

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique



Union – Discipline – Travail



Institut National Polytechnique
Félix HOUPHOUËT-BOIGNY



BLOCKCHAIN

TP2

ENSEIGNANT

M.DJICKO BONNAI

Enseignant Vacataire à l'INPHB

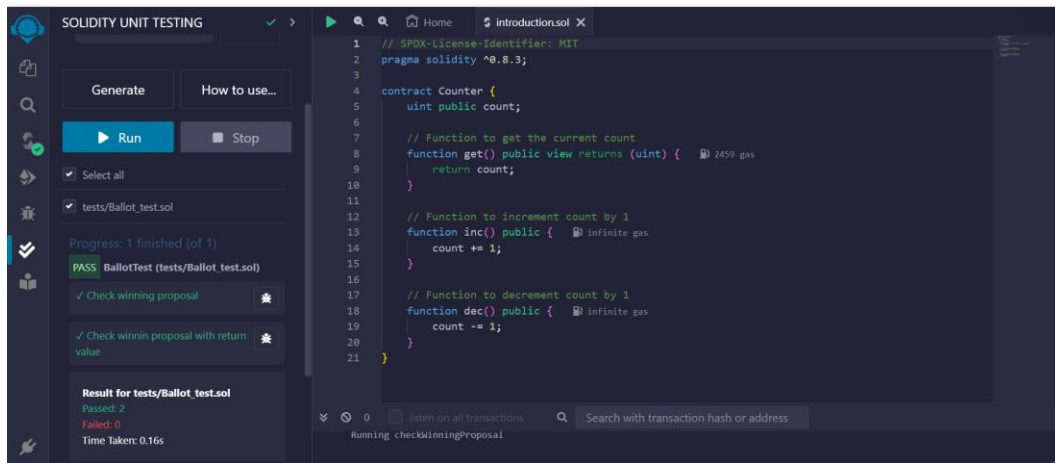
REALISE PAR

LORNG YED BRUCE

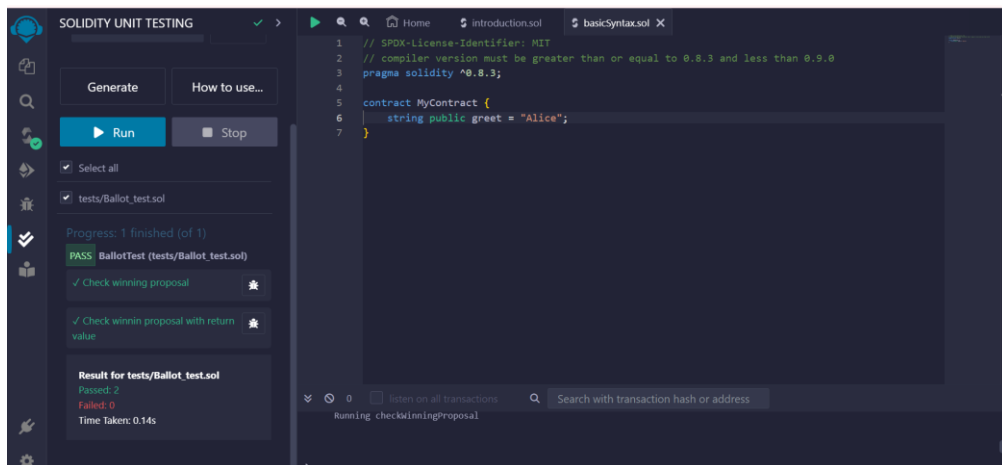
Elèves Ingénieurs en STIC – TLR 3^e année

Année académique : 2023-2024

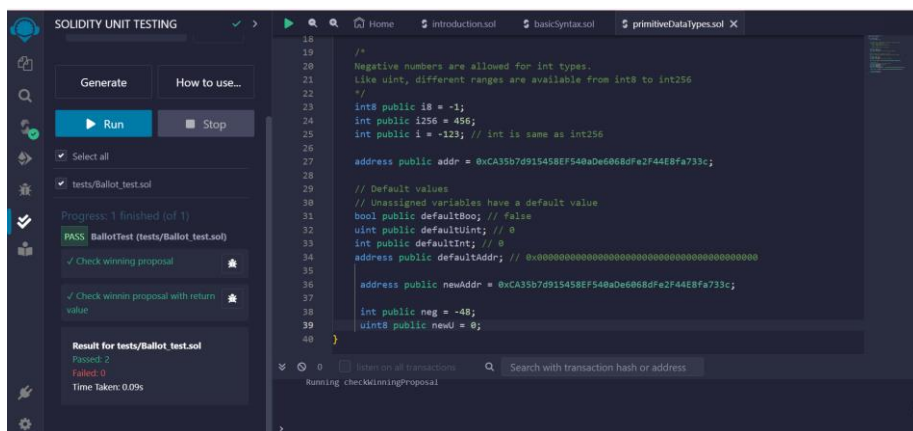
1.



2.



3.

