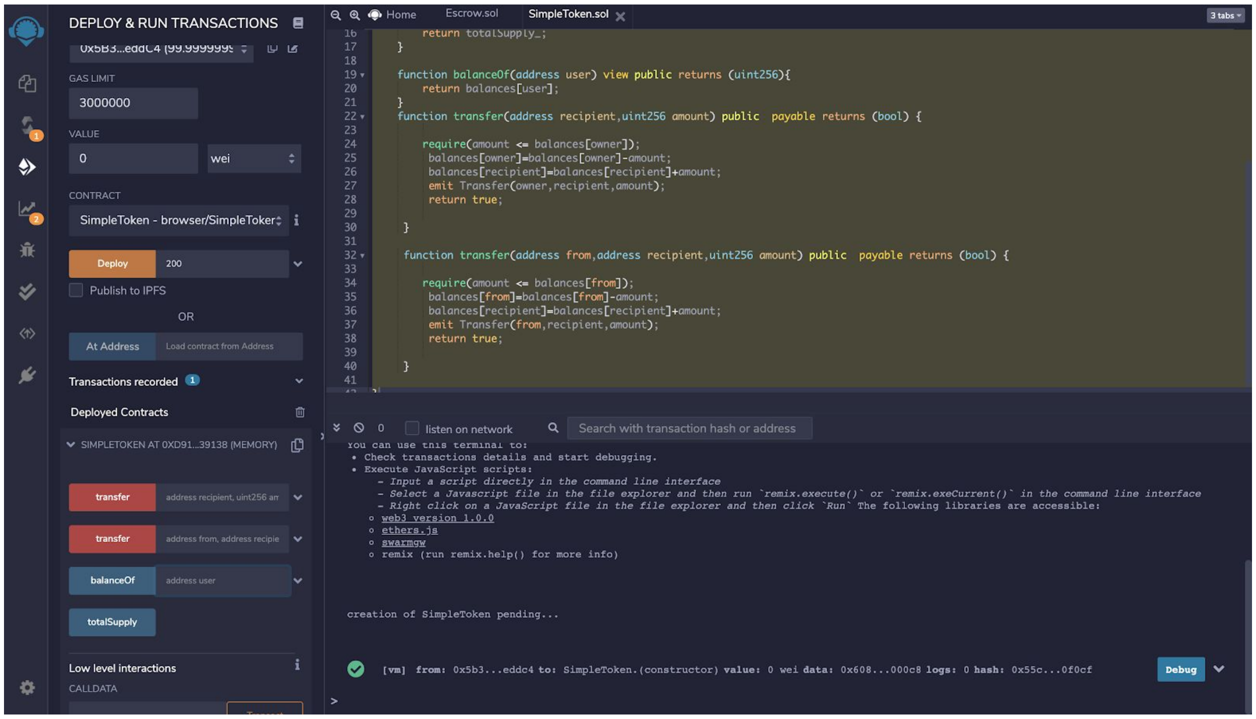
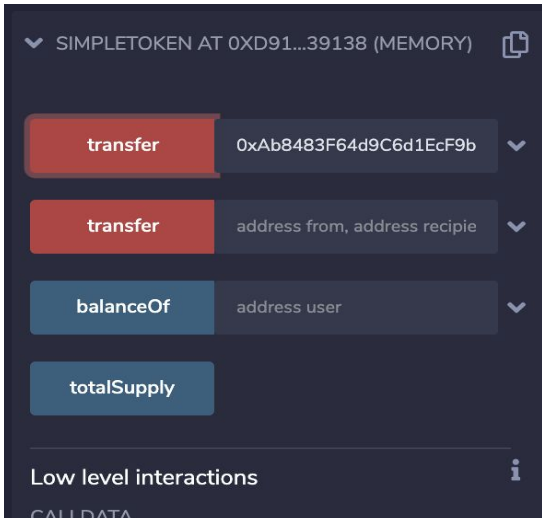
Question 1 : Code : attached in file

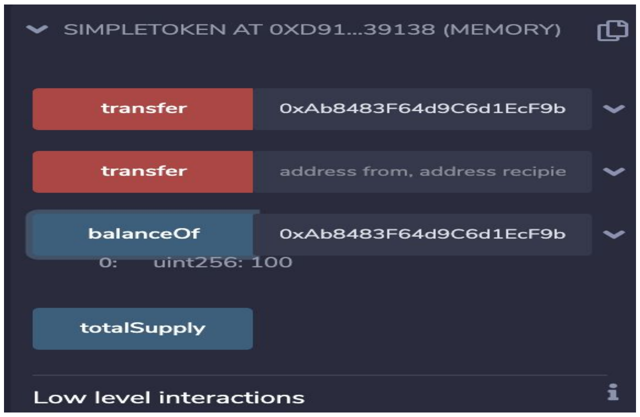
Execution : Total Supply 200 tokens , transfer 100 tokens from one account to other.



