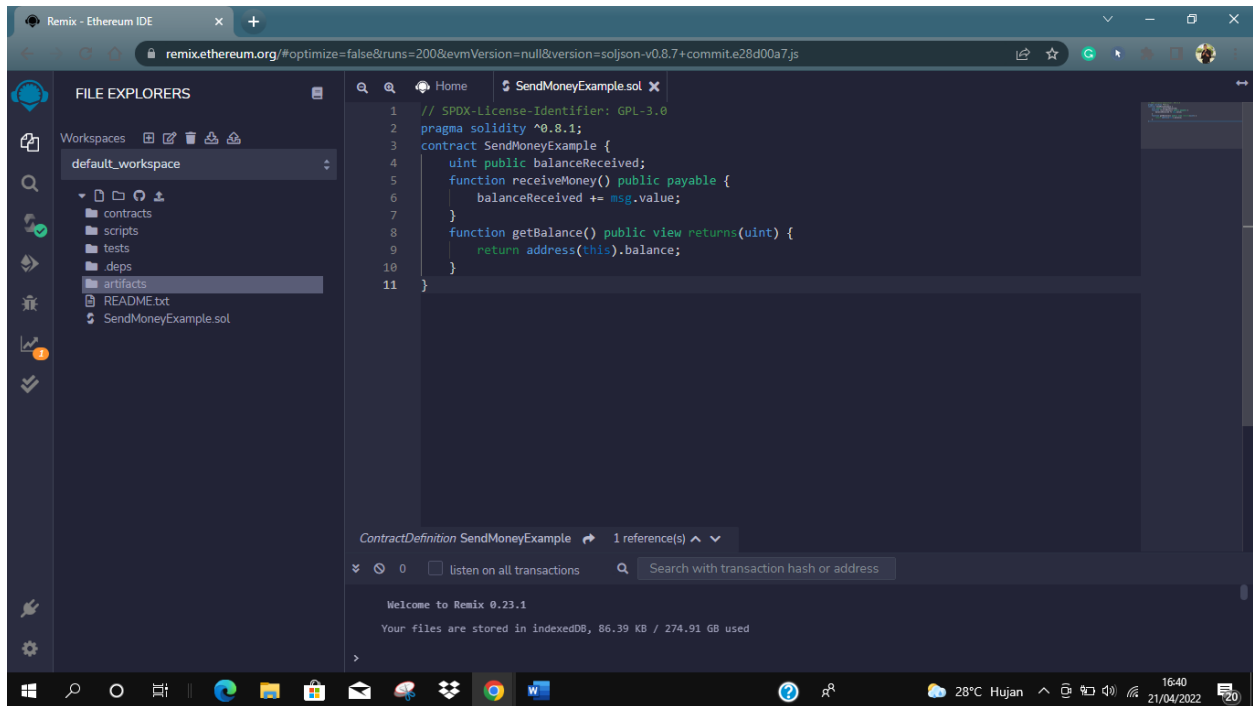


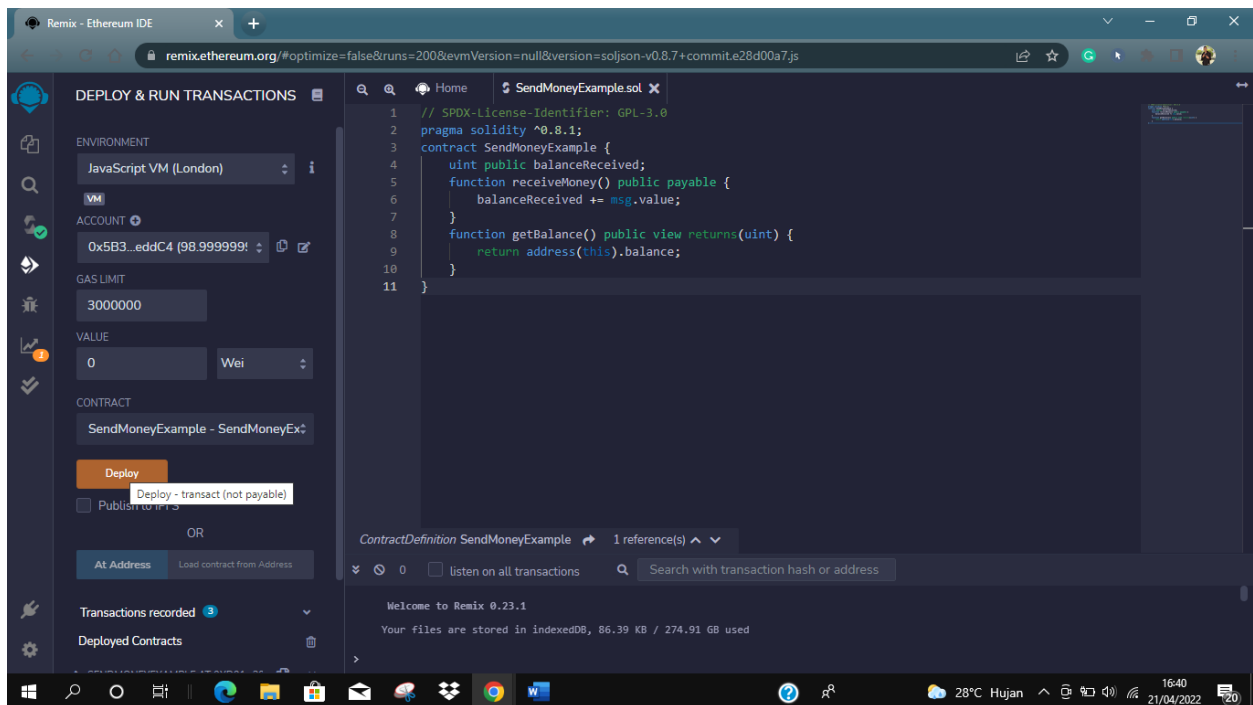
Smart Contract

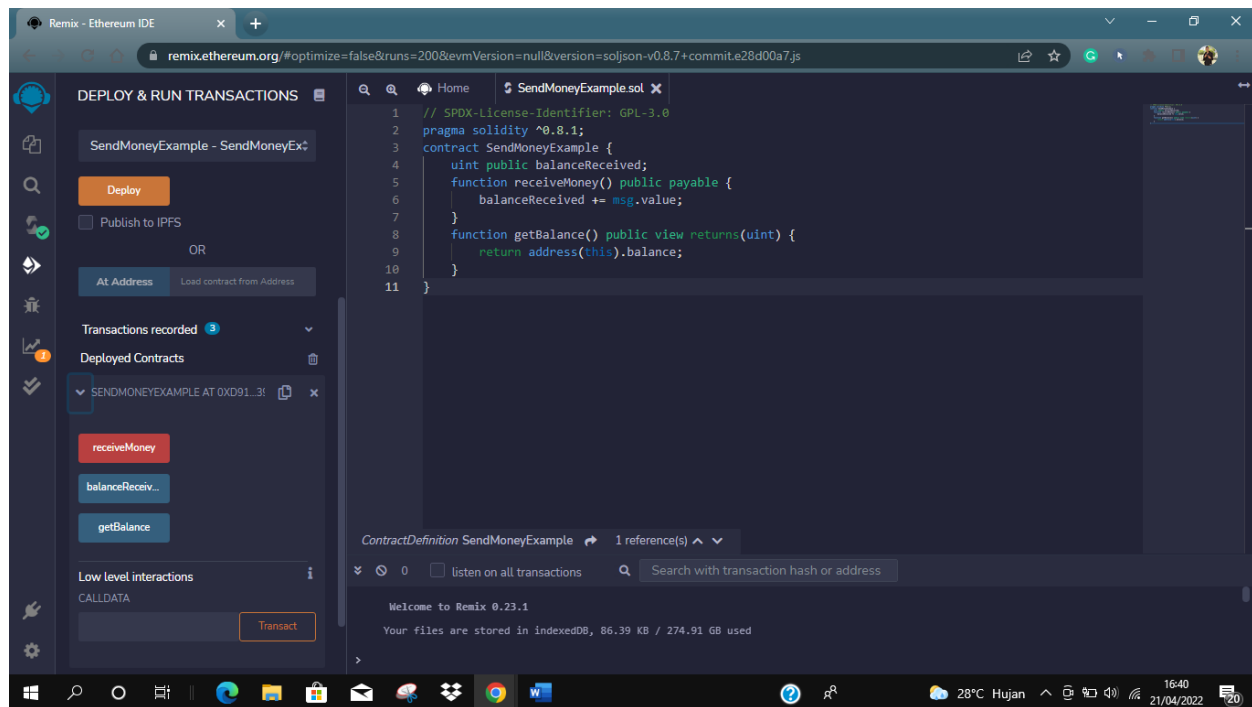
Let's start with a simple Smart Contract. Create a new file in Remix