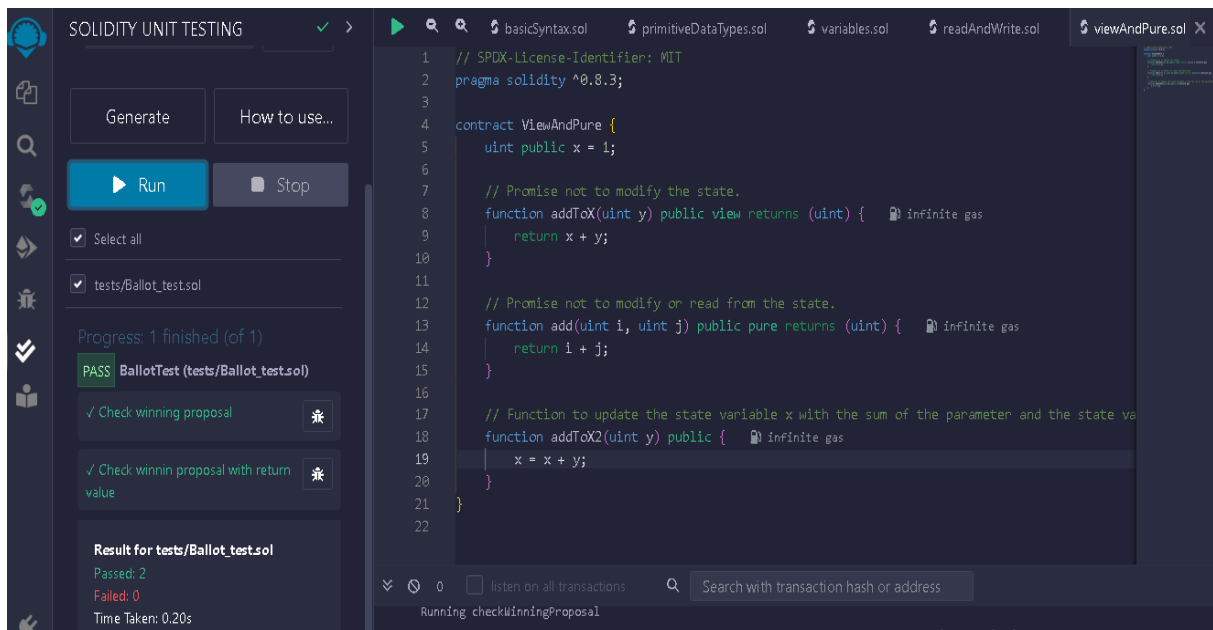


4.

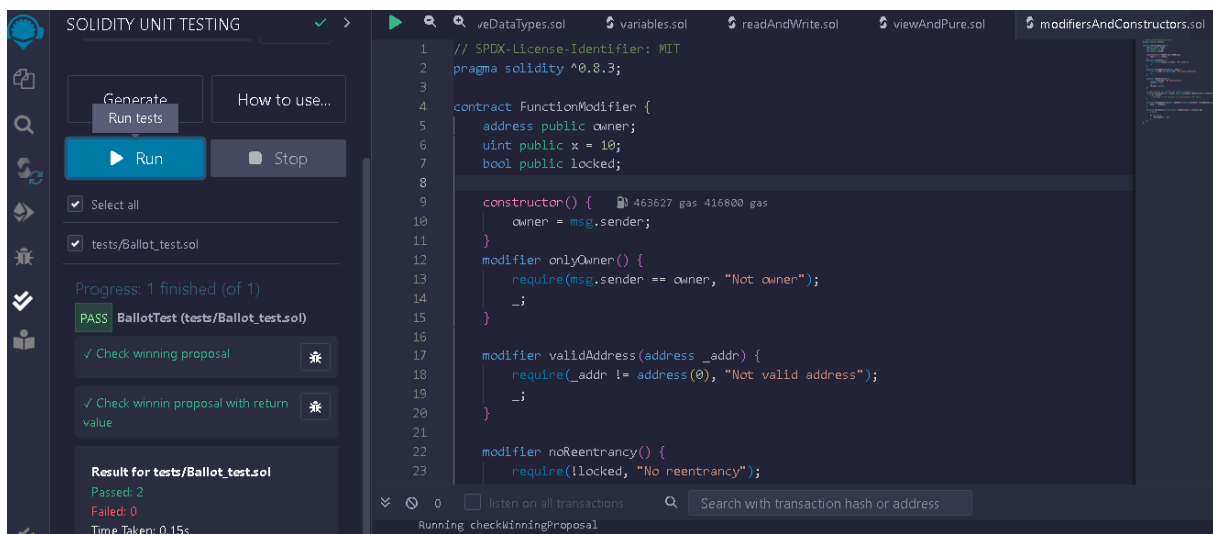
The screenshot shows the Solidity Unit Testing interface. On the left, the 'SOLIDITY UNIT TESTING' sidebar is visible with buttons for 'Generate', 'How to use...', 'Run', and 'Stop'. Below these, there are checkboxes for 'Select all' and 'tests/Ballot_test.sol'. The progress bar indicates 'Progress: 1 finished (of 1)' and 'PASS: BallotTest (tests/Ballot_test.sol)'. Two test cases are listed: '✓ Check winning proposal' and '✓ Check winning proposal with return value', both marked with a star icon. The 'Result for tests/Ballot_test.sol' section shows 'Passed: 2', 'Failed: 0', and 'Time Taken: 0.12s'. The main editor displays the Solidity code for the 'Variables' contract, which includes state variables for 'text', 'num', and 'blockNumber', and a 'doSomething()' function. The bottom status bar shows 'Running checkWinningProposal'.

