《区块链技术》实验报告

实验名称	基于众筹合约的 Dapp 系统			指导老师	程宏兵
成员 1 姓名	温家伟	成员 1 学号	202103151422	成员 1 班级	21 级大数据分析 01
成员 2 姓名	刘易非	成员 2 学号	202103150210	成员 2 班级	21 级网络工程 01

一、团队分工介绍

1.刘易非：合约设计师

确定智能合约的整体架构和设计方向，负责项目的设计规划和技术指导。
具体任务：
完成需求分析和功能规格说明，明确智能合约的具体功能和交互逻辑。
设计智能合约的数据结构，包括定义合约中的状态变量、事件和函数接口。
确定智能合约的逻辑流程， 与开发者协商设计细节，解决设计方面的问题并提供支持。

2.温家伟：开发者

负责根据合约设计师的设计方案和文档，编写智能合约的具体代码实现。
具体任务：
根据合约设计师提供的设计文档，使用 Solidity 语言编写智能合约的代码。实现智能合约的各个功能模块，并确保代码的质量和可读性。用 streamlit 库和 web3 库构建前端交互，并且用 Ganache 和 Metamask 把链部署在本地。进行单元测试和集成测试，修复代码中的 bug 并优化性能。主动沟通交流，及时反馈问题并修改代码以满足设计需求。

二、项目概述和架构设计

1.项目概述：

该智能合约是一个众筹合约（CrowdFunding.sol），用于处理众筹活动的资金募集和管理。合约中包括项目的基本信息（例如标题、发起人、结束时间、目标金额、详细信息等），记录众筹的资金流向（当前已募集金额、剩余目标金额、捐赠人数等），以及记录具体资金用途（每个具体用途的内容、所需资金、赞成人数、反对人数等）。

2.架构设计：

本项目实现了众筹合约的基本逻辑，并借助 streamlit 和 web3 库做了和前端的交互，用 Ganache 和 Metamask 把链部署在本地。以下是合约具体内容的设计。

（1）基本信息和状态变量：

合约包含的基本信息有标题（title）、发起人（initiator）、结束时间（endTime）、目标金额（goal）、详细信息（info）等。

合约中维护了当前已筹集金额（current）、剩余目标金额（remainder）、捐赠人数（funderNum）、众筹是否成功的状态（isSuccessful）等状态变量。

通过结构体（struct）Use 来记录具体的资金用途，包括内容、所需资金、赞成人数、反对人数以及投票情况。

（2）构造函数：

在构造函数中初始化项目的基本信息，包括标题、发起人、结束时间、目标金额、详细信息等，并初始化当前已筹集金额和剩余目标金额。

（3）捐款（contribute）函数：

用户可以调用该函数向项目捐款，传入捐款金额和捐款人地址，要求捐款金额大于 0 且小于等于目标金额与已募集金额之差，捐款时间在结束时间之前，如果众筹已成功则无法再进行捐款。

更新当前已筹集金额、记录捐款人的捐款金额，并维护捐款人的信息（包括捐款金额、捐款人列表等）。

并且，在众筹成功的时候，会自动执行空投函数的逻辑，即向所有参与者发放奖励。

（4）退款（returnMoney）函数：

捐款人可以调用该函数申请退款，要求捐款人存在且捐款金额大于 0，当前已筹集金额小于目标金额。

将捐款金额返还给捐款人，更新当前已筹集金额，清除捐款人的信息。

（5）创建使用请求（newUse）函数：

用于创建新的使用案例(Use)。函数需要接收三个参数：_funder（资助方的地址）、_content（案例内容）、_money（资金数额）。

在函数中，首先会进行一系列的 require 断言，包括检查调用者是否为发起人(initiator)、资金是否不超过余额(remainder)、目标是否与当前状态(current)相符。如果满足所有 require 条件，将扣除相应资金，然后将新的使用案例信息（包括内容、资金、赞同数和反对数）添加到 allUses 数组中。