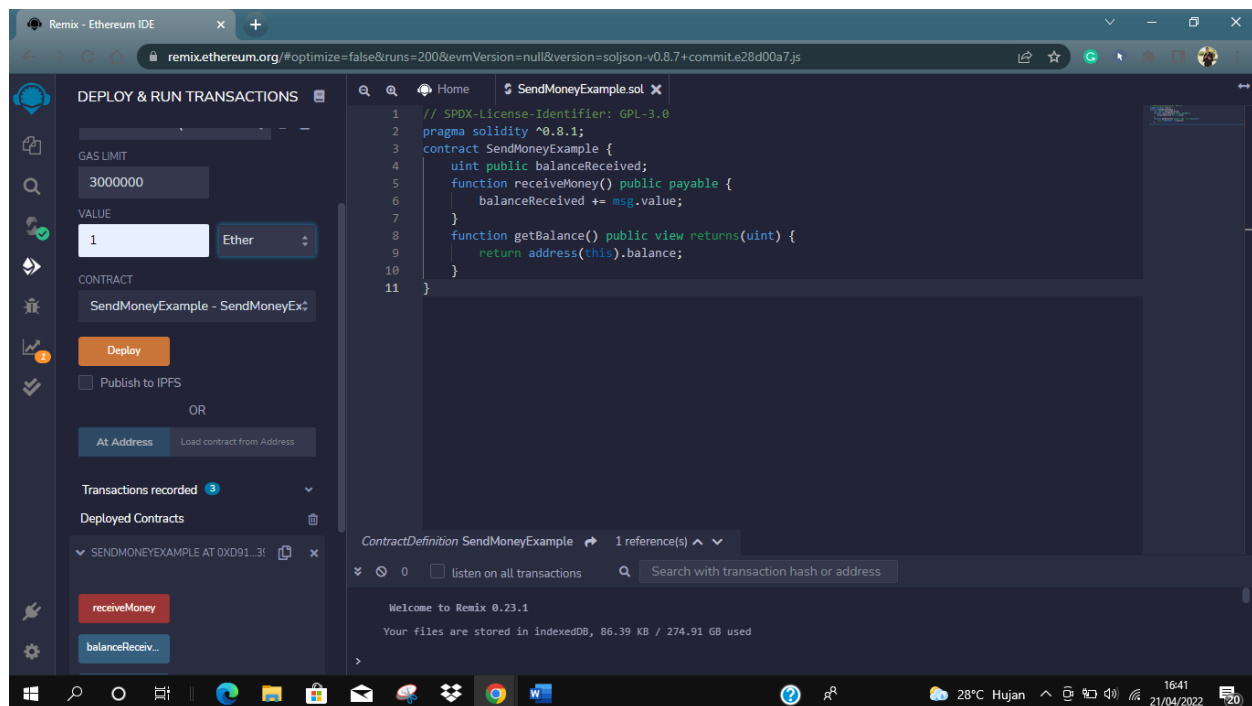
Deploy and Use the Smart Contract

Head over to the Deploy and Run Transactions Plugin and deploy the Smart Contract into the JavaScript VM