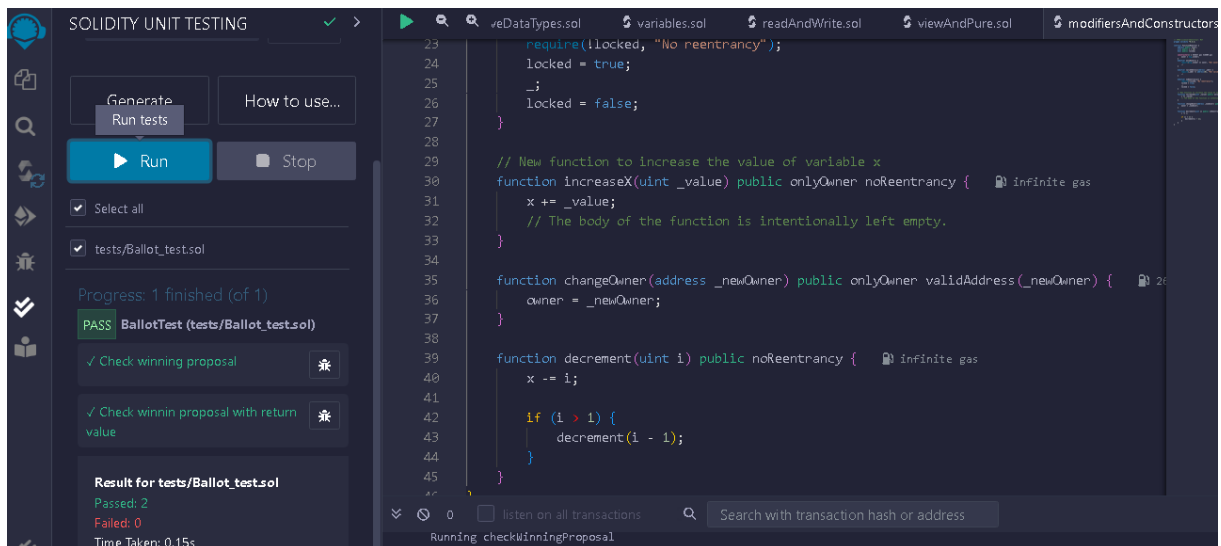
The screenshot shows the Solidity Unit Testing interface. On the left, the 'SOLIDITY UNIT TESTING' sidebar is visible with buttons for 'Generate', 'How to use...', 'Run', and 'Stop'. Below these, there are checkboxes for 'Select all' and 'tests/Ballot_test.sol'. The progress bar indicates 'Progress: 1 finished (of 1)' and 'PASS: BallotTest (tests/Ballot_test.sol)'. Two test cases are listed: '✓ Check winning proposal' and '✓ Check winning proposal with return value', both marked with a star icon. The 'Result for tests/Ballot_test.sol' section shows 'Passed: 2', 'Failed: 0', and 'Time Taken: 0.15s'. The main editor displays the Solidity code for the 'SimpleStorage' contract, which includes state variables for 'num' and 'b', and functions 'set()', 'get()', and 'get_b()'. The bottom status bar shows 'Running checkWinningProposal'.

5.

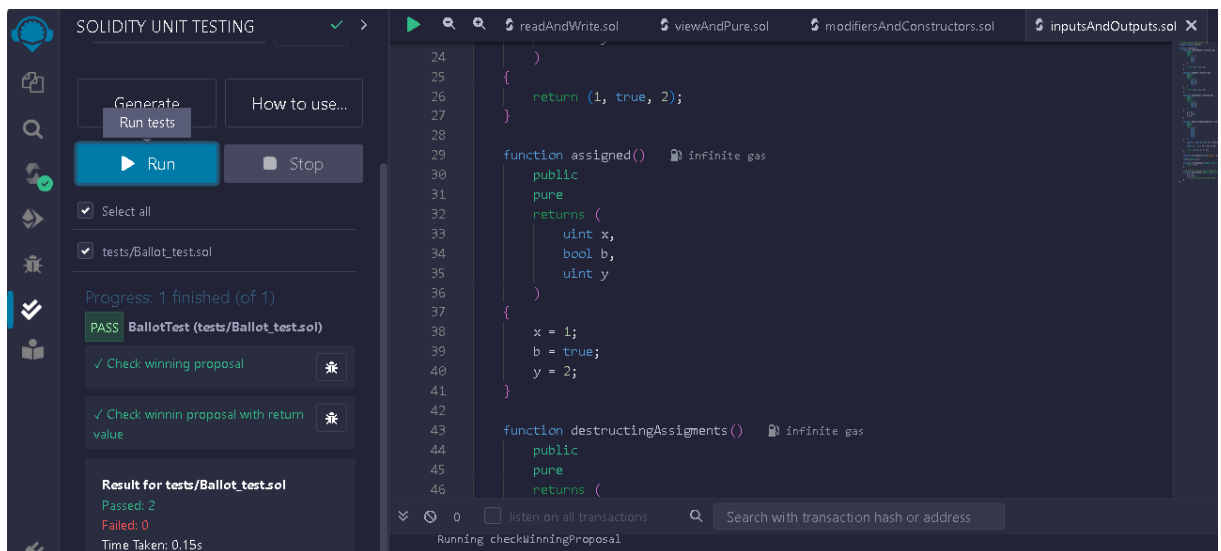
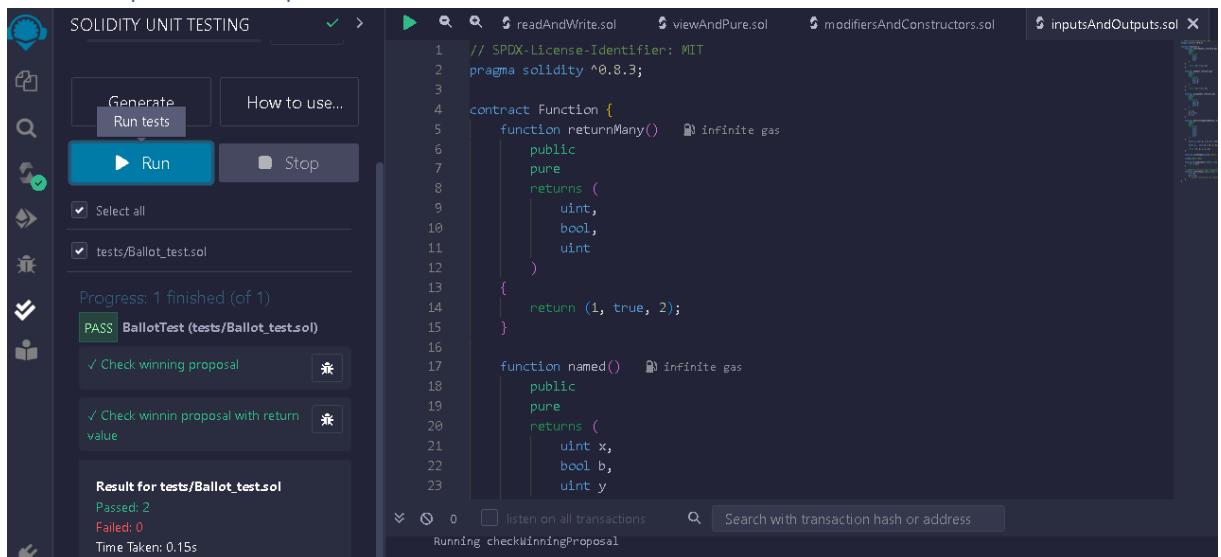


