Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Лабораторна робота №3

Програмування смарт-контрактів

Тема: Паттерни в мові Solidity

Виконав
студент групи ІП-11:
Панченко С. В.

Перевірив:
Яланецький В.А.

Київ 2025

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Ознайомитися з паттернами в Solidity, навчитися використовувати паттерни в смарт-контрактах.

2 ЗАВДАННЯ

За допомогою паттернів модифікувати смарт-контракт, створений у попередній роботі та протестувати його роботу.

3 ВИКОНАННЯ

3.1 Вибір варіанту. Governance

Мій день народження 6-го числа, 6 — парне число, тому виконую паттерн governance.

3.2 Опис контрактів

Під час написання коду створив три контракти:

1. MyToken. Цей контракт реалізує простий токен ERC20, де власник отримує всі токени при розгортанні. Контракт дозволяє перевіряти загальну пропозицію токенів через функцію totalSupply(), перевіряти баланс будь-якого користувача через balanceOf() та переказувати токени від відправника до іншої адреси через функцію transfer(), яка спрацьовує лише якщо відправник має достатню кількість токенів. Власник контракту визначається при створенні та має спеціальні права через модифікатор onlyOwner().
2. Governance. Цей контракт реалізує систему децентралізованого управління (governance), яка дозволяє власникам токенів голосувати за пропозиції та виконувати їх. Користувачі з принаймні одним токеном можуть створювати пропозиції через функцію propose(), голосувати за, проти або утриматись через функцію vote(), де їх голос зважується кількістю токенів. Пропозиція вважається успішною, якщо кількість голосів "за" перевищує кількість голосів "проти". Після успішного голосування, пропозицію можна виконати через функцію execute(), яка викликає вказаний контракт з переданими параметрами та значенням ETH. Контракт також має функцію receive(), що дозволяє йому отримувати ETH.
3. TargetContract. Використовується як приклад для демонстрації з Governance контрактом.

3.3 Приклад використання

Для початку створимо об’єкт MyToken, Governance та TargetContract на рисунках 3.1, 3.2.

