Project Setup and Testing Guide

1. Install dependencies by running the following command:

```
npm install
```

2. Add Network Configuration to `hardhat.config.js`

```
networks: {
  sepolia: {
    url: "https://eth-sepolia.g.alchemy.com/v2/YOUR-PROVIDER-API-KEY",
    accounts:["YOUR-WALLET-PRIVATE-KEY" ]
  }
  }
```

In this project, we use Alchemy as a blockchain infrastructure provider.

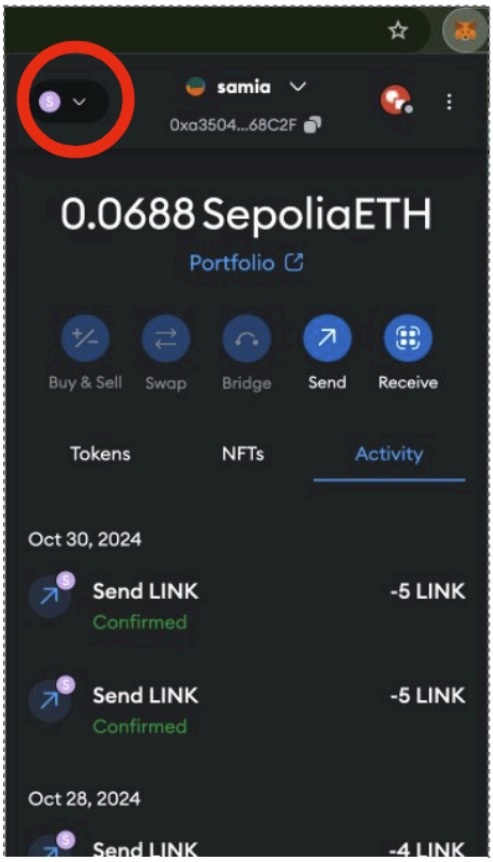
2.1 Sign up on Alchemy's website and create a new project, this process will generate an API key, copy this API key in

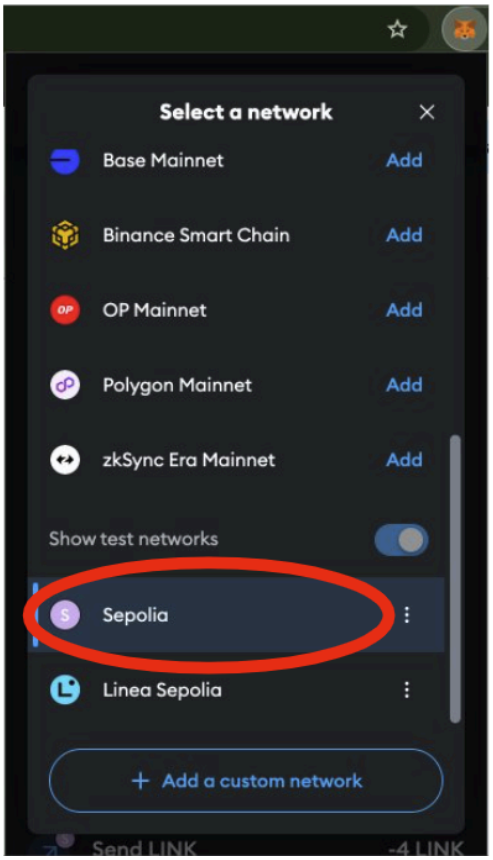
`url:`

`"https://eth-sepolia.g.alchemy.com/v2/YOUR-PROVIDER-API-KEY",`

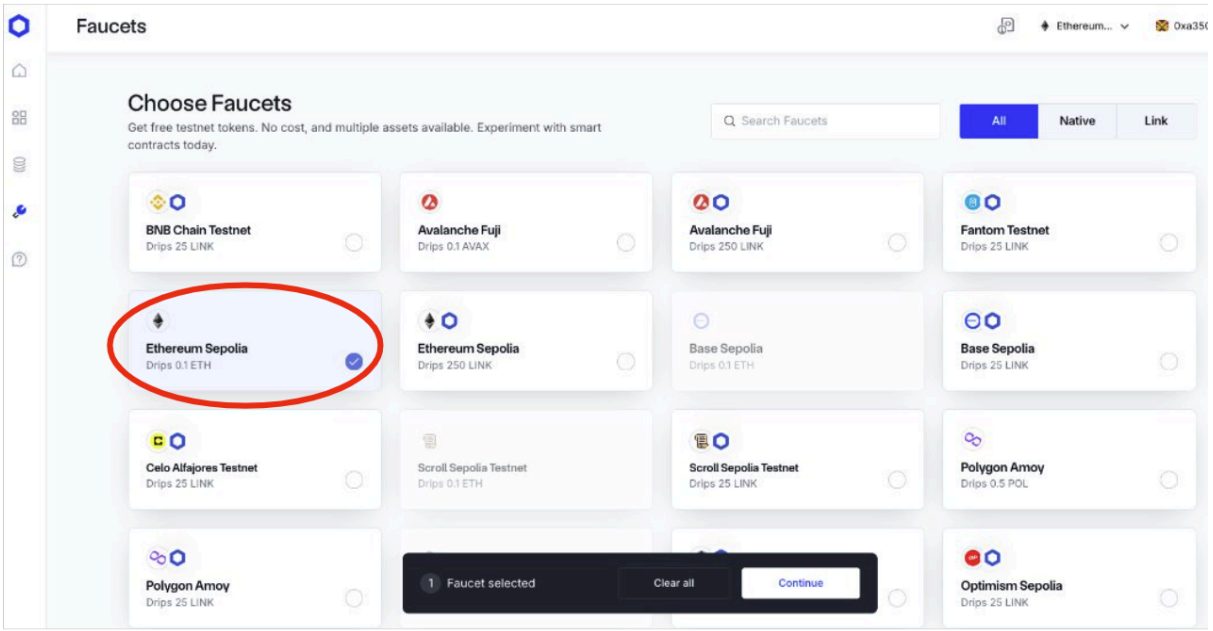
2.2 Create wallet e.g MetaMask wallet (add Metamask extension to your browser) and set the Ethereum Sepolia Testnet as a deployment network.

2.3 Enter your wallet private key to the `accounts:` [“YOUR-WALLET-PRIVATE-KEY”]





2.4 Fund your wallet with Sepolia Ether (e.g., using the Chainlink faucet). Select Sepolia Ethereum and click "Continue," and then enter your wallet's public address.



3.Compile the contracts, run in your terminal

```
npx hardhat compile
```

Upon successful compilation, you should see a message similar to:

Compiled 27 Solidity file successfully (EVM target: Paris)

Compilation Output

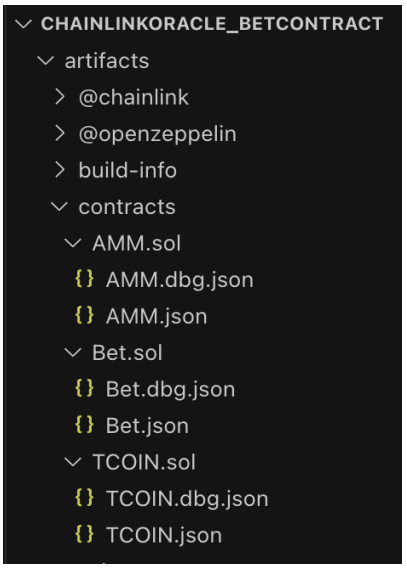
A new directory called /`artifacts` will be created, storing the compiled contract files. Each compiled contract will have two files in this folder:

contract.json: Contains the contract's ABI (Application Binary Interface) and bytecode.

contract.dbg.json: Provides debugging information for the contract.

These files provide the essential data for local testing

.