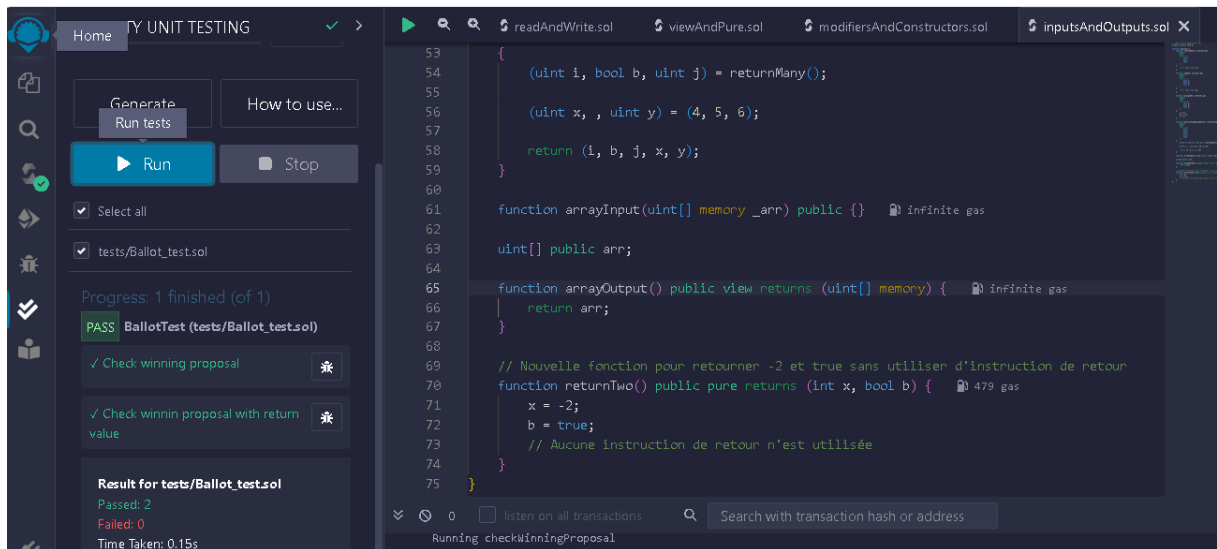
6. ModifiersAndConstructor.sol



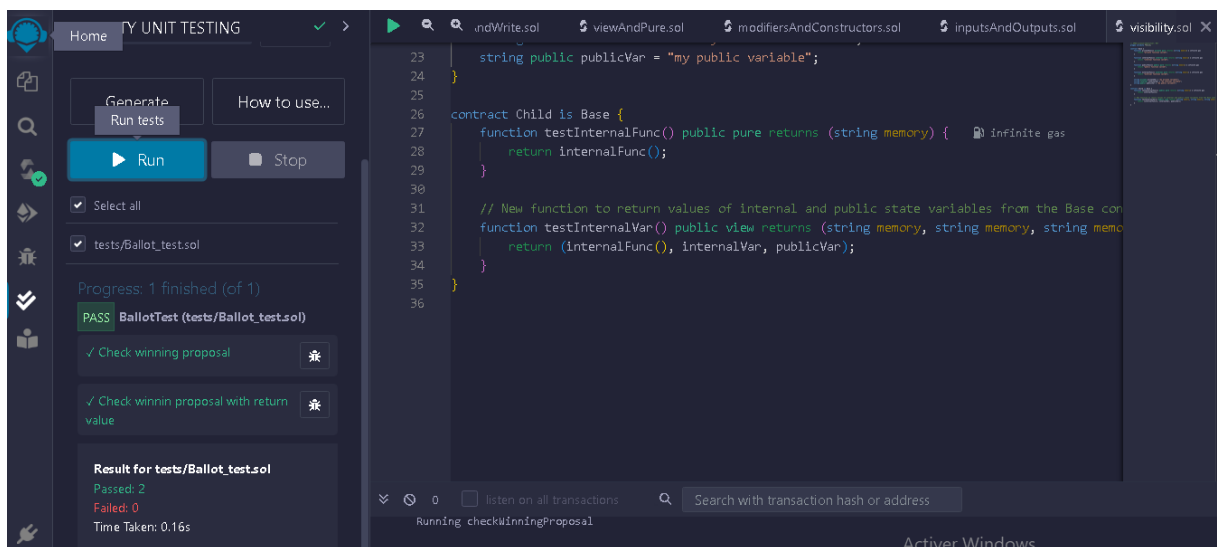
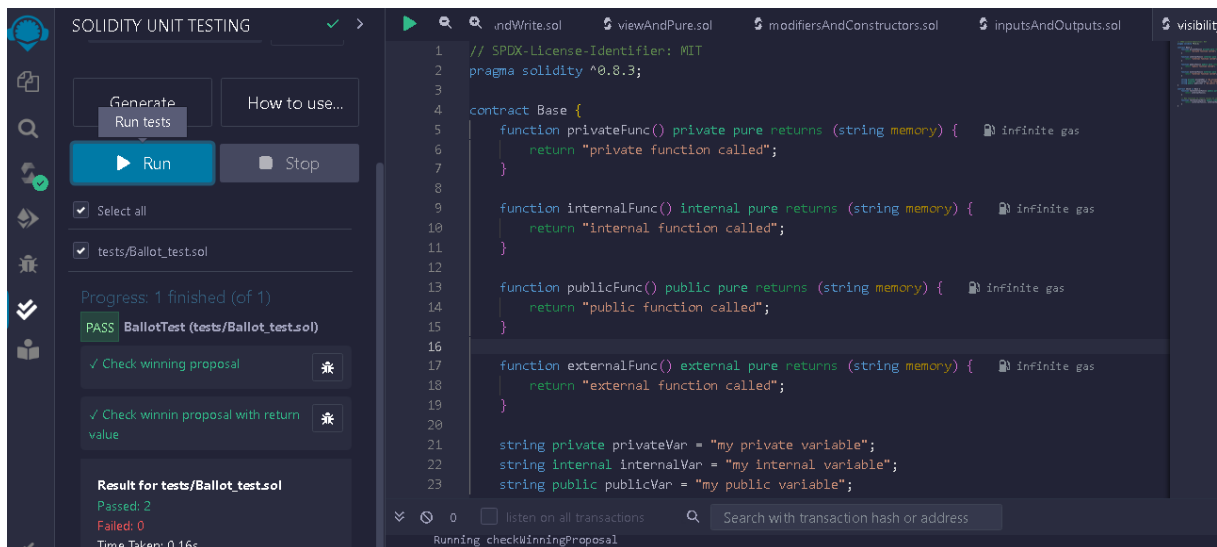


