Debugging transactions

This walkthrough tutorial covers debugging transaction using Remix IDE

Debugging transactions

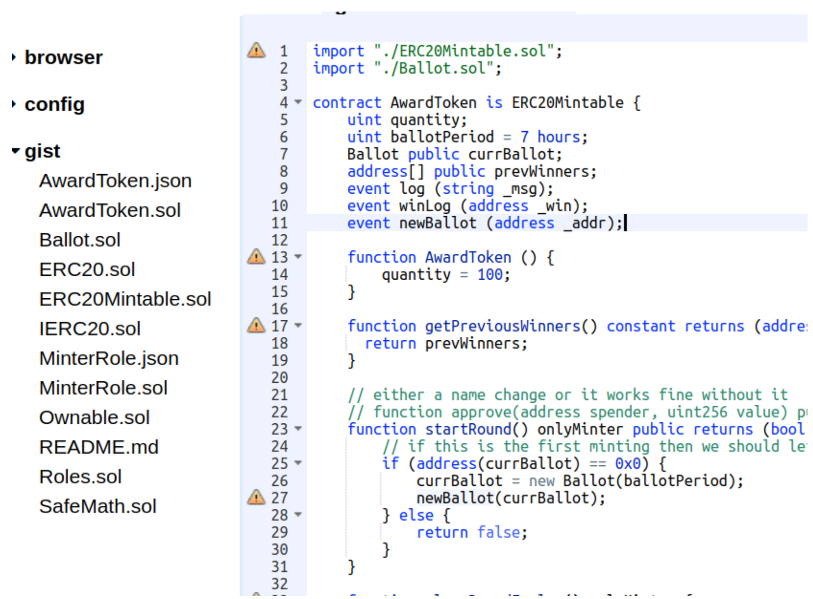
We will use the AwardToken in this tutorial as a short example for debugging the AwardToken.