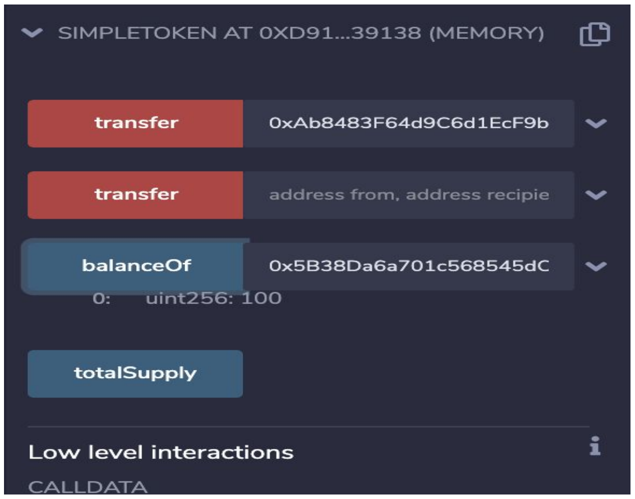
```
        return totalSupply_;
    }

    function balanceOf(address user) view public returns (uint256){
        return balances[user];
    }
    function transfer(address recipient,uint256 amount) public  payable returns (bool) {

        require(amount <= balances[owner]);
         balances[owner]=balances[owner]-amount;
         balances[recipient]=balances[recipient]+amount;
         emit Transfer(owner,recipient,amount);
         return true;

     }

     function transfer(address from,address recipient,uint256 amount) public  payable returns (bool) {

        require(amount <= balances[from]);
         balances[from]=balances[from]-amount;
         balances[recipient]=balances[recipient]+amount;
         emit Transfer(from,recipient,amount);
         return true;

     }
```

```
You can use this terminal to:
 • Check transactions details and start debugging.
 • Execute JavaScript scripts:
     - Input a script directly in the command line interface
     - Select a Javascript file in the file explorer and then run `remix.execute()` or `remix.exeCurrent()` in the command line interface
     - Right click on a JavaScript file in the file explorer and then click `Run` The following libraries are accessible:
   o web3 version 1.0.0
   o ethers.js
   o swarmgw
   o remix (run remix.help() for more info)


creation of SimpleToken pending...


[vm] from: 0x5b3...eddc4 to: SimpleToken.(constructor) value: 0 wei data: 0x608...000c8 logs: 0 hash: 0x55c...0f0cf

>
```





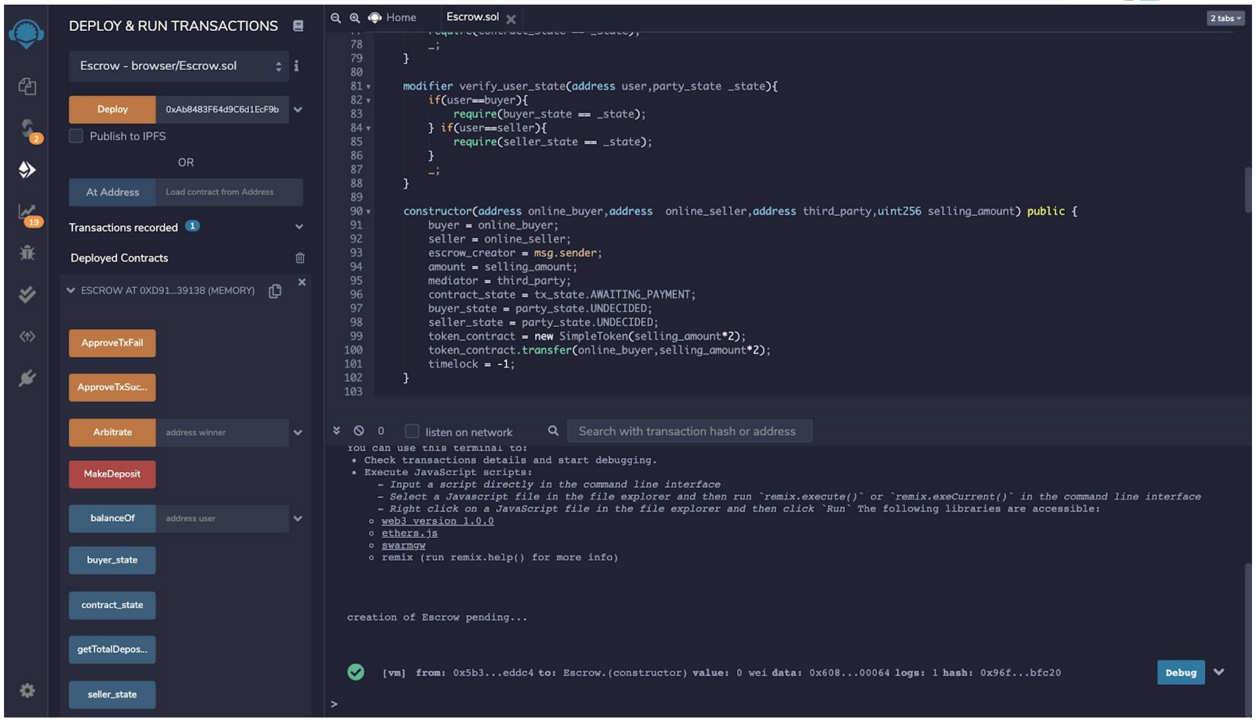
```
0:    uint256: 100
```



```
0:    uint256: 100
```

Question 2) Code Attached With file
Execution: We will pass the following arguments to run escrow smart contract.
Buyer:0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
Seller:0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
Arbitrator:0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB
Selling Amount:100 token