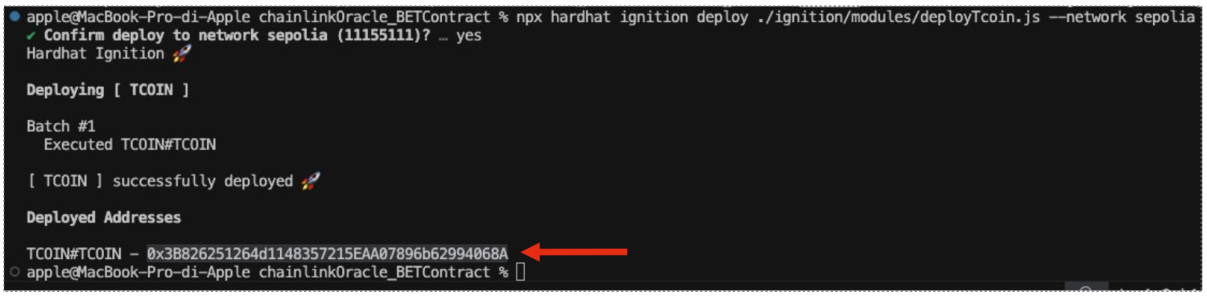


4. Deploy `./contracts/TCOIN.sol` Contract

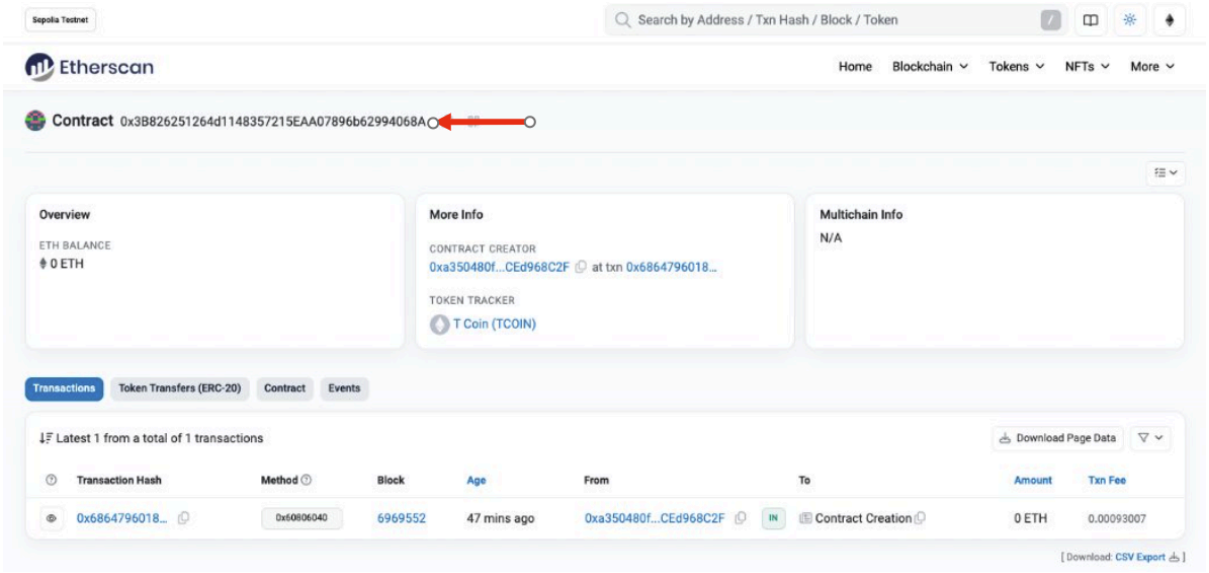
Run the following command

npx hardhat ignition deploy ./ignition/modules/deployTcoin.js –– network sepolia

TCOIN compiling output: The terminal shows the address of the newly deployed contract.



You can also see the contract on Etherscan by entering this address in the search bar.



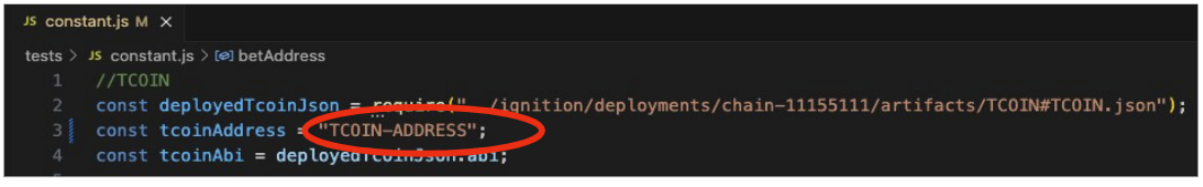
Deployment output:

You'll find two JSON files in ./ignition/deployments/chain-11155112/artifacts/ after deployment:

`TCOIN#TCOIN.dbg.json:` Debugging information.

`TCOIN#TCOIN.json:` ABI and bytecode of the deployed contract on the Sepolia testnet.

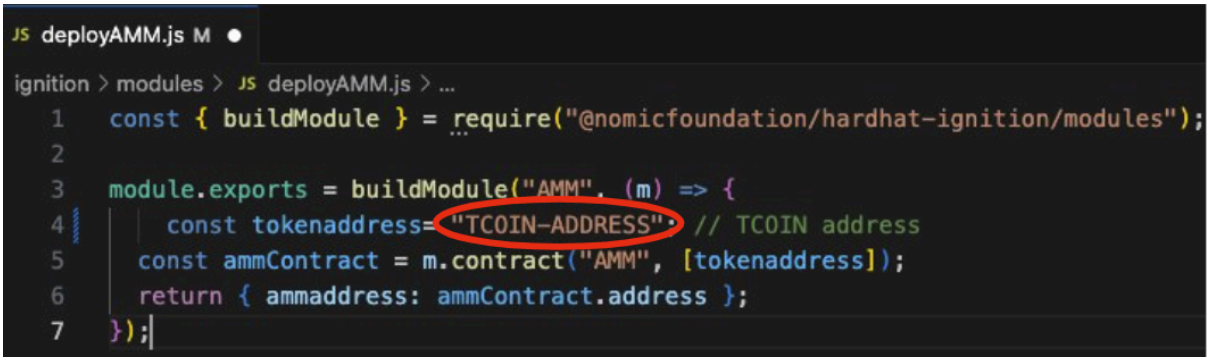
Note: A `constants.js` file is located in the `./tests` folder, where we save the smart contract properties (address and ABI) for future interactions. After deployment, copy the TCOIN contract address from the terminal output and update the `tcoinAddress` variable. Do not change the `deployedTcoin` variable, as it automatically updates with each deployment. **We'll use the same process for any future deployed contracts.**



Unexpected Error Handling Possible: If you want to modify the contract and redeploy it, delete only the entries in the `journals.json` file located in /ignition/deployments/chain-11155111/artifacts/ that are related to this contract. This step is essential to prevent confusion during future deployments.

