区块链原理与实践

实验三 Solidity与智能合约在线编程

实验内容

1. Solidity基础——搭建僵尸工厂
2. Solidity进阶——僵尸猎食系统
3. Solidity高阶理论
4. 扩展实验一：支付系统
5. 扩展实验二：战斗升级系统

实验步骤

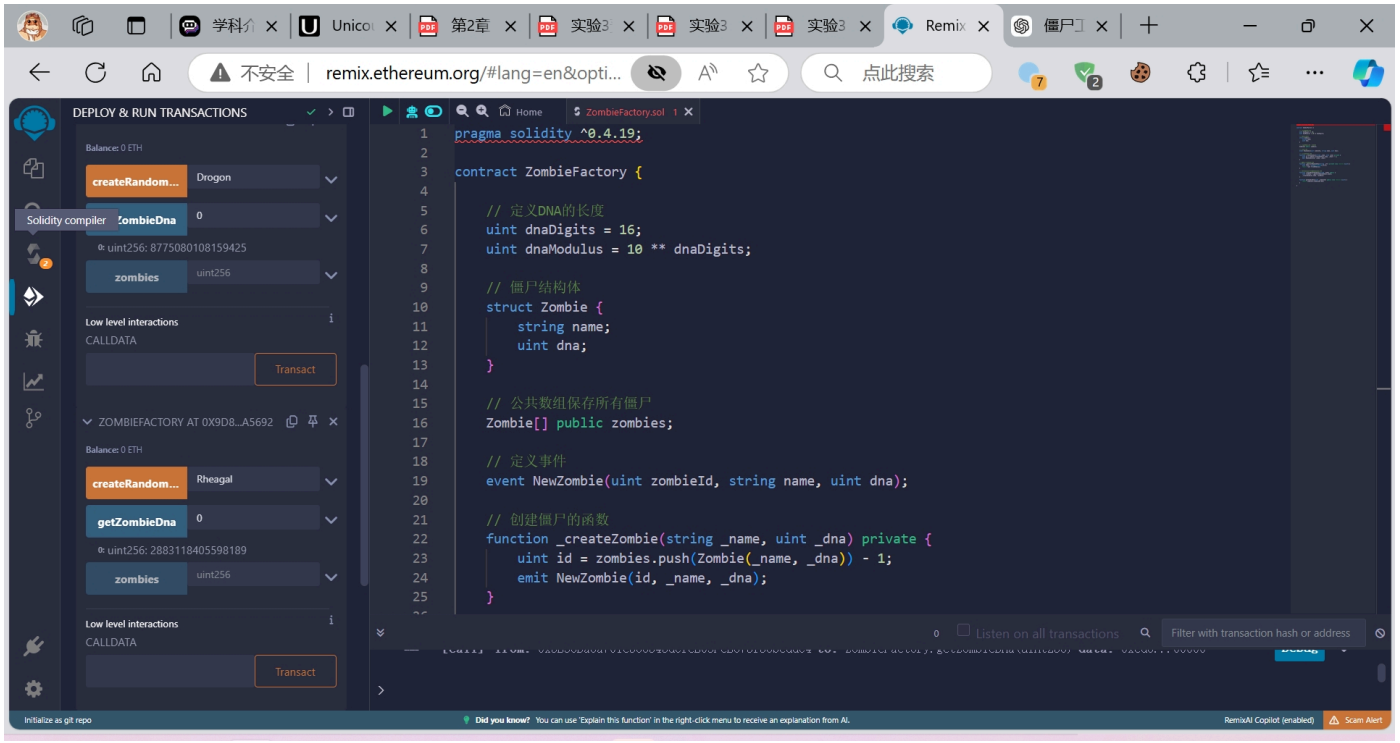
准备阶段