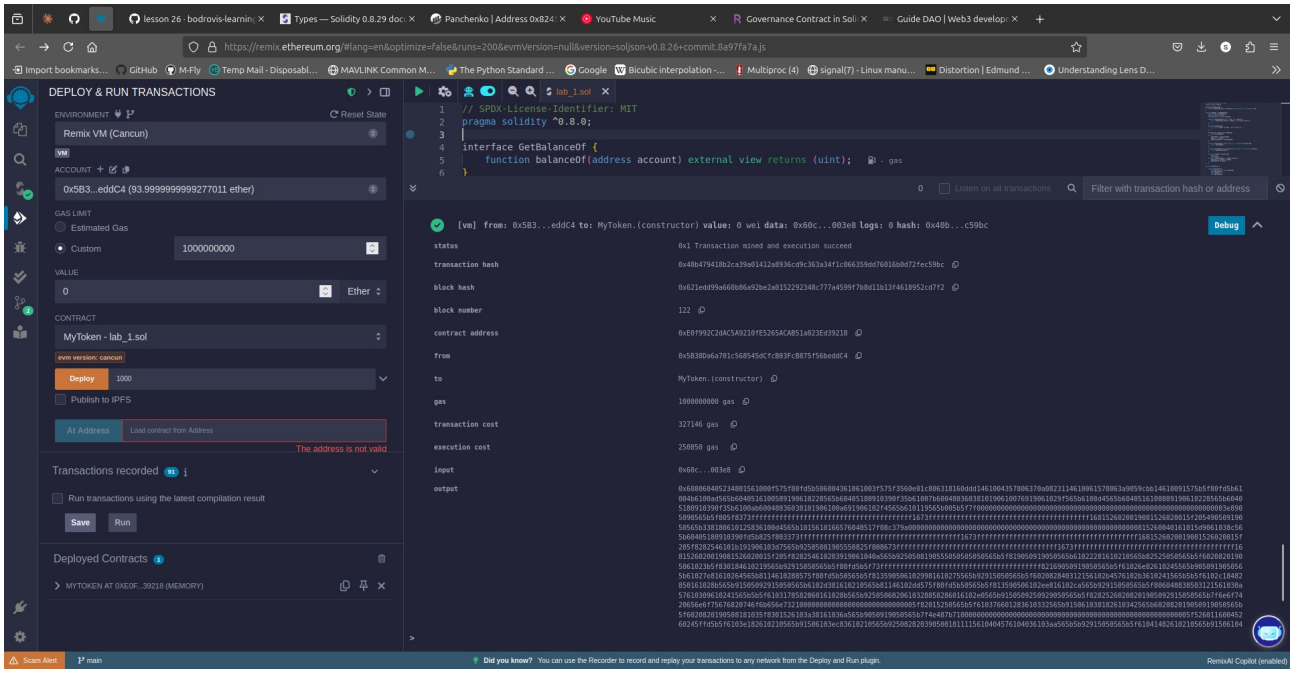

Рисунок 3.1 — створення MyToken з початковою кількістю токенів

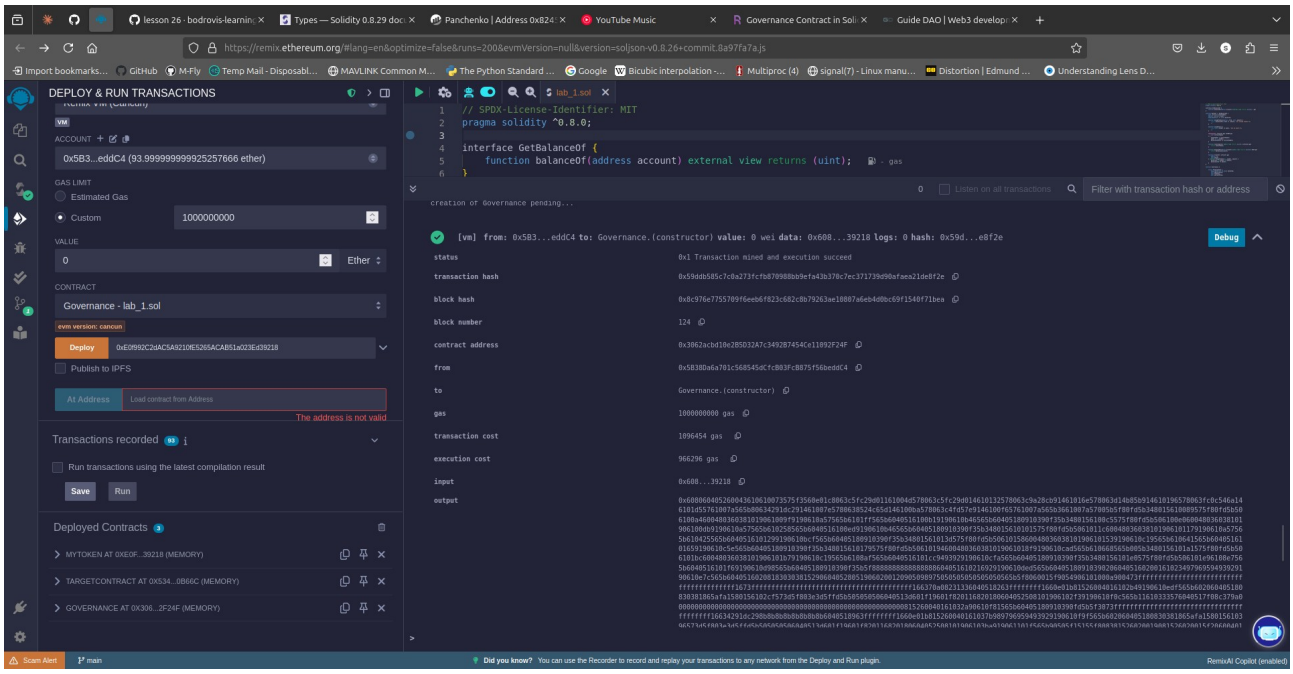

Рисунок 3.2 — створення TargetContract та Governance

Створимо прозицію: викликати метод sayHello у TargetContract на рисунку 3.3.