7. InputAndOutput.sol





8. Visibility.sol



9. IfElse.sol

The screenshot shows the Solidity Unit Testing interface. On the left, a sidebar contains a 'Run' button and a list of tests. The main area displays the Solidity code for the 'IfElse' contract. The code includes a 'foo' function with an 'if-else' statement and a 'ternary' function using a ternary operator. The test results on the left indicate that the tests passed.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract IfElse {
5     function foo(uint x) public pure returns (uint) {
6         if (x < 10) {
7             return 0;
8         } else if (x < 20) {
9             return 1;
10        } else {
11            return 2;
12        }
13    }
14
15    function ternary(uint _x) public pure returns (uint) {
16        // shorthand way to write if / else statement
17        return _x < 10 ? 1 : 2;
18    }
19
20    // New function evenCheck using ternary operator
21    function evenCheck(uint _num) public pure returns (bool) {
22        return _num % 2 == 0 ? true : false;
23    }
24 }
```

Test Results:

- Progress: 1 finished (of 1)
- PASS BallotTest (tests/Ballot_test.sol)
- ✓ Check winning proposal
- ✓ Check winning proposal with return value
- Result for tests/Ballot_test.sol
- Passed: 2
- Failed: 0
- Time Taken: 0.15s

10. Loops.sol

The screenshot shows the Solidity Unit Testing interface. On the left, a sidebar contains a 'Run' button and a list of tests. The main area displays the Solidity code for the 'Loop' contract. The code includes a 'loop' function with a 'for' loop and a 'while' loop. The test results on the left indicate that the tests passed.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Loop {
5     uint public count; // New public uint state variable
6
7     function loop() public {
8         // for loop
9         for (uint i = 0; i < 10; i++) {
10             if (i == 3) {
11                 // Skip to next iteration with continue
12                 continue;
13             }
14             if (i == 5) {
15                 // Exit loop with break
16                 break;
17             }
18             // Increment the count variable by 1 at the beginning of each iteration
19             count++;
20         }
21
22         // while loop
23         uint j;
24     }
25 }
```

Test Results:

- Progress: 1 finished (of 1)
- PASS BallotTest (tests/Ballot_test.sol)
- ✓ Check winning proposal
- ✓ Check winning proposal with return value
- Result for tests/Ballot_test.sol
- Passed: 2
- Failed: 0
- Time Taken: 0.16s

11. Arrays.sol

SOLIDITY UNIT TESTING

Generate Run tests Run Stop

Select all tests/Ballot_test.sol

Progress: 1 finished (of 1)
PASS: BallotTest (tests/Ballot_test.sol)

✓ Check winning proposal ✖

✓ Check winning proposal with return value ✖

Result for tests/Ballot_test.sol
Passed: 2
Failed: 0
Time Taken: 0.18s

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Array {
5     uint[] public arr;
6     uint[] public arr2 = [1, 2, 3];
7     uint[3] public arr3 = [0, 1, 2]; // New fixed-sized array arr3
8
9     function get(uint i) public view returns (uint) { infinite gas
10         return arr[i];
11     }
12
13     // Update getArr to return arr3
14     function getArr() public view returns (uint[3] memory) { infinite gas
15         return arr3;
16     }
17
18     function push(uint i) public { 46829 gas
19         arr.push(i);
20     }
21
22     function pop() public { 29467 gas
23         arr.pop();
24     }
25 }
```

Running checkWinningProposal

SOLIDITY UNIT TESTING

Generate Run tests Run Stop

Select all tests/Ballot_test.sol

Progress: 1 finished (of 1)
PASS: BallotTest (tests/Ballot_test.sol)

✓ Check winning proposal ✖

✓ Check winning proposal with return value ✖

Result for tests/Ballot_test.sol
Passed: 2
Failed: 0
Time Taken: 0.18s