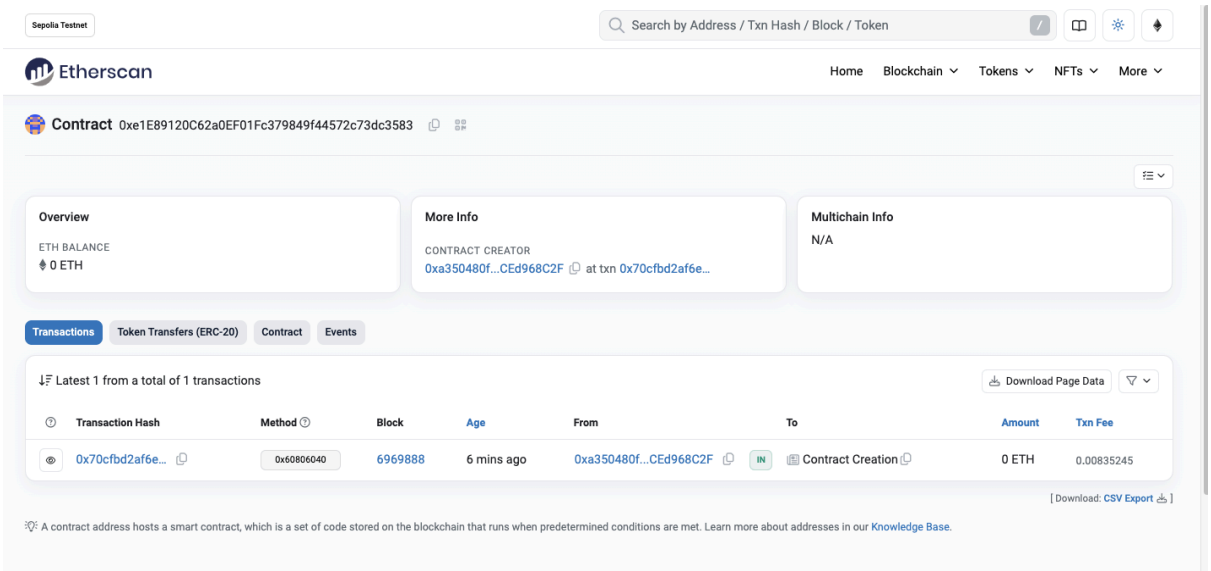
5. Deploy `./contract/AMM.sol` contract

5.1 Paste the TCOIN address into the `contract` function of the `buildModule` class in `./ignition/modules/deployAMM.js.` This address is required for the AMM constructor to use TCOIN as a token for exchanging with Ether.



5.2 Run the following command

npx hardhat ignition deploy ./ignition/modules/deployAMM.js –– network sepolia



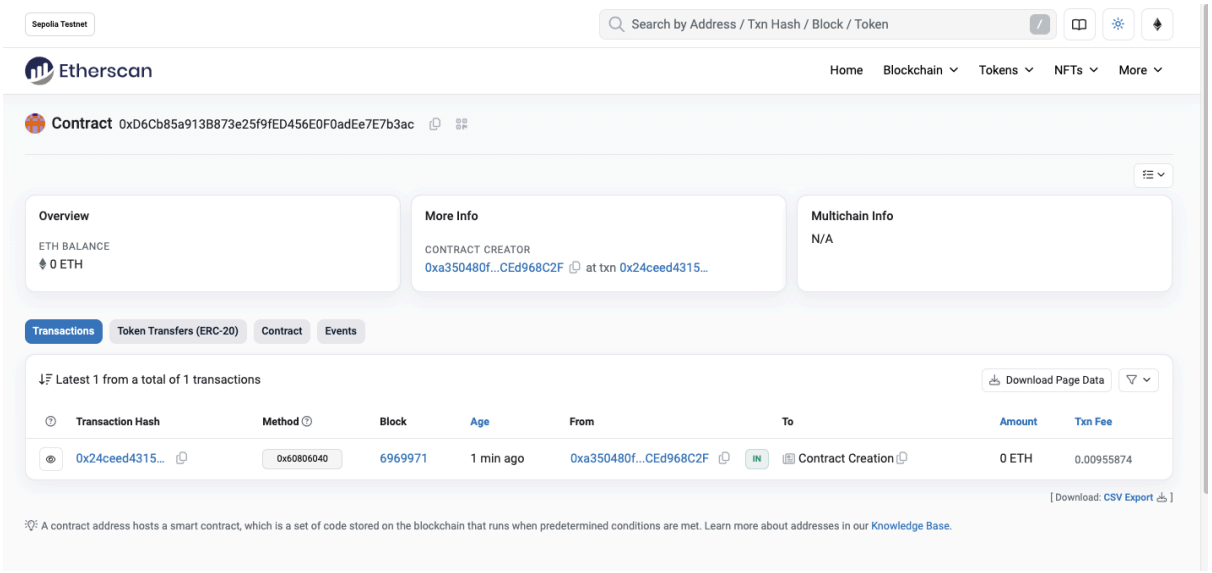
6. Deploy `./contract/Bet.sol` contract

6.1 Paste the TCOIN address and AMM address into the `contract` function of the `buildModule` class in `./ignition/modules/deployBet.js.`

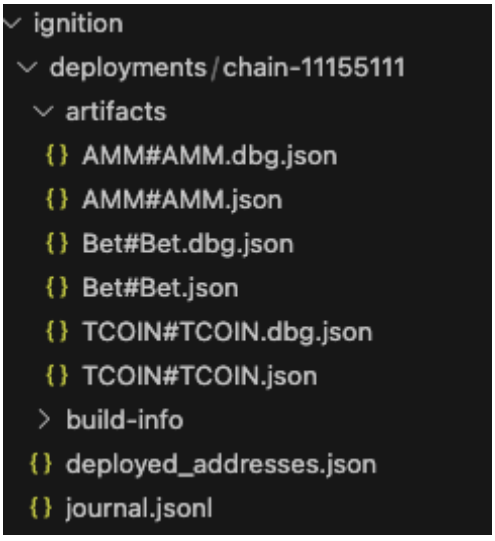
```
JS deployBet.js M ●
ignition > modules > JS deployBet.js > ...
1  const {ethers} = require("hardhat");
2  const { buildModule } = require("@nomicfoundation/hardhat-ignition/modules");
3
4  function convertDateToTimestamp(dateString) {
5    const date = new Date(dateString);
6    const timestamp = Math.floor(date.getTime() / 1000); // Convert to seconds
7    return timestamp;
8  }
9  module.exports = buildModule("Bet", (m) => {
10   const humanReadableDate = "31 october 2024 23:00:00";
11   const timestamp1= convertDateToTimestamp(humanReadableDate);
12
13   const oracle = m.contract("Bet", ["AMM-ADDRESS", "TOKEN-ADDRESS", 1, timestamp1]);
14   return { contractaddress: oracle.address };
15 });
```

6.2 Run the following command

npx hardhat ignition deploy ./ignition/modules/deployBet.js --network sepolia



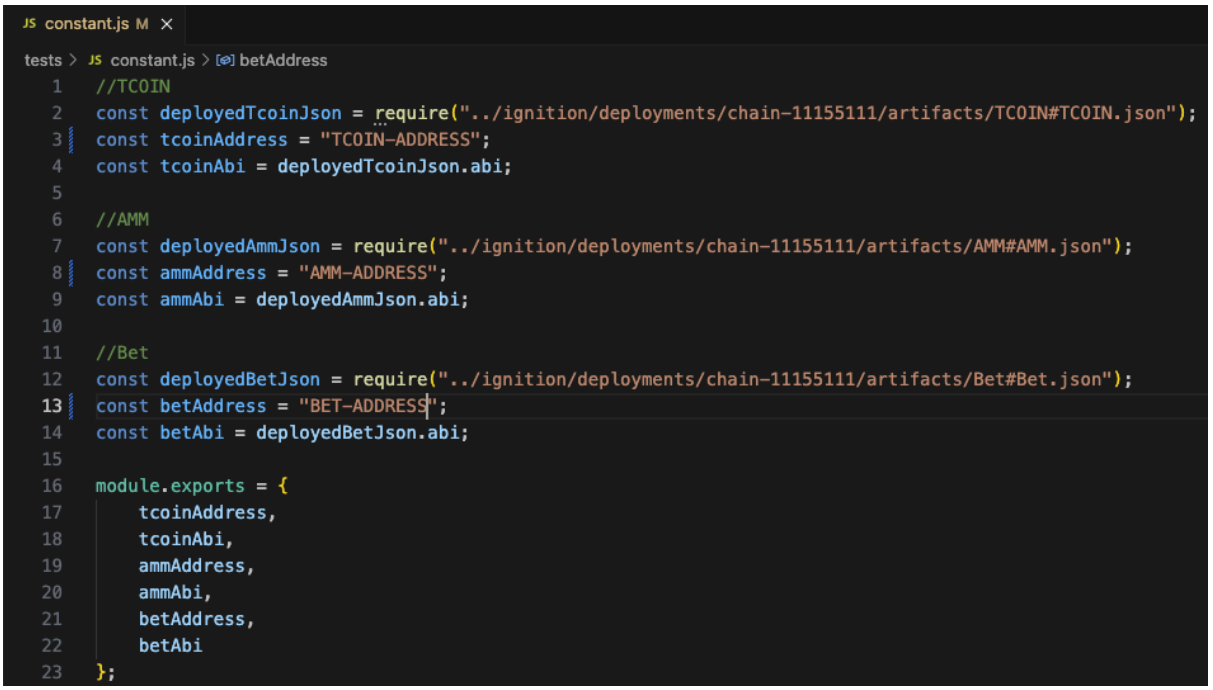
Deployments output:



7. Testing Contracts

The test scripts for contracts are in the `ignition/modules folder`. To test a smart contract on the Ethereum Sepolia network, we need to interact with it and call its functions. This requires creating a contract instance using the contract's ABI, bytecode, and a connected wallet. We will use the variables from the `constants.js` file to create each contract instance.

constant.js:



7.1 Create .env file under `./tests/` and set `YOUR-PROVIDER_API_KEY` to your Alchemy API key and `YOUR-WALLET-PRIVATE_KEY_` to your wallet's private key.

