

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря
СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Звіт з виконання комп'ютерного практикума
Реалізація смарт-контракту або анонімної криптовалюти

Виконали студенти
групи ФЕ-31мп
Мятка І.І.
Кирилюк Д.В.
Столярчук Т.В.

Перевірила:
Байденко П. В.

Мета роботи: Отримання навичок роботи із смарт-контрактами або анонімними криптовалютами.

Для другого типу лабораторних робіт:

Розробка власного смарт-контракту.

Теоретичні відомості:

Смарт-контракти - це набір інструкцій, які можуть бути виконані без втручання третіх сторін. Код смарт-контракту визначає, як він реагує на вхідні дані, як і код будь-якої іншої комп'ютерної програми.

Найпопулярнішою мовою для написання смарт-контрактів на ланцюжках Ethereum і EVM є Solidity. Вона була створена Ethereum Foundation спеціально для розробки смарт-контрактів і постійно оновлюється. Існують й інші мови для написання смарт-контрактів на ланцюжках Ethereum і EVM.

Як правило, усі контракти Solidity містять такі елементи:

- **Директиви *Pragma*** - це ключове слово, за допомогою якого компілятор перевіряє відповідність версії Solidity необхідній. Якщо версії збігаються, файл може бути виконаний. В іншому разі компілятор повертає помилку.
- **Змінні стану** - є основними елементами будь-якого вихідного файлу Solidity. Значення змінних стану назавжди зберігаються у сховищі контрактів. У визначенні будь-якої змінної необхідно вказувати її тип та ім'я.

Крім того, ви можете задати видимість змінної стану таким чином:

- `public` - елемент інтерфейсу контрактів, доступний з інших контрактів.
- `internal` - елемент, доступний тільки з поточного контракту.
- `private` - елемент, доступний тільки в тому контракті, в якому він визначений.
- `external` - неможливо отримати доступ зсередини, тільки ззовні
- **Функції** - у рамках контракту виконувані одиниці коду називаються функціями. За допомогою функцій описуються окремі дії, необхідні для виконання загального завдання. Функції можна використовувати повторно і викликати з інших вихідних файлів, наприклад бібліотек. Поведінка функцій у Solidity аналогічна іншим мовам програмування. Підтримуються такі описувачі видимості функцій: `public`, `private`, `internal` і `external`. За допомогою модифікаторів можна змінювати поведінку функцій. Якщо задано модифікатор, перед виконанням функції перевіряється відповідна умова.

У визначенні функції також можна використовувати такі модифікатори:

- `pure` - описує функції, для яких не допускається зміна стану або доступ до відомостей про нього.
- `view` - описує функції, для яких не допускається зміна стану.
- `payable` - описує функції, які можуть приймати «ефіри».
- **Події** – за допомогою подій описуються дії, які виконуються в контракті. Як і функції, події використовують параметри, які необхідно задавати під час їхнього виклику. Щоб викликати подію, необхідно використовувати ключове слово `emit`, вказавши ім'я події та її параметри. Під час виклику події її фіксують як транзакцію в журналі транзакцій, який являє собою спеціальну структуру даних у блокчейні. Такі журнали пов'язуються з адресою контракту, включаються в блокчейн і залишаються в ньому назавжди. Журнал і дані про події, що містяться в ньому, недоступні з контрактів і не можуть бути змінені.

Хід виконання:

В якості прикладу розробки власного смарт-контракту була розроблена гра, де в якості плати користувачу запропоновано внести плату в деяку кількість ЕТН. Також була визначена логіка вибору «рандомного» числа. Плата за гру вноситься на банк, якщо користувач правильно вгадає число то отримує весь банк який накопився. В якості прикладу також була реалізована функція зняття балансу для власника контракту.

1) Написання коду

Код було написано на мові Solidity з використанням IDE Ethereum Remix.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract NumberGuessingGame {
    address public owner;
    uint256 public randomNumber;
    address public lastWinner;
    uint256 public bankBalance;
    uint256 public guessCost = 1 ether;

    event NumberGuessed(address player, uint256 guessedNumber, uint256
randomNumber, bool won);
    event NewGameStarted(address owner, uint256 randomNumber);

    constructor() {
```

```

    owner = msg.sender;
}

modifier onlyOwner() {
    require(msg.sender == owner, "Only owner can call this function.");
    _;
}

function generateRandomNumber() internal {
    uint256 blockHashRandom = uint256(blockhash(block.number - 1));
    randomNumber = (blockHashRandom % 10) + 1;
    emit NewGameStarted(owner, randomNumber);
}

function guessNumber(uint256 _guessedNumber) external payable {
    require(msg.value >= guessCost, "Insufficient funds. Please send at least 1 ETH to play.");
    require(_guessedNumber >= 1 && _guessedNumber <= 10, "Invalid guess. Please guess a number between 1 and 10.");

    if (lastWinner == address(0) || msg.sender == lastWinner) {
        generateRandomNumber();
    }

    emit NumberGuessed(msg.sender, _guessedNumber, randomNumber, _guessedNumber == randomNumber);

    if (_guessedNumber == randomNumber) {
        payable(msg.sender).transfer(bankBalance + guessCost);
        bankBalance = 0;
        lastWinner = msg.sender;
    } else {
        bankBalance += guessCost;
    }
}

function withdraw() external onlyOwner {
    require(bankBalance > 0, "Bank balance is empty.");
    payable(owner).transfer(bankBalance);
    bankBalance = 0;
}