```
26 function getLength() public view returns (uint) { 2489 gas
27     return arr.length;
28 }
29
30 function remove(uint index) public { 7596 gas
31     delete arr[index];
32 }
33
34 contract CompactArray {
35     uint[] public arr;
36
37     function remove(uint index) public { infinite gas
38         arr[index] = arr[arr.length - 1];
39         arr.pop();
40     }
41
42     function test() public { infinite gas
43         arr.push(1);
44         arr.push(2);
45         arr.push(3);
46         arr.push(4);
47     }
48 }
```

Running checkWinningProposal

12. Mappings. Sol

The screenshot shows the Solidity IDE interface. On the left, the 'Home' tab is active, displaying 'TY UNIT TESTING'. The 'Run tests' button is highlighted. Below it, the 'Select all' checkbox is checked, and the test file 'tests/Ballot_test.sol' is selected. The progress bar shows 'Progress: 1 finished (of 1)'. The test results for 'BallotTest (tests/Ballot_test.sol)' are displayed as 'PASS'. The test suite includes 'Check winning proposal' and 'Check winnin proposal with return value'. The 'Result for tests/Ballot_test.sol' shows 'Passed: 2', 'Failed: 0', and 'Time Taken: 0.19s'.

The main editor displays the 'mappings.sol' file, which defines a 'Mapping' contract. The contract includes a 'mapping' function to create a new entry in the 'balances' mapping, a 'get' function to retrieve the value, and a 'remove' function to delete an entry. The gas costs for each function are indicated: 2885 gas for 'get' and 5554 gas for 'remove'.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Mapping {
5     // Updated mapping from address to uint
6     mapping(address => uint) public balances; // Renamed myMap to balances
7
8     // Updated get function to use balances mapping
9     function get(address _addr) public view returns (uint) { 2885 gas
10         return balances[_addr];
11     }
12
13     // Updated set function to create a new entry in the balances mapping
14     function set(address _addr, uint _i) public { 22854 gas
15         balances[_addr] = _i;
16     }
17
18     // Updated remove function to work with the balances mapping
19     function remove(address _addr) public { 5554 gas
20         delete balances[_addr];
21     }
22 }
23
```

The bottom status bar shows 'Running checkWinningProposal' and a search bar with the text 'Search with transaction hash or address'.

The screenshot shows the Solidity IDE interface. On the left, the 'Home' tab is active, displaying 'TY UNIT TESTING'. The 'Run tests' button is highlighted. Below it, the 'Select all' checkbox is checked, and the test file 'tests/Ballot_test.sol' is selected. The progress bar shows 'Progress: 1 finished (of 1)'. The test results for 'BallotTest (tests/Ballot_test.sol)' are displayed as 'PASS'. The test suite includes 'Check winning proposal' and 'Check winnin proposal with return value'. The 'Result for tests/Ballot_test.sol' shows 'Passed: 2', 'Failed: 0', and 'Time Taken: 0.19s'.

The main editor displays the 'mappings.sol' file, which defines a 'NestedMapping' contract. The contract includes a 'mapping' function to create a new entry in the 'nested' mapping, a 'get' function to retrieve the value, and a 'remove' function to delete an entry. The gas costs for each function are indicated: 3178 gas for 'get' and 25060 gas for 'remove'.

```
22 }
23
24 contract NestedMapping {
25     mapping(address => mapping(uint => bool)) public nested;
26
27     function get(address _addr1, uint _i) public view returns (bool) { 3178 gas
28         return nested[_addr1][_i];
29     }
30
31     function set( 25217 gas
32         address _addr1,
33         uint _i,
34         bool _boo
35     ) public {
36         nested[_addr1][_i] = _boo;
37     }
38
39     function remove(address _addr1, uint _i) public { 25060 gas
40         delete nested[_addr1][_i];
41     }
42 }
43
```

The bottom status bar shows 'Running checkWinningProposal' and a search bar with the text 'Search with transaction hash or address'.

13. Stucts.sol

The screenshot shows the Solidity Unit Testing interface. On the left, there's a sidebar with a 'Run' button and a 'Stop' button. Below them, it shows 'Progress: 1 finished (of 1)' and 'PASS BallotTest (tests/Ballot_test.sol)'. The main area displays the Solidity code for 'Stucts.sol'. The code defines a 'Todos' contract with a 'struct Todo' containing 'string text' and 'bool completed'. It includes a 'create' function to add new todos and a 'get' function to retrieve a todo by index. The bottom status bar indicates 'Running checkWinningProposal'.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Todos {
5     struct Todo {
6         string text;
7         bool completed;
8     }
9
10    Todo[] public todos;
11
12    function create(string memory _text) public {
13        todos.push(Todo(_text, false));
14        todos.push(Todo({text: _text, completed: false}));
15        Todo memory todo;
16        todo.text = _text;
17        todos.push(todo);
18    }
19
20    function get(uint _index) public view returns (string memory text, bool completed) {
21        Todo storage todo = todos[_index];
22        return (todo.text, todo.completed);
23    }
24 }
```

This screenshot shows the continuation of the Solidity code in 'Stucts.sol'. It includes an 'update' function to modify a todo's text, a 'toggleCompleted' function to flip the 'completed' status, and a 'remove' function to delete a todo by index. The 'remove' function includes a gas cost of 29006. The interface shows the same sidebar and status bar as the previous screenshot.