Test for Fair Player: No MEV Attempt and No Interaction with Bet Contract Dependency (AMM Contract)

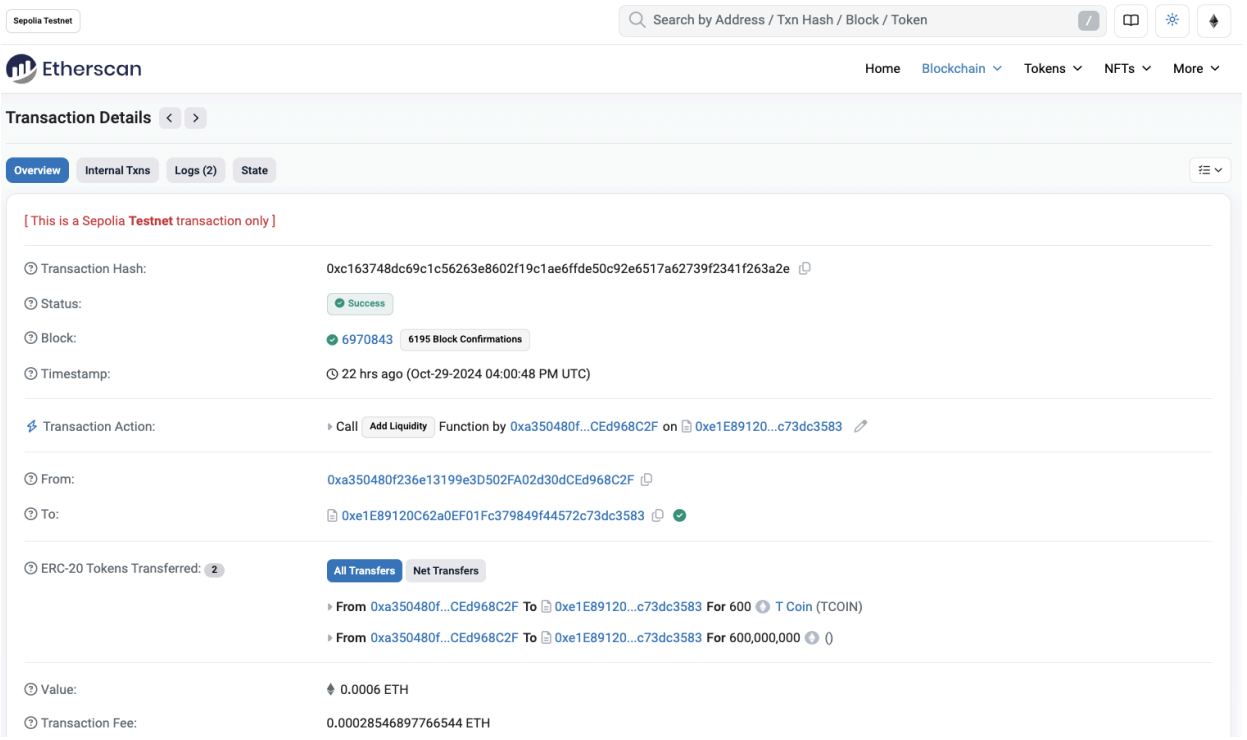
1. Interact with AMM contract

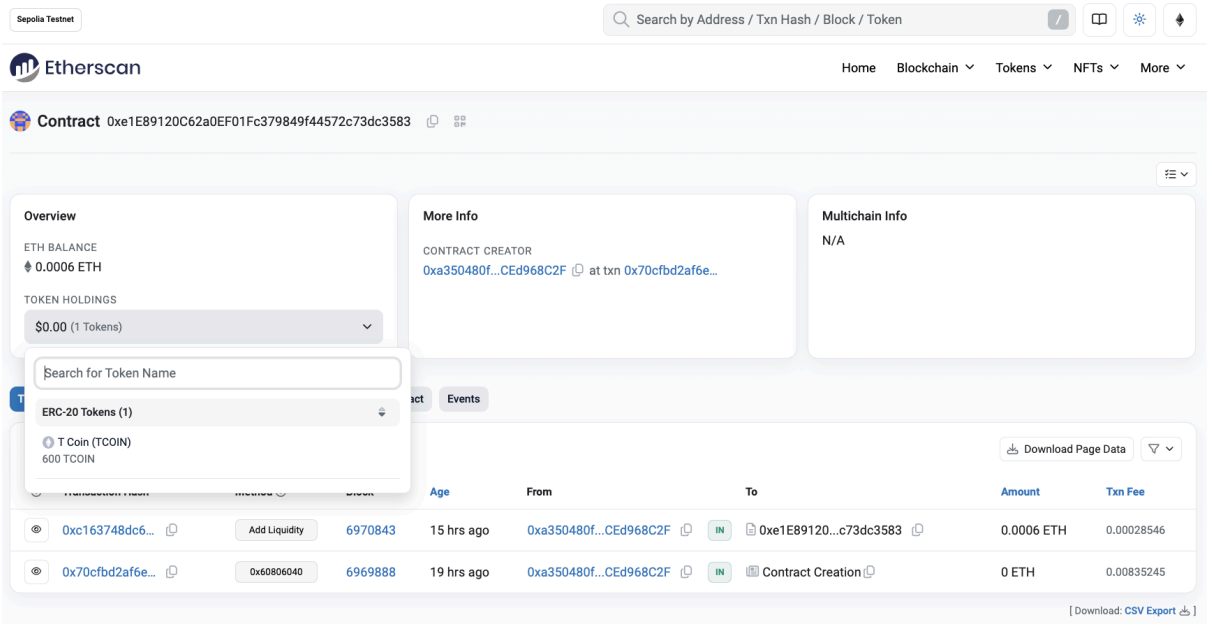
Note: Comment Out the swap() Function Call IB ./tests/interactWithAmm.js