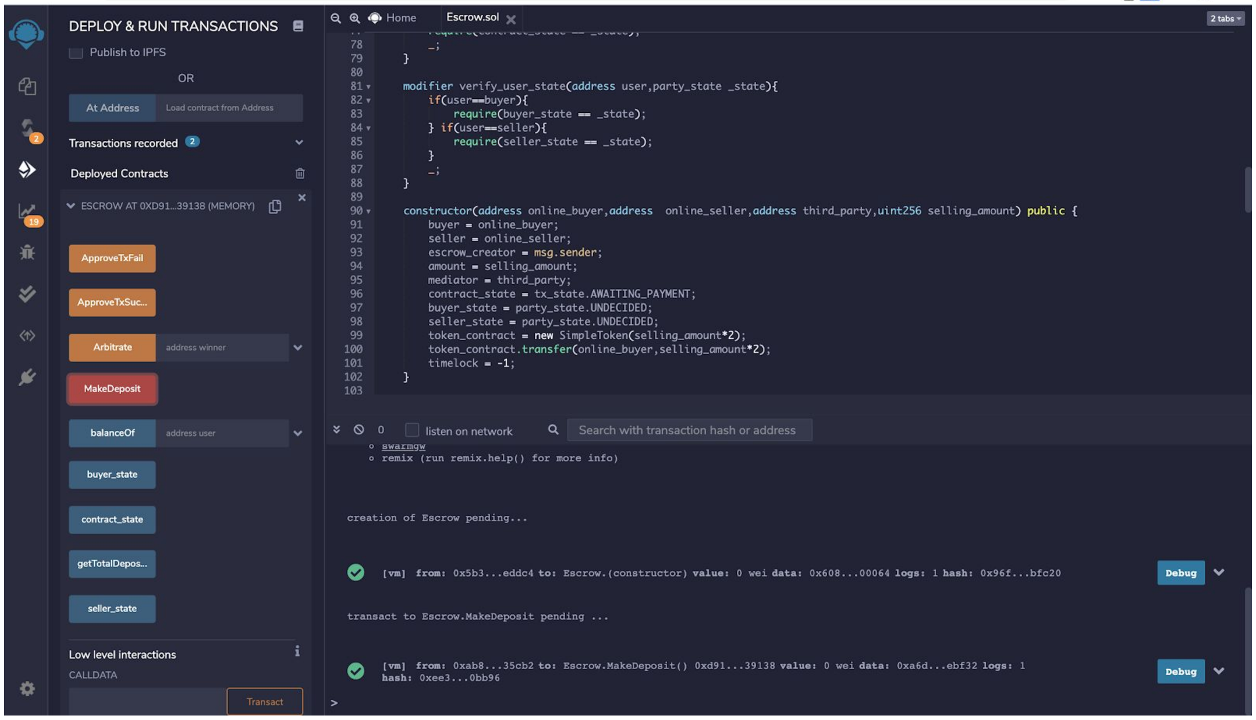
1)



```
        _;
    }

    modifier verify_user_state(address user,party_state _state){
        if(user==buyer){
            require(buyer_state == _state);
        } if(user==seller){
            require(seller_state == _state);
        }
        _;
    }

    constructor(address online_buyer,address  online_seller,address third_party,uint256 selling_amount) public {
        buyer = online_buyer;
        seller = online_seller;
        escrow_creator = msg.sender;
        amount = selling_amount;
        mediator = third_party;
        contract_state = tx_state.AWAITING_PAYMENT;
        buyer_state = party_state.UNDECIDED;
        seller_state = party_state.UNDECIDED;
        token_contract = new SimpleToken(selling_amount*2);
        token_contract.transfer(online_buyer,selling_amount*2);
        timelock = -1;
    }
```

```
You can use this terminal to:
 • Check transactions details and start debugging.
 • Execute JavaScript scripts:
     - Input a script directly in the command line interface
     - Select a Javascript file in the file explorer and then run `remix.execute()` or `remix.exeCurrent()` in the command line interface
     - Right click on a JavaScript file in the file explorer and then click `Run` The following libraries are accessible:
   o web3 version 1.0.0
   o ethers.js
   o swarmgw
   o remix (run remix.help() for more info)


creation of Escrow pending...


 [vm]  from: 0x5b3...eddc4 to: Escrow.(constructor) value: 0 wei data: 0x608...00064 logs: 1 hash: 0x96f...bfc20

>
```

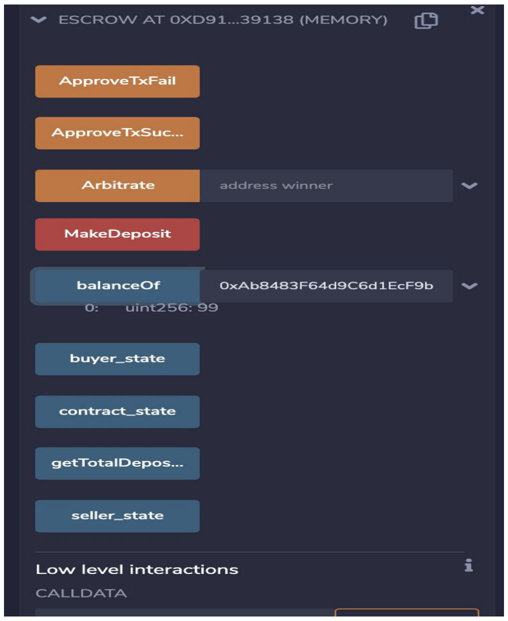
2) Calling make deposit from buyer account

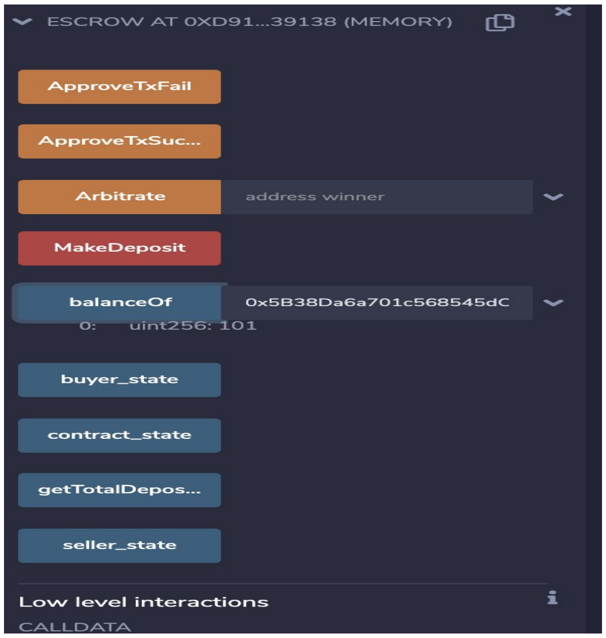


```
        _;
    }

    modifier verify_user_state(address user,party_state _state){
        if(user==buyer){
            require(buyer_state == _state);
        } if(user==seller){
            require(seller_state == _state);
        }
        _;
    }

    constructor(address online_buyer,address  online_seller,address third_party,uint256 selling_amount) public {
        buyer = online_buyer;
        seller = online_seller;
        escrow_creator = msg.sender;
        amount = selling_amount;
        mediator = third_party;
        contract_state = tx_state.AWAITING_PAYMENT;
        buyer_state = party_state.UNDECIDED;
        seller_state = party_state.UNDECIDED;
        token_contract = new SimpleToken(selling_amount*2);
        token_contract.transfer(online_buyer,selling_amount*2);
        timelock = -1;
    }
```