Debugging transactions

Let’s keep the loading process simple,

Run in the terminal

`remix.loadgist('add9633c2b0101f6fda0aadcfe350f60')`

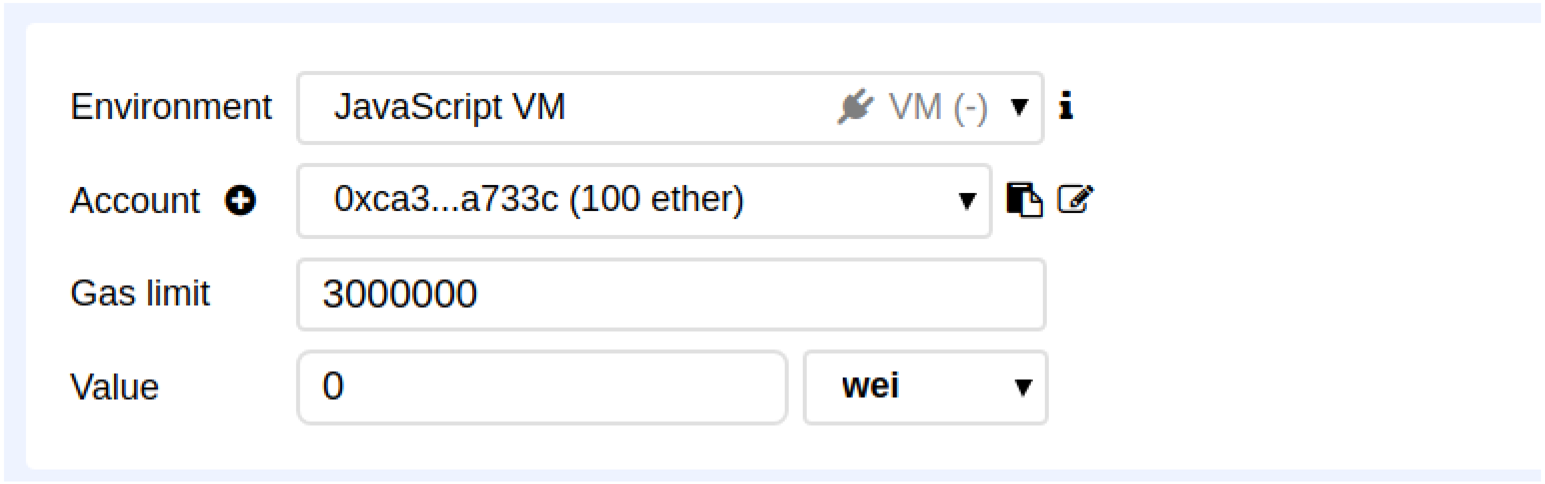


- browser
- config
- gist
  - AwardToken.json
  - AwardToken.sol
  - Ballot.sol
  - ERC20.sol
  - ERC20Mintable.sol
  - IERC20.sol
  - MinterRole.json
  - MinterRole.sol
  - Ownable.sol
  - README.md
  - Roles.sol
  - SafeMath.sol

```
import "./ERC20Mintable.sol";
import "./Ballot.sol";

contract AwardToken is ERC20Mintable {
    uint quantity;
    uint ballotPeriod = 7 hours;
    Ballot public currBallot;
    address[] public prevWinners;
    event log (string _msg);
    event winLog (address _win);
    event newBallot (address _addr);

    function AwardToken () {
        quantity = 100;
    }

    function getPreviousWinners() constant returns (addre
      return prevWinners;
    }

    // either a name change or it works fine without it
    // function approve(address spender, uint256 value) p
    function startRound() onlyMinter public returns (bool
        // if this is the first minting then we should le
        if (address(currBallot) == 0x0) {
            currBallot = new Ballot(ballotPeriod);
            newBallot(currBallot);
        } else {
            return false;
        }
    }

```

Debugging transactions

In the run tab, make sure the JavaScript VM is selected in the environment section.



Environment: JavaScript VM — VM (-)

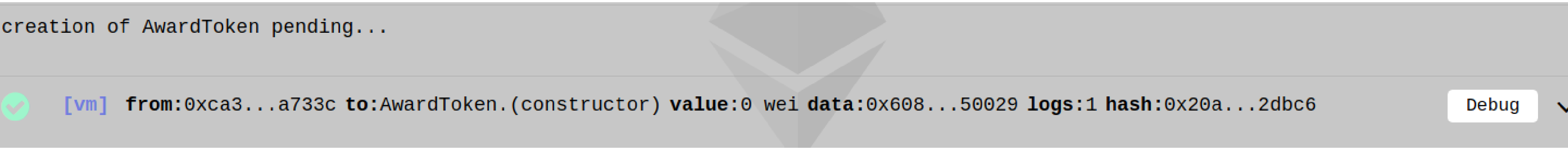
Account: 0xca3...a733c (100 ether)

Gas limit: 3000000

Value: 0 wei

Debugging transactions

Deploy the AwardToken



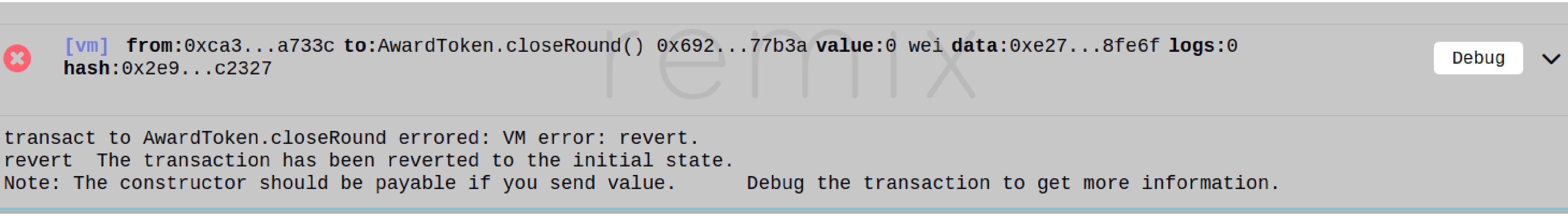
```
creation of AwardToken pending...

[vm] from:0xca3...a733c to:AwardToken.(constructor) value:0 wei data:0x608...50029 logs:1 hash:0x20a...2dbc6
```

Debugging transactions

Let’s try a function which should fail ;)

Execute ‘CloseRound’, it should fails because the round is not even started.

