

28 November 2025



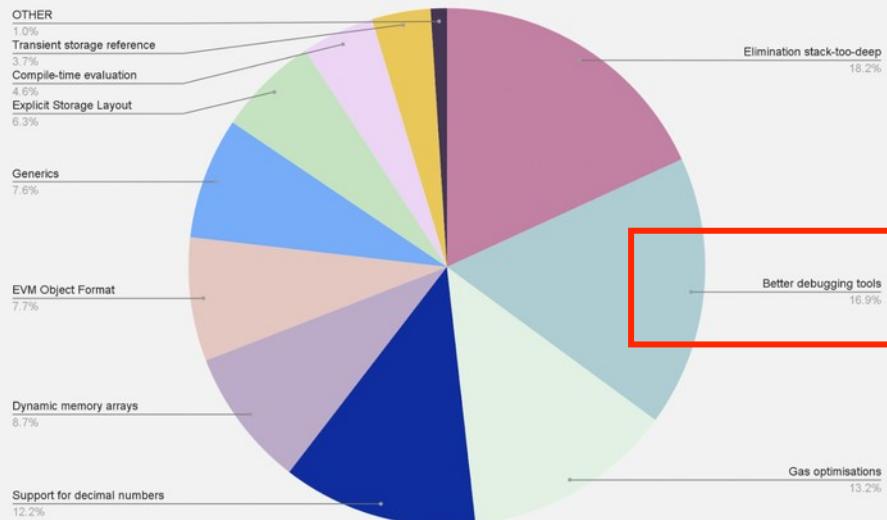
debugging solidity

roman from walnut.dev





What's the most anticipated feature you'd like to see in Solidity in the near-term future?





Agenda

1. Demo of the best Solidity Debugger
with most amazing UX 🎉
2. Challenges
3. Proposed Improvements

Web3 Fullstack Starter



Secure Web3 Voting

Participate in decentralized voting with our token-gated dApp. Mint your NFTs to gain access and have your voice heard in a transparent, secure blockchain environment.

[My NFTs](#)[How It Works](#)

Ballot 1

Voted

Ballot 1 description

[Reveal results](#)

Ballot 2

Voted

Ballot 2 description

[Reveal results](#)

WEB3-FULLSTACK-STARTER

```
> .vscode  
> node_modules  
v packages  
  v blockchain  
    > cache  
    > lib  
    > node_modules  
    > out  
    v src  
      & BallotContract.sol M  
        BallotNFT.sol  
        > test  
        & .gitignore  
        & foundry.toml  
        () package.json  
        ! pnpm-lock.yaml  
        README.md  
        > site  
        & .gitignore  
        & .gitmodules  
        () package-lock.json  
        () package.json  
        ! pnpm-lock.yaml  
        ! pnpm-workspace.yaml  
        README.md
```

BallotNFT.sol

BallotContract.sol M X

```
packages > blockchain > src > BallotContract.sol > BallotContract > getResults  
9  contract BallotContract {  
10 }  
11  
12   function castBallot(  
13     uint256 ballotId,  
14     uint256 tokenId,  
15     uint256 choice  
16   ) external {  
17     require(  
18       ballotNFT.ownerOf(tokenId) == msg.sender,  
19       "Caller does not own the token"  
20     );  
21     require(!isTokenUsed[tokenId], "Token has already been used");  
22     require(!isClosed[ballotId], "Ballot is closed");  
23     require(choice < ballots[ballotId].choices.length, "Invalid choice"  
24     require(  
25       !hasVoted[msg.sender][ballotId],  
26       "User has already voted"  
27     );  
28  
29     console.log("Ballot ID:", ballotId);  
30     emit DebugCastBallot(ballotId, tokenId, choice);  
31   }
```

At what point did software engineers get tired of console-log debugging and decide to build visual debuggers?

Here's the short version:

1960s–1970s: No debuggers → print-style debugging

Early programmers had *no* interactive tools. Debugging meant printing values to a console or paper. This is the ancestor of today's `printf` / `console.log` debugging.

Late 1970s–1980s: First visual/interactive debuggers

As programs grew larger and languages more structured, developers needed better ways to inspect state:

| Ask anything



Demo:

The best Solidity Debugger
with most amazing UX 

The screenshot displays a web-based development interface for a blockchain application. On the left, there are four ballot boxes labeled Ballot 1, Ballot 2, Ballot 3, and Ballot 4. Each ballot box has a "Voted" button and a "Reveal results" button. Ballot 1 has a "Ballot 1 description". Ballot 2 has a "Ballot 2 description". Ballot 3 has a "Ballot 3 description". Ballot 4 has a "Ballot 4 description".

On the right, the interface shows the `BallotContract.sol` file in the `Editor` tab. The code defines a `BallotContract` with functions for minting NFTs, casting ballots, and getting results. A red box highlights the `TRANSACTIONS` section in the `CALL STACK` panel, which lists several transaction logs, including铸造和投票操作。

```
BallotContract.sol — web3-fullstack-starter
packages > blockchain > src > BallotContract.sol > BallotContract > mintBallotNFT
8 contract BallotContract {
    ...
    function mintBallotNFT() external {
        ballotNFT.safeMint(msg.sender); Chinthaka Weerakkodi
    }

    function castBallot(
        uint256 ballotId,
        uint256 tokenId,
        uint256 choice
    ) external {
        require(
            ballotNFT.ownerOf(tokenId) == msg.sender,
            "Caller does not own the token"
        );
        require(!isTokenUsed[tokenId], "Token has already been used");
        require(!isClosed[ballotId], "Ballot is closed");
        require(choice < ballots[ballotId].choices.length, "Invalid choice");
        require(
            !hasVoted[msg.sender][ballotId],
            "User has already voted"
        );

        hasVoted[msg.sender][ballotId] = true;
        isTokenUsed[tokenId] = true;
        results[ballotId][choice] += 1;

        emit VoteCast(ballotId, tokenId, choice);
    }

    function getResults(uint id) public view returns (uint[] memory result) {
        uint[] memory result = new uint[](ballots[id].choices.length);
        ...
    }
}
```

The screenshot shows a web-based development interface for a Solidity smart contract named `BallotContract.sol`. The interface includes a sidebar with four ballot boxes (Ballot 1, Ballot 2, Ballot 3, Ballot 4) and a central editor window displaying the contract code.