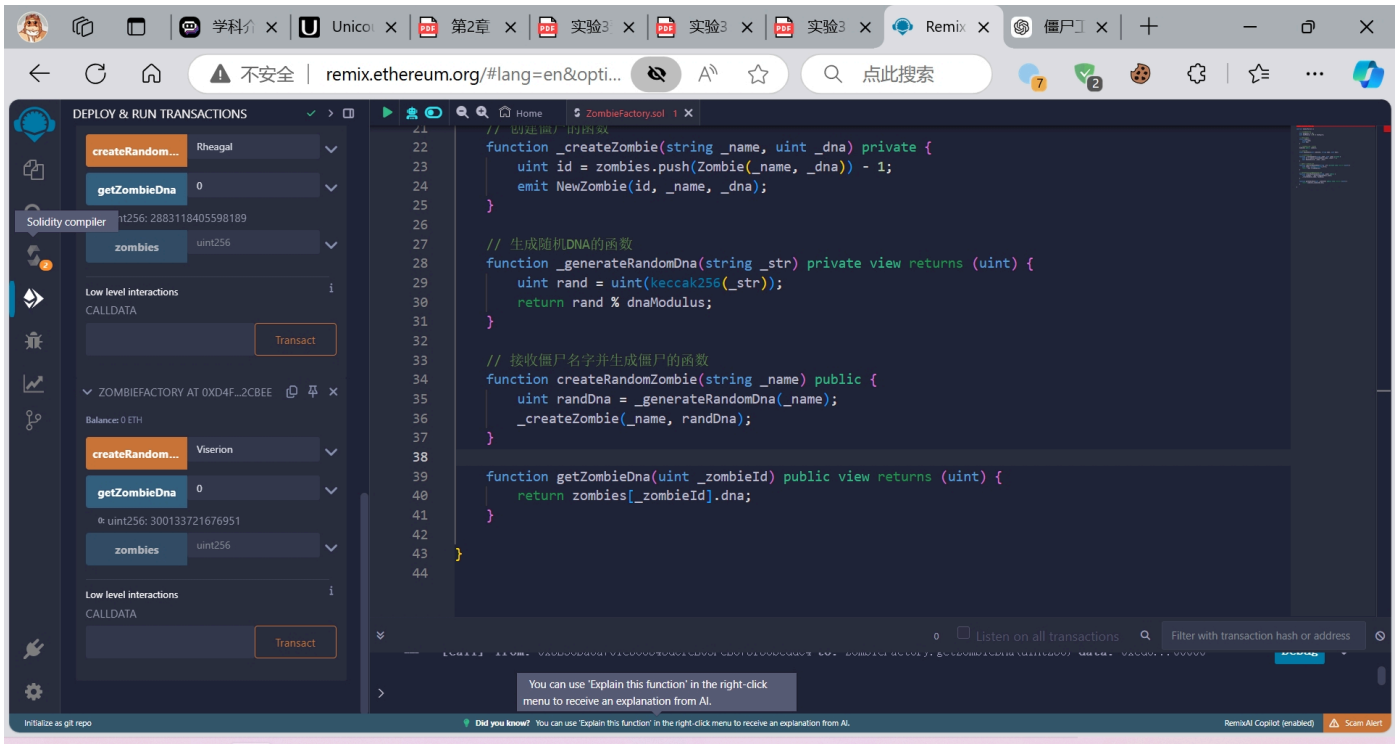
阅读Remix预习文档、Solidity基础知识文档

一 搭建僵尸工厂

按照实验说明的步骤，先后设置僵尸的私有DNA，僵尸结构体，以及公有的僵尸数组，并维护创建僵尸信号与创建僵尸函数之间的联系。为方便查找僵尸DNA，可以额外构建公有函数返回所查询序号位置的僵尸DNA。