Call addLiquidity() function with 0.0006 Ether and 600 TCOIN, run the following command:

```
node tests/interactWithAmm.js
```

addLiquidity() transaction in etherscan



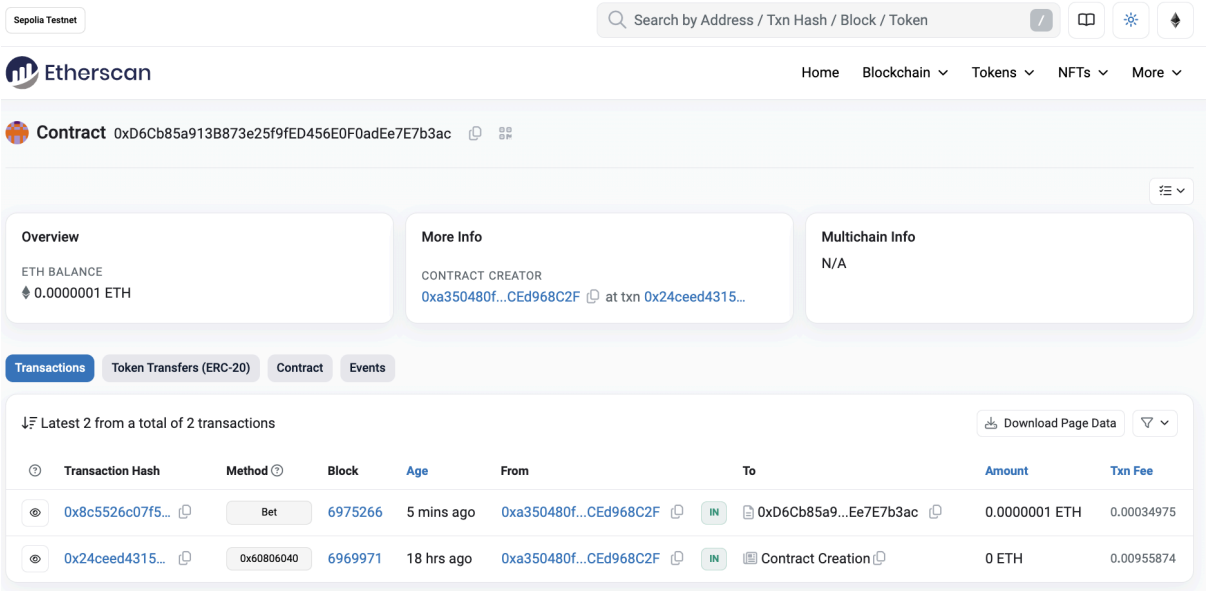


2. Interact with Bet Contract

Note: Comment Out the win() Function Call in tests/interactWithBet.js

Call bet() function, run the following command in your terminal:

node tests/interactWithBet.js



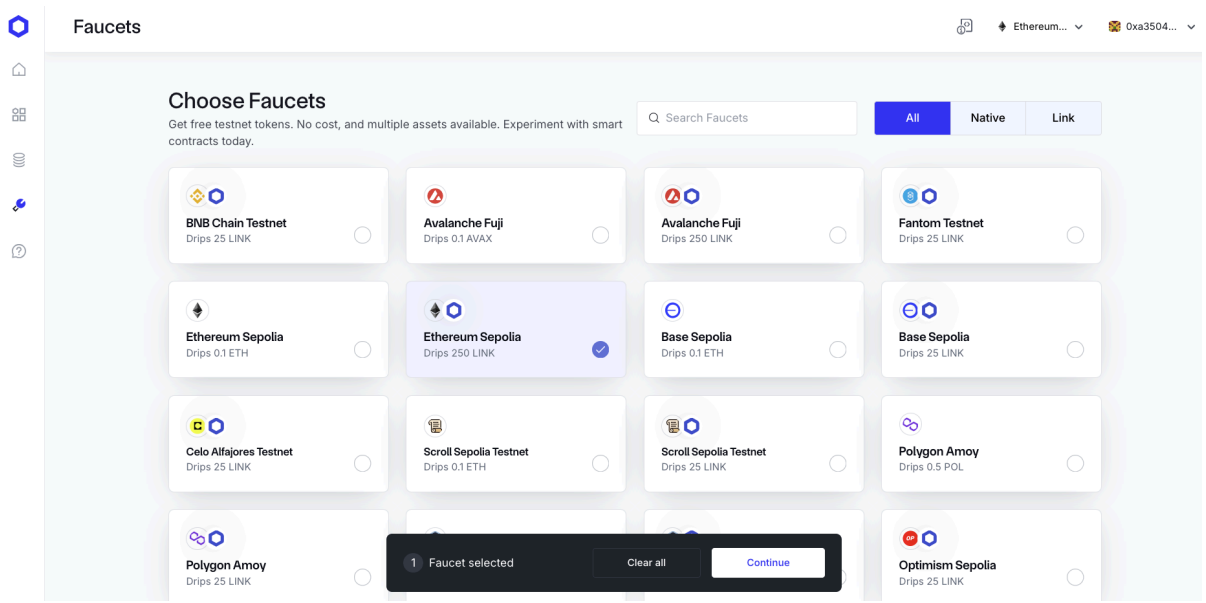
3. Interact with Bet Contract : Call win () function

Note: Comment Out the bet() Function Call in tests/interactWithBet.js

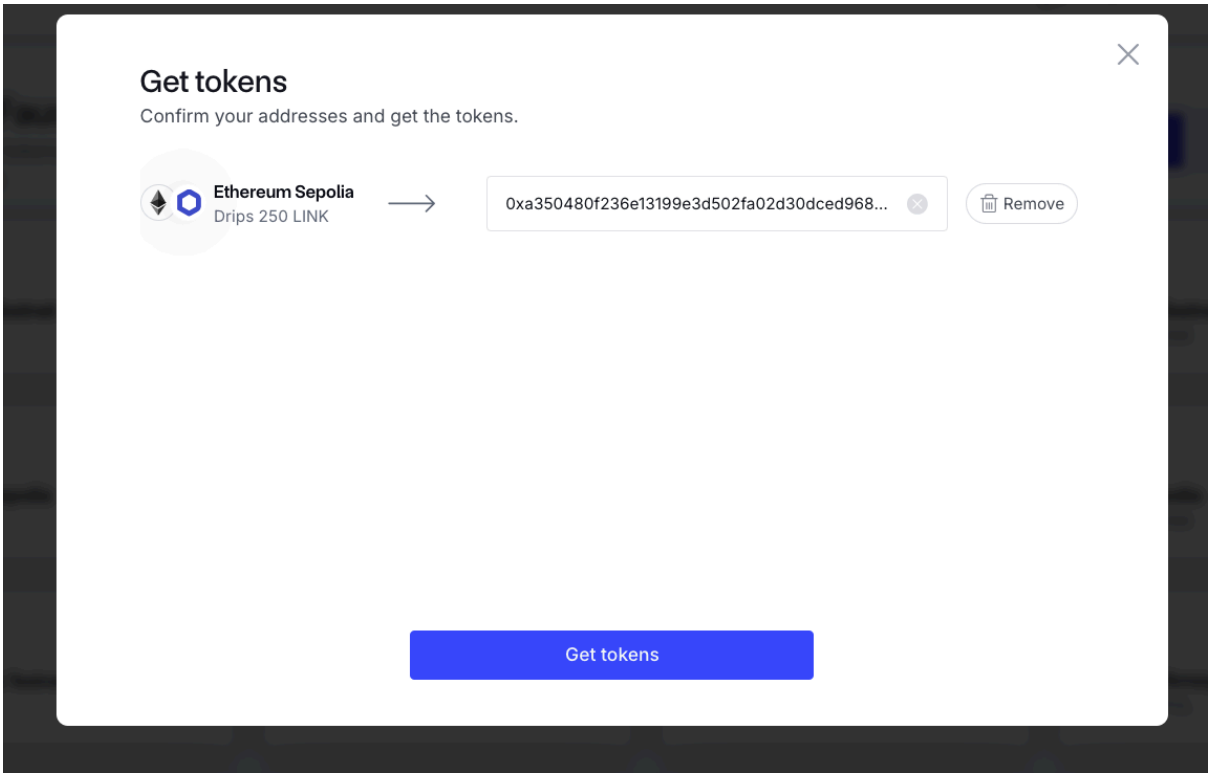
3.1 Ensure your wallet has sufficient LINK Tokens

The win() function requires a LINK token balance in the Bet contract to cover fees for interacting with the Chainlink oracle for that you need to fund the Bet contract with enough LINK, to do that first Fund Your Wallet with LINK from Chainlink faucet, then send a sufficient amount of LINK |(e.h 10 LINK) from your wallet to the Bet contract address.