But let’s debug it.

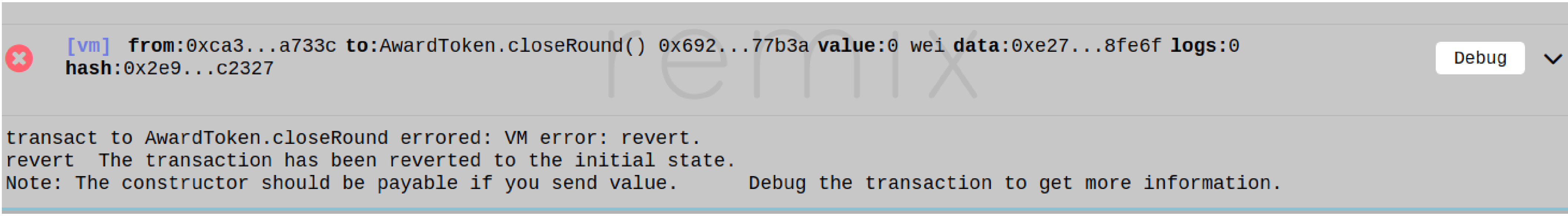


```
[vm] from:0xca3...a733c to:AwardToken.closeRound() 0x692...77b3a value:0 wei data:0xe27...8fe6f logs:0
hash:0x2e9...c2327
```

```
transact to AwardToken.closeRound errored: VM error: revert.
revert   The transaction has been reverted to the initial state.
Note: The constructor should be payable if you send value.     Debug the transaction to get more information.
```

Debugging transactions

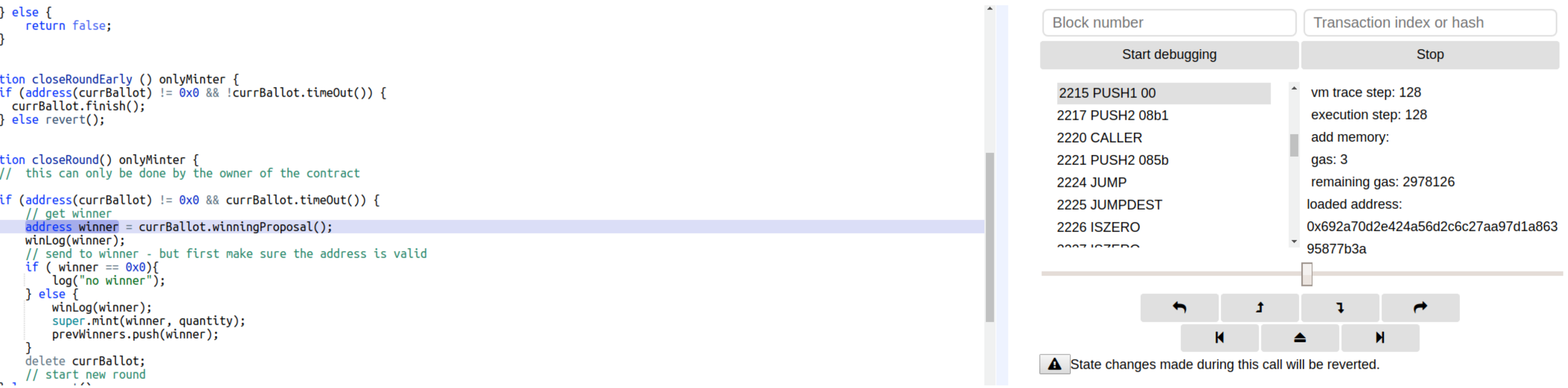
Click on ‘debug’



```
[vm] from:0xca3...a733c to:AwardToken.closeRound() 0x692...77b3a value:0 wei data:0xe27...8fe6f logs:0
hash:0x2e9...c2327

transact to AwardToken.closeRound errored: VM error: revert.
revert  The transaction has been reverted to the initial state.
Note: The constructor should be payable if you send value.     Debug the transaction to get more information.
```