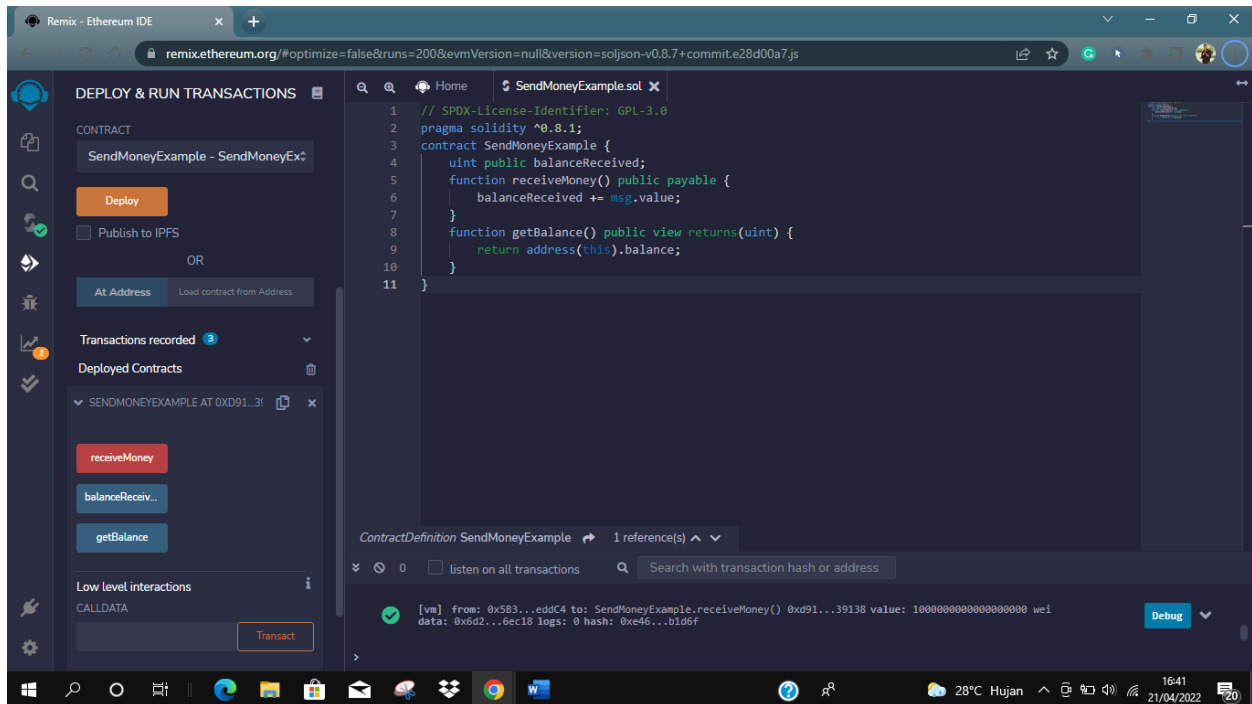




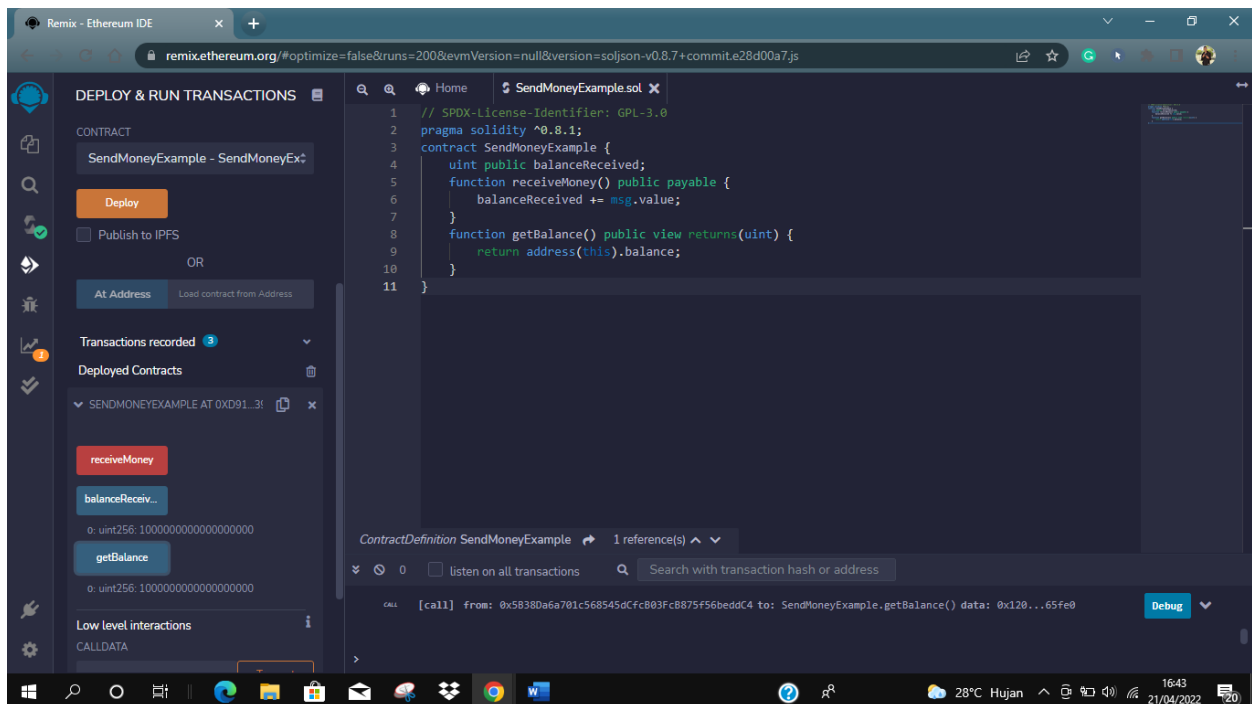
Scroll up to the "value" field and put "1" into the value input field and select "ether" from the dropdown. Then scroll down to the Smart Contract and hit the red "receiveMoney" button:



Also observe the terminal, see that there was a new transaction sent to "the network" (although just a simulation in the

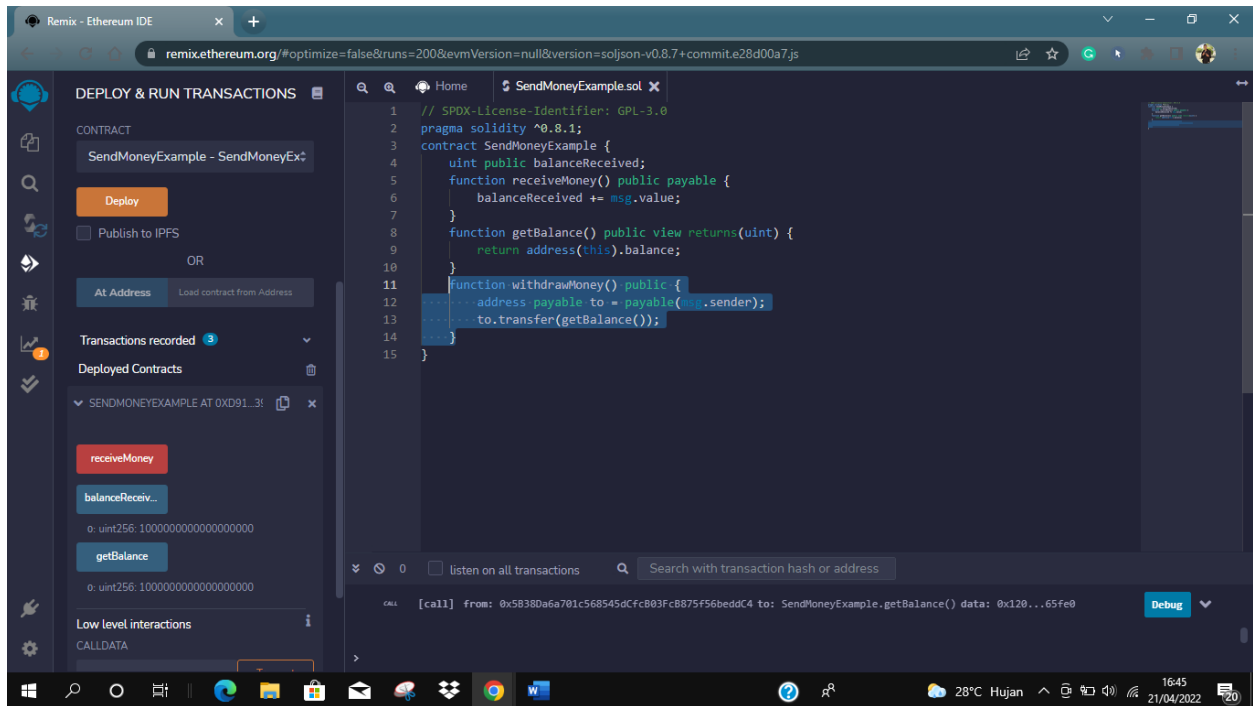


Now we sent 1 Ether, or 10^{18} Wei, to the Smart Contract. According to our code the variable balanceReceived and the function getBalance() should have the same value.