Editor Window Content:

```
BallotNFT. 8 packages > blockchain > src > BallotContract.sol > BallotContract > mintBallotNFT
contract BallotContract {
    function mintBallotNFT() external {
        ballotNFT.safeMint(msg.sender); Chinthaka Weerakkod
    }

    function castBallot(
        uint256 ballotId,
        uint256 tokenId,
        uint256 choice
    ) external {
        require(
            ballotNFT.ownerOf(tokenId) == msg.sender,
            "Caller does not own the token"
        );
        require(!isTokenUsed[tokenId], "Token has already been used");
        require(!isClosed[ballotId], "Ballot is closed");
        require(choice < ballots[ballotId].choices.length, "Invalid choice");
        require(
            !hasVoted[msg.sender][ballotId],
            "User has already voted"
        );

        hasVoted[msg.sender][ballotId] = true;
        isTokenUsed[tokenId] = true;
        results[ballotId][choice] += 1;

        emit VoteCast(ballotId, tokenId, choice);
    }

    function getResults(uint id) public view returns (uint[] memory result) {
        uint[] memory result = new uint[](ballots[id].choices.length);
        for (uint i = 0; i < result.length; i++) {
            result[i] = results[id][i];
        }
        return result;
    }
}
```

Sidebar Elements:

- My NFTs:** Shows a purple button labeled "Vote".
- How It Works:** Shows a grey button labeled "Voted".
- Ballot 1:** Shows a grey button labeled "Reveal results".
- Ballot 2:** Shows a grey button labeled "Reveal results".
- Ballot 3:** Shows a purple button labeled "Vote". This button is highlighted with a red rectangular border.
- Ballot 4:** Shows a purple button labeled "Vote".

Toolbars and Status:

- Top toolbar: Agents, Editor, Launchpad, SolDB (web3-fullstack-starter).
- Bottom status bar: Ln 76, Col 24, Spaces: 4, UTF-8, LF, Solidity, Prettier.

My NFTs How It Works

Agents Editor

Ballot1 Voted

Ballot 1 description

Reveal results

Ballot 3

Pick a NFT that you want to vote with for Ballot #2

Selected

Ballot NFT
#2

Next

Vote

Ballot 4

Ballot 4 description

Vote

BallotContract.sol — web3-fullstack-starter

SolDB

VARIABLES

CALL STACK

Running

BREAKPOINTS

TRANSACTIONS

- Monitoring transactions at http://localhost:85...
✓ 0x9f52a6..8eea62 mintBallotNFT
- ✓ 0x7ba05a...aad663 create(string,string,string[])
- ✓ 0x59f1f4...c9d5ce castBallot(uint256,uint256,...
⊗ 0xc65ba3...088ef0 castBallot(uint256,uint256,...
✓ 0x19f6bf..7d458c create(string,string,string[])
- ✓ 0x198a6c...fdcf38 mintBallotNFT

WATCH

BallotNFT.

BallotContract.sol > BallotContract > mintBallotNFT

contract BallotContract {

function mintBallotNFT() external {
 ballotNFT.safeMint(msg.sender); Chinthaka Weerakkod

}

function castBallot(
 uint256 ballotId,
 uint256 tokenId,
 uint256 choice
) external {
 require(
 ballotNFT.ownerOf(tokenId) == msg.sender,
 "Caller does not own the token"
);
 require(!isTokenUsed[tokenId], "Token has already been used");
 require(!isClosed[ballotId], "Ballot is closed");
 require(choice < ballots[ballotId].choices.length, "Invalid choice");
 require(
 !hasVoted[msg.sender][ballotId],
 "User has already voted"
);

 hasVoted[msg.sender][ballotId] = true;
 isTokenUsed[tokenId] = true;
 results[ballotId][choice] += 1;

 emit VoteCast(ballotId, tokenId, choice);
}

function getResults(uint id) public view returns (uint[] memory result) {
 uint[] memory result = new uint[](ballots[id].choices.length);
 for (uint i = 0; i < result.length; i++) {
 result[i] = results[id][i];
 }
 return result;

Ln 76, Col 24 Spaces: 4 UTF-8 LF ⚙️ solidity ⚙️ Prettier ⚙️

The screenshot shows a web-based development interface for a blockchain application. On the left, there are three cards: "Ballot 1" (status: Voted), "Ballot 3" (status: Pick a NFT that you want to vote with for Ballot #2, with a "Selected" NFT thumbnail labeled "Ballot NFT #2" and a red box around the "Next" button), and "Ballot 4" (status: Ballot 4 description). The central part of the interface is an "Editor" tab showing Solidity code for the `BallotContract.sol` file. The code defines a `BallotContract` with functions `mintBallotNFT` and `castBallot`. The `mintBallotNFT` function uses `ballotNFT.safeMint(msg.sender)`. The `castBallot` function checks if the voter has already voted and increments the vote count. The right side of the interface includes a "CALL STACK" section showing transaction history and a "WATCH" section.

```
BallotContract.sol — web3-fullstack-starter
packages > blockchain > src > BallotContract.sol > BallotContract > mintBallotNFT
8 < contract BallotContract {
    ...
    function mintBallotNFT() external {
        ballotNFT.safeMint(msg.sender); Chinthaka Weerakkod
    }
    ...
    function castBallot(
        uint256 ballotId,
        uint256 tokenId,
        uint256 choice
    ) external {
        require(
            ballotNFT.ownerOf(tokenId) == msg.sender,
            "Caller does not own the token"
        );
        require(!isTokenUsed[tokenId], "Token has already been used");
        require(!isClosed[ballotId], "Ballot is closed");
        require(choice < ballots[ballotId].choices.length, "Invalid choice");
        require(
            !hasVoted[msg.sender][ballotId],
            "User has already voted"
        );
        hasVoted[msg.sender][ballotId] = true;
        isTokenUsed[tokenId] = true;
        results[ballotId][choice] += 1;
        emit VoteCast(ballotId, tokenId, choice);
    }
    ...
    function getResults(uint id) public view returns (uint[] memory result) {
        uint[] memory result = new uint[](ballots[id].choices.length);
        ...
    }
}
```