function getContractBalance() external view returns (uint256) {
    return address(this).balance;
}
}

```

В блоці визначення змінних виділено:

- 1) Змінні типу `address` для власника (`owner`) та останнього переможця (`lastWinner`)
- 2) Змінні типу `uint256` для рандомного числа (`randomNumber`), балансу банку (`randomNumber`) та визначена «константа» для ціни (`guessCost`)

В блоці визначення подій визначені:

- 1) `NumberGuessed` - викликається коли гравець вгадує номер. Подія містить адресу гравця, вгадане число, згенероване випадкове число та інформацію про те, чи виграв гравець (змінна типу `bool`).
- 2) `NewGameStarted` - викликається коли починається нова гра. Подія містить адресу власника контракту та щойно згенероване випадкове число.

В конструкторі визначено значення змінної `owner` як адресу того, хто розмістив контракт.

За ним визначено модифікатор `onlyOwner`, що говорить про те що тільки власник може викликати цю функцію. В даному випадку це відноситься до функції `withdraw`.

Для формування рандомного числа була використана функція `blockHashRandom`, яка використовує `blockhash`. Звісно така практика не є безпечною. Блок-хеш - це випадкове значення `uint256`, але воно вже детерміноване. Кожного разу викликаючи функцію `blockhash()` з одним і тим же параметром, ми будемо отримувати одне і те ж значення. Тому легко написати так звану «атаку» на контракт, де можливо буде 100% вгадати число. Для запобігання цьому прийнято використовувати рандомні значення, які були добуті НЕ в блокчейні. Прийнятою практикою є використання **Chainlink VRF**, це чесний генератор випадкових чисел, який дозволяє смарт-контрактам отримувати доступ до випадкових значень без шкоди для безпеки та зручності використання. Для кожного запиту **Chainlink VRF** генерує одне або декілька випадкових значень і криптографічний доказ того, як ці значення були визначені. Доказ публікується і перевіряється в ланцюжку, перш ніж будь-який додаток, що його використовує, зможе його використати. Цей процес гарантує, що результати не можуть бути підроблені або маніпульовані жодним суб'єктом, включаючи майнерів, користувачів або розробників смарт-контрактів.

В контексті лабораторної роботи, враховуючи що це тестова мережа та не критичний додаток, було прийнято рішення використати простіший спосіб з використанням `blockhash`. В результаті виконання «рандомне» число

перекидується аргументом до виклику події `NewGameStarted` разом з адресою власника.

Основна функція `guessNumber` має параметр `payable`, що дозволяє здійснювати надходження ЕТН. Звісно перед початком перевіряється умова на те чи `msg.value`, тобто вхідний ЕТН, був рівним вказаному раніше 1 ЕТН. Також перевіряється умова «вгаданого» числа в межах від 1 до 10. При невиконанні однієї з умов контракт не пройде та видасть відповідну помилку.

Наступна умова змушує контракт виконувати генерацію «рандомного» числа при тому випадку коли гравець виграв або програв. Тобто це зроблено для того щоб при програші виконувалась генерація нового числа.

Після чого виконується тригер події з передачею відповідних значень.

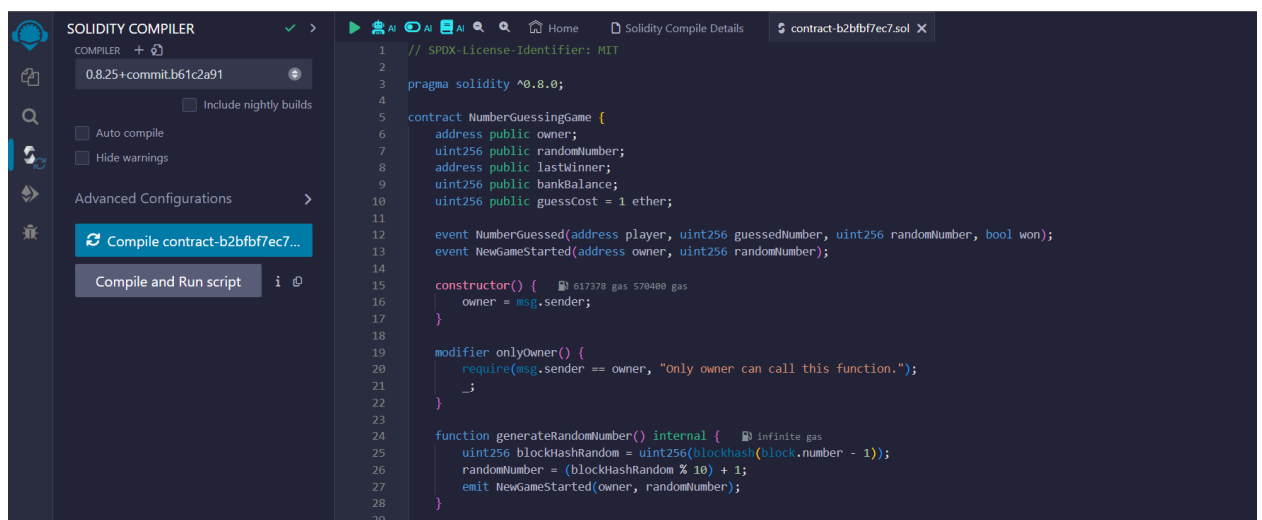
Логіка розподілення фінансів проста: якщо вгадано число правильно то до користувача що вгадав переводиться увесь баланс банку та власне сума ставки. Якщо ж ні то до суми в банку приплюсовується сума ставки.

Функція `withdraw` перевіряє наявність ненульового значення в банку. Знімання коштів виконується лише для власника контракту. Звісно в реальному контракті це буде дещо нечесним, оскільки власник влюбий момент може зняти кошти, не давши навіть шансу відігратись. Проте в якості приклада функції з параметром `external` була реалізована.

Остання функція `getContractBalance` слугує для перевірки балансу банку. Функція має параметр `view`, тобто ми звертаємось до змінної блокчейну, проте не змінюємо її.

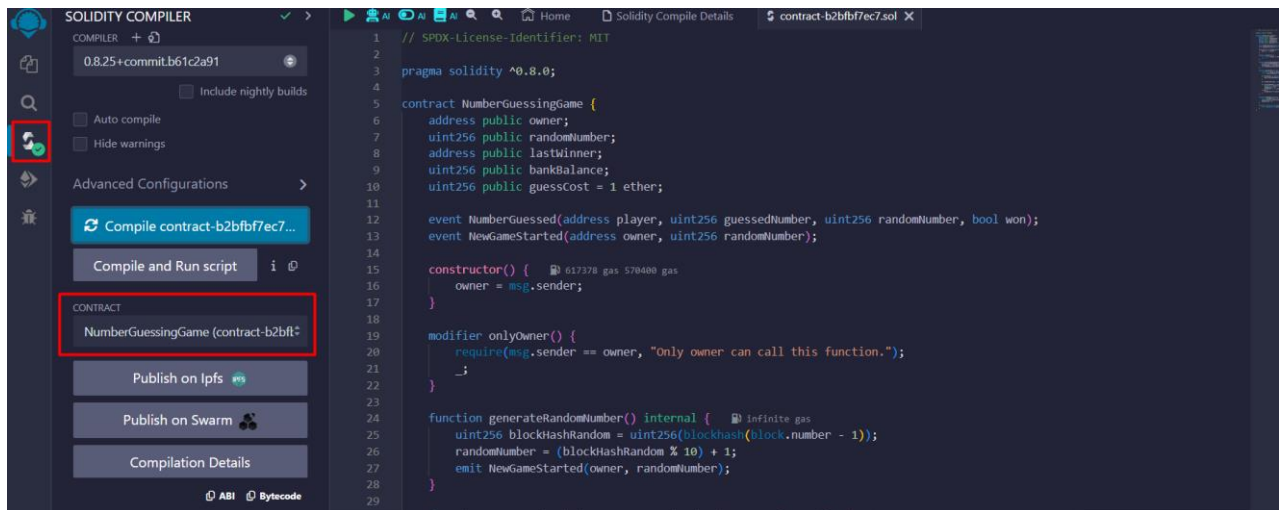
2) Перевірка роботи на Remix

Для початку перенесемо код в IDE:

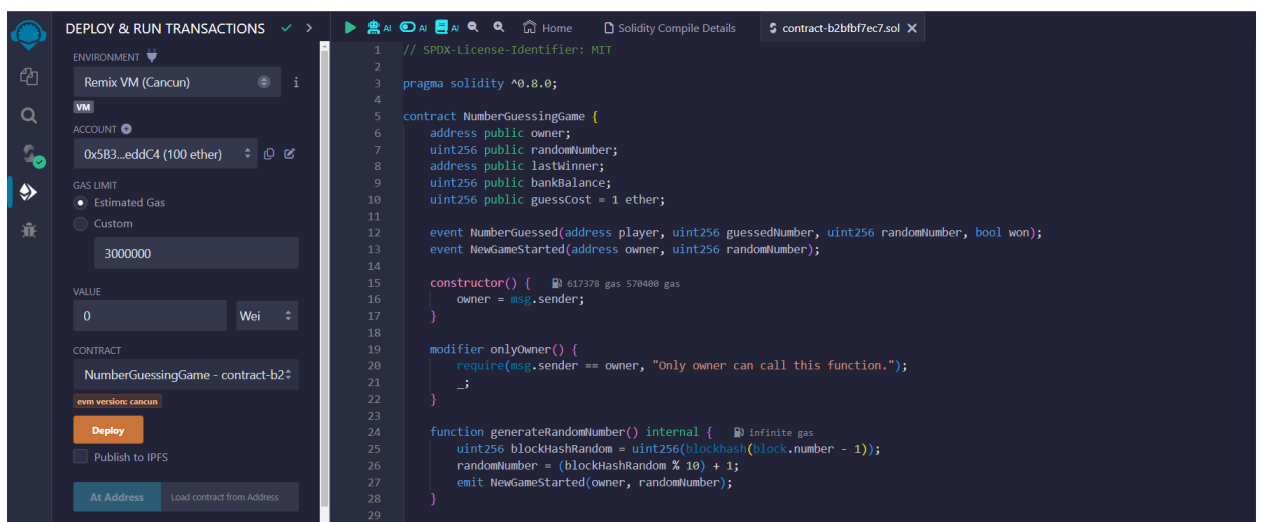