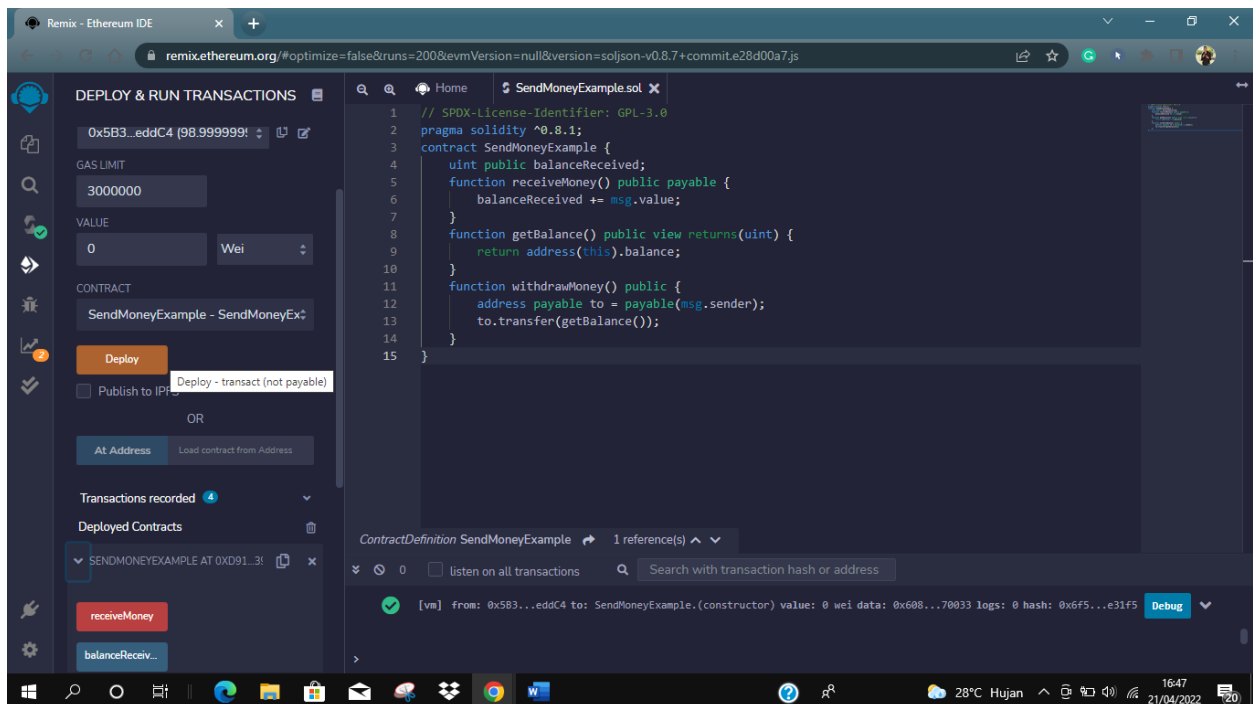


Withdraw Ether From Smart Contract

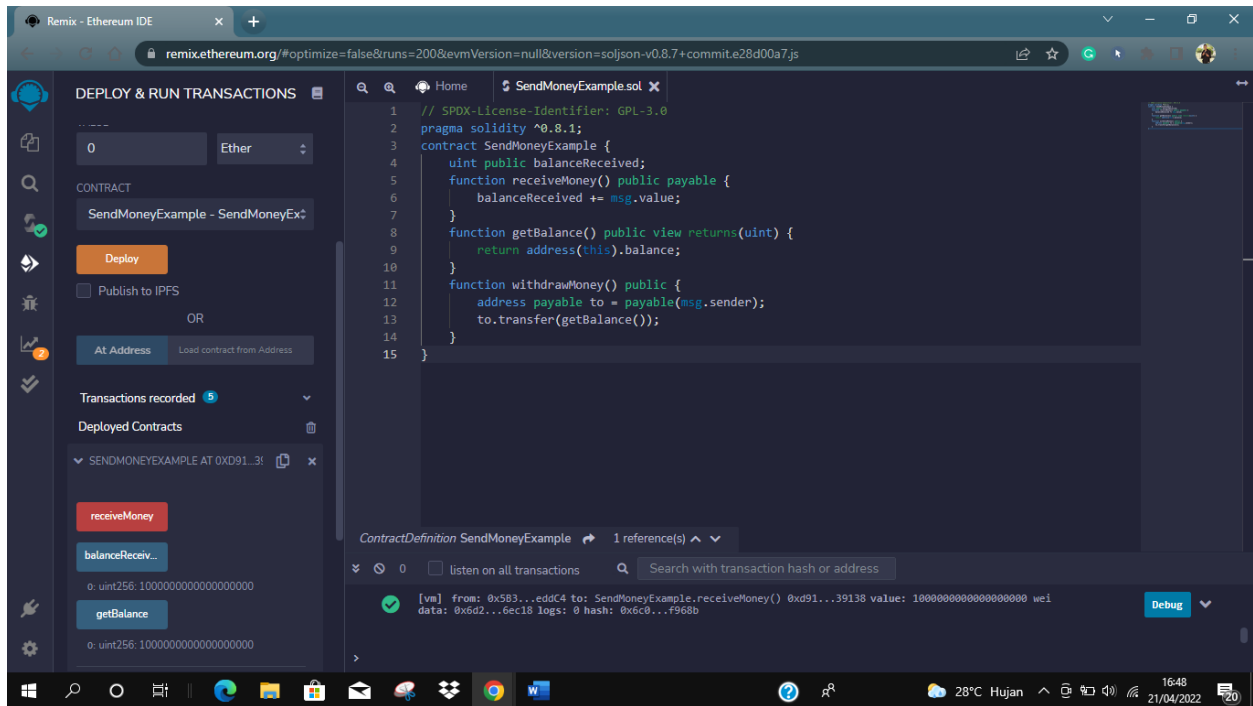
Let's add the following function to the Smart Contract:



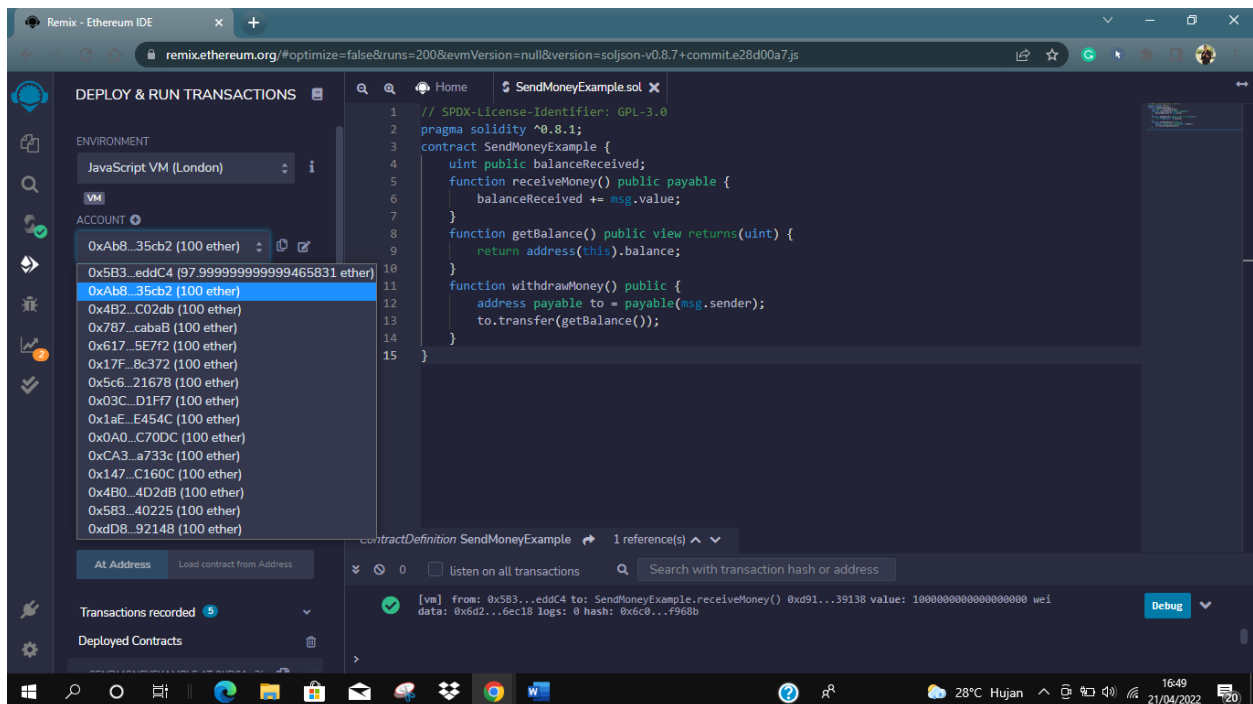
Deploy the new version and send again 1 Ether to the Smart Contract. To avoid confusion, I recommend you close the previous Instance, we won't need it anymore.



Put in "1 Ether" into the value input box. hit "receiveMoney" in your new contract Instance.



Select the second Account from the Accounts dropdown:

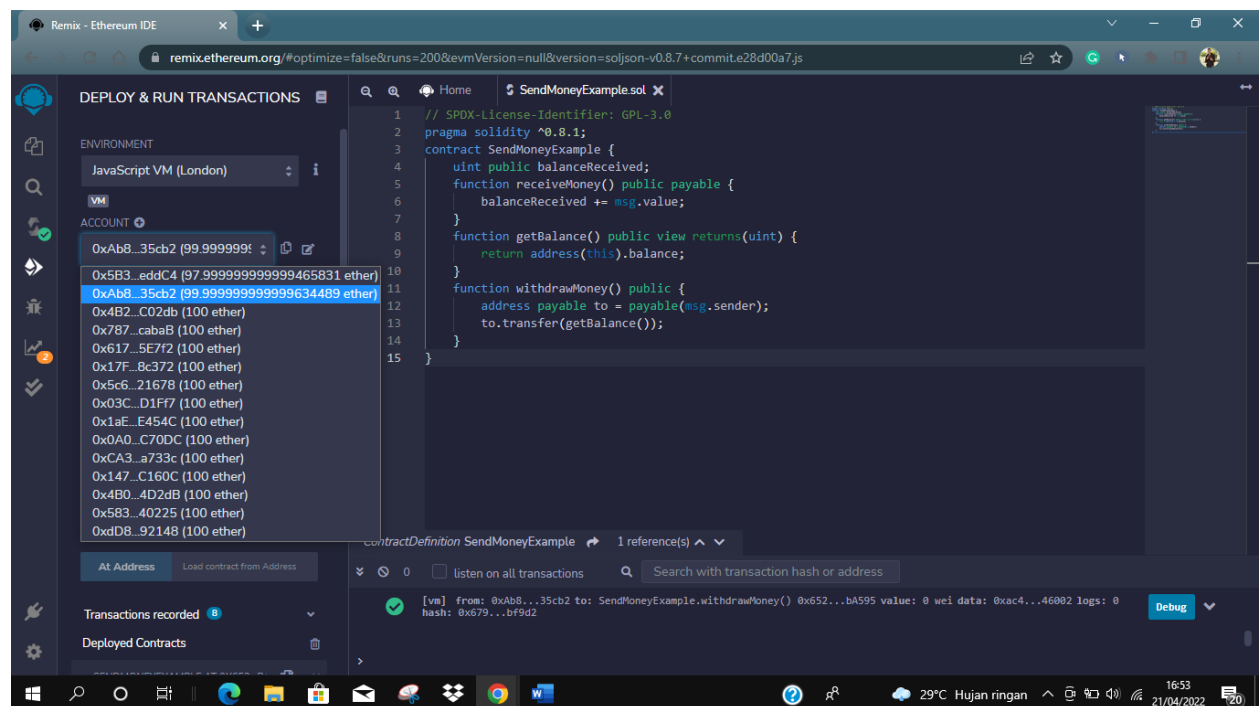


The screenshot displays the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar shows the contract 'SendMoneyExample - SendMoneyEx'. It has a 'Deploy' button and a 'Publish to IPFS' checkbox. Below this, it lists 'Transactions recorded' and 'Deployed Contracts'. The 'SENDMONEYEXAMPLE AT 0X652...BJ' contract is listed with buttons for 'receiveMoney', 'withdrawMoney', 'balanceReceiv...', and 'getBalance'. A tooltip points to the 'withdrawMoney' button, stating 'withdrawMoney - transaction (not payable)'. At the bottom, 'Low level interactions' are visible.

