Solidity Fundamentals 1

Lottery Demo

Flow:

- ❑ The manager deploys the smart contract on blockchain network
- ❑ Each player must pay at least 0.01 ether to join
- ❑ Everyone can see the list of player
- ❑ The manager choose winner randomly from the list

Use lottery.sol

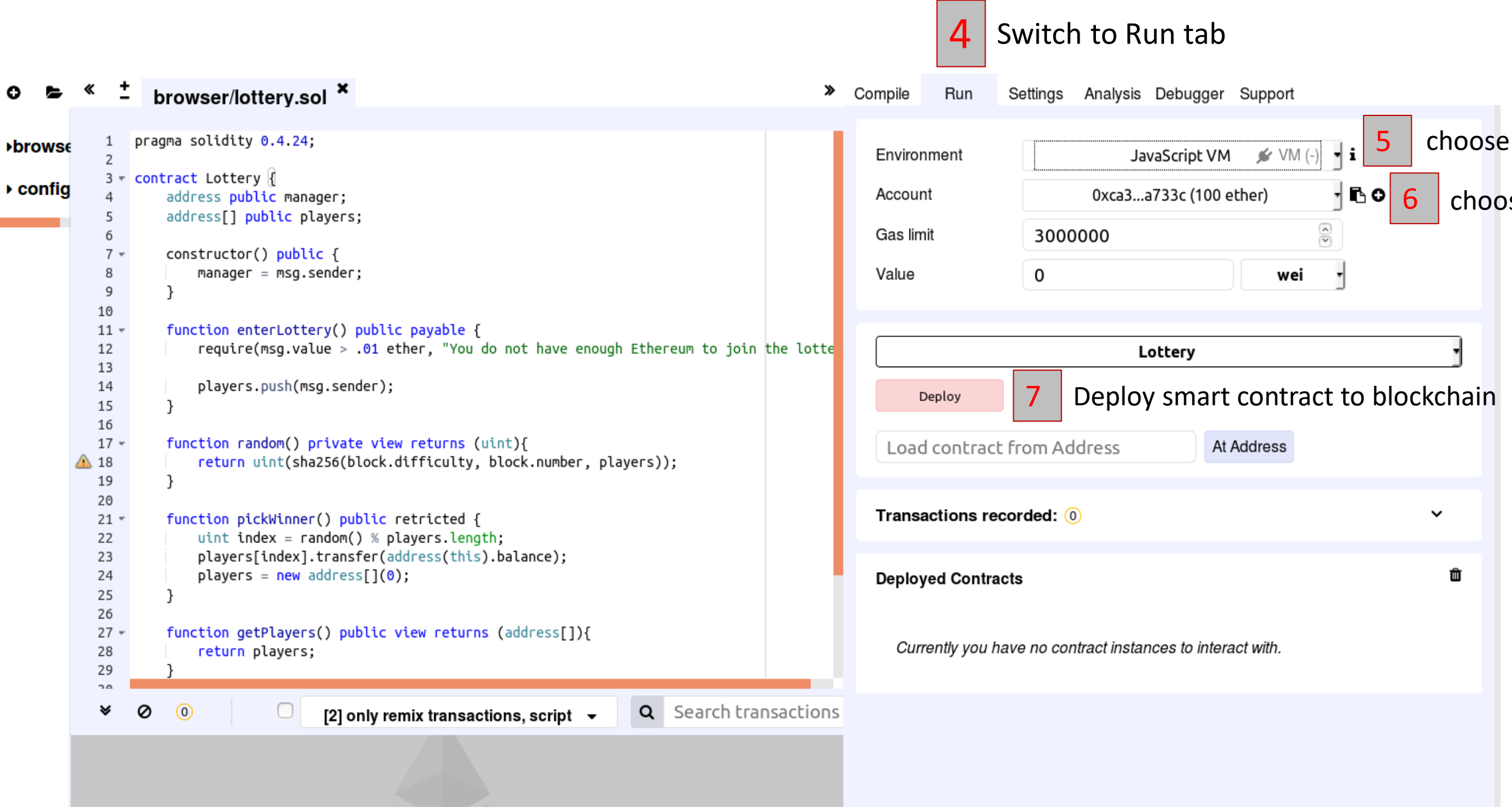
1 Create a new contract

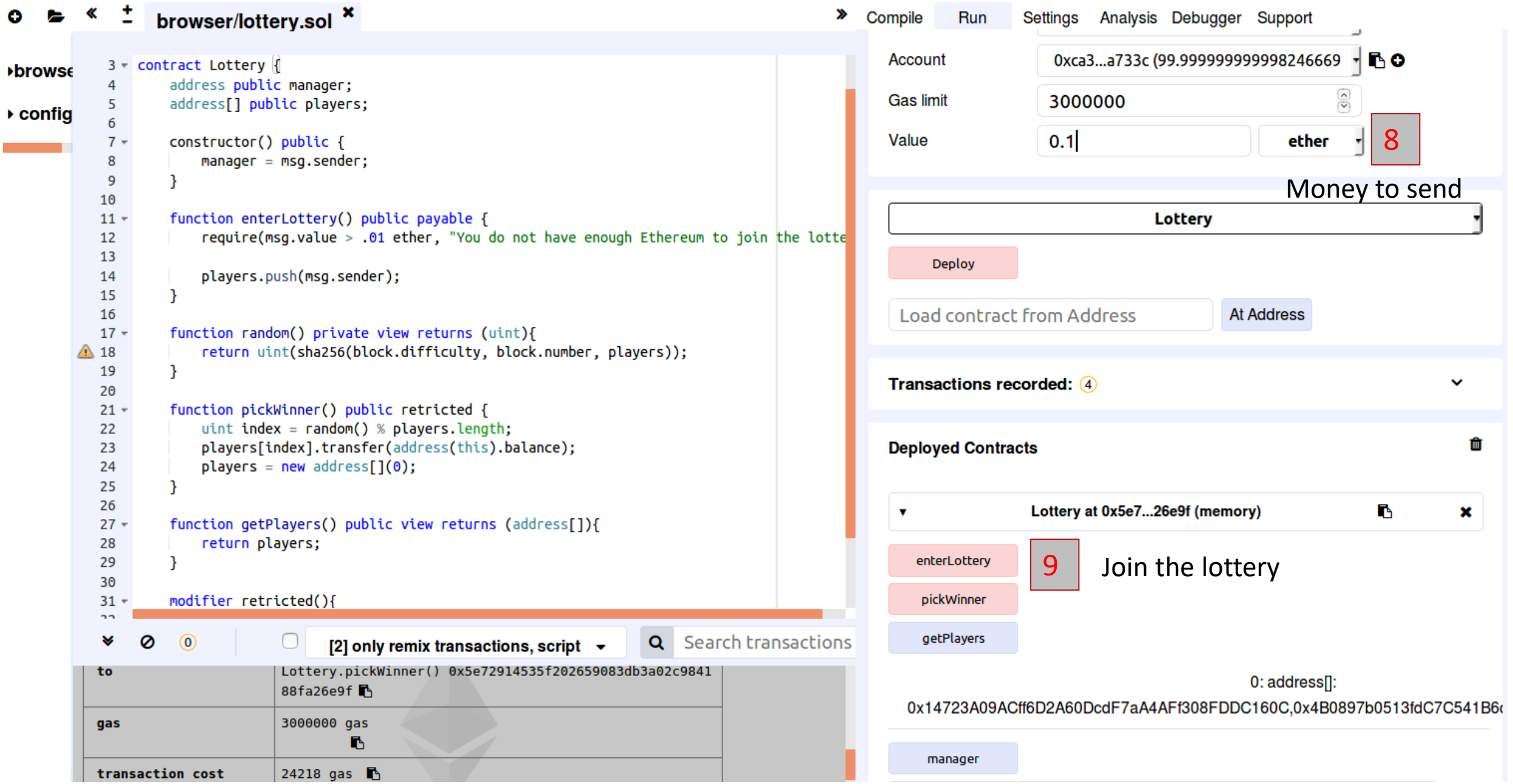
Paste your code here

2