The screenshot shows a web-based development environment for a blockchain application. On the left, there are three ballot interfaces: **Ballot 1**, **Ballot 3**, and **Ballot 4**. **Ballot 3** is active, displaying a list of options (1, 3, 5) and a **Vote** button. **Ballot 4** has a **Ballot 4 description** and a **Vote** button. The central part of the interface is the **Editor** window, which displays the Solidity source code for the **BallotContract.sol** file.

```
BallotNFT. BallotContract.sol -- web3-fullstack-starter
packages > blockchain > src > BallotContract.sol > BallotContract > mintBallotNFT
8 < contract BallotContract {
    Debug
75     function mintBallotNFT() external {
76         ballotNFT.safeMint(msg.sender); Chinthaka Weerakkod
77     }
78
79     function castBallot(
80         uint256 ballotId,
81         uint256 tokenId,
82         uint256 choice
83     ) external {
84         require(
85             ballotNFT.ownerOf(tokenId) == msg.sender,
86             "Caller does not own the token"
87         );
88         require(!isTokenUsed[tokenId], "Token has already been used");
89         require(!isClosed[ballotId], "Ballot is closed");
90         require(choice < ballots[ballotId].choices.length, "Invalid choice");
91         require(
92             !hasVoted[msg.sender][ballotId],
93             "User has already voted"
94         );
95
96         hasVoted[msg.sender][ballotId] = true;
97         isTokenUsed[tokenId] = true;
98         results[ballotId][choice] += 1;
99
100        emit VoteCast(ballotId, tokenId, choice);
101    }
102
103    Debug
104    function getResults(uint id) public view returns (uint[] memory result) {
105        uint[] memory result = new uint[](ballots[id].choices.length);
106        for (uint i = 0; i < result.length; i++) {
107            result[i] = results[id][i];
108        }
109    }
}
```

The editor also shows a call stack with one entry: **Running**. The bottom status bar indicates the current file is **SolDB (web3-fullstack-starter)**, line 76, column 24, with 4 spaces, encoding type **UTF-8**, and the file is **solidity**.

The screenshot shows a web-based application interface for a ballot system, with the source code of the smart contract visible in the background.

Left Panel (Ballot 3 View):

- Header:** Ballot 1, Voted
- Description:** Ballot 1 description
- Buttons:** Reveal results, Ballot 3
- Ballot 3 View:**
 - Pick a choice
 - Options: 1, 3, 5
 - Vote button:** This button is highlighted with a red border.
 - Back button:**
- Ballot 4 View:**
 - Balot 4 description
 - Vote button

Right Panel (Editor):

- Agents:** My NFTs, How It Works
- Editor:** BallotContract.sol — web3-fullstack-starter
- Solidity Compiler:** SolDB
- VARIABLES:** None listed.
- CALL STACK:** Running
- BREAKPOINTS:** Monitoring transactions at http://localhost:85...
 - ✓ 0x9f52a6...8eea62 mintBallotNFT
 - ✓ 0x7ba05a...aad663 create(string, string, string[])
 - ✓ 0x59f1f4...c9d5ce castBallot(uint256, uint256, uint256)
 - ⌚ 0xc65ba3...088ef0 castBallot(uint256, uint256, uint256)
 - ✓ 0x19f6bf...7d458c create(string, string, string[])
 - ✓ 0x198a6c...fdcf38 mintBallotNFT
- WATCH:** None listed.
- Code:** BallotContract.sol

```
8 // SPDX-License-Identifier: MIT
9
10 pragma solidity ^0.8.0;
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75     function mintBallotNFT() external {
76         ballotNFT.safeMint(msg.sender);    Chinthaka Weerakkod
77     }
78
79     function castBallot(
80         uint256 ballotId,
81         uint256 tokenId,
82         uint256 choice
83     ) external {
84         require(
85             ballotNFT.ownerOf(tokenId) == msg.sender,
86             "Caller does not own the token"
87         );
88         require(!isTokenUsed[tokenId], "Token has already been used");
89         require(!isClosed[ballotId], "Ballot is closed");
90         require(choice < ballots[ballotId].choices.length, "Invalid choice");
91         require(
92             !hasVoted[msg.sender][ballotId],
93             "User has already voted"
94         );
95
96         hasVoted[msg.sender][ballotId] = true;
97         isTokenUsed[tokenId] = true;
98         results[ballotId][choice] += 1;
99
100        emit VoteCast(ballotId, tokenId, choice);
101    }
102
103    function getResults(uint id) public view returns (uint[] memory result) {
104        uint[] memory result = new uint[](ballots[id].choices.length);
105        for (uint i = 0; i < result.length; i++) {
106            result[i] = results[id][i];
107        }
108    }
109}
```

Bottom Status Bar: Launchpad, 0, SolDB (web3-fullstack-starter), Ln 76, Col 24, Spaces: 4, UTF-8, LF, Solidity, Prettier.

The screenshot displays a web-based development interface for a blockchain application. On the left, there are four separate ballot boxes labeled Ballot 1, Ballot 2, Ballot 3, and Ballot 4. Each ballot box has a "Reveal results" button. The Ballot 3 and Ballot 4 boxes have a "Voted" status indicator. Below each ballot box is a brief description.

The right side of the interface shows the source code for the `BallotContract.sol` file, specifically the `BallotContract` contract. The code includes functions for minting NFTs, casting ballots, and getting results. A red box highlights the transaction hash `0x70a6ca..f2a9ca` in the CALL STACK, which corresponds to the cast ballot action for Ballot 3.

```
BallotContract.sol — web3-fullstack-starter

packages > blockchain > src > BallotContract.sol > BallotContract > mintBallotNFT

contract BallotContract {
    function mintBallotNFT() external {
        ballotNFT.safeMint(msg.sender); Chinthaka Weerakkodi
    }

    function castBallot(
        uint256 ballotId,
        uint256 tokenId,
        uint256 choice
    ) external {
        require(
            ballotNFT.ownerOf(tokenId) == msg.sender,
            "Caller does not own the token"
        );
        require(!isTokenUsed[tokenId], "Token has already been used");
        require(!isClosed[ballotId], "Ballot is closed");
        require(choice < ballots[ballotId].choices.length, "Invalid choice");
        require(
            !hasVoted[msg.sender][ballotId],
            "User has already voted"
        );
        hasVoted[msg.sender][ballotId] = true;
        isTokenUsed[tokenId] = true;
        results[ballotId][choice] += 1;

        emit VoteCast(ballotId, tokenId, choice);
    }

    function getResults(uint id) public view returns (uint[] memory)
        uint[] memory result = new uint[](ballots[id].choices.length);
        for (uint i = 0; i < result.length; i++) {
            result[i] = results[id][i];
        }
        return result;
    }
}
```