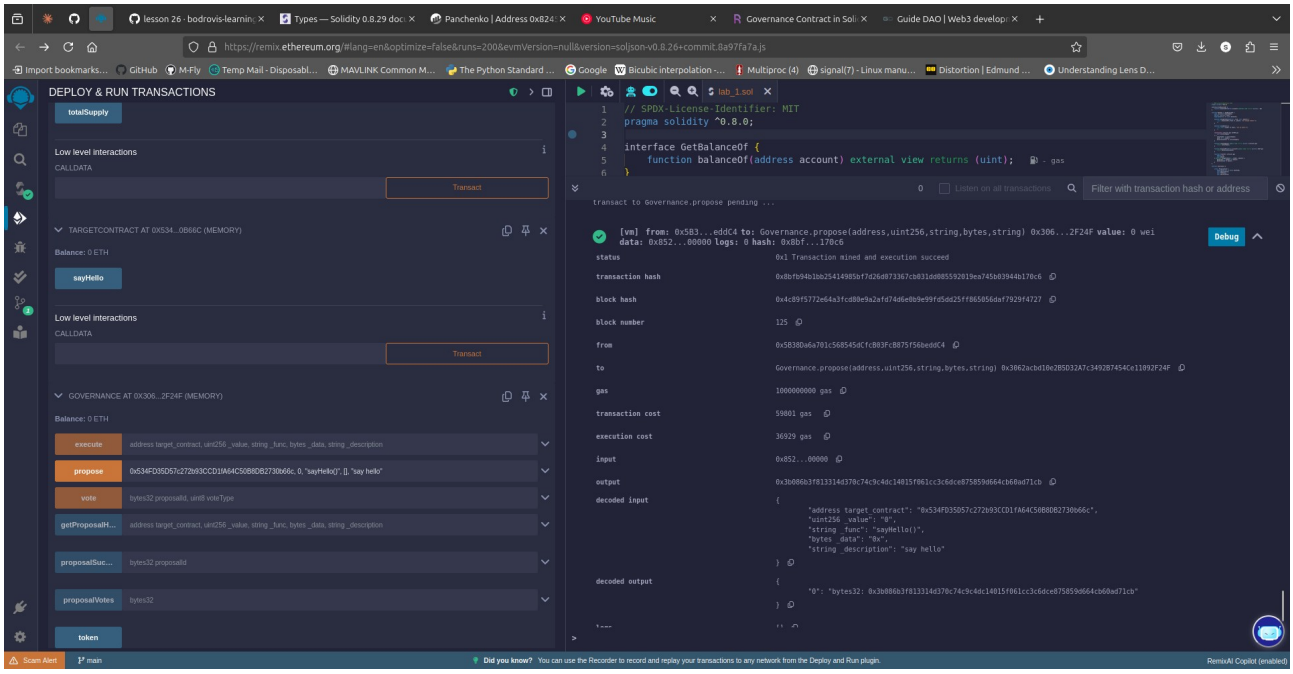

Рисунок 3.3 — створення пропозиції

Далі у тестовій мережі трьом різним акаунтам роздамо токени на рисунку 3.4.