为保持报告的简洁性，故将本实验后的`ZombieFactory.sol`与实验二后的版本合并展示。

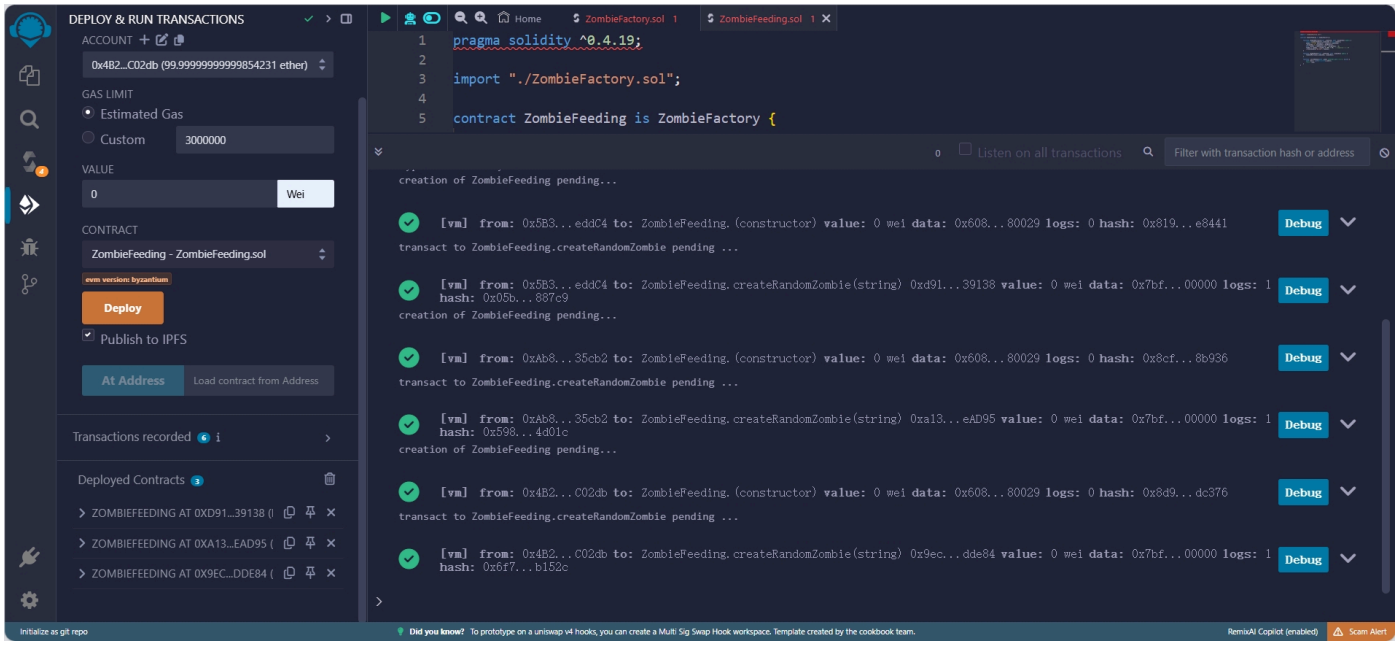




二 僵尸猎食系统

按照实验说明的步骤，在ZombieFactory.sol中创建并维护僵尸DNA与所有者地址间的映射，确保僵尸的所有者关系；使用require保证只有在用户没有僵尸时才能创建。

在ZombieFeeding.sol中，引用继承ZombieFactory.sol，增加_catchAHuman函数实现抓人（实际返回随机数），增加feedOnHuman实现抓人与繁殖，增加feedAndMultiply实现新DNA的计算与新僵尸的生成，实验二结束后两个文件的实验表现及具体代码如下：



ZombieFactory

```
pragma solidity ^0.4.19;

contract ZombieFactory {

    uint dnaDigits = 16;
    uint dnaModulus = 10 ** dnaDigits;

    struct Zombie {
        string name;
        uint dna;
    }

    Zombie[] public zombies;

    mapping (uint => address) public zombieToOwner;
    mapping (address => uint) public ownerZombieCount;

    event NewZombie(uint zombieId, string name, uint dna);

    function _createZombie(string _name, uint _dna) internal {
        uint id = zombies.push(Zombie(_name, _dna)) - 1;
        zombieToOwner[id] = msg.sender;
        ownerZombieCount[msg.sender]++;
        emit NewZombie(id, _name, _dna);
    }

    function _generateRandomDna(string _str) private view returns (uint) {
        uint rand = uint(keccak256(_str));
        return rand % dnaModulus;
    }

    function createRandomZombie(string _name) public {
        require(ownerZombieCount[msg.sender] == 0);
        uint randDna = _generateRandomDna(_name);
        _createZombie(_name, randDna);
```

```
    }

    function getZombieDna(uint _zombieId) public view returns (uint) {
        return zombies[_zombieId].dna;
    }
}
```