My NFTs How It Works

Agents Editor

BallotContract.sol — web3-fullstack-starter

SolDB

VARIABLES

CALL STACK

Running

Ballot1 Voted

Ballot 1 description

Reveal results

Ballot 4

Pick a NFT that you want to vote with for Ballot #3

Selected

Ballot NFT #2

Next

Reveal results

Ballot 4

Ballot 4 description

Vote

BallotNFT.

packages > blockchain > src > BallotContract.sol > BallotContract > mintBallotNFT

```
8 contract BallotContract {  
    ...  
    function mintBallotNFT() external {  
        ballotNFT.safeMint(msg.sender); Chinthaka Weerakkod  
    }  
    ...  
    function castBallot(  
        uint256 ballotId,  
        uint256 tokenId,  
        uint256 choice  
    ) external {  
        require(  
            ballotNFT.ownerOf(tokenId) == msg.sender,  
            "Caller does not own the token"  
        );  
        require(!isTokenUsed[tokenId], "Token has already been used");  
        require(!isClosed[ballotId], "Ballot is closed");  
        require(choice < ballots[ballotId].choices.length, "Invalid choice");  
        require(  
            !hasVoted[msg.sender][ballotId],  
            "User has already voted"  
        );  
  
        hasVoted[msg.sender][ballotId] = true;  
        isTokenUsed[tokenId] = true;  
        results[ballotId][choice] += 1;  
  
        emit VoteCast(ballotId, tokenId, choice);  
    }  
    ...  
    function getResults(uint id) public view returns (uint[] memory result) {  
        uint[] memory result = new uint[](ballots[id].choices.length);  
        ...  
    }  
}
```

Ln 76, Col 24 Spaces: 4 UTF-8 LF ⚙️ solidity ⚙️ Prettier ⚙️

My NFTs How It Works Agents Editor BallotContract.sol — web3-fullstack-starter

Ballot1 Voted
Ballot 1 description
Reveal results

Ballot 4
Pick a NFT that you want to vote with for Ballot #3
Selected
Balot NFT #2
Next

Reveal results

Ballot 4
Ballot 4 description
Vote

VARIABLES
CALL STACK Running
BREAKPOINTS
TRANSACTIONS
Monitoring transactions at http://localhost:85...
✓ 0x9f52a6..8eea62 mintBallotNFT
✓ 0x7ba05a...aad663 create(string,string,string[])
✓ 0x59f1f4..c9d5ce castBallot(uint256,uint256,...)
⊗ 0xc65ba3..088ef0 castBallot(uint256,uint256,...)
✓ 0x19f6bf..7d458c create(string,string,string[])
✓ 0x198a6c..fdcf38 mintBallotNFT
✓ 0x70a6ca..f2a9ca castBallot(uint256,uint256,...)

WATCH

packages > blockchain > src > BallotContract.sol > BallotContract > mintBallotNFT

```
8 contract BallotContract {  
    ...  
    function mintBallotNFT() external {  
        ballotNFT.safeMint(msg.sender); Chinthaka Weerakkod  
    }  
    ...  
    function castBallot(  
        uint256 ballotId,  
        uint256 tokenId,  
        uint256 choice  
    ) external {  
        require(  
            ballotNFT.ownerOf(tokenId) == msg.sender,  
            "Caller does not own the token"  
        );  
        require(!isTokenUsed[tokenId], "Token has already been used");  
        require(!isClosed[ballotId], "Ballot is closed");  
        require(choice < ballots[ballotId].choices.length, "Invalid choice");  
        require(  
            !hasVoted[msg.sender][ballotId],  
            "User has already voted"  
        );  
  
        hasVoted[msg.sender][ballotId] = true;  
        isTokenUsed[tokenId] = true;  
        results[ballotId][choice] += 1;  
  
        emit VoteCast(ballotId, tokenId, choice);  
    }  
    ...  
    function getResults(uint id) public view returns (uint[] memory result) {  
        uint[] memory result = new uint[](ballots[id].choices.length);  
        ...  
    }  
}
```

Ln 76, Col 24 Spaces: 4 UTF-8 LF ⚙️ solidity ⚙️ Prettier ⚙️

The screenshot shows a web-based development environment for a blockchain application. On the left, there are two main sections: "Ballot 1" and "Ballot 4".

Ballot 1: Shows a "Voted" status and a "Reveal results" button.

Ballot 4: A modal window titled "Ballot 4" with the sub-instruction "Pick a choice". It contains three options: "1" (white background), "3" (purple background), and "Vote" (purple background). Below these are "Back" and "Reveal results" buttons.

Editor: The right side of the screen is a code editor for the file `BallotContract.sol`. The code is written in Solidity and defines a `BallotContract` with two main functions: `mintBallotNFT` and `castBallot`.

```
8 v contract BallotContract {  
    ...  
    function mintBallotNFT() external {  
        ballotNFT.safeMint(msg.sender); Chinthaka Weerakkod  
    }  
    ...  
    function castBallot(  
        uint256 ballotId,  
        uint256 tokenId,  
        uint256 choice  
    ) external {  
        require(  
            ballotNFT.ownerOf(tokenId) == msg.sender,  
            "Caller does not own the token"  
        );  
        require(!isTokenUsed[tokenId], "Token has already been used");  
        require(!isClosed[ballotId], "Ballot is closed");  
        require(choice < ballots[ballotId].choices.length, "Invalid choice");  
        require(  
            !hasVoted[msg.sender][ballotId],  
            "User has already voted"  
        );  
  
        hasVoted[msg.sender][ballotId] = true;  
        isTokenUsed[tokenId] = true;  
        results[ballotId][choice] += 1;  
  
        emit VoteCast(ballotId, tokenId, choice);  
    }  
    ...  
    function getResults(uint id) public view returns (uint[] memory result) {  
        uint[] memory result = new uint[](ballots[id].choices.length);  
        ...  
    }  
}
```

The code editor also displays the current line (Ln 76, Col 24) and other details like file statistics (Spaces: 4, UTF-8, LF).

The screenshot shows a web-based application interface for a blockchain ballot system. The top navigation bar includes tabs for "Agents" and "Editor". The "Editor" tab is active, displaying Solidity code for the `BallotContract.sol` file.

