GoBang 3.0: A Blockchain-Based Chess Game

Motivation

Blockchain, as a decentralized pattern, has raised significant attention in recent decades. Some derivative applications, such as Cryptocurrency, Metaverse, and Web3, are currently changing fields like finance, gaming, and networking in the real world. The most noteworthy effect of blockchain might be accelerating the New Creator Economy (NCE) [1]. One way to drive the NCE is by rewarding the creators with Non-Fungible Tokens (NFTs), which are unique and tradable. This is a popular way used by Metaverse-based games, where players are attracted by the potential profits when they 'contribute' to the games. Apart from games, Web 3.0 gets rid of the traditional network (Web 2.0) and enables content creation to be decentralized. The creators can share their content with anyone else in the world without relying on big corporations and are promised to get the revenue (as rewards) directly.

As experienced game players, we understand and believe that the fun of gaming is much more than just playing alone, even though that might earn you some pennies. A competitive Player-vs-Player (PvP) game would be more interesting, and if you want, you can bet on it. One example is Tic Tac Toe, which is simple but popular. We want to challenge ourselves with a more complex game, so in this project, we implement a chess game called Gobang. It will be deployed in a test blockchain and played through a Web3 interface, that is the reason we call it GoBang 3.0.

Game Rules

Gobang (a.k.a., Gomoku or Five in a Row) is a strategy board game similar to Tic Tac Toe. As its name indicates, you need to have five chess in a line, whether it is horizontal, vertical, or diagonal, to win the game. Usually, the board size is 15x15 or 19x19, but it is quite flexible as the players demand. Two players would compete in one game, one is black, and another is white. They will play in turn until one of them wins.

Developing and Testing Settings

Our game is supported by smart contracts previously deployed in the blockchain, and a webpage interface based on Web3 is developed for a better human-computer interaction experience. The blockchain we have chosen is Ethereum, considering its popularity and support for smart contracts. Another reason is its average block time of 10-14 seconds, which suits the time for each move in Gobang. Truffle, a powerful tool using the Ethereum Virtual Machine (EVM) is used to develop smart contracts. Of course, we are not likely to deploy the smart contracts in a real Ethereum chain due to the expense, thus a personal local blockchain created by Ganache is introduced to run the simulations. It provides the same functions as a real-world blockchain, except for the fact that you can not exchange ETH in it for money. The next tool is MetaMask, a popular method to interact with dApps in the front-end. It can be installed as an extension for Chrome or Firefox, which is quite convenient to manage your account and make transactions. In this project, we only test our code with Chrome, but it is also supposed to work with Firefox as well. With regard to the front-end, we implement the webpage with React, a useful open-source JavaScript library for building user interfaces for single-page applications. The README file in the project repository contains more details about how to set the environment and run our game locally.

Project Structure

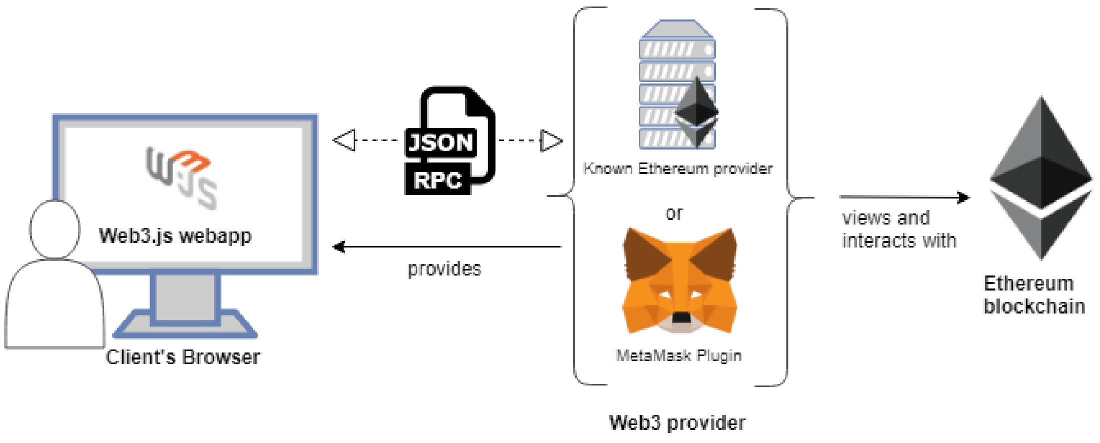

Figure 1 Structure of GoBang 3.0

Figure 1 shows a high-level structure of our project. There are mainly three layers in the structure: client (front-end), Web3 provider, and blockchain (back-end). As MetaMask only works as an intermediate layer, which does not include many implementations, we will focus on the other two layers. The front end is the webpage of our game, which includes some functional buttons like starting the game or joining the game and a Go board that serve visualization purpose. The Web3 page takes input from the user and builds the requests. The requests are then processed by MetaMask, which charges a gas fee or some betting amount from the player's account and sends requests generated by Web3 to the Ethereum blockchain.

The smart contracts are deployed on the Ethereum network in advance and deal with requests from MetaMask. There are typically three kinds of requests processed by our smart contract. One is starting or joining the game, where the player's address is recorded, and some bet may be deposited in the contract. Another type of request is each player's move, which will fit into a predefined board in the contract. The final one is the withdrawal request, which allows the winner to transfer his prize from the contract to his wallet. The first two requests will also produce a reply from the contract, either telling the player that he has successfully joined the game, or one chess is placed in some place on the board. Be careful that it is an abstract idea, while in practice, the user may still need to call some functions in the contract to get a reply.

Features Implemented

Our project is a fully functional program that can be run in a test environment. Here we are proud to introduce the core features it has, even though some of these features might be basic requirements.

- The user can start the game by paying a bet amount of 0.5 ETH, and another user can join with the same amount. A total of 1 ETH can be claimed by and only by the winner after the game.
- The player that starts the game will always take black, and his opponent will always take white. Whether black or white makes the first move is chosen randomly.

- The player can choose his next move by clicking the empty place on the board. A pop-up window from MetaMask would appear and request the player to pay a small amount of gas fee to submit his move to the contract. The player can regret that by rejecting the payment, choosing another place, and submitting the move again. Clicking on a place with chess will not work.
- After the transaction is published in the blockchain, the opponent would get the move and it would be displayed on the opponent's board.
- There are texts that suggest whether it is the player's round or the opponent's round. The winner's address would also show after the game ends.
- Moves would be stored in the contract in the form of an array. After each move, the contact checks if the player wins the game, and would automatically end the game and announces the winner if someone wins.

Future Work

We are also aware that this game is quite a prototype at the moment, and we have listed some features to be implemented in the future if the game would be placed on the real blockchain.

- The bet amount is hardcoded to be 0.5 ETH now. It should be flexible and can be determined by the players.
- In fact, the contract could only host one game currently, as the participant joins the default game created. Users may want to choose whom he wants to play with, so in the future, the player should be able to input an address and join a specific game.
- There is not a time limit for each move, while in practice, it is important to end the game if either player takes too long to submit a move. This problem is more complicated, as sometimes the transaction may not be published due to situations like gas fees increasing. It would be a critical task to set a proper time limit.

References

[1] Y. Özkardes-Cheung, "The New Creative Economy Within Web 3.0," Entrepreneur, 14 May 2022. [Online]. Available: https://www.entrepreneur.com/article/423776. [Accessed 26 May 2022].