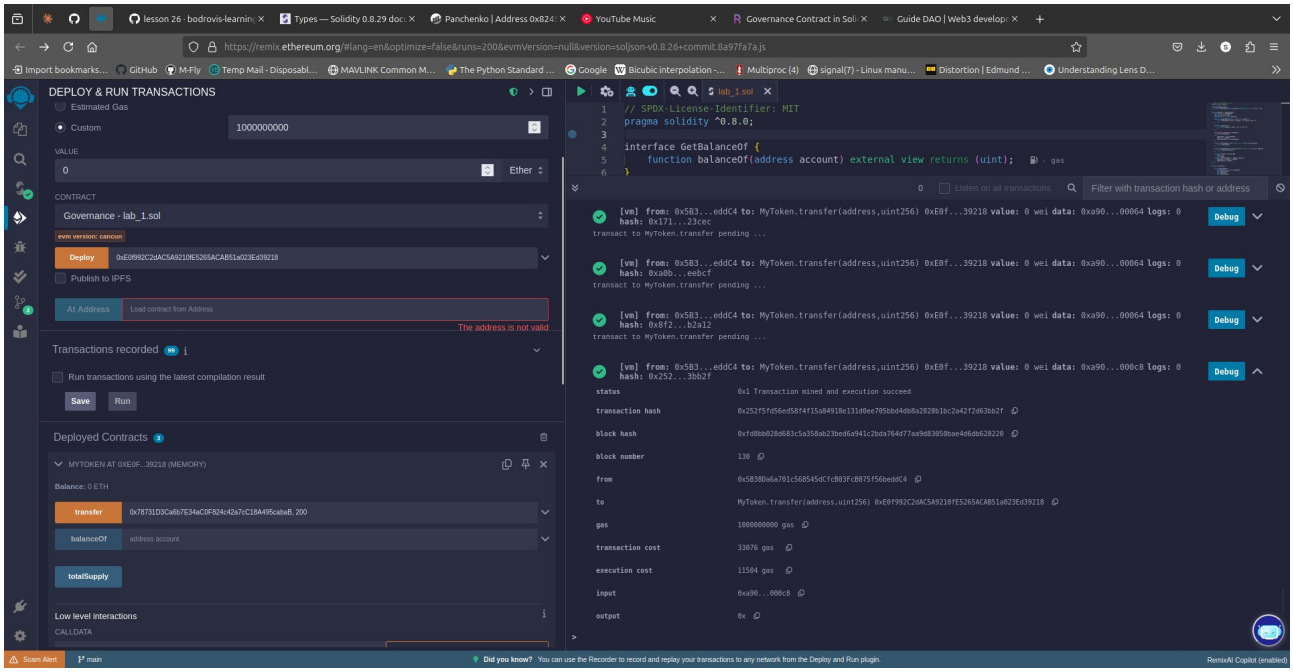

Рисунок 3.4 — роздача токенів

Проголосуємо з трьох різних акаутів за чи проти на рисунку 3.5.

