Distributed Systems Project Report

Go Ethereum

1. Idea

The main idea of this project is to implement the game Go using Ethereum's BlockChain and smart contracts.

1.1. What is Go?

Go is a strategy board game that was first invented more than 2500 years ago in China: it then became popular also in other countries such as Korea and Japan, and it developed different rules over the course of time. Nowadays Go is known and played world-wide, even though the majority of players still reside in Asia. While it is essentially a simple game to learn, it is very difficult to master, as the tree of possible moves that the players can take can quickly become huge.

1.2. How does the game work?

Go is a two player game, with each player having tokens called stones of one color (black or white). Players take turns to place their tokens on the intersections of the lines on the board. The black player plays first, and both players are allowed to pass their turns whenever they want. When the two players pass their turns consecutively, the game ends.

The objective of the game is to capture more area on the board than the opponent, by surrounding it with our stones.

2. Implementation

2.1. Client

The application is built using Truffle, a development environment for Ethereum. The interface is done using HTML5, CSS3 and Bootstrap, and JavaScript and jQuery are used to manage the backend of the application. The interaction with the smart contracts is done through Truffle's interface/API, which uses web3.js and provides additional tools to simplify the communication with the contracts.

When the user runs the application, he can use the user interface to create games, join existing games created by other users, and he can play in the games that he is part of.

2.2. Contracts

The contracts are written using the language Solidity.

We have two contracts: `Go.sol` and `GoGame.sol`. `Go.sol` is a contract that is always deployed: it is unique, and it acts as a "Factory" for the Go games: when the user wants to create a new game, a method of `Go.sol` will be called, that then creates a new instance of `GoGame.sol` and adds it to its list of existing games.

`GoGame.sol` represents a single Go game: it keeps track of the board, the players and the state of the game. When a player makes a move, its validity is checked directly on the contract, so that we ensure the fairness of the game. If the move turns out to be valid, the state of the game gets updated (we update the board accordingly and we change the turn of the player).

If instead of making a move the player passes the turn, it is again checked on the contract if it is valid. If that is the case, we update the turn of the game, and if it was the second consecutive pass (both players pass one after the other), the game ends, in which case the contract calculates who the winner is and stores that in the contract.

3. Using BlockChain

3.1. Benefits

The main benefit of developing such a game using smart contracts and BlockChain is the possibility to ensure the fairness of the game, since the logic and the validity of the moves are checked on the contract. This would not be possible if we were to implement the game using a normal client-server approach.

Another advantage is that we do not depend on the availability and reliability of a server, since the BlockChain is a distributed system.

3.2. Challenges

One of the drawbacks of working with BlockChain is that we cannot rely on the timing of the transactions, as we only have a rough idea of when the blocks will be mined, and even if we did know the exact time, it might simply be too slow to be useful for certain applications.

Also, writing smart contracts can be quite troublesome, as it is difficult to debug applications, there is no simple way to understand what is happening during the execution of the transactions, and the error messages can be quite cryptic.

Most of the technologies used are recent and still in development, too: there is not much documentation and some features might be lacking, which again doesn't make the development easier.

Finally, smart contracts' transactions cost money to be executed, which means that the code and the algorithms used have to be highly optimised to reduce the costs, and when possible it should be considered to move parts of the logic of the application outside of the contracts, to simplify the operations of the transactions.