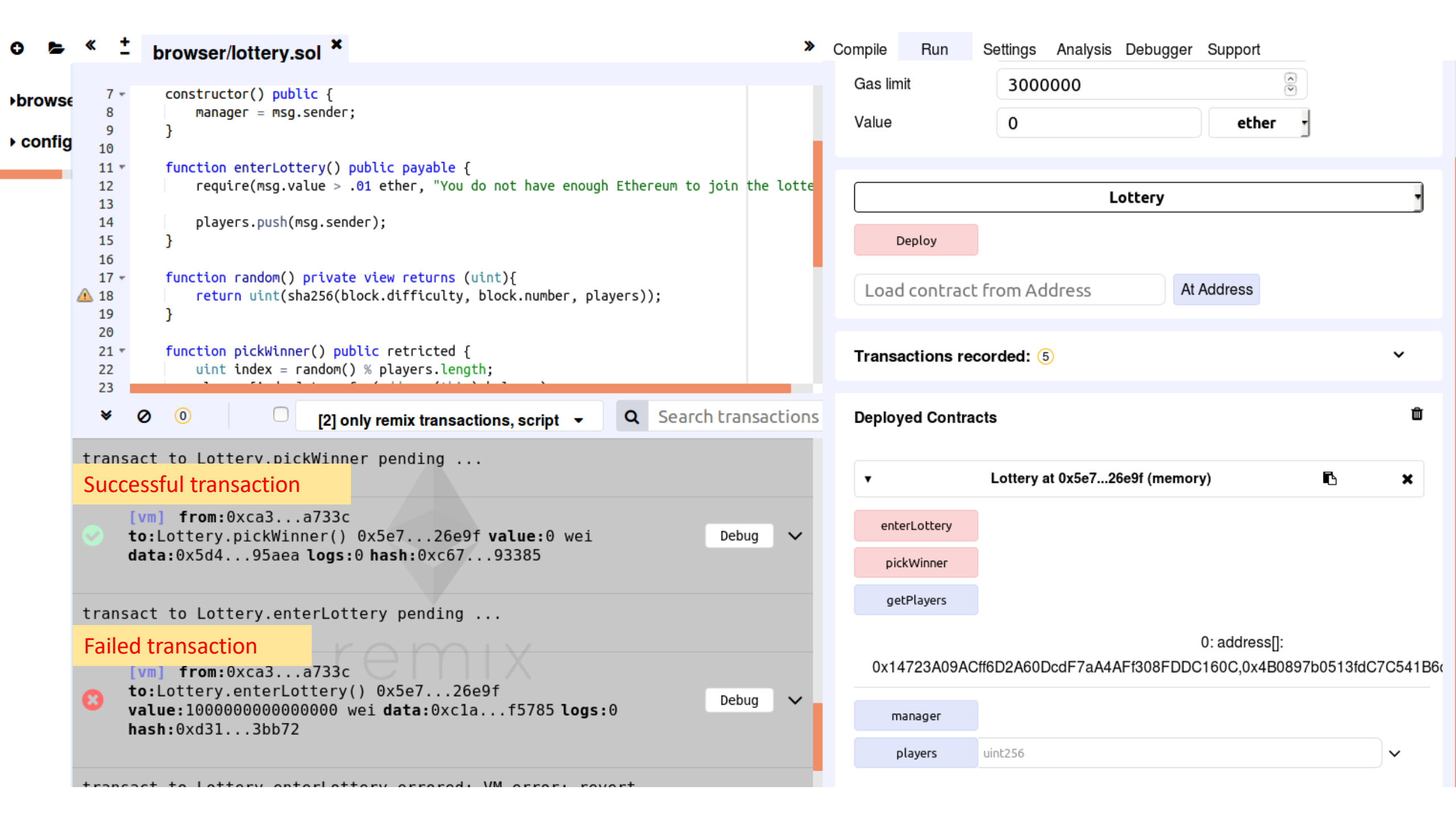
3 Compile your code


```
pragma solidity 0.4.24;

contract Lottery {
    address public manager;
    address[] public players;

    constructor() public {
        manager = msg.sender;
    }

    function enterLottery() public payable {
        require(msg.value > .01 ether, "You do not have enough Ethereum to join the lotte

        players.push(msg.sender);
    }

    function random() private view returns (uint){
        return uint(sha256(block.difficulty, block.number, players));
    }

    function pickWinner() public retricted {
        uint index = random() % players.length;
        players[index].transfer(address(this).balance);
        players = new address[](0);
    }

    function getPlayers() public view returns (address[]){
        return players;
    }
```