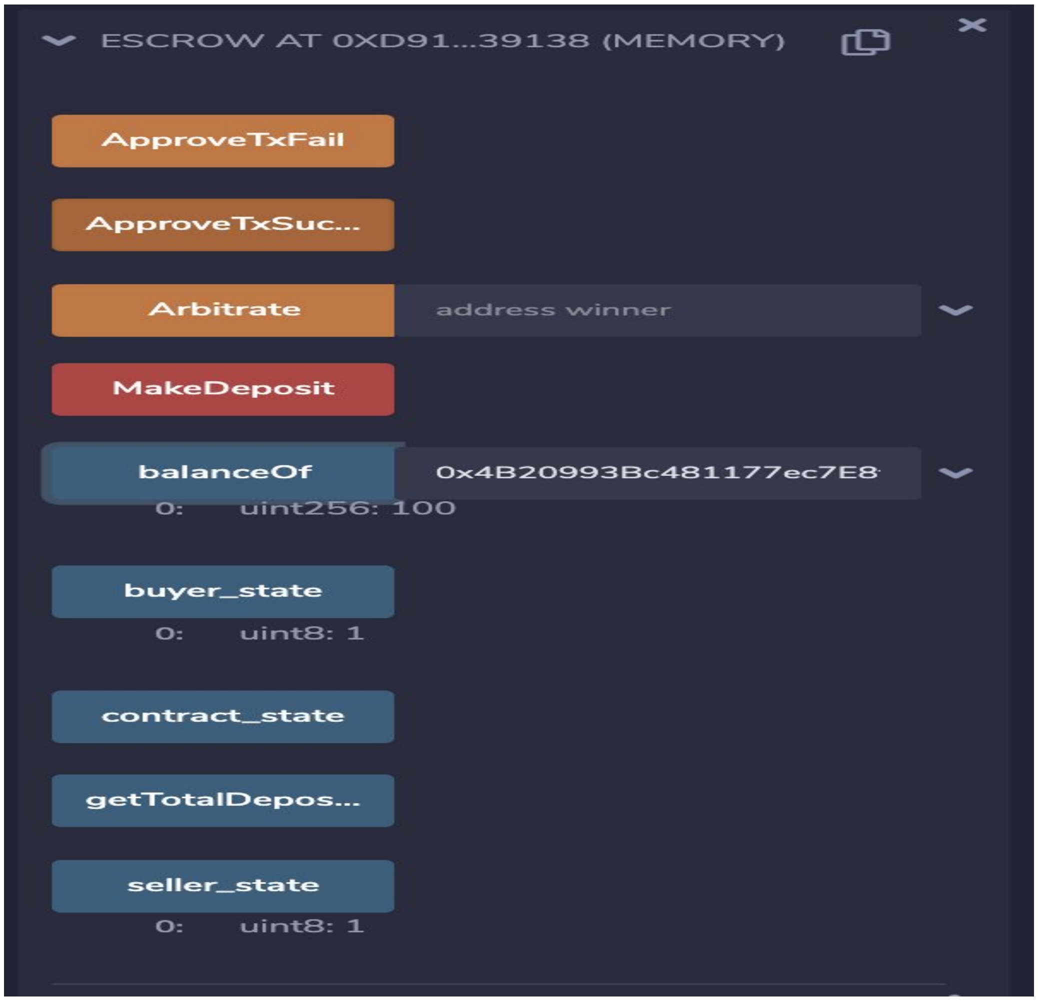
```
   o swarmgw
   o remix (run remix.help() for more info)


creation of Escrow pending...


 [vm]  from: 0x5b3...eddc4 to: Escrow.(constructor) value: 0 wei data: 0x608...00064 logs: 1 hash: 0x96f...bfc20

transact to Escrow.MakeDeposit pending ...

 [vm]  from: 0xab8...35cb2 to: Escrow.MakeDeposit() 0xd91...39138 value: 0 wei data: 0xa6d...ebf32 logs: 1
 hash: 0xee3...0bb96
>
```

3) Checking balance of buyer and Agent Smart Contract: Initial balance of buyer was 200 and selling amount is 100 , so buyer will pay 101 (100 + 1% fee) to agent.