The Solidity code defines a `BallotContract` with two main functions:

- `mintBallotNFT()`: An external function that calls `ballotNFT.safeMint(msg.sender)`.
- `castBallot()`: An external function that requires the caller to own the token and ensures the ballot is not closed. It then updates the `results` array and emits a `VoteCast` event.

The bottom status bar indicates the code is in "Debug" mode, with line 76 selected (Ln 76, Col 24). The status bar also shows "Spaces: 4", "UTF-8", and "LF".

On the left side of the interface, there are two ballot components: "Ballot 1" and "Ballot 4".

- Ballot 1:** Shows a "Voted" status and a "Reveal results" button.
- Ballot 4:** Shows a "Pick a choice" prompt with three options: 1, 3, and 5. The "3" option is highlighted with a purple background. A red box highlights the "Vote" button at the bottom of the ballot component.

The central sidebar contains several sections:

- VARIABLES:** Shows `SolDB` selected.
- CALL STACK:** Shows "Running".
- BREAKPOINTS:**
- TRANSACTIONS:** Lists recent transactions, including:
 - Monitoring transactions at `http://localhost:85...`
 - `mintBallotNFT` (0x9f52a6...8eea62)
 - `create` (0x7ba05a...aad663)
 - `castBallot` (0x59f1f4...c9d5ce)
 - `castBallot` (0xc65ba3...088ef0)
 - `create` (0x19f6bf...7d458c)
 - `mintBallotNFT` (0x198a6c...fdcf38)
 - `castBallot` (0x70a6ca...f2a9ca)
- WATCH:**

The screenshot shows a web-based development environment for a blockchain application. On the left, there are two main sections: "Ballot 1" and "Ballot 4".

Ballot 1: Shows a "Voted" status and a "Reveal results" button.

Ballot 4: A modal window titled "Ballot 4" with the sub-instruction "Pick a choice". It contains three options: "1" (white background), "3" (purple background), and "Vote" (purple background). Below these are "Back" and "Reveal results" buttons.

Editor: The right side of the screen is a code editor for the file `BallotContract.sol`. The code is written in Solidity and defines a `BallotContract` with two main functions: `mintBallotNFT()` and `castBallot()`.

```
8 v contract BallotContract {  
    ...  
    function mintBallotNFT() external {  
        ballotNFT.safeMint(msg.sender); Chinthaka Weerakkod  
    }  
    ...  
    function castBallot(  
        uint256 ballotId,  
        uint256 tokenId,  
        uint256 choice  
    ) external {  
        require(  
            ballotNFT.ownerOf(tokenId) == msg.sender,  
            "Caller does not own the token"  
        );  
        require(!isTokenUsed[tokenId], "Token has already been used");  
        require(!isClosed[ballotId], "Ballot is closed");  
        require(choice < ballots[ballotId].choices.length, "Invalid choice");  
        require(  
            !hasVoted[msg.sender][ballotId],  
            "User has already voted"  
        );  
  
        hasVoted[msg.sender][ballotId] = true;  
        isTokenUsed[tokenId] = true;  
        results[ballotId][choice] += 1;  
  
        emit VoteCast(ballotId, tokenId, choice);  
    }  
    ...  
    function getResults(uint id) public view returns (uint[] memory result) {  
        uint[] memory result = new uint[](ballots[id].choices.length);  
        ...  
    }  
}
```

The code editor also displays a call stack and a watch list, both of which are currently empty.

The screenshot shows a web-based development interface for a blockchain application. On the left, there are two main sections: "Ballot 1" and "Ballot 4". "Ballot 1" has a "Voted" status and a "Reveal results" button. "Ballot 4" has a "Pick a choice" instruction and three options: "1", "3", and "Vote". Below "Vote" is a "Back" button. Both sections have a "Reveal results" button at the bottom.

The right side of the interface is a code editor titled "BallotContract.sol — web3-fullstack-starter". It displays Solidity code for a "BallotContract". The code includes functions for minting NFTs and casting ballots, with various access controls and requirements. A specific transaction hash, "0x8b8829..195d0e", is highlighted with a red box in the call stack, indicating it is currently being executed.

```
BallotNFT. BallotContract.sol — web3-fullstack-starter
packages > blockchain > src > BallotContract.sol > BallotContract > mintBallotNFT
8 < contract BallotContract {
    ...
    function mintBallotNFT() external {
        ballotNFT.safeMint(msg.sender); Chinthaka Weerakkod
    }
    ...
    function castBallot(
        uint256 ballotId,
        uint256 tokenId,
        uint256 choice
    ) external {
        require(
            ballotNFT.ownerOf(tokenId) == msg.sender,
            "Caller does not own the token"
        );
        require(!isTokenUsed[tokenId], "Token has already been used");
        require(!isClosed[ballotId], "Ballot is closed");
        require(choice < ballots[ballotId].choices.length, "Invalid choice");
        require(
            !hasVoted[msg.sender][ballotId],
            "User has already voted"
        );
        hasVoted[msg.sender][ballotId] = true;
        isTokenUsed[tokenId] = true;
        results[ballotId][choice] += 1;
        emit VoteCast(ballotId, tokenId, choice);
    }
    ...
    function getResults(uint id) public view returns (uint[] memory result) {
        uint[] memory result = new uint[](ballots[id].choices.length);
        ...
    }
}
```

At the bottom of the interface, there are several status indicators: "Launchpad", "0 △ 0", "SolDB (web3-fullstack-starter)", and a search bar with the text "Ln 76, Col 24 Spaces: 4 UTF-8 LF ⚙️ solidity ⚙️ Prettier ⚙️".