Debugging transactions

Use the slide or the navigation action to go forward or backward.



```
} else {
    return false;
}

tion closeRoundEarly () onlyMinter {
if (address(currBallot) != 0x0 && !currBallot.timeOut()) {
  currBallot.finish();
} else revert();

tion closeRound() onlyMinter {
//  this can only be done by the owner of the contract

if (address(currBallot) != 0x0 && currBallot.timeOut()) {
    // get winner
    address winner = currBallot.winningProposal();
    winLog(winner);
    // send to winner - but first make sure the address is valid
    if ( winner == 0x0){
        log("no winner");
    } else {
        winLog(winner);
        super.mint(winner, quantity);
        prevWinners.push(winner);
    }
    delete currBallot;
    // start new round
```

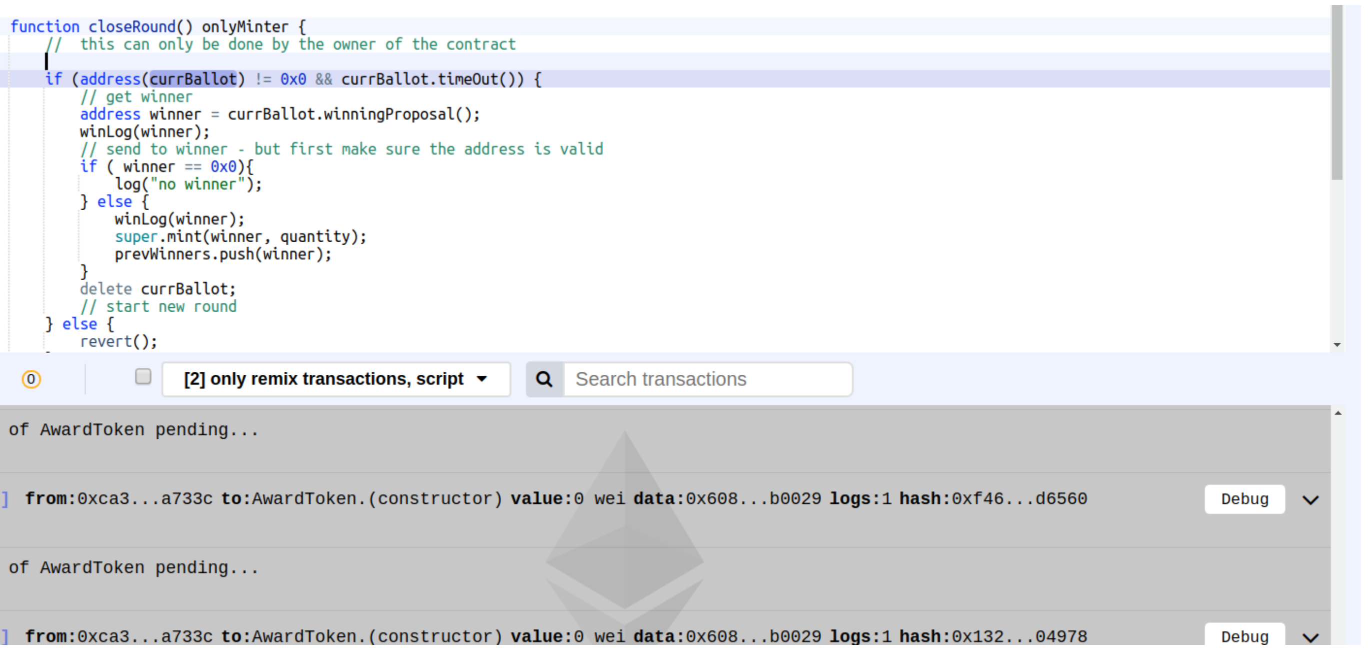
```
2215 PUSH1 00
2217 PUSH2 08b1
2220 CALLER
2221 PUSH2 085b
2224 JUMP
2225 JUMPDEST
2226 ISZERO
```

```
vm trace step: 128
execution step: 128
add memory:
gas: 3
remaining gas: 2978126
loaded address:
0x692a70d2e424a56d2c6c27aa97d1a863
95877b3a
```

State changes made during this call will be reverted.