ESCROW AT 0XD91...39138 (MEMORY)
ApproveTxFail
ApproveTxSuc...
Arbitrate
address winner
MakeDeposit
balanceOf
0xAb8483F64d9C6d1EcF9b
0: uint256: 99
buyer_state
contract_state
getTotalDepos...
seller_state
Low level interactions
CALLDATA


ESCROW AT 0XD91...39138 (MEMORY)
ApproveTxFail
ApproveTxSuc...
Arbitrate
address winner
MakeDeposit
balanceOf
0x5B38Da6a701c568545dC
0: uint256: 101
buyer_state
contract_state
getTotalDepos...
seller_state
Low level interactions
CALLDATA

4) Buyer and seller both agree about transaction success , so money is transferred to the seller



```
0:    uint256: 100
0:    uint8: 1
0:    uint8: 1
```

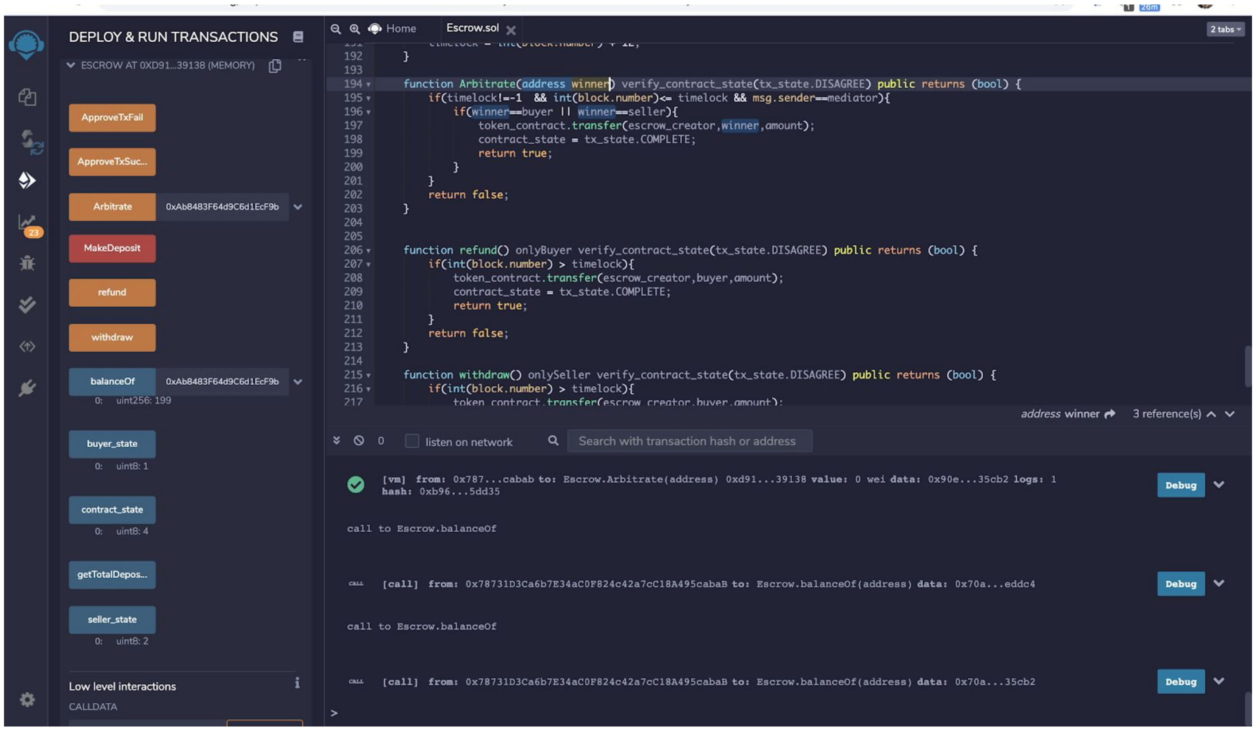
.

Question 3) Along with code in the previous question below are the additional methods which are added in order to handle dispute scenarios.

Code:

```
   function enterTimeLock() internal {
      uint256 current_block = block.number;
      timelock = int(block.number) + 12;
   }

   function Arbitrate(address winner) verify_contract_state(tx_state.DISAGREE) public returns
(bool) {
      if(timelock!=-1  && int(block.number)<= timelock && msg.sender==mediator){
         if(winner==buyer || winner==seller){
            token_contract.transfer(escrow_creator,winner,amount);
            contract_state = tx_state.COMPLETE;
            return true;
            }
        }
        return false;
    }

function refund() onlyBuyer verify_contract_state(tx_state.DISAGREE) public returns (bool) {
        if(int(block.number) > timelock){
            token_contract.transfer(escrow_creator,buyer,amount);
            contract_state = tx_state.COMPLETE;
            return true;
        }
        return false;
    }

   function withdraw() onlySeller verify_contract_state(tx_state.DISAGREE) public returns (bool)
{
        if(int(block.number) > timelock){
            token_contract.transfer(escrow_creator,buyer,amount);
            contract_state = tx_state.COMPLETE;
            return true;
        }
        return false;
    }
```

Execution Screenshot:
In the execution buyers say the transaction was successful , while the seller says the transaction failed which results in a disputed state. Mediator verifies that the buyer was correct so the smart contract refunds the money to the buyer . At the end the buyer will have 199 token (1 token was given to the mediator as a security deposit.