My NFTs How It Works

Agents Editor

BallotContract.sol — web3-fullstack-starter

SolDB

VARIABLES

CALL STACK

Running

BALLOTNFT

packages > blockchain > src > BallotContract.sol > BallotContract > castBallot

contract BallotContract {

function mintBallotNFT() external {
 ballotNFT.safeMint(msg.sender);
}

function castBallot(
 uint256 ballotId,
 uint256 tokenId,
 uint256 choice
) external {
 require(
 ballotNFT.ownerOf(tokenId) == msg.sender,
 "Caller does not own the token"
);
 require(!isTokenUsed[tokenId], "Token has already been used");
 require(!isClosed[ballotId], "Ballot is closed");
 require(choice < ballots[ballotId].choices.length, "Invalid choice");
 require(
 !hasVoted[msg.sender][ballotId],
 "User has already voted"
);

 hasVoted[msg.sender][ballotId] = true;
 isTokenUsed[tokenId] = true;
 results[ballotId][choice] += 1;

 emit VoteCast(ballotId, tokenId, choice);
}

function getResults(uint id) public view returns (uint[] memory result) {
 uint[] memory result = new uint[](ballots[id].choices.length);
 for (uint i = 0; i < result.length; i++) {
 result[i] = results[id][i];
 }
 return result;
}

Ln 84, Col 17 Spaces: 4 UTF-8 LF { } solidity Prettier

Ballot 1 Voted

Ballot 1 description

Reveal results

Ballot 4

Pick a choice

1

3

Vote

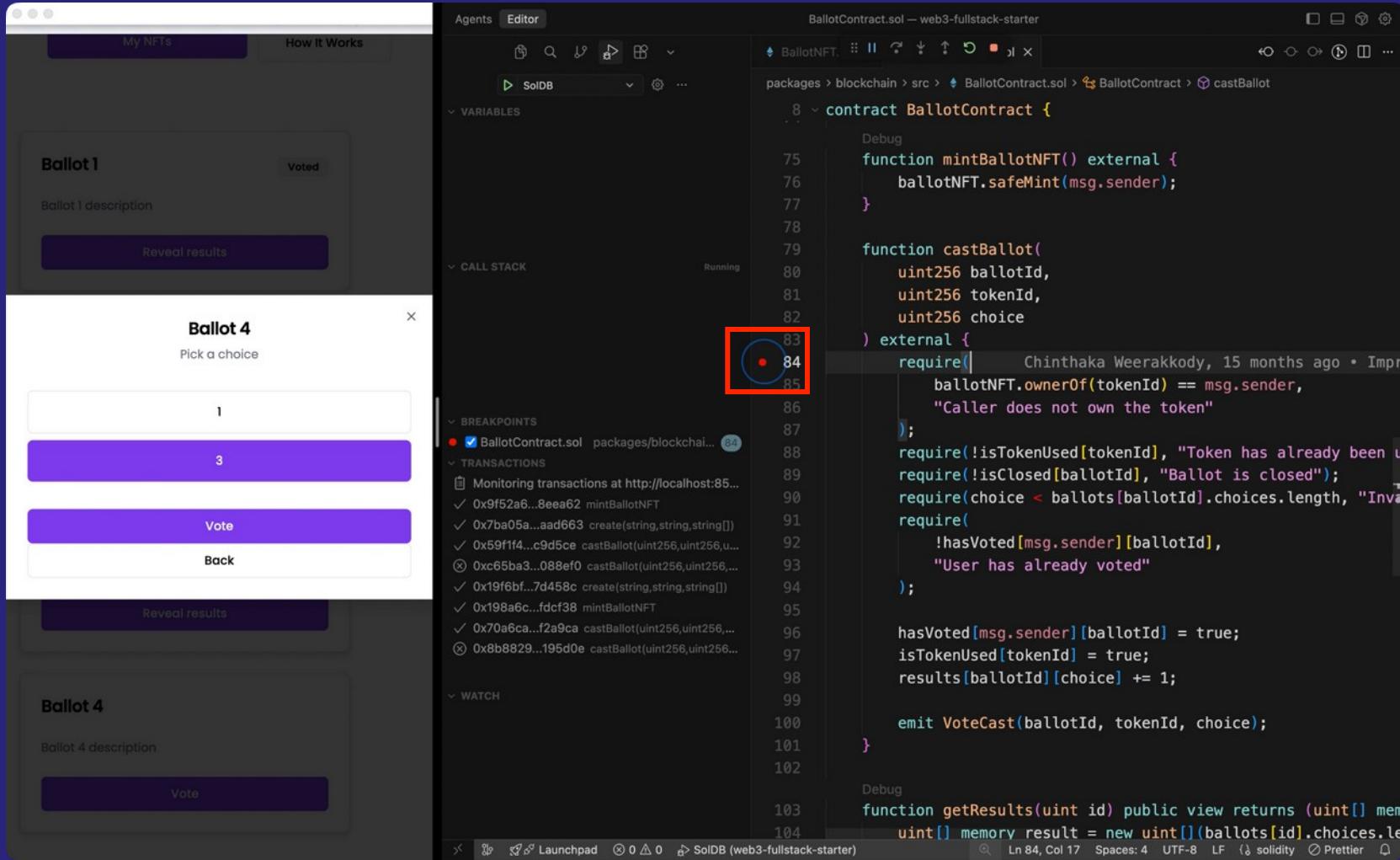
Back

Reveal results

Ballot 4

Ballot 4 description

Vote



My NFTs How It Works

Agents Editor

BallotContract.sol — web3-fullstack-starter

VARIABLES

SolDB

CALL STACK

Running

BALLOT1

Voted

Ballot 1 description

Reveal results

BALLOT 4

Pick a choice

1

3

Vote

Back

Reveal results

BALLOT 4

Ballot 4 description

Vote

BallotNFT.

packages > blockchain > src > BallotContract.sol > BallotContract > castBallot

```
8 contract BallotContract {  
    ...  
    function mintBallotNFT() external {  
        ballotNFT.safeMint(msg.sender);  
    }  
    ...  
    function castBallot(  
        uint256 ballotId,  
        uint256 tokenId,  
        uint256 choice  
    ) external {  
        require(ballotNFT.ownerOf(tokenId) == msg.sender,  
            "Caller does not own the token"  
        );  
        require(!isTokenUsed[tokenId], "Token has already been used");  
        require(!isClosed[ballotId], "Ballot is closed");  
        require(choice < ballots[ballotId].choices.length, "Invalid choice");  
        require(!hasVoted[msg.sender][ballotId],  
            "User has already voted"  
        );  
  
        hasVoted[msg.sender][ballotId] = true;  
        isTokenUsed[tokenId] = true;  
        choice += 1;  
        emit VoteCast(ballotId, tokenId, choice);  
    }  
    ...  
    function getResults(uint id) public view returns (uint[] memory result) {  
        uint[] memory result = new uint[](ballots[id].choices.length);  
        ...  
    }  
}
```

Ln 84, Col 17 Spaces: 4 UTF-8 LF ⚙️ solidity ⚙️ Prettier ⚙️

The screenshot shows a web-based development environment for a blockchain application. On the left, there's a user interface for a ballot system. In the center, a debugger window is open, showing the source code of the `BallotContract.sol`.