The central code editor shows the Solidity code for 'SendMoneyExample.sol':

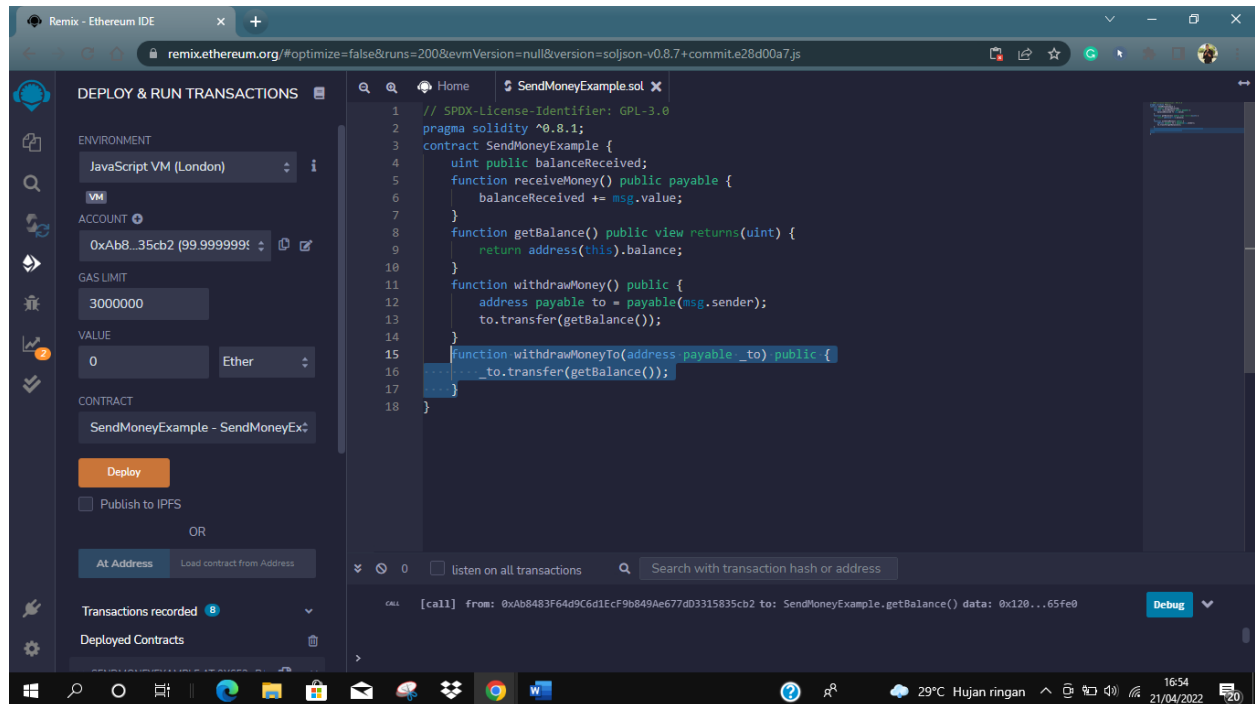
```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity ^0.8.1;
3 contract SendMoneyExample {
4     uint public balanceReceived;
5     function receiveMoney() public payable {
6         balanceReceived += msg.value;
7     }
8     function getBalance() public view returns(uint) {
9         return address(this).balance;
10    }
11    function withdrawMoney() public {
12        address payable to = payable(msg.sender);
13        to.transfer(getBalance());
14    }
15 }
```

The bottom console shows the deployment transaction details: '[vm] from: 0xAb0...35cb2 to: SendMoneyExample.(constructor) value: 0 wei data: 0x608...70033 logs: 0 hash: 0x8e6...f8324'. A 'Debug' button is next to the transaction details.