```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.0;
4
5 contract NumberGuessingGame {
6     address public owner;
7     uint256 public randomNumber;
8     address public lastWinner;
9     uint256 public bankBalance;
10    uint256 public guessCost = 1 ether;
11
12    event NumberGuessed(address player, uint256 guessedNumber, uint256 randomNumber, bool won);
13    event NewGameStarted(address owner, uint256 randomNumber);
14
15    constructor() {
16        // 617378 gas 570480 gas
17        owner = msg.sender;
18    }
19
20    modifier onlyOwner() {
21        require(msg.sender == owner, "Only owner can call this function.");
22        _;
23    }
24
25    function generateRandomNumber() internal {
26        // infinite gas
27        uint256 blockHashRandom = uint256(blockhash(block.number - 1));
28        randomNumber = (blockHashRandom % 10) + 1;
29        emit NewGameStarted(owner, randomNumber);
30    }
31}
```

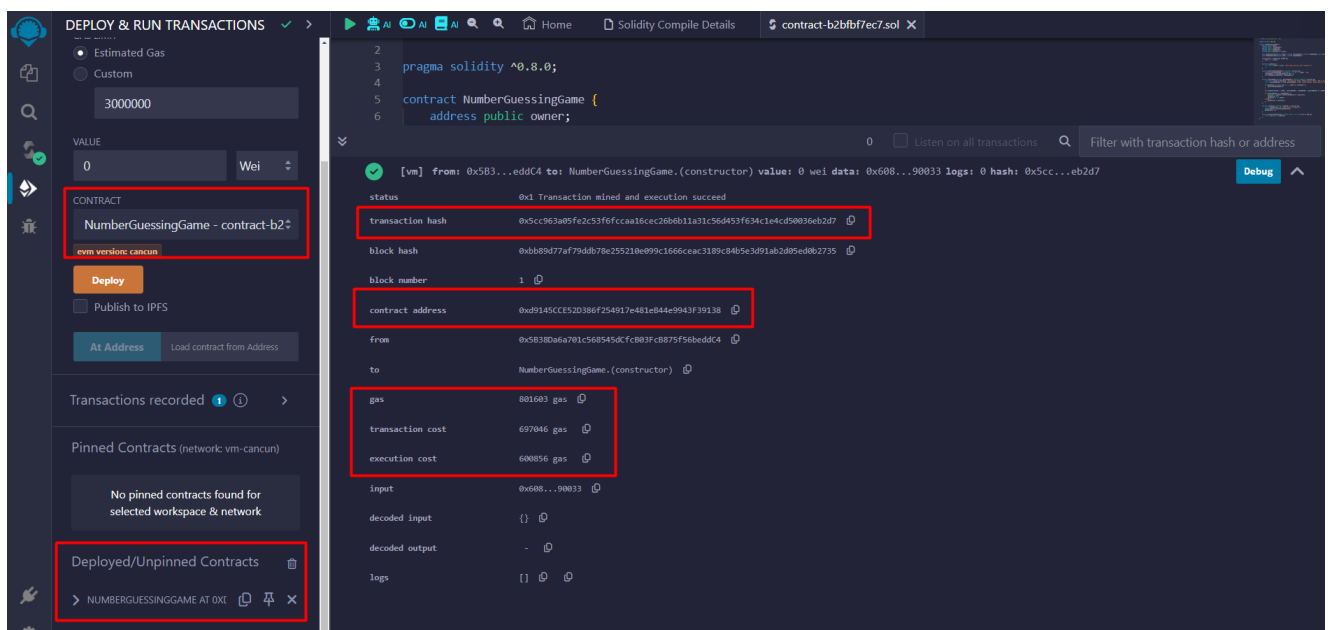
Скомпілюємо контракт:



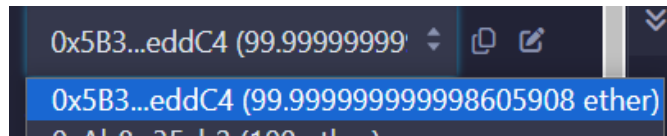
Перейдемо в розділ Deploy & run transactions:



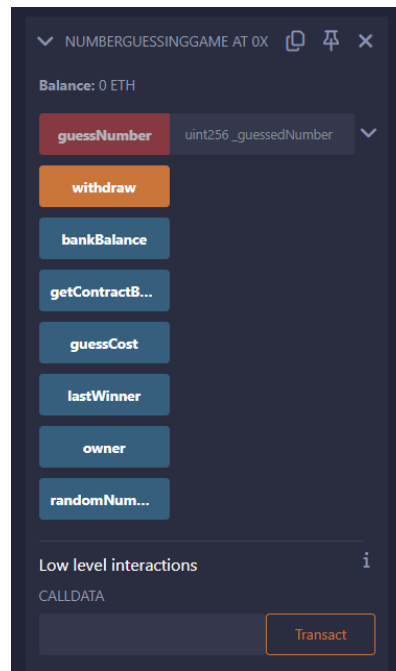
Для початку виконаємо операції в тестовому блокчейні Remix. Для цього з першого акаунту 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 виконаємо деплой контракту.



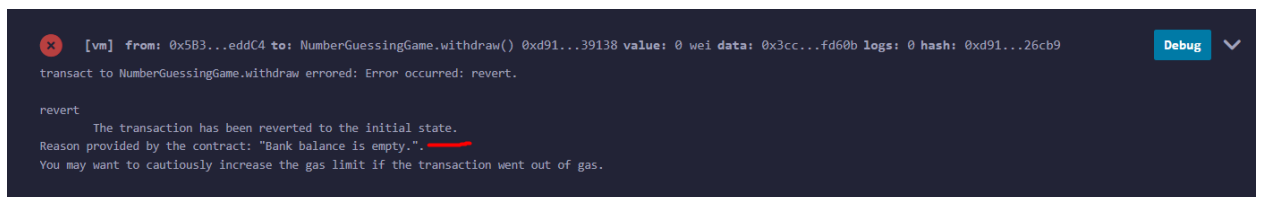
На одержаному скріншоті видно к-сть газу, яка була використана при деплої, хеш транзакції та хеш самого контракту. З'явилась також форма для смарт-контракту в лівій нижній частині. Після виконання деплою к-сть ефіру на 1-му акаунті становить:



Перейдемо до самої форми:






Спробуємо виконати зняття коштів:



Отримали помилку з відповідним повідомленням.

Тепер перейдемо на 2-й акаунт з адресою 0xA**b**8483F64d9C6d1EcF9b849Ae677dD3315835cb2. Встановимо значення суми в 5 gwei та вгадане число як 3:

ACCOUNT 

0xAb8...35cb2 (100 ether)  


GAS LIMIT




☒ Estimated Gas

☐ Custom


3000000




VALUE

5 Gwei 

NUMBERGUESSINGGAME AT 0X   


Balance: 0 ETH

guessNumber 3 

 [vm] from: 0xAb8...35cb2 to: NumberGuessingGame.guessNumber(uint256) 0xd91...39138 value: 5000000000 wei data: 0xb43...00003 logs: 0
hash: 0x2ef...07118  Debug 


transact to NumberGuessingGame.guessNumber errored: Error occurred: revert.



revert

The transaction has been reverted to the initial state.
Reason provided by the contract: "Insufficient funds. Please send at least 1 ETH to play.". 
You may want to cautiously increase the gas limit if the transaction went out of gas.

Отримано помилку з відповідним повідомленням.

Тепер встановимо суму в 1 ЕТН та вгадане число як 11:

ACCOUNT 

0xAb8...35cb2 (99.99999999!)  


GAS LIMIT




☒ Estimated Gas

☐ Custom


3000000

VALUE

1 Ether 

NUMBERGUESSINGGAME AT 0X   

Balance: 0 ETH

guessNumber 11 