Debugging transactions

Solidity local variables and state variable can also help figuring out why the transaction fails.

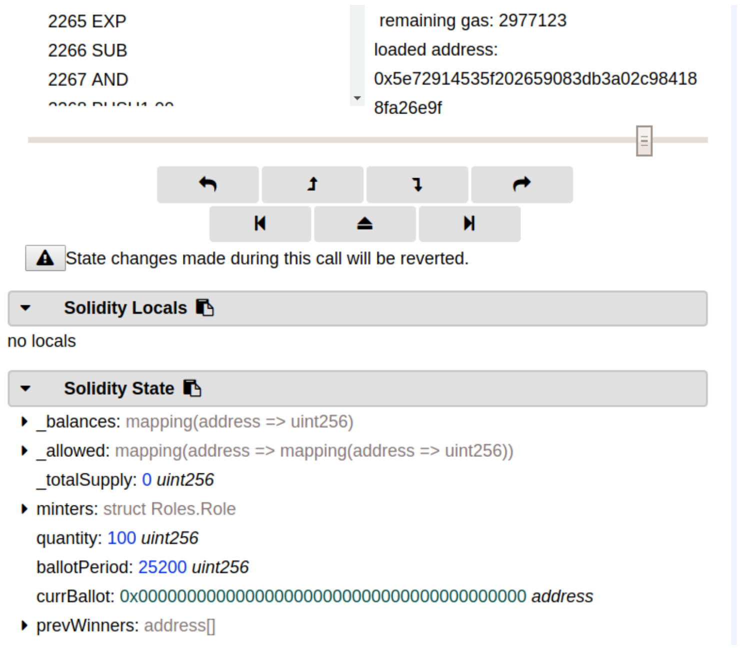


```
function closeRound() onlyMinter {
    //  this can only be done by the owner of the contract

    if (address(currBallot) != 0x0 && currBallot.timeOut()) {
        // get winner
        address winner = currBallot.winningProposal();
        winLog(winner);
        // send to winner - but first make sure the address is valid
        if ( winner == 0x0){
            log("no winner");
        } else {
            winLog(winner);
            super.mint(winner, quantity);
            prevWinners.push(winner);
        }
        delete currBallot;
        // start new round
    } else {
        revert();
```

[2] only remix transactions, script

Search transactions

```
of AwardToken pending...

] from:0xca3...a733c to:AwardToken.(constructor) value:0 wei data:0x608...b0029 logs:1 hash:0xf46...d6560

of AwardToken pending...

] from:0xca3...a733c to:AwardToken.(constructor) value:0 wei data:0x608...b0029 logs:1 hash:0x132...04978
```



```
2265 EXP
2266 SUB
2267 AND
```

remaining gas: 2977123
loaded address:
0x5e72914535f202659083db3a02c98418
8fa26e9f

State changes made during this call will be reverted.

Solidity Locals

no locals

Solidity State

- _balances: mapping(address => uint256)
- _allowed: mapping(address => mapping(address => uint256))
- _totalSupply: 0 *uint256*
- minters: struct Roles.Role
- quantity: 100 *uint256*
- ballotPeriod: 25200 *uint256*
- currBallot: 0x0000000000000000000000000000000000000000 *address*
- prevWinners: address[]