ZombieFeeding.sol

```
pragma solidity ^0.4.19;

import "./ZombieFactory.sol";

contract ZombieFeeding is ZombieFactory {

    function feedAndMultiply(uint _zombieId, uint _targetDna) public {
        require(zombieToOwner[_zombieId] == msg.sender);
        Zombie storage myZombie = zombies[_zombieId];
        _targetDna = _targetDna % dnaModulus;
        uint newDna = (myZombie.dna + _targetDna) / 2;
        newDna = newDna - (newDna % 100) + 99;  // 将最后两位设为99
        _createZombie("No-one", newDna);
    }

    function feedOnHuman(uint _zombieId, uint _humanDna) public {
        feedAndMultiply(_zombieId, _humanDna);
    }

    function _catchAHuman(uint _name) internal pure returns (uint) {
        uint rand = uint(keccak256(_name));
        return rand;
    }
}
```

三 Solidity高阶理论

onlyOwner的作用

修饰符`onlyOwner`用于限制某些函数的访问权限，确保只有合约的拥有者可以调用这些函数。`onlyOwner`通常与`Ownable`合约结合使用，以实现权限控制。

具体而言，`Ownable`基础合约通过定义部分有关所有权的函数来控制合约权限，其他合约可以通过继承`Ownable`来获得功能。一个基础的`Ownable`合约如下:

```
pragma solidity ^0.8.0;

contract Ownable {
```

```
    address public owner;

    constructor() {
        owner = msg.sender; // 部署合约的人是拥有者
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Not the contract owner");
        _;
    }

    function transferOwnership(address newOwner) public onlyOwner {
        owner = newOwner; // 转移合约拥有权
    }
}
```

完成实验与结果展示

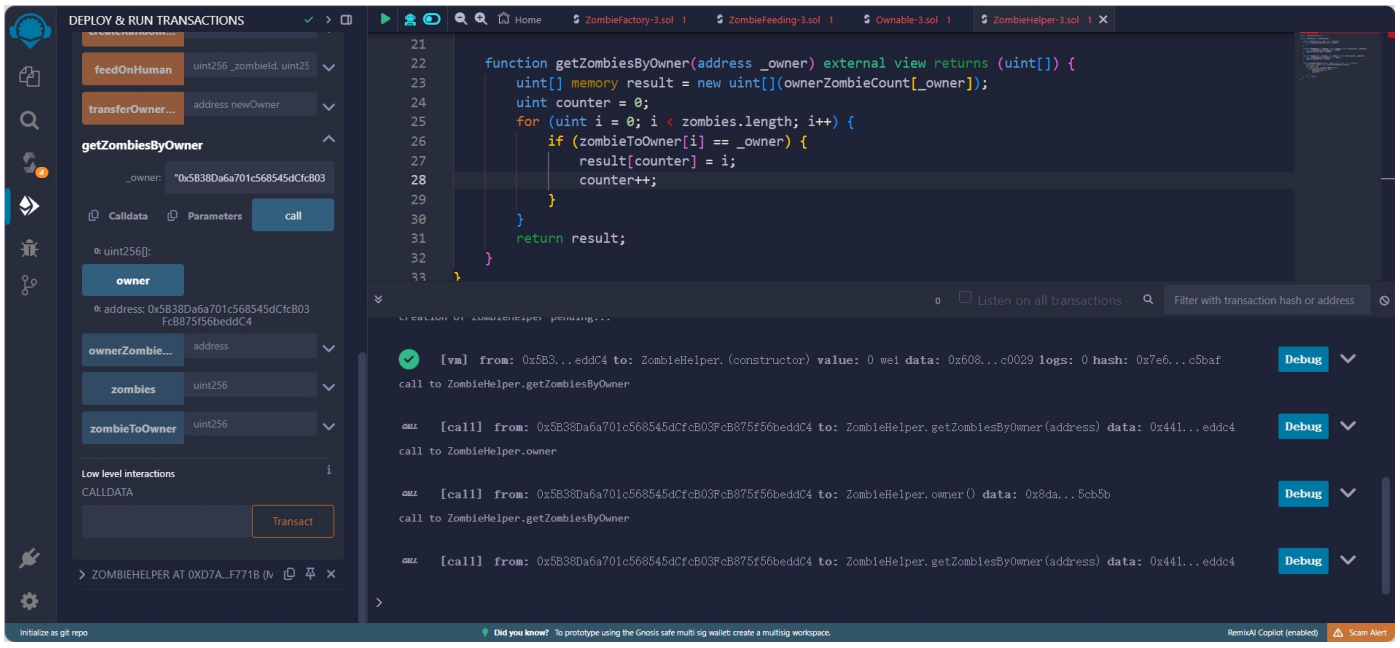
为什么“getZombiesByOwner不花费任何gas”：

getZombiesByOwner函数在不改变合约状态的情况下执行，同时其使用内存数组来存储结果，使得数组的创建和操作不会影响合约的持久化状态。因此调用该函数不会消耗Gas。