4
Switch to Run tab
browser/lottery.sol
Compile
Run
Settings
Analysis
Debugger
Support
Environment
JavaScript VM
VM (-)
5
choose
Account
0xca3...a733c (100 ether)
6
choose
Gas limit
3000000
Value
0
wei
Lottery
Deploy
7
Deploy smart contract to blockchain
Load contract from Address
At Address
Transactions recorded: 0
Deployed Contracts
Currently you have no contract instances to interact with.
[2] only remix transactions, script
Search transactions




```
contract Lottery {
    address public manager;
    address[] public players;

    constructor() public {
        manager = msg.sender;
    }

    function enterLottery() public payable {
        require(msg.value > .01 ether, "You do not have enough Ethereum to join the lotte

        players.push(msg.sender);
    }

    function random() private view returns (uint){
        return uint(sha256(block.difficulty, block.number, players));
    }

    function pickWinner() public retricted {
        uint index = random() % players.length;
        players[index].transfer(address(this).balance);
        players = new address[](0);
    }

    function getPlayers() public view returns (address[]){
        return players;
    }

    modifier retricted(){
```


browser/lottery.sol
Compile
Run
Settings
Analysis
Debugger
Support
Gas limit
3000000
Value
0
ether
Lottery
Deploy
Load contract from Address
At Address
Transactions recorded: 5
Deployed Contracts
Lottery at 0x5e7...26e9f (memory)
enterLottery
pickWinner
getPlayers
0: address[]:
0x14723A09ACff6D2A60DcdF7aA4AFf308FDDC160C,0x4B0897b0513fdC7C541B6
manager
players
uint256
[2] only remix transactions, script
Search transactions
transact to Lottery.pickWinner pending ...
Successful transaction
[vm] from:0xca3...a733c
to:Lottery.pickWinner() 0x5e7...26e9f value:0 wei
data:0x5d4...95aea logs:0 hash:0xc67...93385
Debug
transact to Lottery.enterLottery pending ...
Failed transaction
[vm] from:0xca3...a733c
to:Lottery.enterLottery() 0x5e7...26e9f
value:1000000000000000 wei data:0xc1a...f5785 logs:0
hash:0xd31...3bb72
Debug
function enterLottery() public payable {
require(msg.value > .01 ether, "You do not have enough Ethereum to join the lotte
players.push(msg.sender);
function random() private view returns (uint){
return uint(sha256(block.difficulty, block.number, players));
function pickWinner() public retricted {
uint index = random() % players.length;

Mapping

Example:

- Store a list of address
- Each address has a unique ID
- Given the ID, anyone can see the address

Key	Value
ID 1	Address 1
ID 2	Address 2
ID 3	Address 3

```
mapping(uint => address) public addressList
```

Syntax:

```
mapping(keyType => valueType) visibility varName
```

- **keyType**: Data type of key (uint)
- **valueType**: Data type of value (address)
- **visibility**: public, private
- **varName**: name of the reference(pointer) to the mapping object

See test2.sol

Mapping

Example:

- Store a list of address **array**
- Each address array has a unique ID
- Given the ID, anyone can see the address **array**

Key	Value
ID 1	Address Array 1
ID 2	Address Array 2
ID 3	Address Array 3

```
mapping(uint => address[]) public addressArrayList
```

*** The [] declares an array**

Access values stored in mapping

varName[key]

*We can not get all the keys and values in a mapping object. The value can only be accessed by key.

See test3.sol

Mapping

Problem:

- Store a list of unsigned integer array.
- Each unsigned integer array is linked to an address
- Given the address, anyone can access the integer array

Key	Value
Address 1	Integer array 1
Address 2	Integer array 2
Address 3	Integer array 3