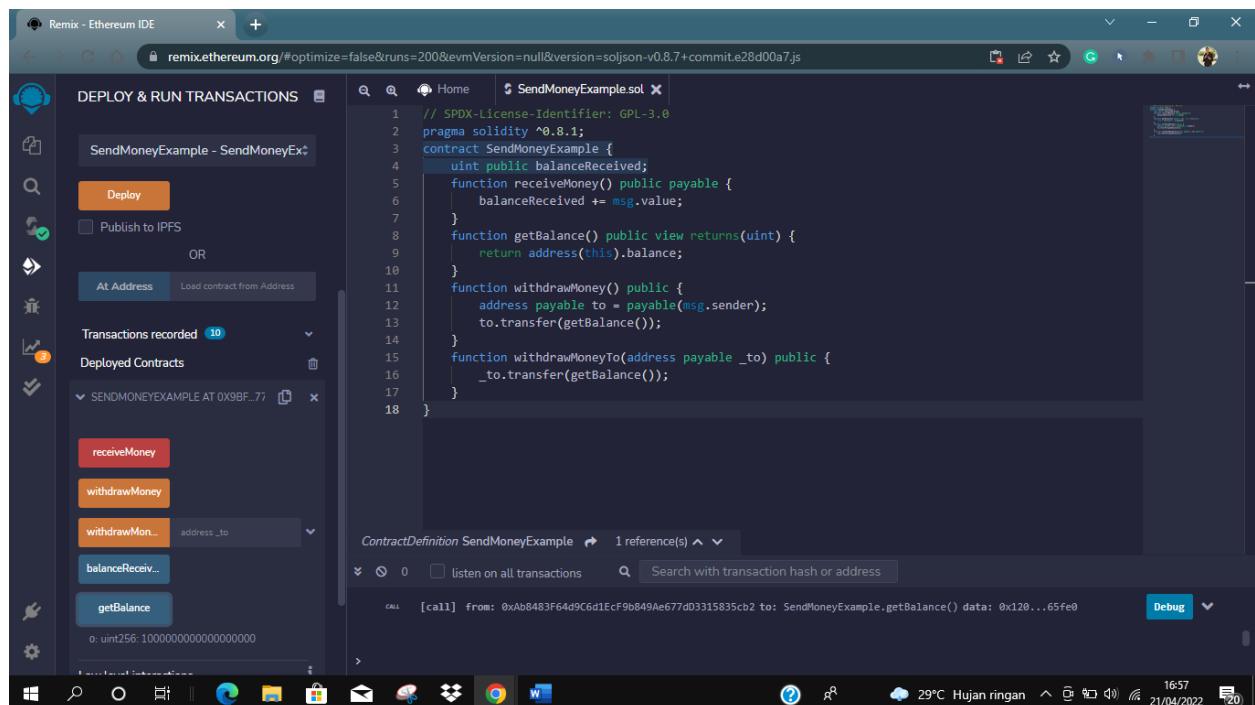


Withdraw To Specific Account

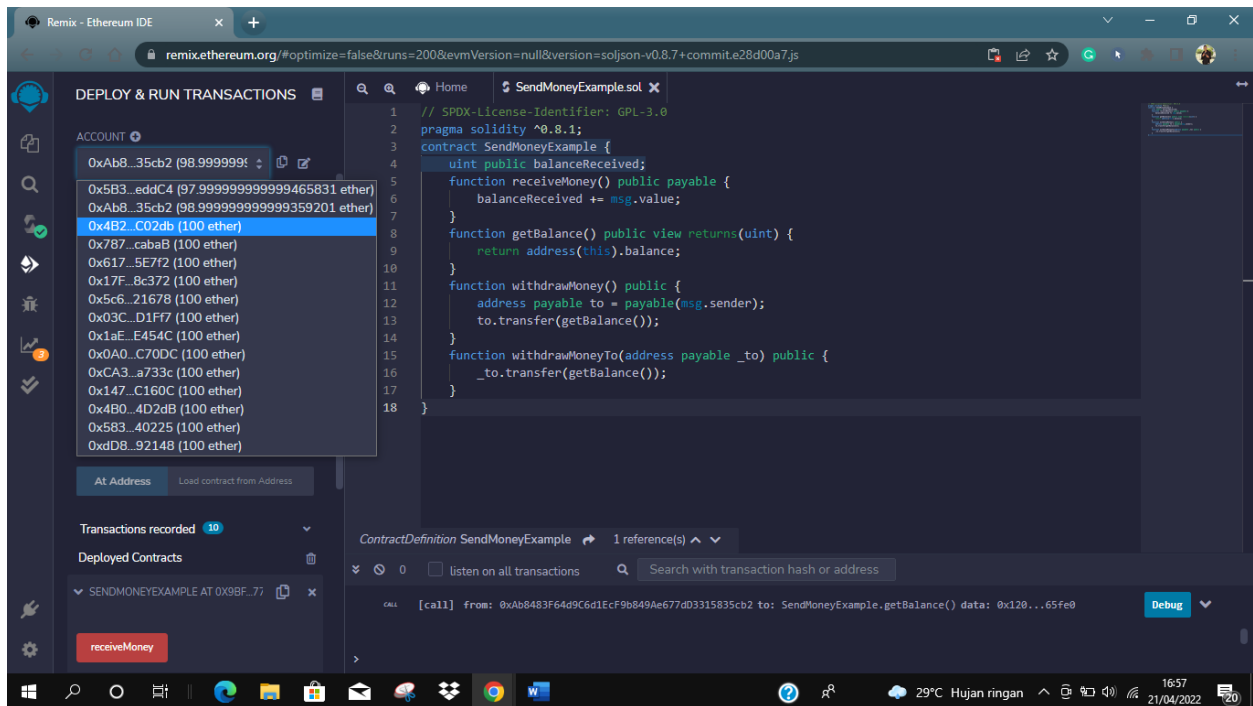
Previously we had our Smart Contract just blindly send the Ether to whoever called the Smart Contracts "withdrawMoney" function. Let's extend this a bit so that the Funds can be send to a specific Account.



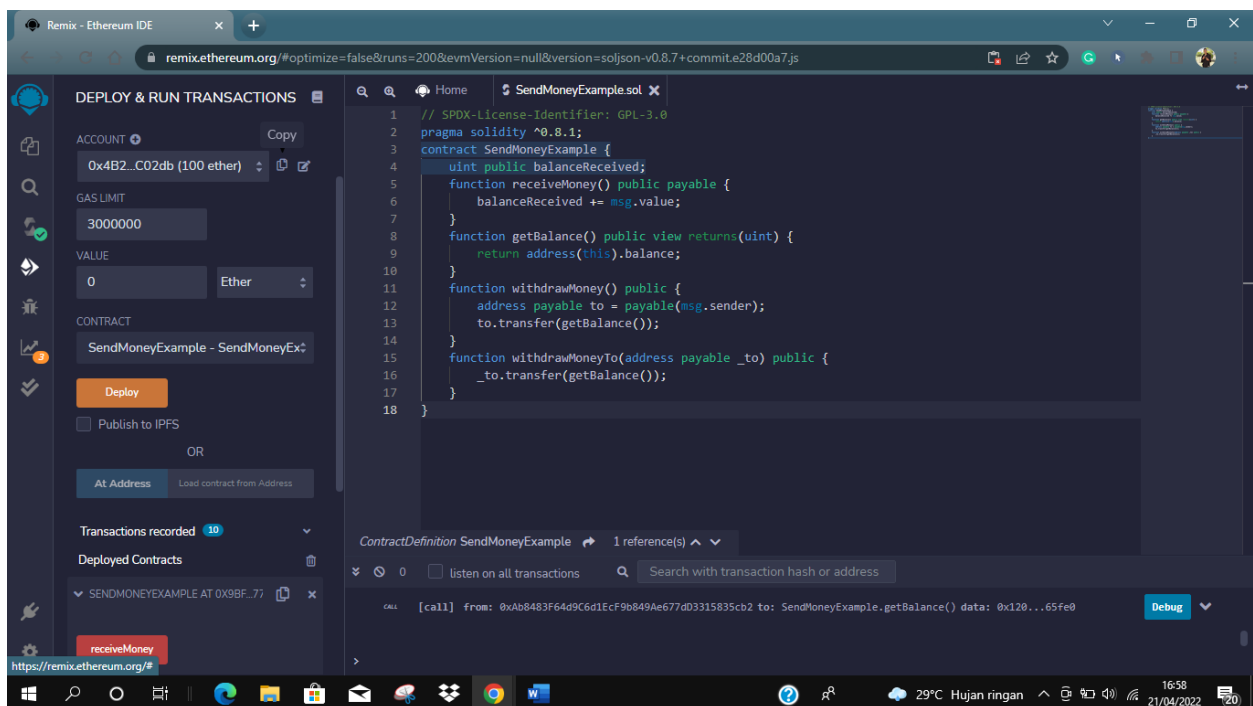
Deploy the Smart Contract. Close the old Instance. Send 1 Ether to the Smart Contract (don't forget the value input field!). Make sure the Balance shows up correctly.



Select the third account from the dropdown



Hit the little "copy" icon:



Switch back to the first Account. Paste the Account you copied into the input field next to "withdrawMoneyTo": Now open the Accounts dropdown. See the balance of your third Account? 101 Ether!!!