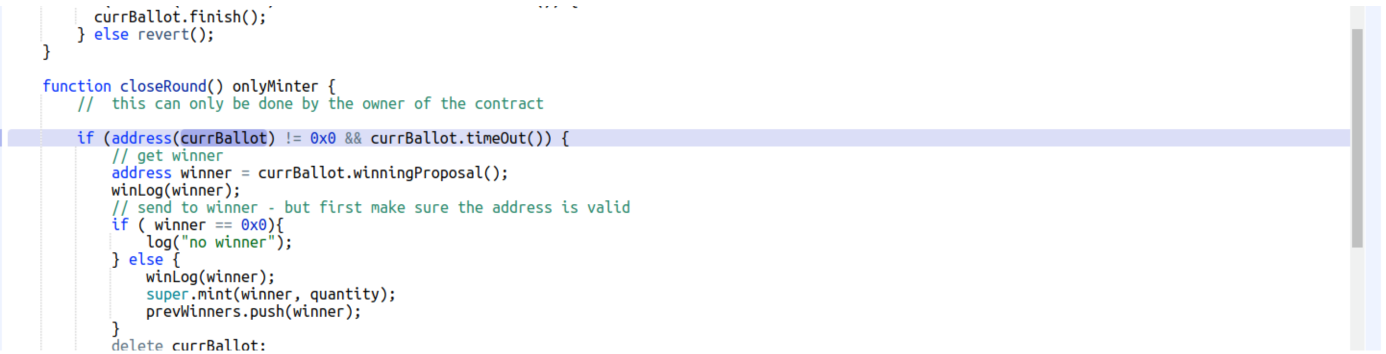
Debugging transactions

Now we just let you try to find why it has failed :-)

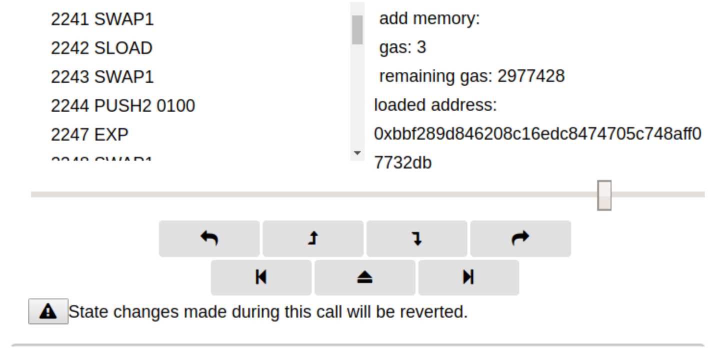
Let’s continue a little bit...

Debugging transactions

Moreover adding breakpoint and clicking on ‘jump to next breakpoint’ would help reaching the part of code you want to verify.



```
        currBallot.finish();
    } else revert();
}

function closeRound() onlyMinter {
    //  this can only be done by the owner of the contract

    if (address(currBallot) != 0x0 && currBallot.timeOut()) {
        // get winner
        address winner = currBallot.winningProposal();
        winLog(winner);
        // send to winner - but first make sure the address is valid
        if ( winner == 0x0){
            log("no winner");
        } else {
            winLog(winner);
            super.mint(winner, quantity);
            prevWinners.push(winner);
        }
        delete currBallot;
```



```
2241 SWAP1          add memory:
2242 SLOAD          gas: 3
2243 SWAP1          remaining gas: 2977428
2244 PUSH2 0100     loaded address:
2247 EXP            0xbbf289d846208c16edc8474705c748aff0
                    7732db
```

State changes made during this call will be reverted.

Debugging live transactions

The Ethereum Foundation and Infura are maintaining nodes dedicated on providing debugging support.

Supported networks are: Mainnet, Ropsten and Rinkeby.

it allows:

- deploying contract and debugging it against a “live” network (in comparison to the JavaScript VM).
- Take any “recent” transactions and debug it with source highlighting support. let’s dig a bit in that direction:

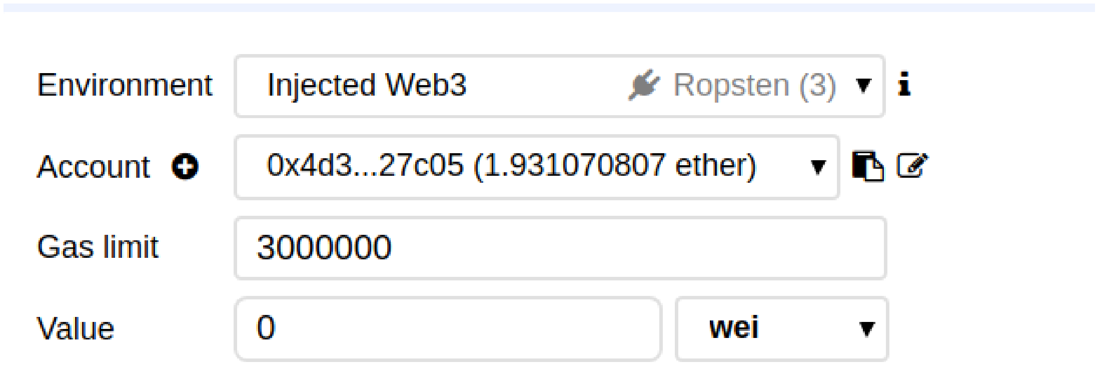
We just deployed the AwardToken contract yesterday and called the ‘startRound’ through the following transaction:

https://ropsten.etherscan.io/tx/0x3ec898b5c0e0fdfa48e5414ce20850552bbb7391f962dce09b1924f2a1f43bd2

We already have the source code, but it would be easy to copy/paste the code from EtherScan (if the contract is verified).

Anyway,

Please switch to Ropsten



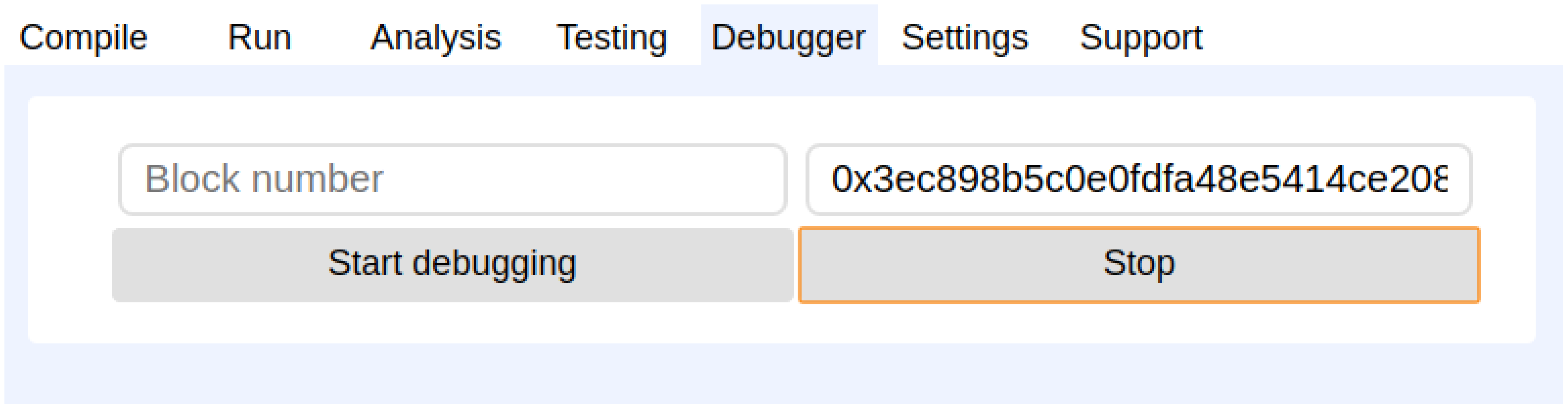
Environment	Injected Web3 · Ropsten (3)
Account	0x4d3...27c05 (1.931070807 ether)
Gas limit	3000000
Value	0 wei

Take the following transaction hash:

`0x3ec898b5c0e0fdfa48e5414ce20850552bbb7391f962dce09b1924f2a1f43bd2`

Select the debugger tab

input the transaction hash, and start debugging


Compile
Run
Analysis
Testing
Debugger
Settings
Support
Block number
0x3ec898b5c0e0fdfa48e5414ce208
Start debugging
Stop