Turn Based Game on Blockchain

Arora, Harsh MCS2022
Chakraborty, Arghyadip MCS202206
Gangamreddypalli, Namratha Reddy MCS202105
Gantait, Arunava MCS202201
Iyer, Paarth MCS202218

December 2022

Contents

1 Problem Statement

The aim of this project is to implement a simple turn-based game on the blockchain. The particular game implemented is **Antakshari**, which is played as follows:

- The first player submits a word
- At each subsequent turn, player submit a word which begins with the last letter of the previous word.

The validity of each submitted word must be checked in some way, e.g. by making calls to a dictionary maintained by some trusted source outside the blockchain. *Timing constraints* and *elimination criteria* may also be introduced. Passing on a turn may be allowed.

Finally, the game should have some consistent termination and winning criteria.

2 Main Idea

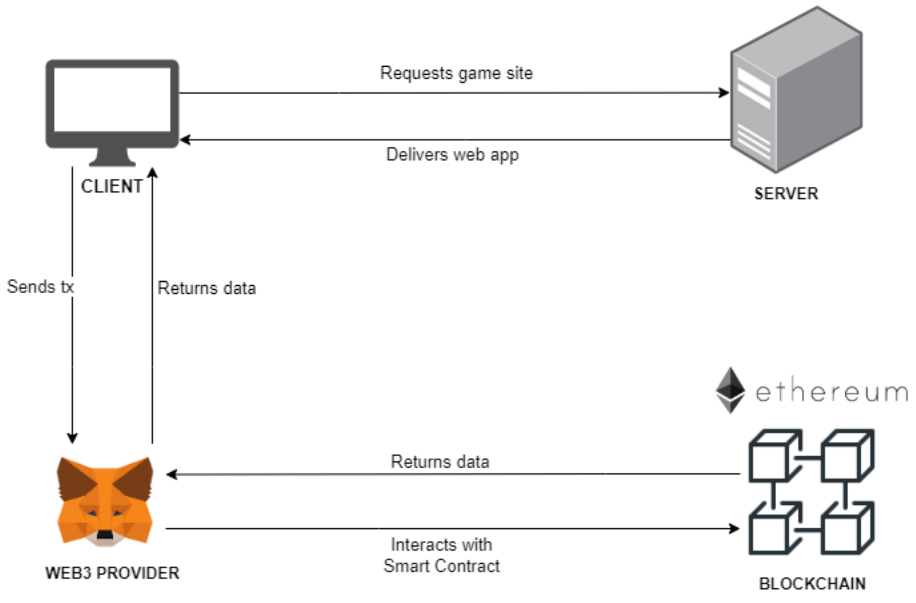

Figure 1: Architecture diagram

The client accesses the game website by entering the url in their browser with the Metamask extension. After which they play the game by using our frontend and sending transactions via Metamask, which interacts with the Ethereum blockchain, and hence our smart contracts deployed on it.

Key Components

Ethereum: It is a decentralized, open-source blockchain with smart contract functionality. Ether is the native cryptocurrency of the platform.

Smart Contract: They are the building blocks we use to create blockchain applications. They are programs that we can write with source code and deploy to the blockchain. They are written in the Solidity programming language.

MetaMask: It serves as a web3 provider and a crypto wallet, whose purpose is to interact with the blockchain. It is distributed as a browser extension.

Ethers js: It is a popular library that basically encodes and decodes any data prior to, or after interaction with the blockchain.

3 Implementation

3.1 Game features

Judge: There is a predecided judge who is responsible for approving (validating) each submitted word in the game

Validation Condition: Apart from the condition mentioned in the problem, a word is deemed valid by the judge if it has not been used previously (thus we do not allow repitition of words) and also if it comes from the dictionary implemented in `worddict.js`

Lives: We have implemented a system of lives. The issuer of the game contract (the *creator*) gets 3 lives, and any new player gets 5. A life is lost upon submission of an invalid word. Losing all lives results in elimination

Terminating/winning condition: Our terminating condition is through **last man standing**. When all but one player have been eliminated through the process described above, the remaining player is declared the winner.

3.2 Development tools

3.2.1 Truffle

The Truffle Suite: Truffle advertises itself as "the most comprehensive suite of tools for smart contract development".

How it can be used : Truffle can be used as a development environment, testing framework and asset pipeline for blockchains using the *Ethereum Virtual Machine* (**EVM**). It manages the entire workflow efficiently for any blockchain-based development. Its key features include:

- Built-in smart contract compilation, linking, deployment and binary management.
- Deployments and transactions through MetaMask to protect our mnemonic.
- External script runner that executes scripts within a Truffle environment.

How we use it : Our backend is Truffle-based. We have used it for testing our software without having to use the actual Ethereum blockchain and spending actual gas on it.

Ganache: Ganache is a tool for producing "One-click blockchains". It is a personal blockchain for developing software for Ethereum or even Corda.

How it can be used : Using Ganache enables:

- `console.log` in Solidity
- Forking any Ethereum network without waiting to sync
- Ethereum JSON/RPC support
- Impersonating any account without requiring any actual private keys

- Listening for JSON-RPC 2.0 requests over HTTPS/WebSockets
- Mining blocks instantly, on demand, or at an interval.

among other things.

How we use it : Ganache has also been used for testing, by simulating players using fake private keys and addresses and observing how our code interacts with requests and events.

3.2.2 User Interface tools

- HTML
- Svelte: It is a front-end, open-source JavaScript framework for building web applications.

3.3 Backend

3.3.1 Code

```
truffle
├── contracts
│   ├── strings.sol
│   ├── WordGame.sol
│   └── WordGameFactory.sol
├── migrations
│   └── 1_deploy_contract.js
├── test
│   ├── wordgame_test.js
│   └── wordgamefactory_test.js
└── truffle-config.js
```

Files

- strings.sol:
- WordGame.sol :
- WordGameFactory.sol :
- 1_deploy_contract.js :
- wordgame_test.js :
- wordgamefactory_test.js :
- truffle-config.js :

3.4 Frontend

The frontend allows the user to create games, join existing games, look at words already played.

3.4.1 Code

```
client
└─ src
   ├─ lib
   │  ├─ Game.svelte
   │  ├─ GameScreen.svelte
   │  ├─ MetaMask.svelte
   │  ├─ Navbar.svelte
   │  ├─ PlayersTab.svelte
   │  ├─ StartJoin.svelte
   │  ├─ WaitingScreen.svelte
   │  └─ WordTab.svelte
   └─ App.svelte
```

Files

- Game.svelte :
- GameScreen.svelte : Contains the html UI elements of the game screen, like text field to enter the word, button to submit the word etc, and the code for events related to these elements.
- MetaMask.svelte : Checks if the user has installed a MetaMask plugin and connects to the users MetaMask account
- Navbar.svelte : Contains html for the navigation bar
- PlayersTab.svelte : Shows the list of players who currently joined the game.
- StartJoin.svelte : Contains buttons to Start and Join game, and their functionality.
- WaitingScreen.svelte :
- WordTab.svelte :

4 Challenges/Improvements

Could not include a timer because you have to keep track of transaction hashrate: essentially the only way to know time is when a new block is added to the blockchain. For an auction this is fine since any transaction after the time limit can be discarded. However for our game we want to ensure that every player has a time limit after which the turn proceeds to the next player and every player needs to know which player failed to answer within the time limit.

Could not include a Merkle tree due to lack of time which would have gotten rid of the judge. Instead we have used another contract (the judge).