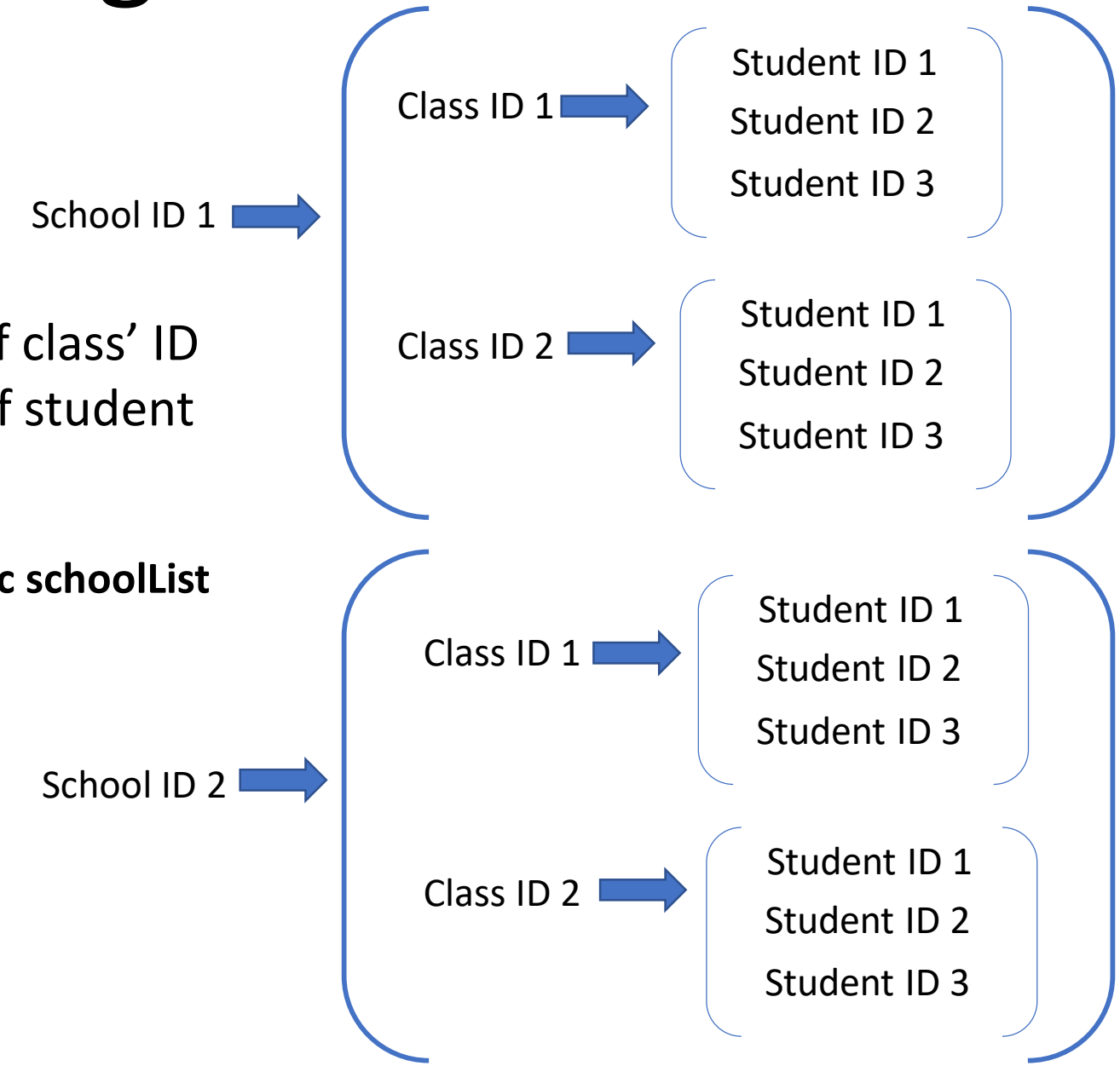
Mapping of Mapping

Example:

- Store a list of school ID
- Each school ID is used to access a list of class' ID
- Each class'ID can give access to a list of student ID

mapping(uint => (mapping(uint => uint[])) public schoolList

See test4.sol



Mapping of Mapping

Problem

- Store a list of school address
- Each school address is used to access a list of class' address
- Each class 's address can give access to an array of student ID

Struct

See test5.sol

Example:

- Each student has an ID and address of his father and address of his close friend)

```
struct Student {
    uint ID
    address father
    address closeFriend
}
```

Syntax:

```
struct structName {
        propertyType1 propertyName1
        propertyType2 propertyName2
...
}
```

Mapping of Struct

See test6.sol

Example:

- Store a list of student addresses
- Each student address is mapped to a student object that contains his ID and address of his father

Mapping of Struct

Problem

- Store a list of school address
- Each school address is used to access a list of class' address
- Each class 's address can give access to an array of student objects
- Each student object contains his ID and address of his father