```
[vm] from: 0xAb8...35cb2 to: NumberGuessingGame.guessNumber(uint256) 0xd91...39138 value: 1000000000000000000 wei data: 0xb43...0000b logs: 0
hash: 0x4bb...cf06b
transact to NumberGuessingGame.guessNumber errored: Error occurred: revert.

revert
  The transaction has been reverted to the initial state.
Reason provided by the contract: "Invalid guess. Please guess a number between 1 and 10.".
You may want to cautiously increase the gas limit if the transaction went out of gas.
```

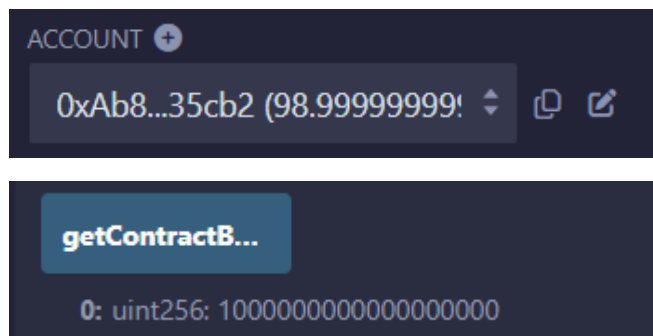
Отримана помилка з відповідним повідомленням.

Тепер вкажимо реальне число та суму, наприклад 3:

```
[vm] from: 0xAb8...35cb2 to: NumberGuessingGame.guessNumber(uint256) 0xd91...39138 value: 1000000000000000000 wei data: 0xb43...00003 logs: 2
hash: 0x714...6b491
status 0x1 Transaction mined and execution succeed
transaction hash 0x71475252b632fb66e707cf3f7ccc06745d5770b9360e9415305748f36ae6b491
block hash 0x208bae55c15a876c92542b89701326fd0866e0d9db37e34768abbf2eda0fa4f0
```