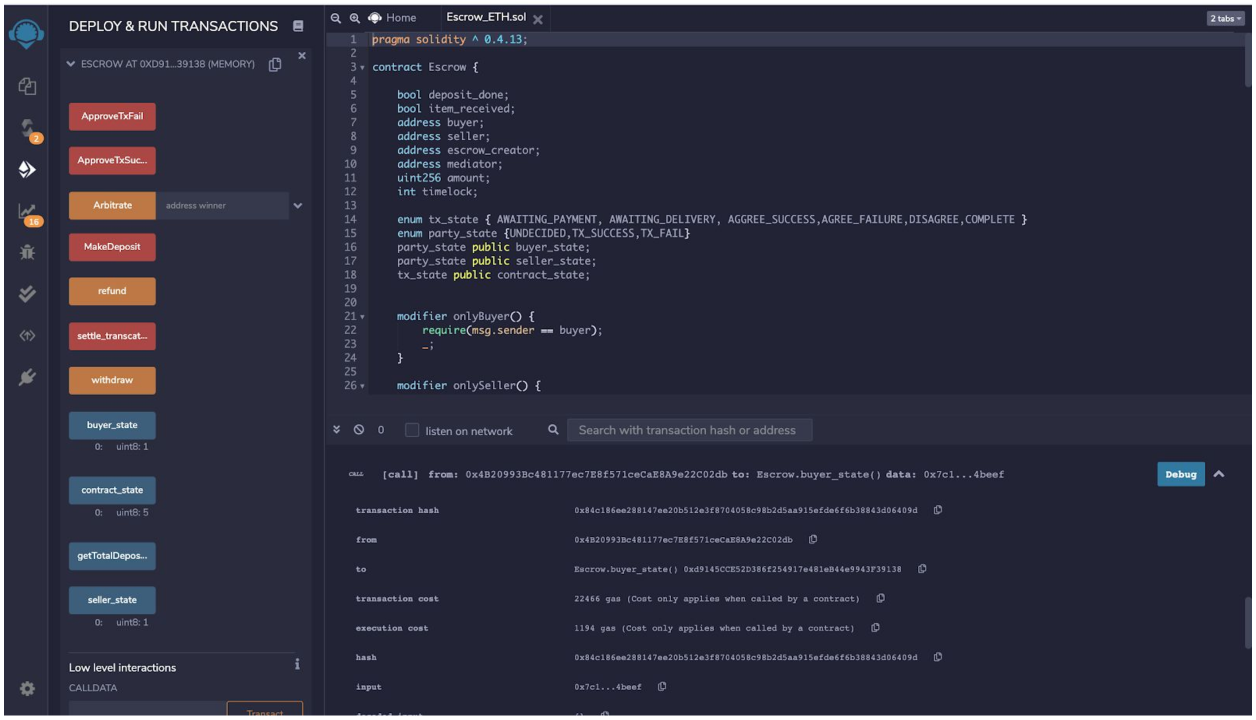


```
    }

    function Arbitrate(address winner) verify_contract_state(tx_state.DISAGREE) public returns (bool) {
        if(timelock!=-1  && int(block.number)<= timelock && msg.sender==mediator){
            if(winner==buyer || winner==seller){
                token_contract.transfer(escrow_creator,winner,amount);
                contract_state = tx_state.COMPLETE;
                return true;
            }
        }
        return false;
    }


    function refund() onlyBuyer verify_contract_state(tx_state.DISAGREE) public returns (bool) {
        if(int(block.number) > timelock){
            token_contract.transfer(escrow_creator,buyer,amount);
            contract_state = tx_state.COMPLETE;
            return true;
        }
        return false;
    }

    function withdraw() onlySeller verify_contract_state(tx_state.DISAGREE) public returns (bool) {
        if(int(block.number) > timelock){
            token_contract.transfer(escrow_creator,buyer,amount);
```

```
[vm] from: 0x787...cabab to: Escrow.Arbitrate(address) 0xd91...39138 value: 0 wei data: 0x90e...35cb2 logs: 1
hash: 0xb96...5dd35

call to Escrow.balanceOf

CALL [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB to: Escrow.balanceOf(address) data: 0x70a...eddc4

call to Escrow.balanceOf

CALL [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB to: Escrow.balanceOf(address) data: 0x70a...35cb2
>
```

Question 4)
Code : Attached in File

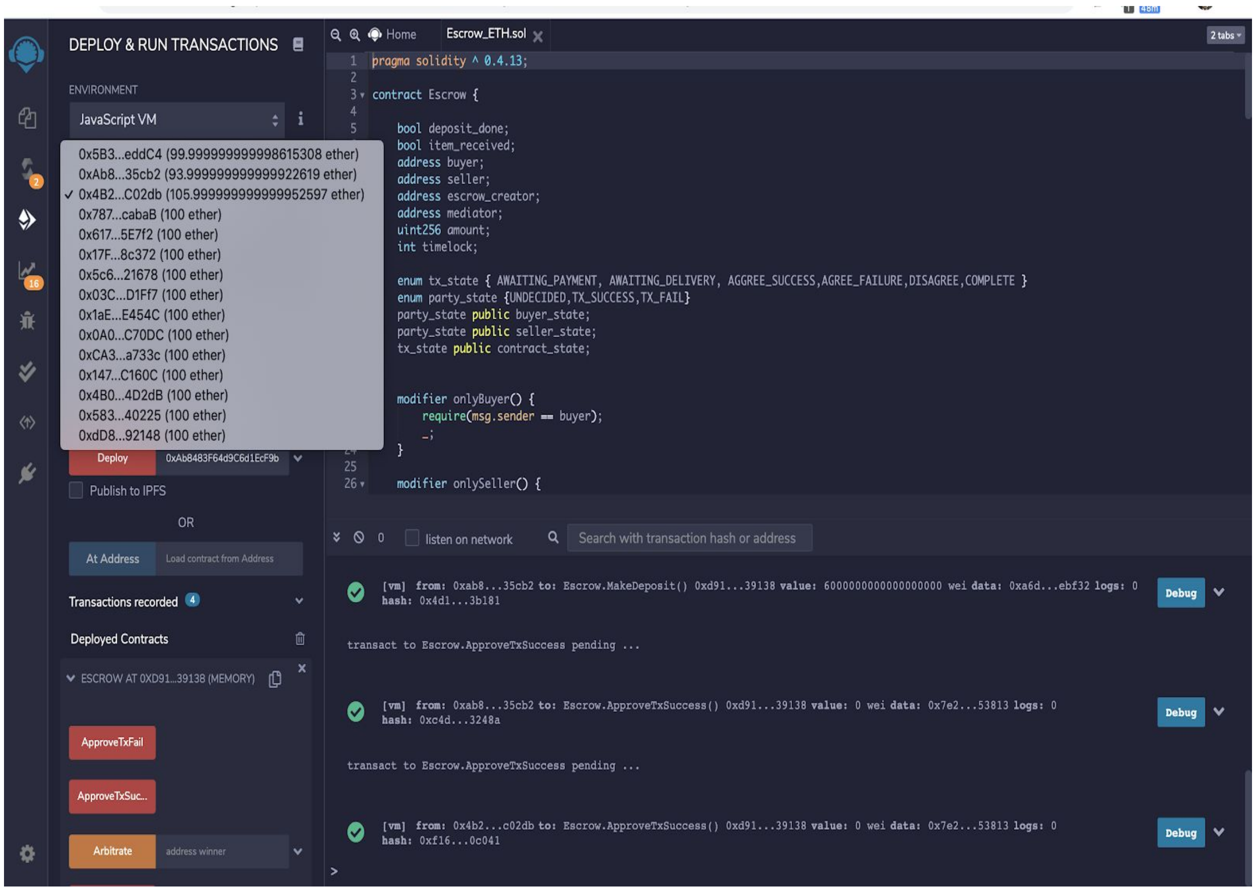
Execution : In this example below are the parameters.
Buyer:0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
Seller : 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
Arbitrator : 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB
Price : 5 Ether
After successful execution state of smart contract :



```
pragma solidity ^ 0.4.13;

contract Escrow {

    bool deposit_done;
    bool item_received;
    address buyer;
    address seller;
    address escrow_creator;
    address mediator;
    uint256 amount;
    int timelock;

    enum tx_state { AWAITING_PAYMENT, AWAITING_DELIVERY, AGGREE_SUCCESS,AGREE_FAILURE,DISAGREE,COMPLETE }
    enum party_state {UNDECIDED,TX_SUCCESS,TX_FAIL}
    party_state public buyer_state;
    party_state public seller_state;
    tx_state public contract_state;


    modifier onlyBuyer() {
        require(msg.sender == buyer);
        _;
    }

    modifier onlySeller() {
```

```
CALL [call] from: 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db to: Escrow.buyer_state() data: 0x7c1...4beef

transaction hash      0x84c186ee288147ee20b512e3f8704058c98b2d5aa915efde6f6b38843d06409d
from                  0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
to                    Escrow.buyer_state() 0xd9145CCE52D386f254917e481eB44e9943F39138
transaction cost      22466 gas (Cost only applies when called by a contract)
execution cost        1194 gas (Cost only applies when called by a contract)
hash                  0x84c186ee288147ee20b512e3f8704058c98b2d5aa915efde6f6b38843d06409d
input                 0x7c1...4beef
```

