since the Blockchain maintains a public ledger, a *CodeMaker* cannot publish the secret code on the ledger, otherwise it would be visible to the codebreaker.

On the other side, the blockchain is exploited:

- to guarantee that the secret chosen by the codemaker at the beginning of each turn is not modified later, before the *CodeMake*r discloses it;
- to check that each feedback given by the *CodeMaker* to the *CodeBreaker*, at each turn, is consistent with the secret code chosen by the *CodeMaker* at the beginning of the game;
- to implement a *rewarding mechanism* for winning players;
- to define a *penalty mechanism* for cheating *CodeMakers* and for players that take no action at a particular turn, thus preventing the advancement of the game.

In the following, we will give a detailed specification of each phase of the game.

Starting the game

The two players agree on a common value of *Wei* to be sent to the smart contract. This is called *game stake.* The total amount sent by the two players will be used to implement the rewardings and punishments.

After this, one of the two users is chosen at random by the smart contract as *CodeMaker*, and the other as *CodeBreaker*. Then, to guarantee that the secret chosen by the *CodeMaker* at the beginning of each turn is not modified later, at each turn, the player chosen as *CodeMaker* chooses the secret code and commits *its hash* on the blockchain.

We suppose that *NT,* the number of turns, and *NG,* the number of guesses for each turn, are statically defined and are the same for all the games.

Playing phase

After this step, the *CodeBreaker* starts to send their guesses to the smart contact. In turn, the *CodeMaker* sends to the smart contract their feedback about the guess (i.e. the values of *CC* and *NC*). Each guess and the feedback values of *CC* and *NC* are *registered in the smart contract.*

End of a turn

At the end of each turn (either by reaching the number allowed number of wrong guesses or by correctly guessing the secret code before the limit), the *CodeMaker* reveals the secret code and the smart contract checks that its hash is equal to that committed at the beginning of the game. After the code is revealed, the *CodeBreaker* has a time period *TDisp*, to possibly dispute any feedback returned by the *CodeMaker* during the current turn. To start a dispute, the *CodeBreaker* sends to the smart contract a reference to the guesses they are disputing. The smart contract decides if the *CodeMaker* has unintentionally or maliciously cheated, by giving a wrong feedback, or if the *CodeBreaker* has unjustly accused the other player. In the first case, the *CodeMaker* is punished by sending the game stake to the *CodeBreaker,* in the second case the other way round. In both cases the game ends. After *TDisp*, if no notification is received from the *CodeBreaker*, the smart contract computes the points earned by the *CodeMaker*.

End of the game

After *NT* turns, the game is finished, and the smart contract can establish the winner and send them all the game stake.

Away from keyboard (AFK)

At any point, a player *A* may accuse the other player *P* of *"away from keyboard" (AFW)*, i.e. the opponent is taking too long to make a move. For instance, in the initial phase of the game, the user appointed as *CodeMaker* may delay the commitment of the hash, while in the game phase, the *CodeBreaker*/*CodeMaker* may delay the guess/feedback. *P* may react

within a timeframe *T*, by performing the action required by that phase of the game (for instance if the *CodeMaker* accuses the *CodeBreaker*, the *CodeBreaker* must react by proposing the next guess). If *P* does not properly react within the time frame *T*, it is punished by sending all the game stake to *A*, and the game is over.

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 (Mastermind 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 *C* of the game can specify the Ethereum address of the user *U* they want to play with. In that case, *C* shares the game *ID* with *U*, *U* specifies the game *ID* when they want to join the game, and the smart contract verifies that *U* is allowed to join that 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 amount to commit may be simply chosen off-chain by the two users or defined by a simple protocol executed on chain by them.
- 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, the hash of the secret, and so on…
- To implement the *AWK strategy*, a player notifies the contract that they 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 *B blocks*. If the time runs out, the accuser may claim the whole reward, and the game is concluded. The opponent may make the proper action, within the *B blocks delay*, to indicate they are still playing, in this case the game goes ahead in the regular way.
- the value of *N*, *M*, *K*, and of the time slots may be chosen arbitrarily.

4. Submission Rules

The project must be developed by a group of at most 2 students. The material to be submitted for evaluation is:

- The smart contracts and the **scripts** Hardhat that allow the deployment and the interaction with the smart contracts
- It is recommended to develop smart contracts using the Hardhat framework as covered in class. Therefore, it is suggested to develop the project, conduct a testing phase to verify the behavior of the smart contracts related to the game, including error handling and specific game situations, and deploy the smart contracts using the functionalities provided by the framework.
- An evaluation of the cost of the gas of the functions provided by the smart contract
- An analysis of the potential vulnerabilities of the contracts.
- A brief pdf report containing:
    - the main decisions made during the implementation of the smart contracts:
    - a user manual with the instructions to deploy and interact with the contract

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 me (laura.ricci@unipi.it) by e-mail, we will fix a meeting on Teams or a physical meeting.