展示僵尸的冷却cd：

```
21
22      function _triggerCooldown(Zombie storage _zombie) internal {
23          _zombie.readyTime = uint32(now + cooldownTime);
24      }
25
26      function isReady(Zombie storage _zombie) internal view returns (bool) {
27          return (_zombie.readyTime <= now);
28      }
29  }
30
```

展示 getZombiesByOwner 函数效果



展示在 zombiefactory.sol 中的僵尸结构，说明为什么你的结构能够节省Gas。

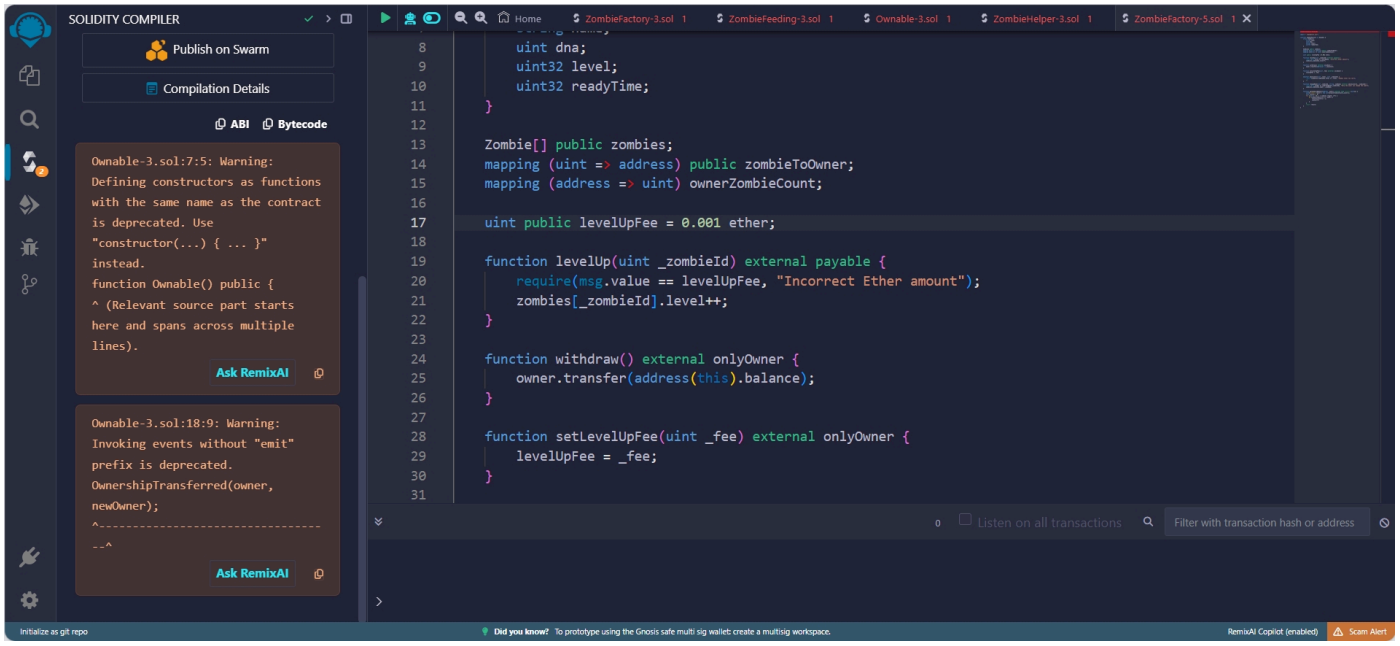
使用存储空间会提高Gas开销、写入存储也是非常昂贵的操作，所以为减少僵尸结构对Gas的使用，我们试图将可以合并的变量放在同一个存储槽内，即：将结构体中的小类型变量（如 uint32 和 uint16）放在一起，保证在一个存储槽中能尽可能多地容纳变量，避免浪费存储槽。同时，我们删除了在实验一中添加的不必要函数`getZombieDNA`以削减可能的额外开销（虽然应该不会有额外开销）。