```
22 return (todo.text, todo.completed);
23 }
24
25 function update(uint _index, string memory _text) public {
26     Todo storage todo = todos[_index];
27     todo.text = _text;
28 }
29
30 function toggleCompleted(uint _index) public {
31     Todo storage todo = todos[_index];
32     todo.completed = !todo.completed;
33 }
34
35 // New function to remove a struct member with the given index
36 function remove(uint _index) public {
37     require(_index < todos.length, "Index out of bounds");
38     // Move the last element into the place to delete
39     todos[_index] = todos[todos.length - 1];
40     // Remove the last element
41     todos.pop();
42 }
43
44 }
```

14. Enums.sol

The screenshot shows the Solidity IDE interface. On the left, the 'Home' tab is active, displaying 'TY UNIT TESTING' with buttons for 'Generate', 'Run tests', 'Run', and 'Stop'. Below these are checkboxes for 'Select all' and 'tests/Ballot_test.sol'. The 'Progress' section shows '1 finished (of 1)' with a 'PASS' status for 'BallotTest (tests/Ballot_test.sol)'. Two test cases are listed: '✓ Check winning proposal' and '✓ Check winnin proposal with return value'. The 'Result for tests/Ballot_test.sol' shows 'Passed: 2', 'Failed: 0', and 'Time Taken: 0.16s'. The main editor displays the 'enums.sol' file with the following code:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3 contract Enum {
4     enum Status {
5         Pending,
6         Shipped,
7         Accepted,
8         Rejected,
9         Canceled
10    }
11    Status public status;
12    function get() public view returns (Status) { 2590 gas
13        return status;
14    }
15    function set(Status _status) public { undefined gas
16        status = _status;
17    }
18    function cancel() public { 24497 gas
19        status = Status.Canceled;
20    }
21    function reset() public { 24430 gas
22        delete status;
23    }
24 }
```

The bottom status bar indicates 'Running checkWinningProposal' and 'Active Windows'.