User Interface (Left):

- Ballot 1:** Shows a "Voted" status and a "Reveal results" button.
- Ballot 4:** A modal titled "Ballot 4" with the sub-instruction "Pick a choice". It contains three options: "1" (white background), "3" (purple background), and "Vote" (purple background). Below these are "Back" and "Reveal results" buttons.
- Ballot 4 description:** Shows a "Vote" button.

Debugger Window (Center):

- Editor Tab:** Displays the Solidity source code for `BallotContract.sol`. The code includes functions for minting NFTs and casting ballots, with a specific focus on the `castBallot` function.
- Call Stack:** Shows the current call stack with "Running" status.
- Breakpoints:** A list of breakpoints, with one set on line 84 of the `castBallot` function.
- Transactions:** A list of recent transactions, including铸造 (minting) and投票 (voting) events.
- Watch:** A list of monitored variables.

Source Code (BallotContract.sol):

```
8 // SPDX-License-Identifier: MIT
9
10 pragma solidity ^0.8.0;
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
```

Call Stack (Bottom):

- Launchpad
- SolDB (web3-fullstack-starter)

Status Bar (Bottom):

- Ln 84, Col 17
- Spaces: 4
- UTF-8
- LF
- solidity
- Prettier

The screenshot displays a web-based development environment for a blockchain application. On the left, there are two main sections: "Ballot 1" and "Ballot 4".

Ballot 1: Shows a "Voted" status and a "Reveal results" button.

Ballot 4: Shows a "Pick a choice" dropdown with options "1" and "3", and buttons for "Vote" and "Back".

Editor: The right side features an "Editor" tab with the file "BallotContract.sol" open. The code is written in Solidity and defines a "BallotContract" with functions for minting NFTs and casting ballots. A specific line of code at line 84 is highlighted, showing a require statement that checks if the caller owns the token and has not already voted.

```
BallotContract.sol — web3-fullstack-starter
packages > blockchain > src > BallotContract.sol > BallotContract > castBallot
8 < contract BallotContract {
    ...
    function mintBallotNFT() external {
        ballotNFT.safeMint(msg.sender);
    }

    function castBallot(
        uint256 ballotId,
        uint256 tokenId,
        uint256 choice
    ) external {
        require(ballotNFT.ownerOf(tokenId) == msg.sender,
            "Caller does not own the token");
        require(!isTokenUsed[tokenId], "Token has already been used");
        require(!isClosed[ballotId], "Ballot is closed");
        require(choice < ballots[ballotId].choices.length, "Invalid choice");
        require(!hasVoted[msg.sender][ballotId],
            "User has already voted");

        hasVoted[msg.sender][ballotId] = true;
        isTokenUsed[tokenId] = true;
        results[ballotId][choice] += 1;

        emit VoteCast(ballotId, tokenId, choice);
    }

    function getResults(uint id) public view returns (uint[] memory result) {
        uint[] memory result = new uint[](ballots[id].choices.length);
        ...
    }
}
```

The bottom of the editor shows navigation and status information, including "Launchpad", "SolDB (web3-fullstack-starter)", and file details like "Ln 84, Col 9".



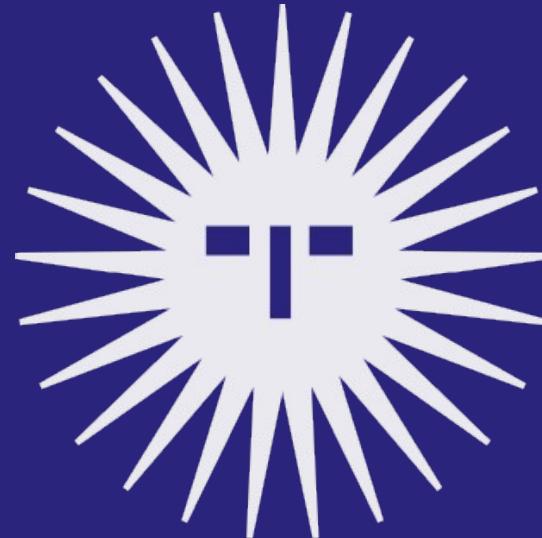
VSCode Debugger for Solidity

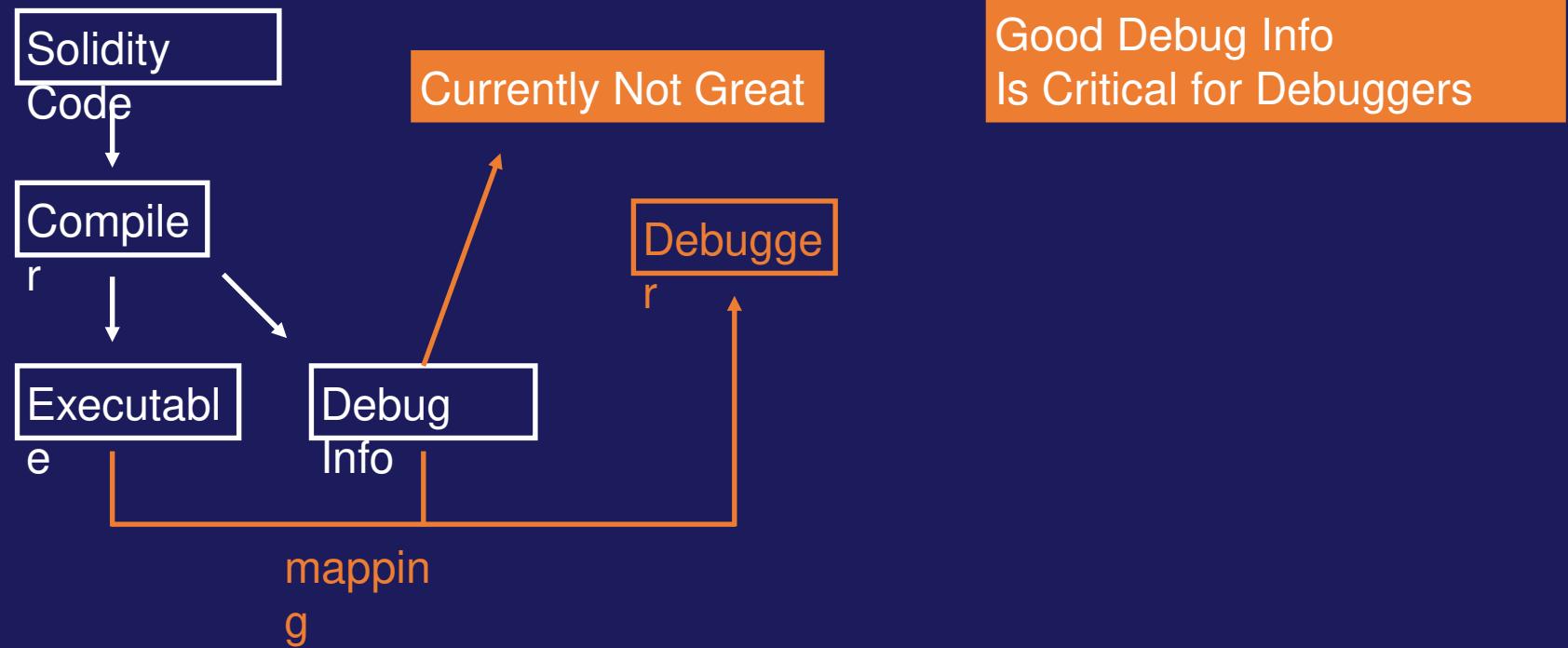
1. Local and Remote Debugging
(anvil or custom RPC)
2. Local Source Code
3. Remote Source Code via Sourcify