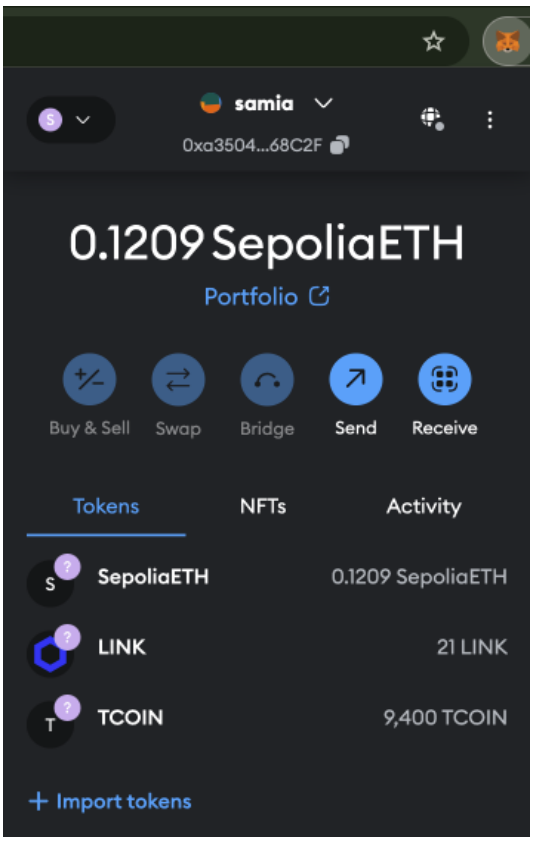
Select Link token for Ethereum Sepolia testnet in Chainlink faucet:



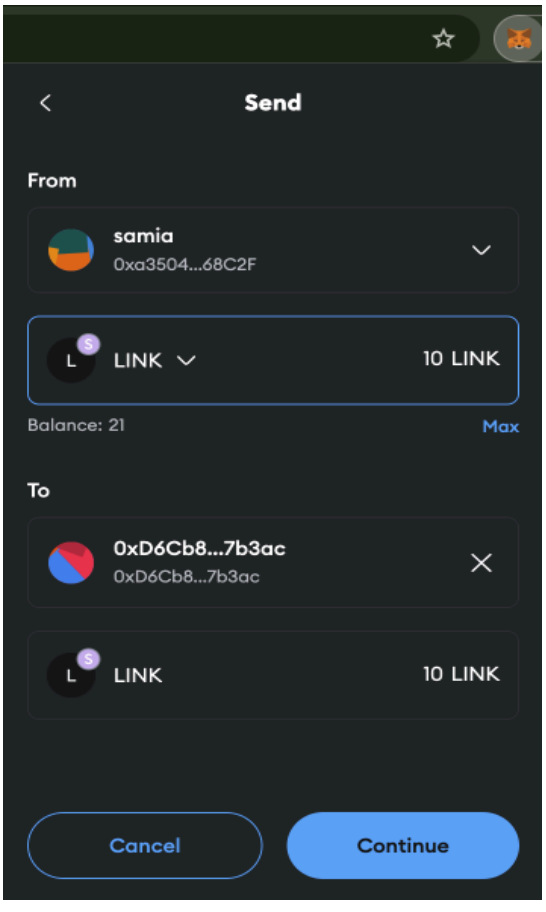
copy your wallet address:



Import the LINK token contract address (0x779877A7B0D9E8603169DdBd7836e478b4624789) into your wallet for proper visibility.



3.2 Fund the Bet Contract with LINK Tokens: Transfer 10 LINK



If the Bet contract lacks sufficient LINK, the transaction will be reverted with an error message:

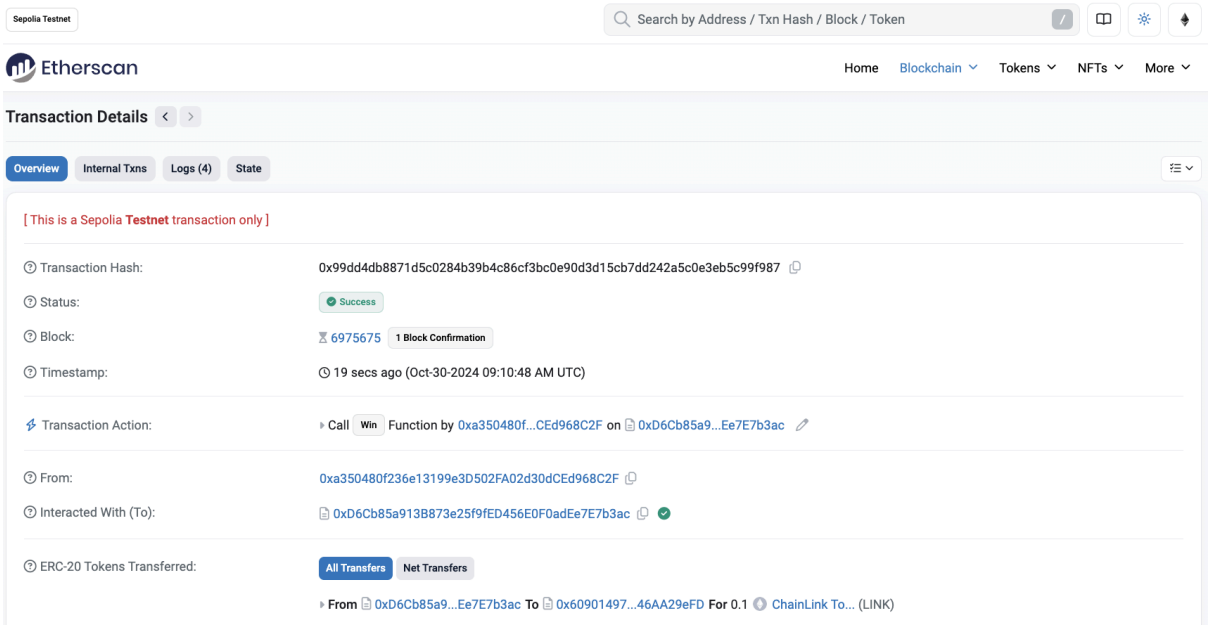
```
invocation: null,
revert: {
  signature: 'Error(string)',
  name: 'Error',
  args: [ 'ERC20: transfer amount exceeds balance' ]
},
```

3.4 Run the Command : call win() function

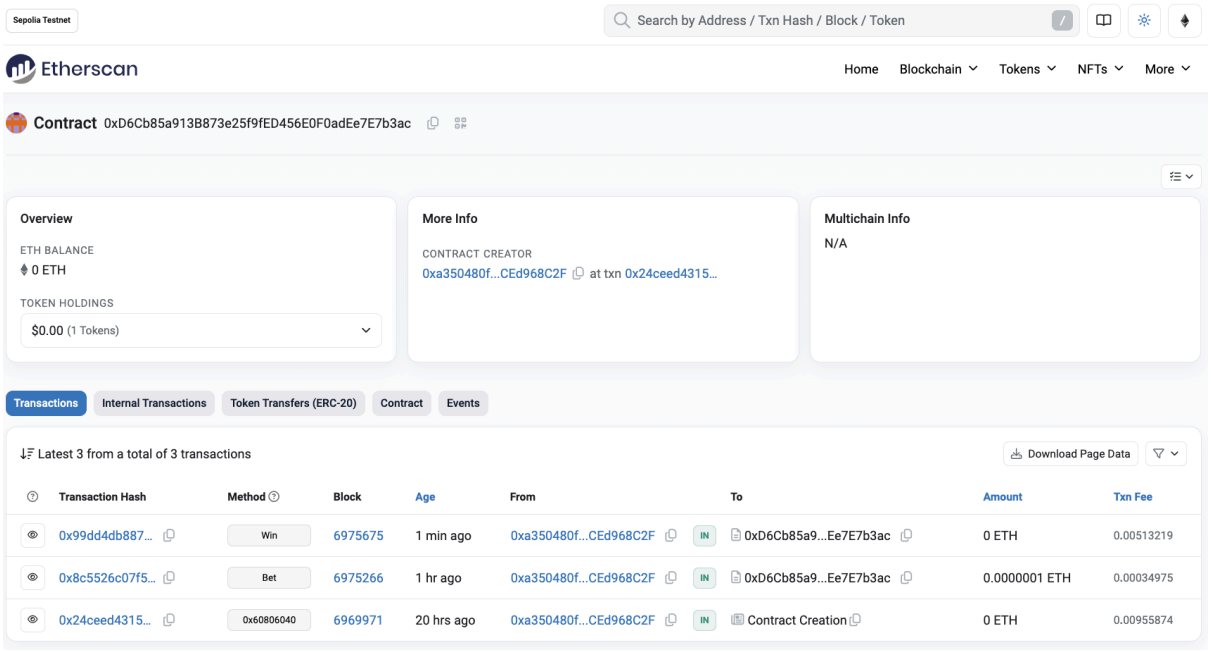
Note: Comment Out the bet() Function Call in tests/interactWithBet.js

node tests/interactWithBet.js

Transaction of the win() function call:



The Bet contract after executing win() function:



As shown in the picture above, the Bet contract transfers its entire balance to the player upon executing the win() function. This transfer is made only after verifying, through the Chainlink oracle, that no MEV attempt has occurred.

Test for UnFair Player: No MEV Attempt and No Interaction with Bet Contract Dependency (AMM Contract)

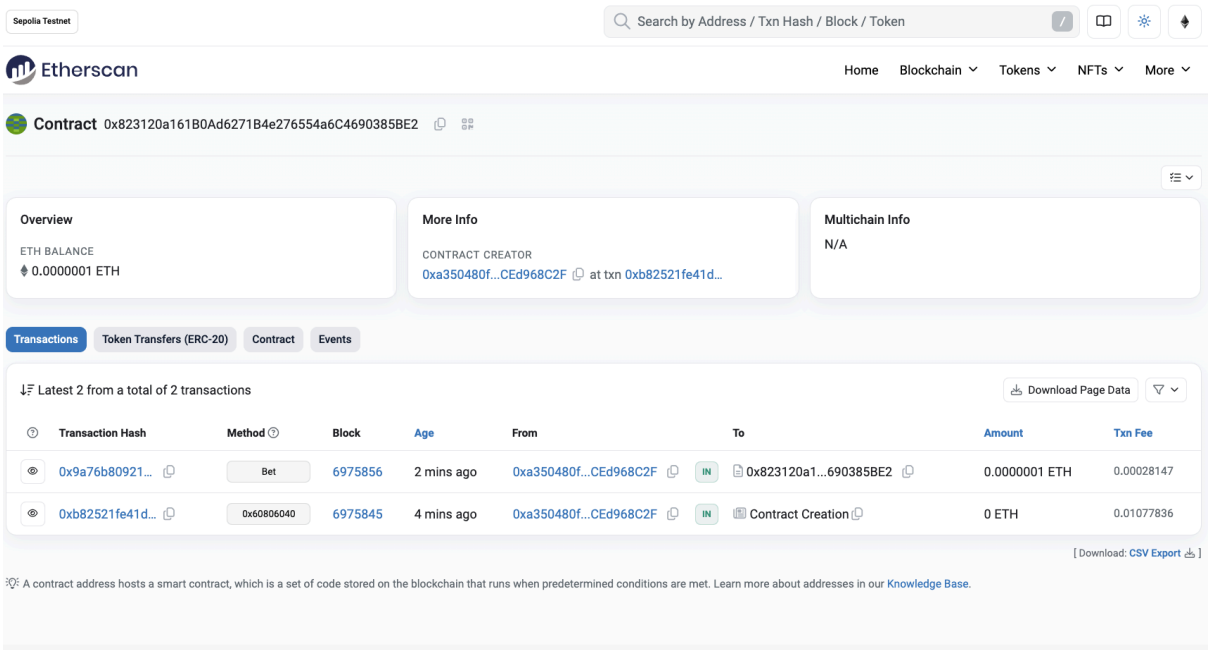
1. redeploy the Bet contract and update the necessary changes in the `constante.js` file
Note: Before redeploying the Bet contract delete only the entries in the journals.json file located in /ignition/deployments/chain-11155111/artifacts/ that are related to this contract. This step is crucial to avoid any confusion during future deployments.

Run the following command:

```
npx hardhat ignition deploy ./ignition/modules/deployBet.js --network sepolia
```

2. Interact with Bet contract
Call bet() function, run the following command in your terminal:
Note: Comment Out the win() Function Call in tests/interactWithBet.js

```
node tests/interactWithBet.js
```



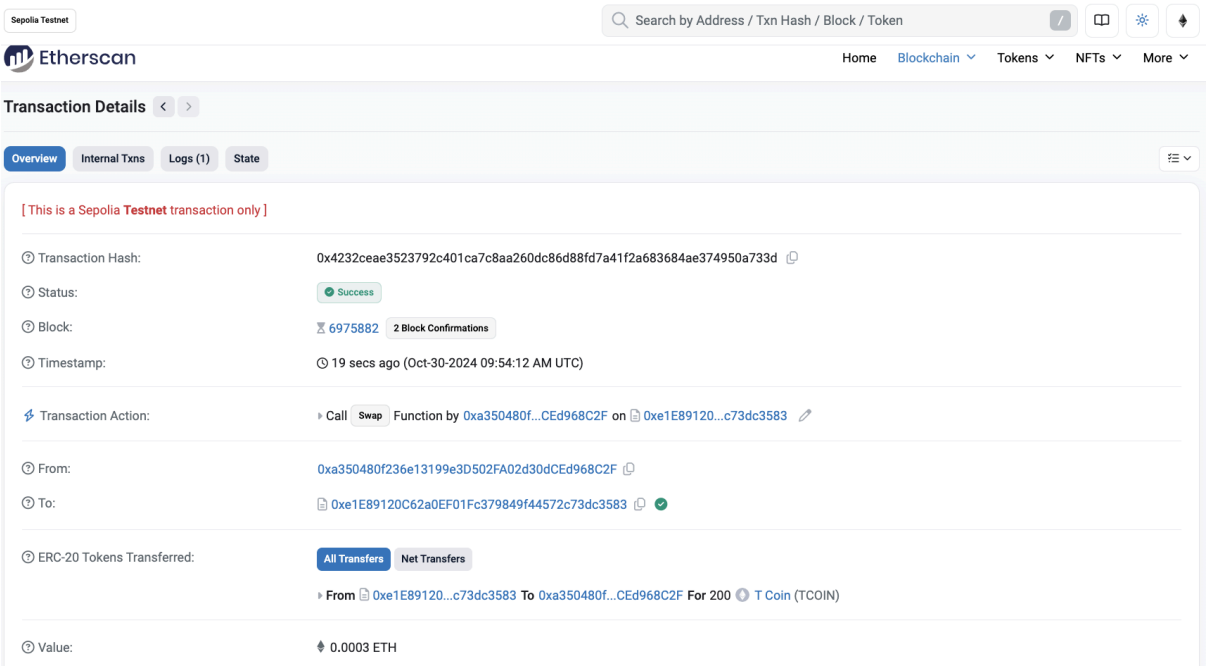
3. Interact with AMM contract

Call swap() function with 0.0003 Ether, run the following command in your terminal:

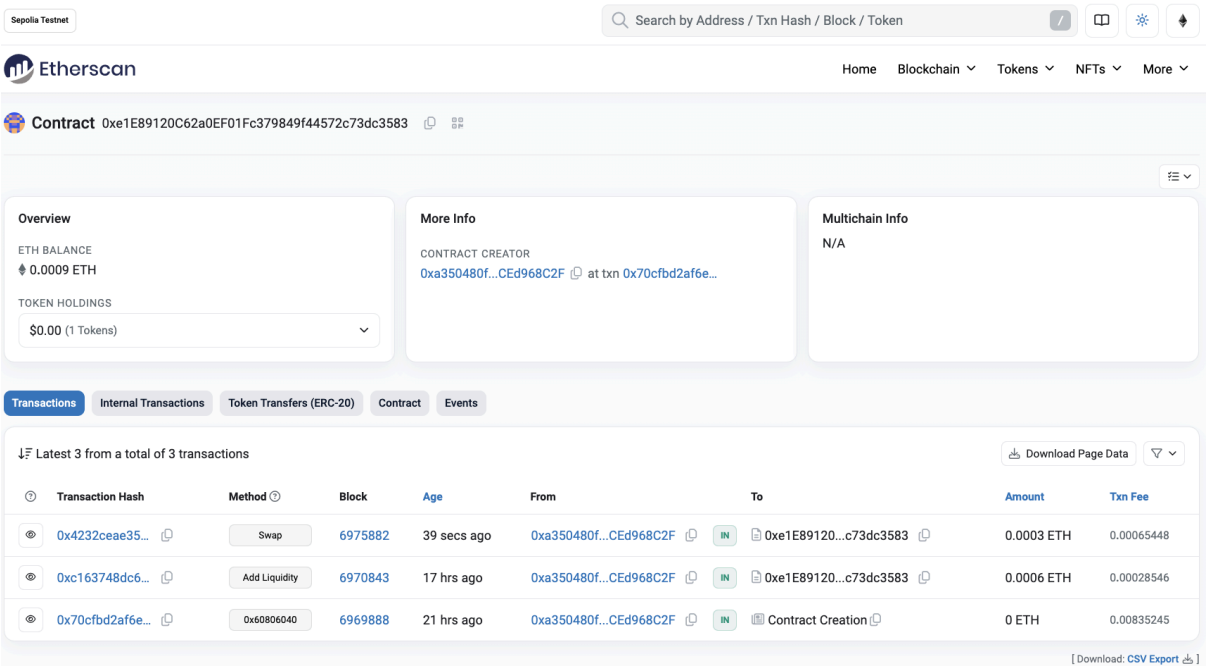
Note: Comment Out the addliquidity() Function Call in tests/interactWithBet.js

node tests/interactWithAmm. js

Transaction of the swap() function call:



The AMM contract after executing swap() function:



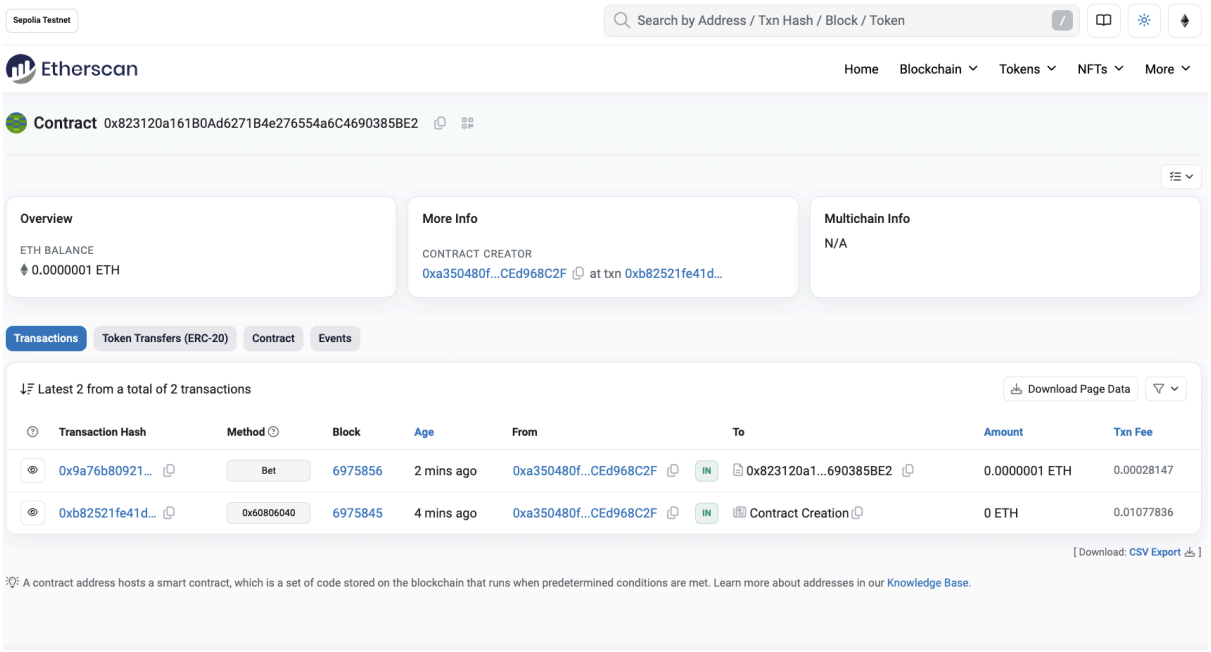
4. Interact with Bet contract

Note: Comment Out the bet() Function Call in tests/interactWithBet.js

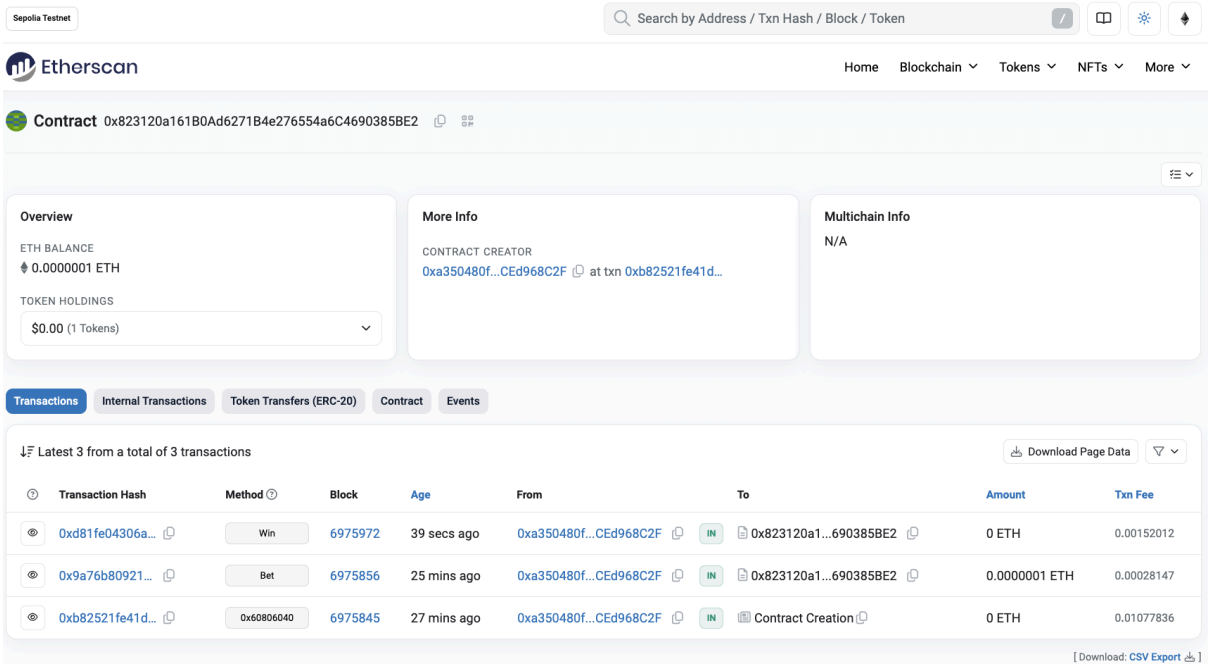
Call win() function, run the following command in your terminal:

node tests/interactWithBet.js

The Bet contract before execution the win() function:



The Bet contract after execution the win() function:



As shown in the picture above, the Bet contract does not transfer its entire balance to the player upon executing the win() function. It verifies through the Chainlink oracle that a MEV attempt has occurred in the player's historical transactions.