```
logs
[
  {
    "from": "0xd9145CCE52D386f254917e481e844e9943F39138",
    "topic": "0x631e75793be1472f707731dcfd097ddce52e764c4fe77f06bb7b2223c3cf1f47",
    "event": "NewGameStarted",
    "args": {
      "0": "0x5838Da6a701c568545dCfcB03Fc8875f56beddC4",
      "1": "7",
      "owner": "0x5838Da6a701c568545dCfcB03Fc8875f56beddC4",
      "randomNumber": "7"
    }
  },
  {
    "from": "0xd9145CCE52D386f254917e481e844e9943F39138",
    "topic": "0x00ca972d572a1573ce61f99cb22d78e85c775628c176b13591bc5d474cb770ec",
    "event": "NumberGuessed",
    "args": {
      "0": "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2",
      "1": "3",
      "2": "7",
      "3": false,
      "player": "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2",
      "guessedNumber": "3",
      "randomNumber": "7",
      "won": false
    }
  }
]
value 1000000000000000000 wei
```

З одержаних логів можна зрозуміти, що «рандомне число» 7, отже ми програли. При цьому відповідно змінився баланс та банк контракту:



Спробуємо зняти банк з цього акаунту:

```
[vm] from: 0xAb8...35cb2 to: NumberGuessingGame.withdraw() 0xd91...39138 value: 0 wei data: 0x3cc...fd60b logs: 0 hash: 0xcd5...7dee6
transact to NumberGuessingGame.withdraw errored: Error occurred: revert.

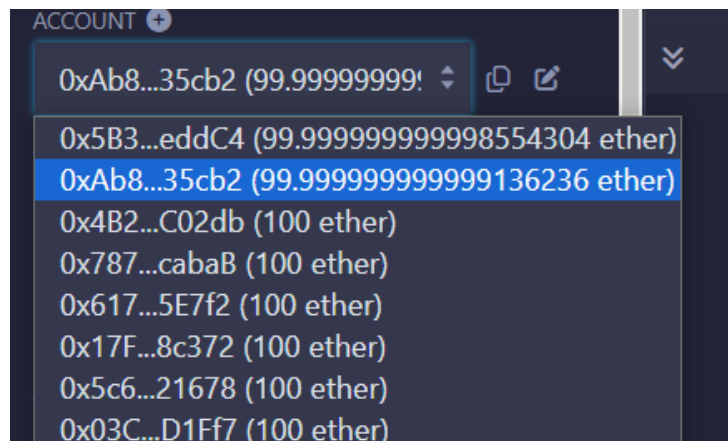
revert
  The transaction has been reverted to the initial state.
Reason provided by the contract: "Only owner can call this function.".
You may want to cautiously increase the gas limit if the transaction went out of gas.
```

Одержали відповідну помилку.

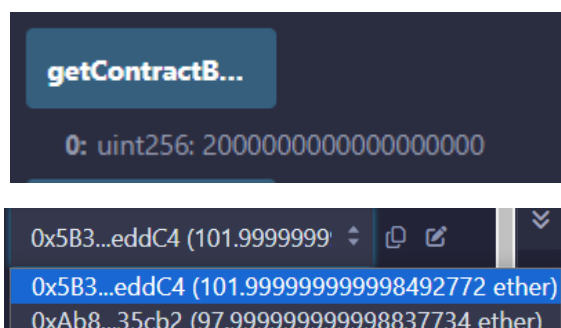
Тепер спробуємо відгадати число:

```
logs
[
  {
    "from": "0xd9145CCE52D386f254917e481eB44e9943F39138",
    "topic": "0x631e75793be1472f707731dcfd097ddce52e764c4fe77f06bb7b2223c3fc1f47",
    "event": "NewGameStarted",
    "args": {
      "0": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
      "1": "3",
      "owner": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
      "randomNumber": "3"
    }
  },
  {
    "from": "0xd9145CCE52D386f254917e481eB44e9943F39138",
    "topic": "0x00ca972d572a1573ce61f99cb22d78e85c775628c176b13591bc5d474cb770ec",
    "event": "NumberGuessed",
    "args": {
      "0": "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2",
      "1": "3",
      "2": "3",
      "3": "3",
      "3": true,
      "player": "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2",
      "guessedNumber": "3",
      "randomNumber": "3",
      "won": true
    }
  }
]
value 1000000000000000000 wei
```

Баланс акаунту:



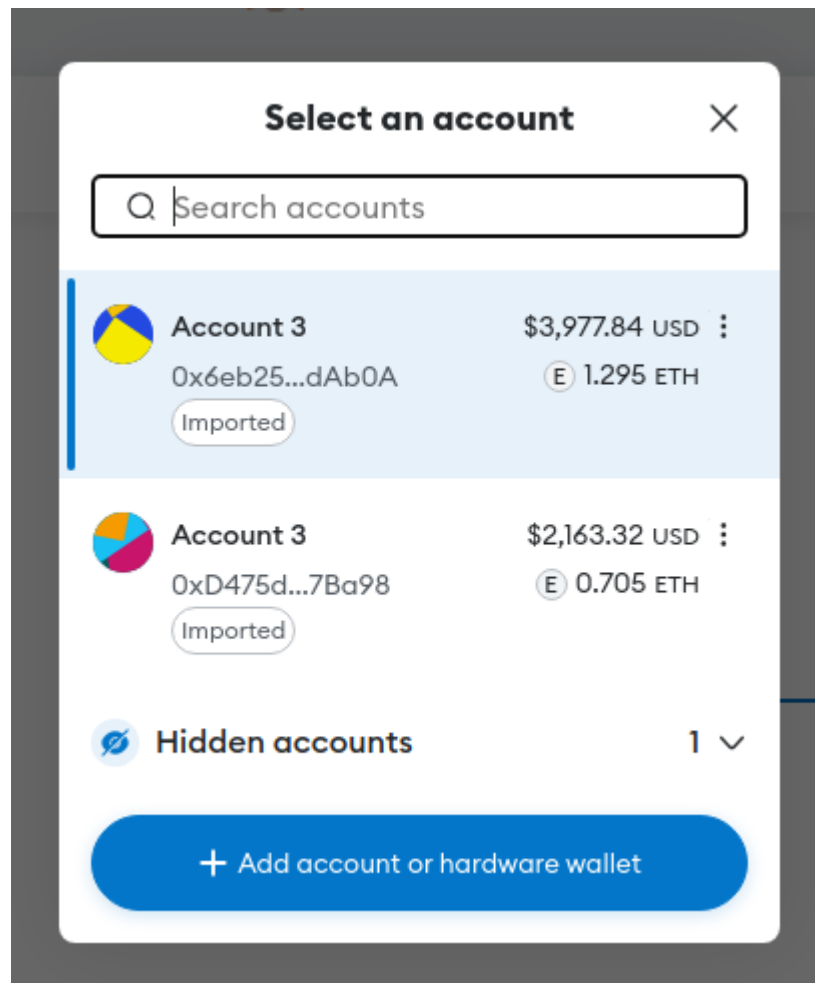
Тепер спробуємо внести щось до банку та зняти баланс з owner акаунту:



Власник успішно вкрав 2 ЕТН.

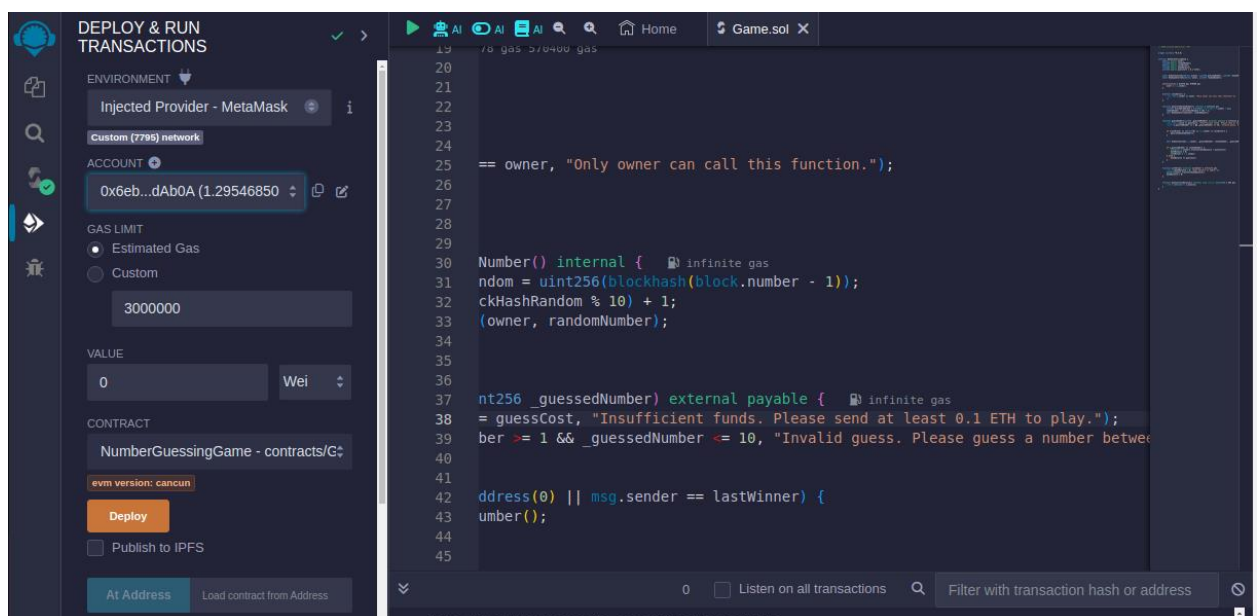
3) Деплой контракту на власну тестову мережу **geth**

Під'єднавши мережу до MetaMask отримали за допомогою імпорту 2 акаунти, які були створені в лабораторній роботі №1.



Сума ставки буде знижена до 0.1 ETH.

Під'єднаємось до нашої тестової мережі з ID 7795:



Виконаємо деплой контракту