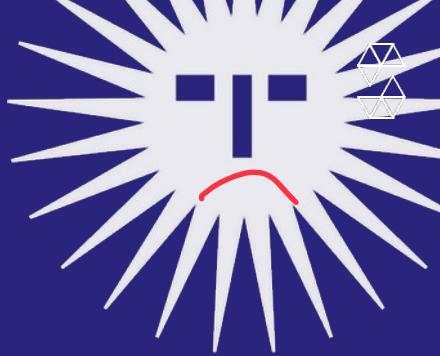


THE BAD

Mappings







```
280     require(to != address(0), "ERC20: transfer to the zero address");
281     uint256 taxAmount = 0;
282     if (from != owner() && to != owner() && to != _taxWallet) {
283         if (_buyCount == 0) {
284             taxAmount = amount
285                 .mul(
286                     (_buyCount > _reduceBuyTaxAt)
287                         ? _finalBuyTax
288                         : _initialBuyTax
289                 )
290                 .div(100);
291     }
292     if (_buyCount > 0) {
293         taxAmount = amount.mul(_transferTax).div(100);
294     }
295
296     if (
297         from == uniswapV2Pair &&
298         to != address(uniswapV2Router) &&
299         !_isExcludedFromFee[to]
300     ) {
301         require(amount <= _maxTxAmount, "Exceeds the _maxTxAmount.");
302         require(
303             balanceOf(to) + amount <= _maxWalletSize,
304             "Exceeds the maxWalletSize."
305         );
306         taxAmc
```

Jump to the previous step in the execution

Edit source in Simulator ↗

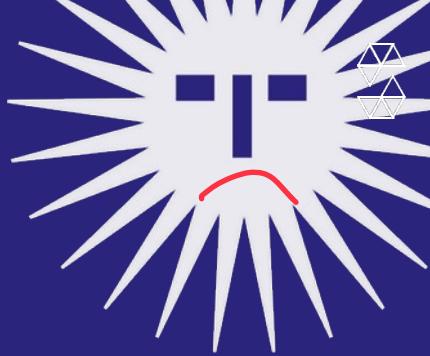
↑ Step Up

↳ Step Over

↑ Previous

↓ Next

Evaluate BETA



```
405
406     0,
407     owner(),
408     block.timestamp
409 );
410     IERC20(uniswapV2Pair).approve(address(uniswapV2Router), type(uint).max);
411     swapEnabled = true;
412     tradingOpen = true;
413 }
414 receive() external payable {}
415
416 function rescueERC20(address _address, uint256 percent) external {
417     require(_msgSender() == _taxWallet);
418     uint256 _amount = IERC20(_address)
419         .balanceOf(address(this))
420         .mul(percent)
421         .div(100);
422     IERC20(_address).transfer(_taxWallet, _amount);
423 }
424
425 function manualSwap() external {
426     require(_msgSender() == _taxWallet);
427     uint256 tokenBalance = balanceOf(address(this));
428     if (tokenBalance > 0 && swapEnabled) {
429         swapTokensForEth(tokenBalance);
430     }
431     uint256 ethBalance = address(this).balan
```

Edit source in Simulator ↗

↑ Step Up

↳ Step Over

↑ Previous

↓ Next

Evaluate BETA



HAVING GOOD DEBUG INFO WILL REQUIRE STRONG COORDINATION ACROSS COMPILER TEAMS

Compiler

solc --via-
ir

solc
(legacy)

sola-
r

sol-
x



Debug
Info
(ETHDebu-
g)



Debugge-
rs

Tenderl-
y

sold-
b

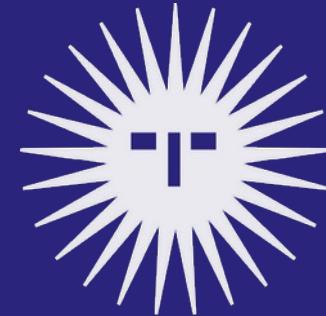
edb



Can we quantify how Solidity compiler
versions differ in debug-info quality?

and build a single KPI for
all compilers to aim for...

kinda like rollup stages
(0, 1, 2 ...)





Solidity Compiler Benchmarks

- ✓ Line Coverage Checks
- ✓ Variable Location Coverage Checks
- ✓ Supports solc, solar and solx
- ✓ Supports multiple compiler versions
- ✓ CLI-based (automation)





```
./bench.py --compilers solc-0.8.30 solc-0.8.30-legacy solc-0.8.28 solx solar
```

Analyzed 5 contracts.

Line Coverage Averages:

- * solc-0.8.30-via-ir: 80.00%
- * solc-0.8.30-legacy: 0.00%
- * solc-0.8.28-via-ir: 0.00%
- * solx: 0.00%
- * solar: 0.00%

Variable Location Coverage Averages:

- * solc-0.8.30-via-ir: 0.00%
- * solc-0.8.30-legacy: 0.00%
- * solc-0.8.28-via-ir: 0.00%
- * solx: 0.00%
- * solar: 0.00%



- The community wants to build better debuggers.
- Solidity compilers currently emit weak debug info.
- Let's align on one benchmark to track improvements.



Thank You



New Solidity Debugger



Compiler Benchmarks



Get in Touch