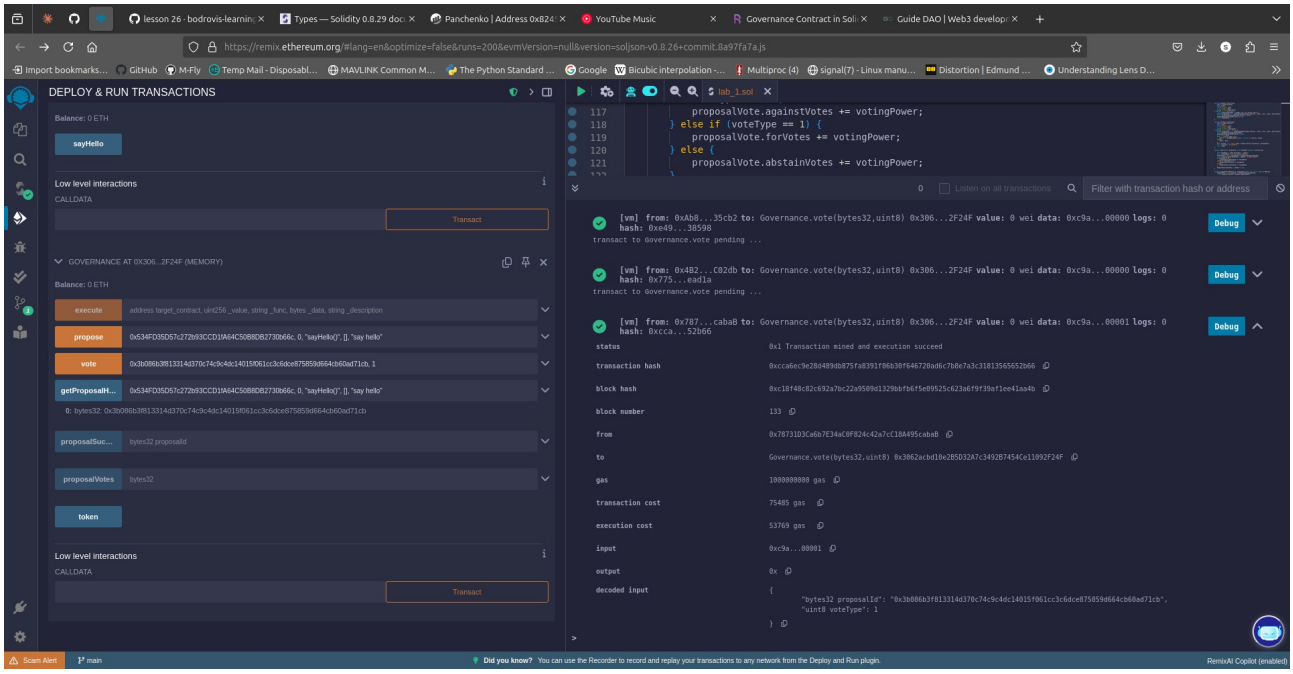

Рисунок 3.5 — процес голосування

Викликаємо метод execute щодо пропозиції та бачимо, що більшість проголосувала за виклик методу sayHello на рисунку 3.6.

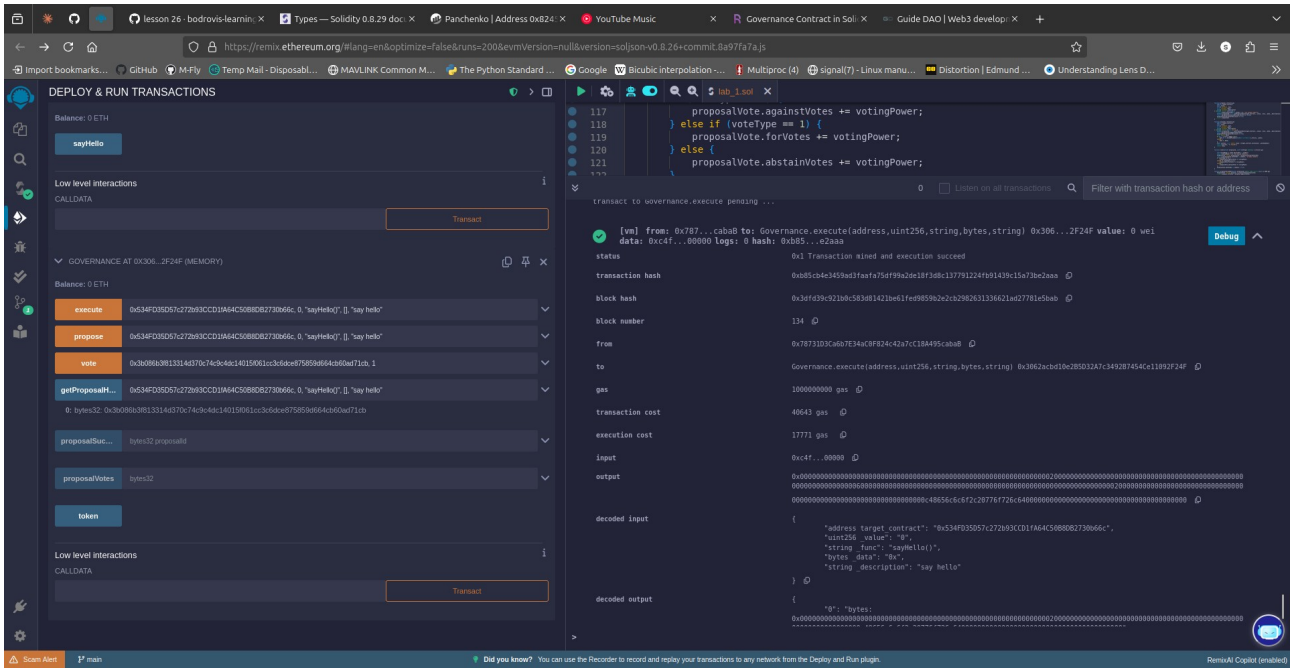

Рисунок 3.6 — виклик методу execute

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду

(Найменування програми (документа))

Жорсткий диск

(Вид носія даних)

(Обсяг програми (документа), арк.)