geth-net

Account 3

New contract

https://remix.ethereum.org

CONTRACT DEPLOYMENT

\$0.00

DETAILS

HEX

Estimated fee

\$41.50 0.0135 ETH

Market -30 sec

Max fee: 0.0135 ETH

Total

\$41.50 0.0135 ETH

Amount + gas fee

Max amount: 0.0135 ETH

Reject

Confirm

geth-net

Tokens

May 7, 2024

Contract deployment

Confirmed

Contract deployment

Status

Confirmed

Add a block explorer

Copy transaction ID

From

0x6eb25...d...

To

New contract

Transaction

Nonce

2

Amount

-0 ETH

Gas Limit (Units)

9000000

Gas Used (Units)

711626

Base fee (GWEI)

0.000000007

Priority fee (GWEI)

1.5

Total gas fee

0.001067 ETH

\$3.28 USD

Max fee per gas

0.000000002 ETH

\$0.00 USD

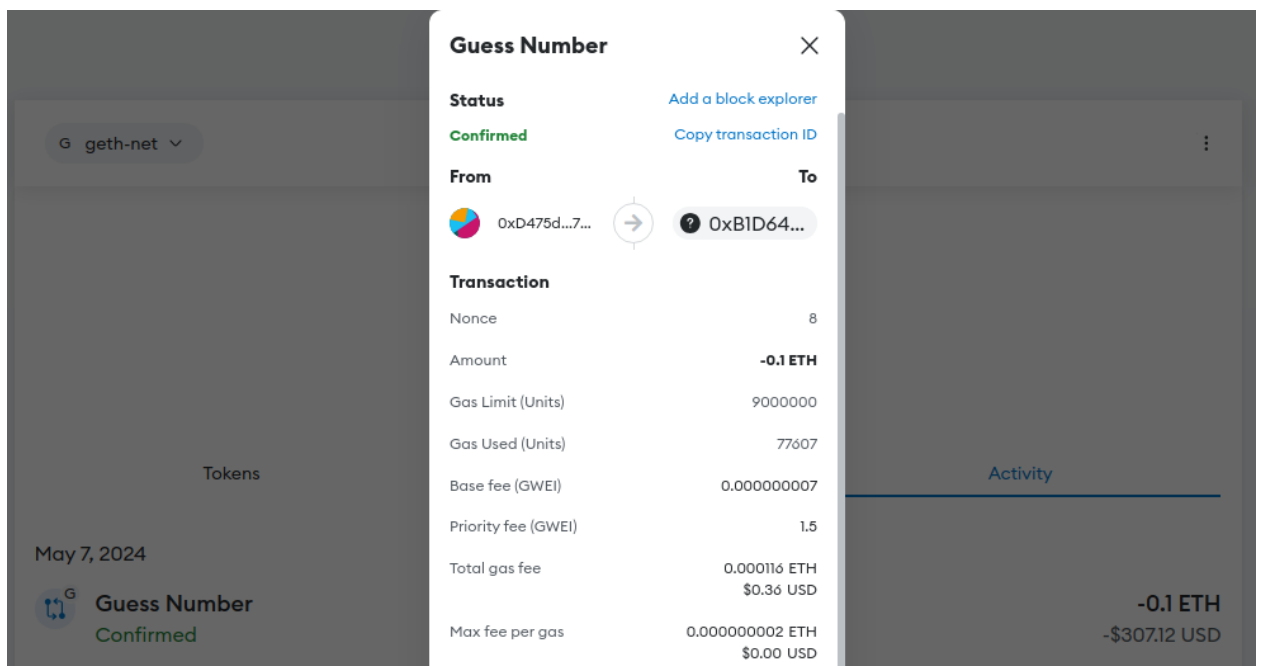
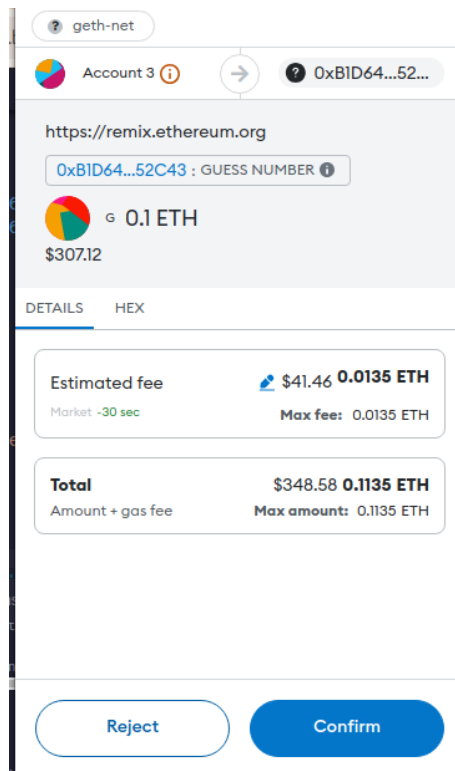
Activity

-0 ETH

-\$0.00 USD

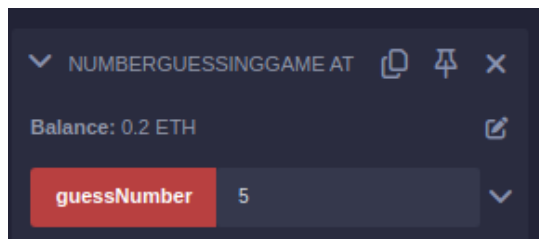
```
INFO [05-07|16:26:28.748] Setting new local account           address=0x6eb25208B5Dfb95654C1a86C70d80c840C8dAb0A
INFO [05-07|16:26:28.754] Submitted contract creation         hash=0xa3b5a95e48d339d90d749d346c138493ac8bf253b909ba75d736a76d103a1c2f from=0x6eb25208B5Dfb95654C1a86C70d80c840C8dAb0A nonce=2 contract=0xB1D64FFA568E190d693263C787a2397497252C43 value=0
INFO [05-07|16:26:29.005] Commit new sealing work             number=3001 s
```

Виконаємо операцію вгадування з іншого акаунту:

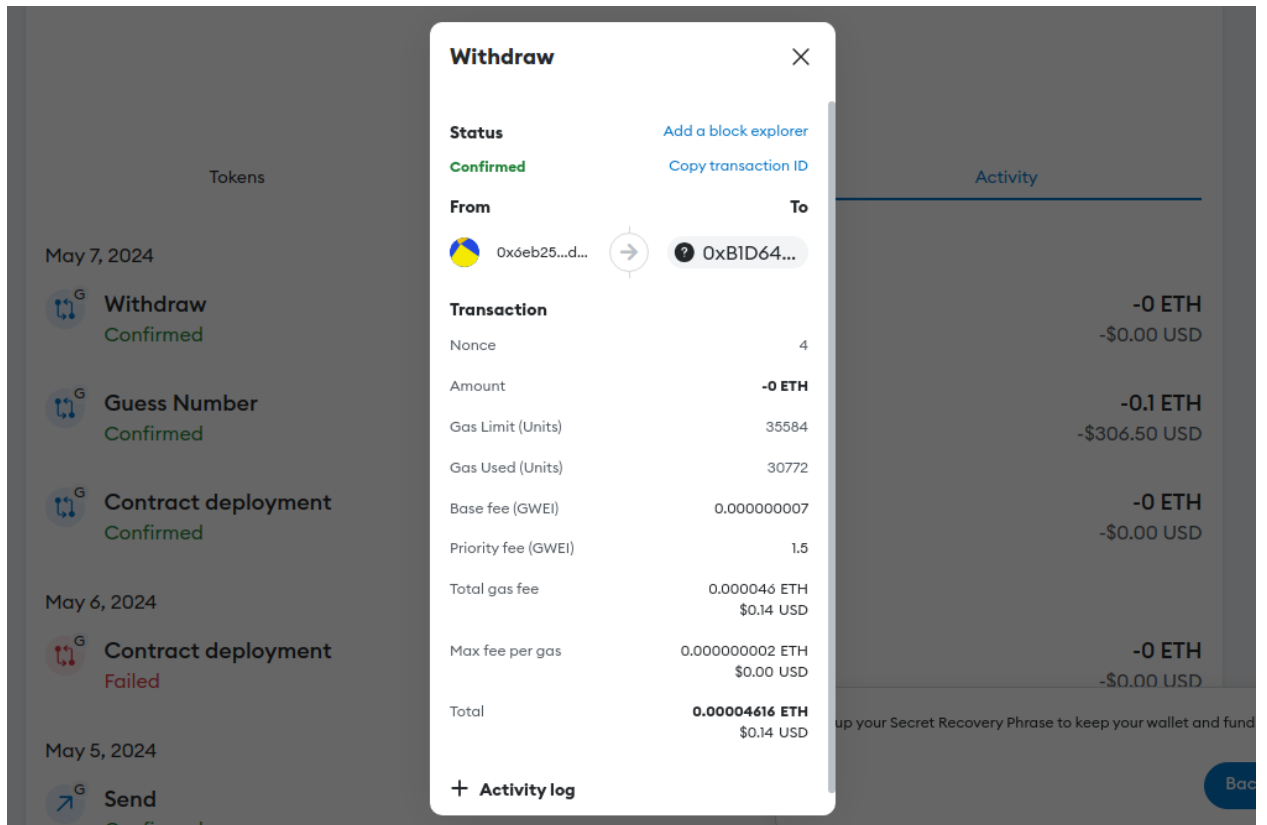