Balance of accounts:



```
0x5B3...eddC4 (99.999999999998615308 ether)
0xAb8...35cb2 (93.999999999999922619 ether)
✓ 0x4B2...C02db (105.999999999999952597 ether)
0x787...cabaB (100 ether)
0x617...5E7f2 (100 ether)
0x17F...8c372 (100 ether)
0x5c6...21678 (100 ether)
0x03C...D1Ff7 (100 ether)
0x1aE...E454C (100 ether)
0x0A0...C70DC (100 ether)
0xCA3...a733c (100 ether)
0x147...C160C (100 ether)
0x4B0...4D2dB (100 ether)
0x583...40225 (100 ether)
0xdD8...92148 (100 ether)
```

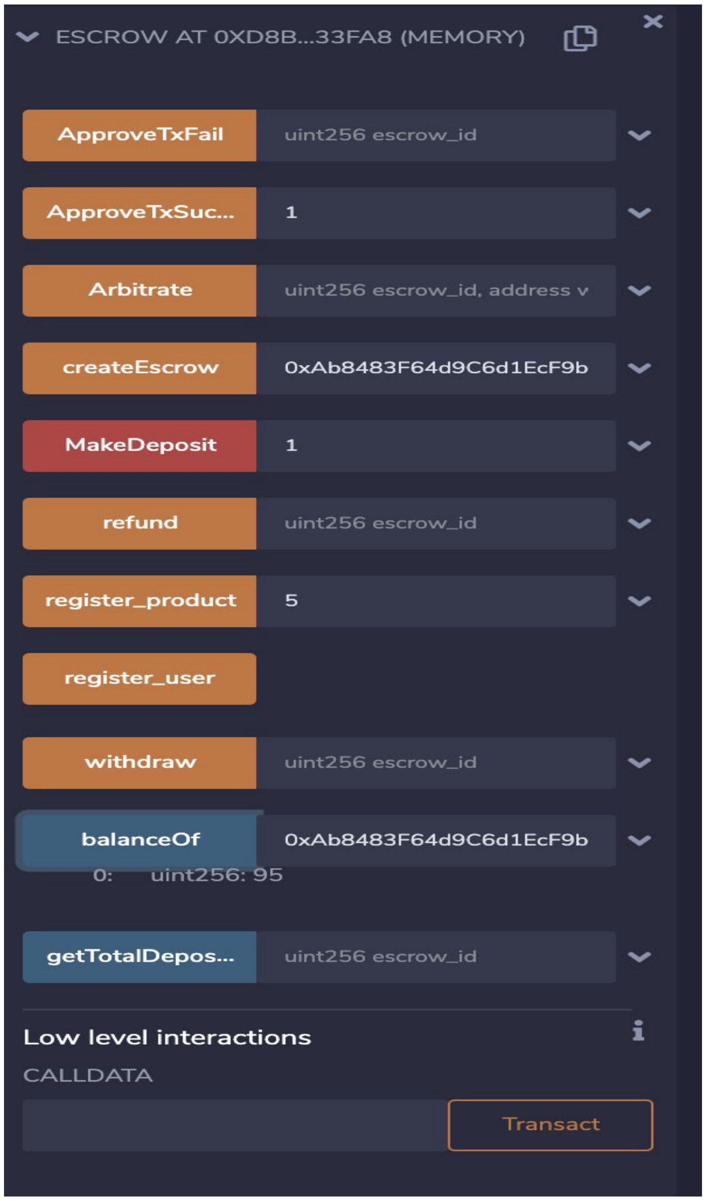
```
pragma solidity ^ 0.4.13;

contract Escrow {

    bool deposit_done;
    bool item_received;
    address buyer;
    address seller;
    address escrow_creator;
    address mediator;
    uint256 amount;
    int timelock;

    enum tx_state { AWAITING_PAYMENT, AWAITING_DELIVERY, AGGREE_SUCCESS,AGREE_FAILURE,DISAGREE,COMPLETE }
    enum party_state {UNDECIDED,TX_SUCCESS,TX_FAIL}
    party_state public buyer_state;
    party_state public seller_state;
    tx_state public contract_state;


    modifier onlyBuyer() {
        require(msg.sender == buyer);
        _;
    }

    modifier onlySeller() {
```

```
[vm] from: 0xab8...35cb2 to: Escrow.MakeDeposit() 0xd91...39138 value: 6000000000000000000 wei data: 0xa6d...ebf32 logs: 0
hash: 0x4d1...3b181

transact to Escrow.ApproveTxSuccess pending ...

[vm] from: 0xab8...35cb2 to: Escrow.ApproveTxSuccess() 0xd91...39138 value: 0 wei data: 0x7e2...53813 logs: 0
hash: 0xc4d...3248a

transact to Escrow.ApproveTxSuccess pending ...

[vm] from: 0x4b2...c02db to: Escrow.ApproveTxSuccess() 0xd91...39138 value: 0 wei data: 0x7e2...53813 logs: 0
hash: 0xf16...0c041
>
```