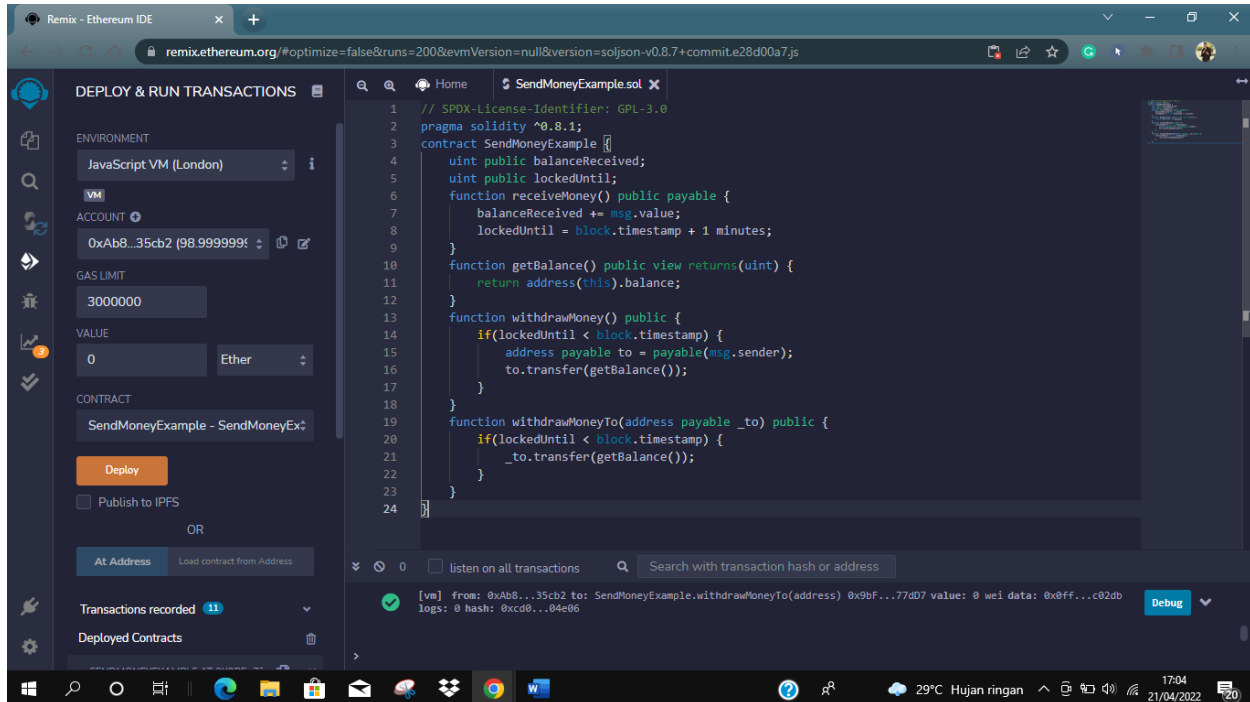
The screenshot displays the Remix Ethereum IDE interface. On the left, the 'ACCOUNT' dropdown menu is open, showing a list of accounts with their balances. The account '0x482...C02db' is highlighted, showing a balance of 101 ether. The main editor shows a Solidity contract named 'SendMoneyExample.sol' with the following code:

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity ^0.8.1;
3 contract SendMoneyExample {
4     uint public balanceReceived;
5     function receiveMoney() public payable {
6         balanceReceived += msg.value;
7     }
8     function getBalance() public view returns(uint) {
9         return address(this).balance;
10    }
11    function withdrawMoney() public {
12        address payable to = payable(msg.sender);
13        to.transfer(getBalance());
14    }
15    function withdrawMoneyTo(address payable _to) public {
16        _to.transfer(getBalance());
17    }
18 }
```

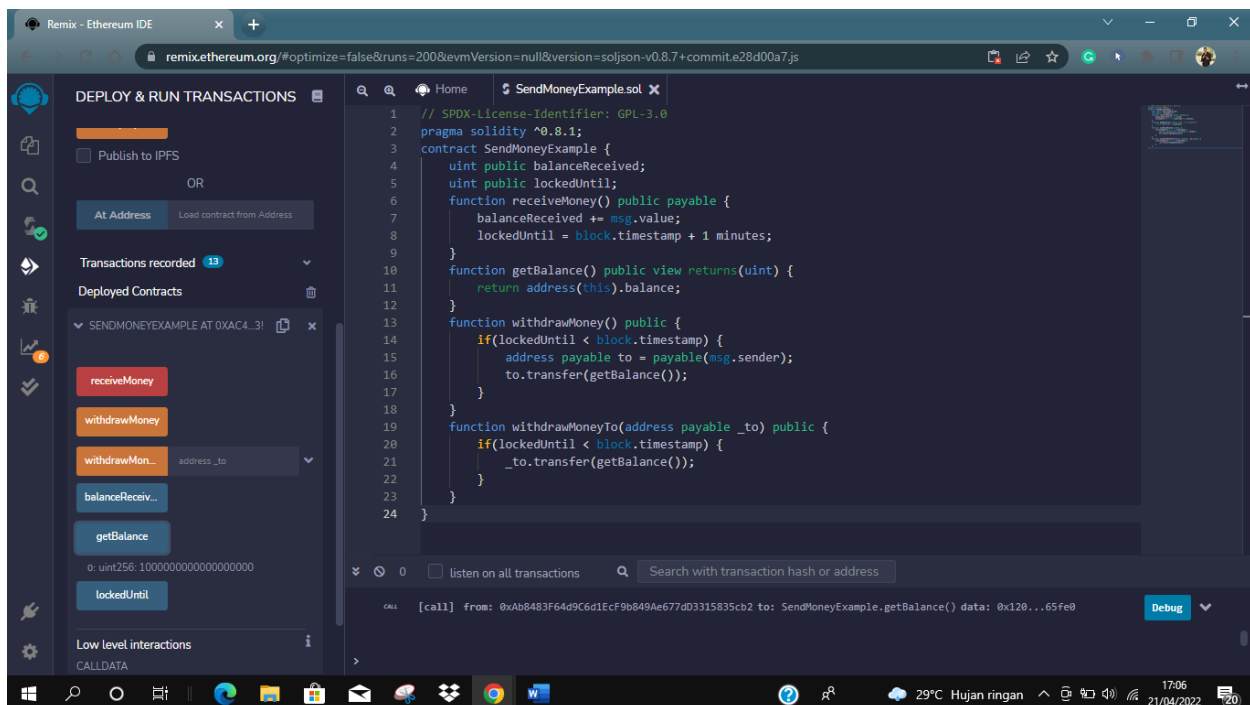
At the bottom, the 'ContractDefinition' panel shows 'SendMoneyExample' with 1 reference(s). The 'Logs' panel shows a transaction from '0xab8...35cb2' to 'SendMoneyExample.withdrawMoneyTo(address) 0x9bf...77d7' with a value of 0 wei and data '0x0ff...c02db'. The 'Debug' button is visible next to the logs.

Withdrawal Locking

Let's extend our Smart Contract to do some locking. You will see that it is very easy to let our code take care of some specific logic to allow/disallow certain actions.



Deploy a new Instance version. Remove the old Instance. Send 1 Ether to the Smart Contract (don't forget the value field) by clicking on "receiveMoney". Check the Balance!



Click "withdrawMoney" - and nothing happens. The Balance stays the same until 1 Minute passed since you hit "receiveMoney".