Студента групи ІП-11 4 курсу

Панченка С. В

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

interface GetBalanceOf {
    function balanceOf(address account) external view returns
(uint);
}

contract MyToken is GetBalanceOf {
    uint immutable totalTokens;
    address immutable owner;
    mapping(address => uint) balances;

    modifier enoughTokens(address _from, uint _amount) {
        require(balanceOf(_from) >= _amount, "not enough
tokens!");
        _;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "not an owner!");
        _;
    }

    constructor(
        uint initialSupply
    ) {
        totalTokens = initialSupply;
        owner = msg.sender;
        balances[owner] += initialSupply;
    }

    function totalSupply() public view returns (uint) {
        return totalTokens;
    }

    function balanceOf(address account) public view returns (uint)
```

```
{
        return balances[account];
    }

    function transfer(
        address to,
        uint amount
    ) external enoughTokens(msg.sender, amount) {
        balances[msg.sender] -= amount;
        balances[to] += amount;
    }
}

contract Governance {

    struct ProposalVote {
        mapping(address => bool) hasVoted;
        uint againstVotes;
        uint forVotes;
        uint abstainVotes;
        bool exists;
    }

    mapping(bytes32 => ProposalVote) public proposalVotes;
    GetBalanceOf public token;

    constructor(address target_contractken) {
        token = GetBalanceOf(target_contractken);
    }

    function getProposalHash(
        address target_contract,
        uint _value,
        string calldata _func,
        bytes calldata _data,
        string calldata _description
    ) external pure returns (bytes32) {
```

```
        return  keccak256(abi.encode(target_contract, _value,
_func, _data, keccak256(bytes(_description))));
    }

    function propose(
        address target_contract,
        uint _value,
        string calldata _func,
        bytes calldata _data,
        string calldata _description
    ) external returns (bytes32) {
        require(token.balanceOf(msg.sender) > 0, "not enough
tokens");
        bytes32 proposalId = this.getProposalHash(target_contract,
_value, _func, _data, _description);
        require(proposalVotes[proposalId].exists == false);
        proposalVotes[proposalId].exists = true;
        return proposalId;
    }

    function execute(
        address target_contract,
        uint _value,
        string calldata _func,
        bytes calldata _data,
        string calldata _description
    ) external returns (bytes memory) {
        bytes32 proposalId = this.getProposalHash(target_contract,
_value, _func, _data, _description);
        require(proposalSucceded(proposalId));
        proposalVotes[proposalId].exists = true;
        bytes memory data;
        if (bytes(_func).length > 0) {
            data =
abi.encodePacked(bytes4(keccak256(bytes(_func))), _data);
        } else {
            data = _data;
```

```
        }
        (bool success, bytes memory resp) =
target_contract.call{value: _value}(data);
        require(success, "tx failed");
        return resp;
    }

    function vote(bytes32 proposalId, uint8 voteType) external {

        uint votingPower = token.balanceOf(msg.sender);
        require(votingPower > 0, "not enough tokens");
        ProposalVote storage proposalVote =
proposalVotes[proposalId];
        require(!proposalVote.hasVoted[msg.sender], "already
voted");
        if (voteType == 0) {
            proposalVote.againstVotes += votingPower;
        } else if (voteType == 1) {
            proposalVote.forVotes += votingPower;
        } else {
            proposalVote.abstainVotes += votingPower;
        }
        proposalVote.hasVoted[msg.sender] = true;
    }

    function proposalSucceeded(bytes32 proposalId) public view
returns (bool) {
        ProposalVote storage proposal = proposalVotes[proposalId];
        return proposal.forVotes > proposal.againstVotes;
    }

    receive() external payable {}
}

contract TargetContract {
    constructor() {}
```

```
    function sayHello() external pure returns (string memory)  {
        return "Hello, world";
    }
}
```