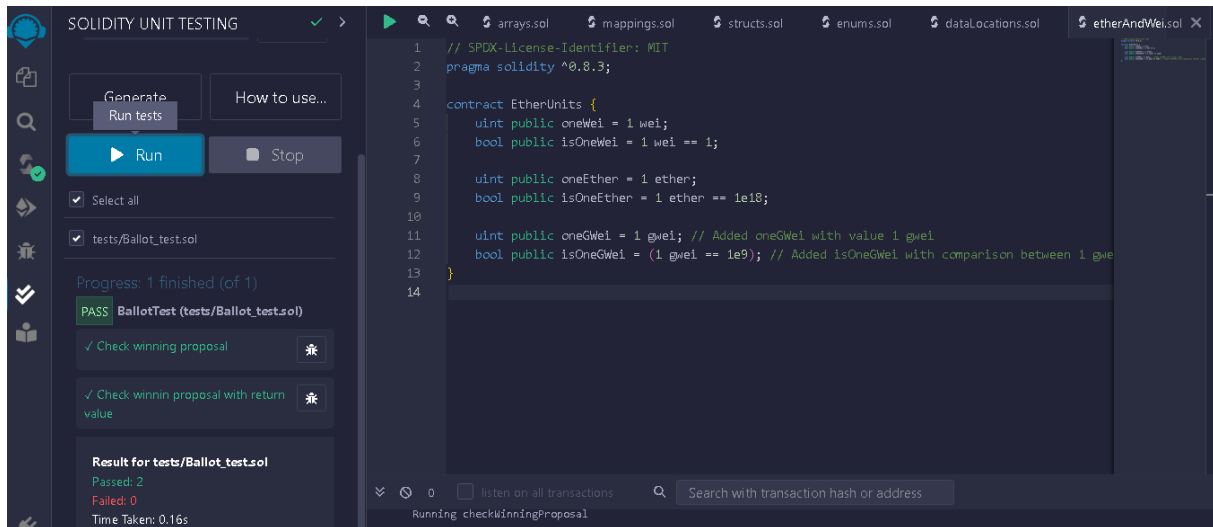
15. dataLocations.sol

The screenshot shows the Solidity IDE interface. On the left, the 'Home' tab is active, displaying 'TY UNIT TESTING' with buttons for 'Generate', 'Run tests', 'Run', and 'Stop'. Below these are checkboxes for 'Select all' and 'tests/Ballot_test.sol'. The 'Progress' section shows '1 finished (of 1)' with a 'PASS' status for 'BallotTest (tests/Ballot_test.sol)'. Two test cases are listed: '✓ Check winning proposal' and '✓ Check winnin proposal with return value'. The 'Result for tests/Ballot_test.sol' shows 'Passed: 2', 'Failed: 0', and 'Time Taken: 0.27s'. The main editor displays the 'dataLocations.sol' file with the following code:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3 contract DataLocations {
4     uint[] public arr;
5     mapping(uint => address) map;
6     struct MyStruct {
7         uint foo;
8     }
9     mapping(uint => MyStruct) myStructs;
10    function f() public returns (MyStruct memory, MyStruct memory) { infinite gas
11        myStructs[1].foo = 4;
12        MyStruct memory myMemStruct = MyStruct(0);
13        myMemStruct.foo = 1;
14        MyStruct memory myMemStruct2 = myStructs[1];
15        myMemStruct2.foo = 3;
16        return (myMemStruct, myMemStruct2);
17    }
18    function _f( undefined gas
19        uint[] storage _arr,
20        mapping(uint => address) storage _map,
21        MyStruct storage _myStruct
22    ) internal {
23    }
24 }
```

The bottom status bar indicates 'Running checkWinningProposal' and 'Active Windows'.

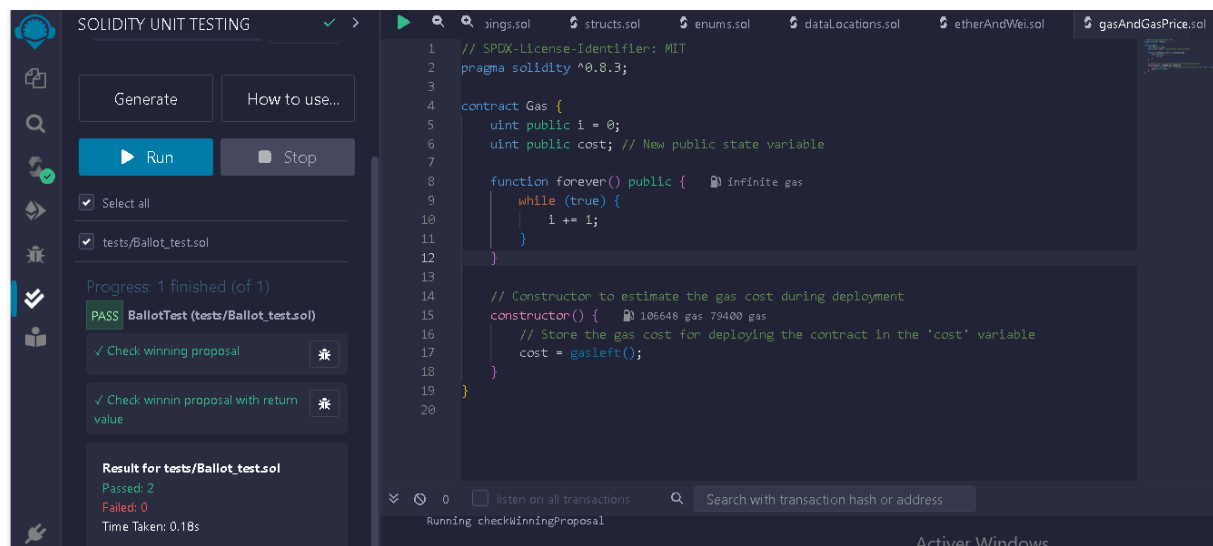
16. etherAndWei.sol



The screenshot shows the Solidity Unit Testing interface. On the left, the 'SOLIDITY UNIT TESTING' sidebar is visible, showing a 'Run' button and a 'Stop' button. Below these, there are checkboxes for 'Select all' and 'tests/Ballot_test.sol'. The progress bar indicates 'Progress: 1 finished (of 1)' and 'PASS: BallotTest (tests/Ballot_test.sol)'. The results section shows 'Result for tests/Ballot_test.sol' with 'Passed: 2' and 'Failed: 0', and 'Time Taken: 0.16s'. The main editor displays the code for 'etherAndWei.sol', which defines a contract 'EtherUnits' with variables 'oneWei', 'isOneWei', 'oneEther', 'isOneEther', 'oneGwei', and 'isOneGwei'. The code is as follows:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract EtherUnits {
5     uint public oneWei = 1 wei;
6     bool public isOneWei = 1 wei == 1;
7
8     uint public oneEther = 1 ether;
9     bool public isOneEther = 1 ether == 1e18;
10
11     uint public oneGwei = 1 gwei; // Added oneGwei with value 1 gwei
12     bool public isOneGwei = (1 gwei == 1e9); // Added isOneGwei with comparison between 1 gwei
13 }
14
```

17. gasAndGasPrice.sol



The screenshot shows the Solidity Unit Testing interface. On the left, the 'SOLIDITY UNIT TESTING' sidebar is visible, showing a 'Run' button and a 'Stop' button. Below these, there are checkboxes for 'Select all' and 'tests/Ballot_test.sol'. The progress bar indicates 'Progress: 1 finished (of 1)' and 'PASS: BallotTest (tests/Ballot_test.sol)'. The results section shows 'Result for tests/Ballot_test.sol' with 'Passed: 2' and 'Failed: 0', and 'Time Taken: 0.18s'. The main editor displays the code for 'gasAndGasPrice.sol', which defines a contract 'Gas' with variables 'i' and 'cost', and a function 'forever()'. The code is as follows:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Gas {
5     uint public i = 0;
6     uint public cost; // New public state variable
7
8     function forever() public {
9         while (true) {
10             i += 1;
11         }
12     }
13
14     // Constructor to estimate the gas cost during deployment
15     constructor() {
16         // Store the gas cost for deploying the contract in the 'cost' variable
17         cost = gasleft();
18     }
19 }
20
```

18. sendingEther.sol

The screenshot shows the Solidity Unit Testing interface in VS Code. The left sidebar displays the 'SOLIDITY UNIT TESTING' panel, which includes a 'Run' button and a 'Stop' button. Below these buttons, there is a section for 'Progress: 1 finished (of 1)' showing a 'PASS' status for 'BallotTest (tests/Ballot_test.sol)'. The results section indicates 'Passed: 2' and 'Failed: 0' with a 'Time Taken: 0.19s'.

The main editor displays the 'sendingEther.sol' file, which contains the following Solidity code:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Charity {
5     address public owner;
6
7     // Constructor to set the owner to the deployer of the contract
8     constructor() {
9         owner = msg.sender;
10     }
11
12     // Function to receive Ether. msg.data must be empty
13     receive() external payable {}
14
15     // Function to donate Ether
16     function donate() public payable {}
17
18     // Function to withdraw the total balance to the owner
19     function withdraw() public {
20         // Only the owner can withdraw
21         require(msg.sender == owner, "You are not the owner");
22
23         // Transfer the total balance to the owner
```

The bottom status bar shows 'Running checkWinningProposal'.