Question 5) Code:attached in file
For this question we have created a new structure called individual_escrow which keeps track of the state for given escrow . Each escrow has an id given to it and you can access escrow by id from mapping . Same for product, each product has an id and you can access product price from id .

Execution : First register buyer to smart contract which will transfer some tokens to the buyer , then register the product and create the escrow which will return escrow id . We will use this escrow id to execute makeDeposit(), ApproveTxSucess() or ApproveTxFail() .

ScreenShot: As we can see after execution buyer has 95 tokens(initial 100) while seller has 5 tokens (initial 0)



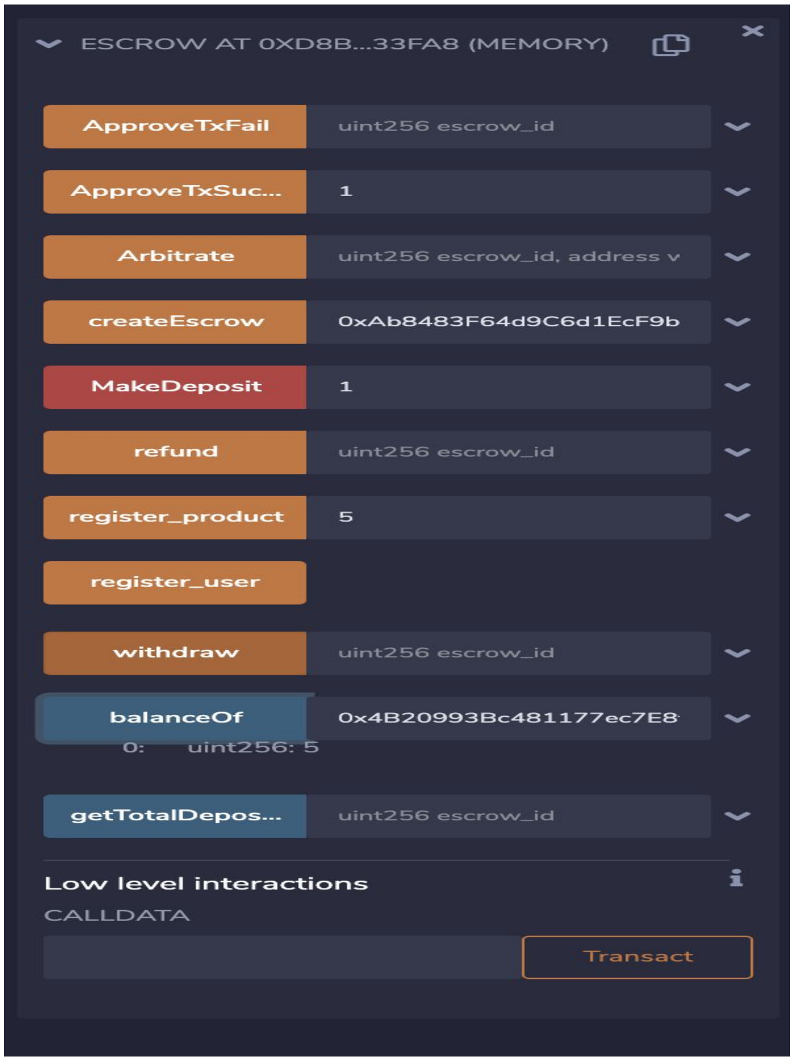
ESCROW AT 0XD8B...33FA8 (MEMORY)

ApproveTxFail	uint256 escrow_id
ApproveTxSuc...	1
Arbitrate	uint256 escrow_id, address v
createEscrow	0xAb8483F64d9C6d1EcF9b
MakeDeposit	1
refund	uint256 escrow_id
register_product	5
register_user	
withdraw	uint256 escrow_id
balanceOf	0xAb8483F64d9C6d1EcF9b

```
0:    uint256: 95
```

getTotalDepos...	uint256 escrow_id

Low level interactions
CALLDATA
Transact



ApproveTxSuc... 1

createEscrow 0xAb8483F64d9C6d1EcF9b

MakeDeposit 1

register_product 5

balanceOf 0x4B20993Bc481177ec7E8

```
0:    uint256: 5
```