扩展实验一

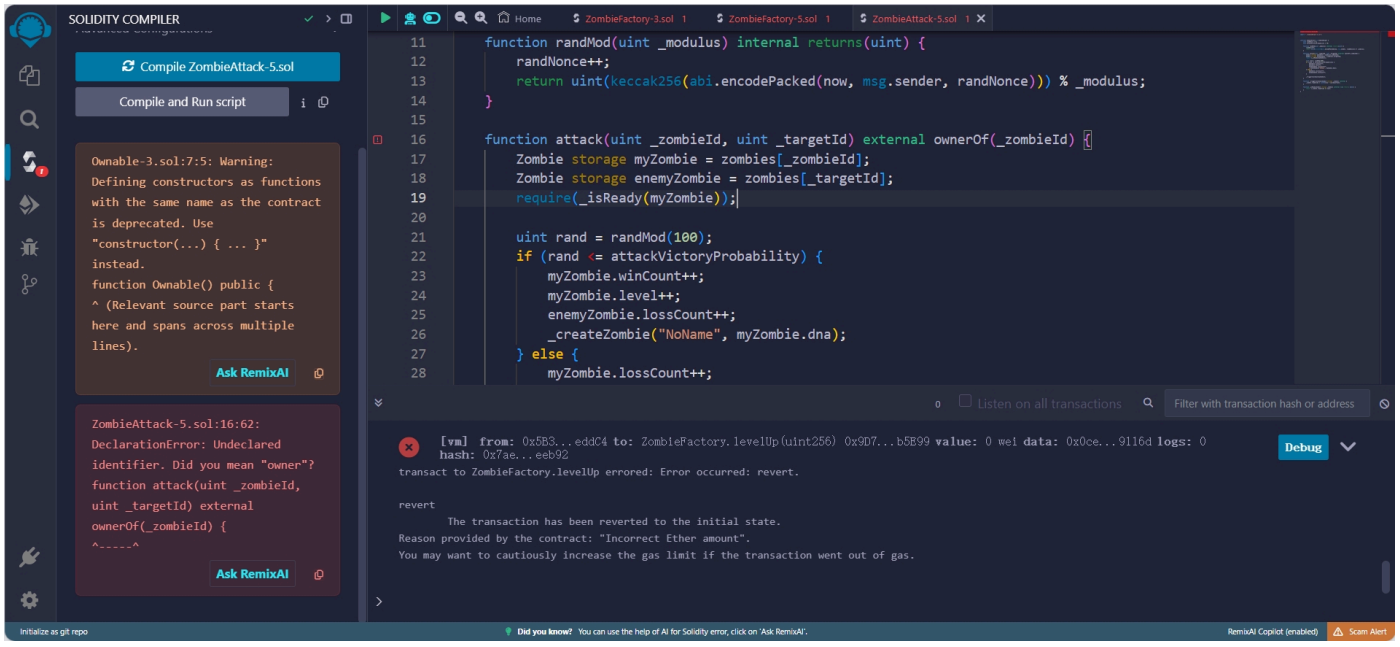
支付系统与提现系统实现如下，但好像由于以太币设置问题会提示报错：

```
transact to ZombieFactory.levelUp errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Incorrect Ether amount".
You may want to cautiously increase the gas limit if the transaction went out of
gas.
```



扩展实验二



根据相关网站的提示与讲解，大致实现了`ZombieAttack.sol`代码，但是在试运行时会产生ownOf报错，多次查阅资料并调试后未果，限于时间只得这样提交了。

实验收获

了解了以太坊智能合约的实现原理，了解了solidity语言语法及相关函数。

参考链接

CryptoZombies

源代码

基础实验（实验三及以前）完成时的代码版本

Ownable-3.sol

ZombieFactory-3.sol

ZombieFeeding-3.sol

ZombieHelper-3.sol

附加实验完成后的代码（未在下方标出则代表没有更新）

ZombieAttack-5.sol

ZombieFactory-5.sol