```
INFO [05-07|16:28:22.266] Setting new local account address=0xD475d859660C75107a001D6BB69adC996C27Ba98
INFO [05-07|16:28:22.266] Submitted transaction hash=0x681d0f
be25cd4540bb942f2d164d5fe668837c3e33f9c497d99f1b569e931840 from=0xD475d859660C75
107a001D6BB69adC996C27Ba98 nonce=8 recipient=0xB1D64FFA568E190d693263C787a239749
7252C43 value=100,000,000,000,000,000
```

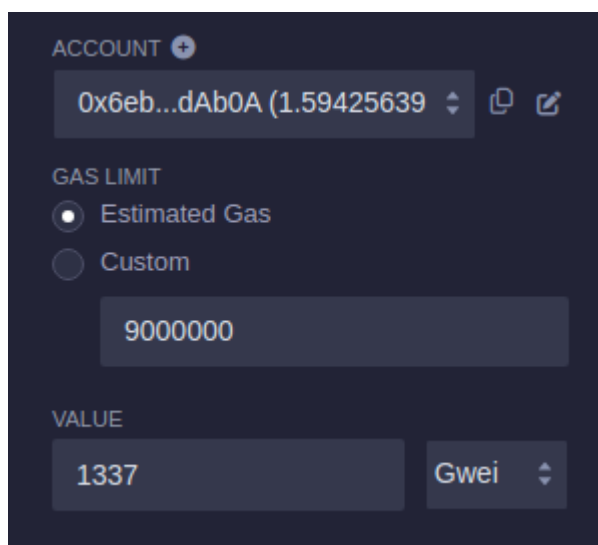
Виконавши ще декілька транзакцій в банкі маємо баланс 0.2 ETH:

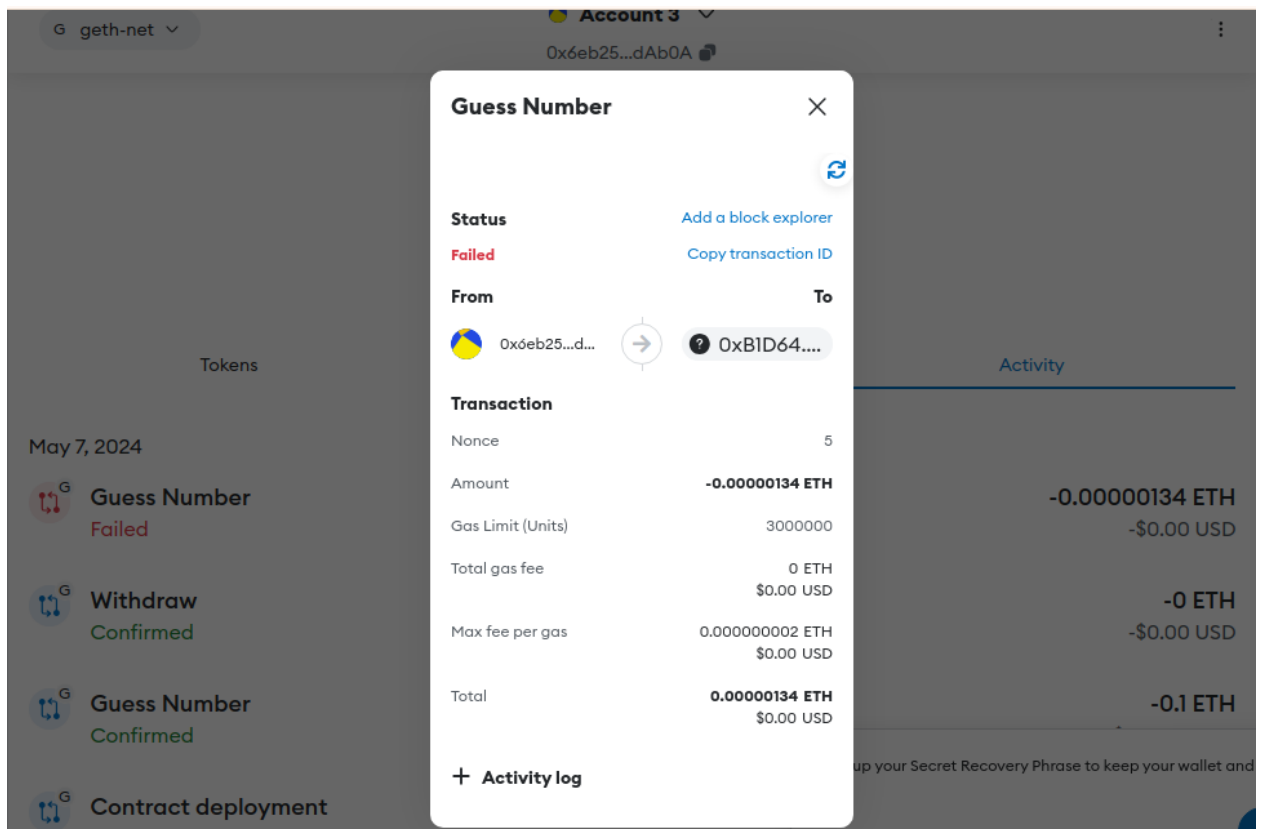
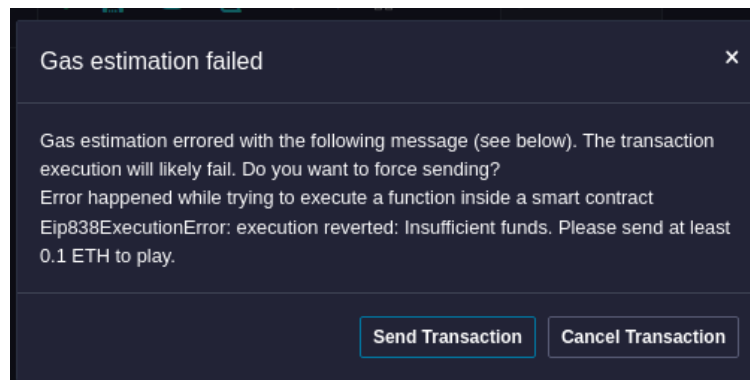


Знімемо їх з першого контракту:



В кінці спробуємо ввести не правильну к-сть ефіру з першого акаунту:





Ще в IDE виникла помилка при виконанні функції, тому очевидно ми отримали статус Failed.

Висновок: загалом для тестування смарт-контрактів зручно використовувати IDE Remix. Тут уже налагоджена своя тестова блокчейн, а очікування між транзакціями зведено до мінімуму. Існують інші тестові мережі, одна з найпопулярніших зараз – Sepolia з ID 11155111. В цій мережі також зручно випробовувати смарт-контракти, проте, як зазначено на сайті, зараз отримати безкоштовні тестові 0.5 «койни» без наявності 0.01 ETH на основному балансі неможливо. Мова програмування Solidity, як і сама технологія блокчейн, активно розвивається, а отже сфера застосувань смарт-контрактів розширюється з кожним днем. Проте важливо утримувати безпеку на належному рівні, а також оптимізувати контракти для оптимального використання газу де це є можливим.