（6）给使用请求投票（voteForUse）函数：

该函数用于为特定使用案例投票。函数接收三个参数：_funder（资助方的地址）、index（案例索引）、agree（是否赞同）。

函数中也有一系列 require 断言，如检查目标是否与当前状态相符、确保投票者未对该案例进行过投票、资助方信息的存在及资金充足等。

如果条件满足，会更新该资助方对该案例的投票状态，并根据是否赞同来更新赞同数或

反对数。若赞同数或反对数达到一定数量（超过目标数的一半），则会触发相应操作，如资金的转移或退还。

三、实现细节

1. 智能合约代码

```
// SPDX-License-Identifier: GPL3.0
pragma solidity ^0.6.0;

// 一个众筹
contract CrowdFunding {
    string public title;                                // 众筹标题
    address payable public initiator;                   // 众筹发起人
    uint public goal;                                   // 众筹目标金额
    string public info;                                 // 众筹详情
    uint public remainder;                              // 众筹剩余可用金额(成功后)
    uint public current;                                // 众筹已筹金额
    uint public funderNum;                              // 参与者数量
    bool public isSuccessful;                           // 众筹是否成功
    address[] public funders;                           // 所有参与者
    mapping(address => uint) public funderMoney;        // 各参与者投的钱
    mapping(address => uint) public funderId;           // 各参与者序号

    // 使用请求
    struct Use {
        string content;     // 请求内容
        uint money;         // 请求金额
        uint agreeNum;      // 同意票数
        uint disagreeNum;   // 不同意票数
        mapping(address => bool) isVote;  // 每个参与者是否投票
    }
    Use[] public allUses;   // 所有使用请求

    constructor(string memory _title, address payable _initiator, uint _goal, string memory _info) public
{
        title = _title;
        initiator = _initiator;
        goal = _goal;
        info = _info;
        current = 0;
        remainder = _goal;
        funderNum = 0;
    }
```

```
    // 往一个众筹里投钱
    function contribute(uint money, address _funder) payable public{
        require(money>0 && money<=goal-current);
        //require(isSuccessful == false);

        current = current + money;
        if(funderId[_funder]!=0){
            funderMoney[_funder] += money;
        }
        else{
            funderMoney[_funder] = money;
            funders.push(_funder);
            funderNum++;
            funderId[_funder] = uint(funders.length);
        }
    }

    // 退钱
    function returnMoney(address _funder) public{
        require(funderId[_funder]!=0 && funderMoney[_funder]>0);
        require(current<goal);

        address payable funderAddr = address(uint160(_funder));
        funderAddr.transfer(funderMoney[_funder]);
        current -= funderMoney[_funder];
        funderId[_funder] = 0;
        funderMoney[_funder] = 0;
    }

    // 创建使用请求
    function newUse(address _funder, string calldata _content, uint _money) public{
        require(_funder == initiator);
        require(_money <= remainder);
        require(goal == current);

        remainder -= _money;
        allUses.push(Use({
            content: _content,
            money: _money,
            agreeNum: 0,
            disagreeNum: 0
        }));
    }
```

```
    // 给使用请求投票
    function voteForUse(address _funder, uint index, bool agree) payable public{
        require(goal == current);
        require(allUses[index].isVote[_funder] != true);
        require(funderId[_funder]!=0 && funderMoney[_funder]>0);
        require(allUses[index].agreeNum < (goal + 1) / 2);
        require(allUses[index].disagreeNum < (goal + 1) / 2);

        //require(isSuccessful == true);

        allUses[index].isVote[_funder] = true;
        uint leastMoney = (goal + 1) / 2;
        if(agree==true){
            allUses[index].agreeNum += funderMoney[_funder];
            if(allUses[index].agreeNum >= leastMoney){
                initiator.transfer(allUses[index].money);
            }
        }
        else{
            allUses[index].disagreeNum += funderMoney[_funder];
            if(allUses[index].disagreeNum >= leastMoney){
                remainder += allUses[index].money;
            }
        }
    }

    // 空投到众筹项目的参与者
    function airdrop(uint totalAirdropAmount) public {
        if (goal==current) {
            // 计算每个参与者的空投份额
            uint perFunderShare = totalAirdropAmount / funderNum;

            // 遍历所有参与者并进行空投
            for (uint j = 0; j < funderNum; j++) {
                address funder = funders[j];
                payable(funder).transfer(perFunderShare);
                remainder -= perFunderShare;
            }
        }
    }

    // 获取使用请求总数
    function getUseNum() public view returns(uint){
```

```
        return allUses.length;
    }

    // 获取使用请求内容
    function getUseContent(uint index) public view returns(string memory){
        return allUses[index].content;
    }

    // 获取使用请求金额
    function getUseMoney(uint index) public view returns(uint){
        return allUses[index].money;
    }

    // 获取使用请求同意数量
    function getUseAgreeNum(uint index) public view returns(uint){
        return allUses[index].agreeNum;
    }

    // 获取使用请求不同意数量
    function getUseDisagreeNum(uint index) public view returns(uint){
        return allUses[index].disagreeNum;
    }

    // 获取参与者是否已对某使用请求投票
    function getUseVote(uint index, address funder) public view returns(bool){
        return allUses[index].isVote[funder];
    }

    // 获取参与者在一个众筹里投的钱
    function getFunderMoney(address funder) public view returns(uint){
        return funderMoney[funder];
    }

    // 获取众筹地址
    function getAddress() public view returns(address payable){
        return address(this);
    }

    // 合约余额查询
    function getContractBalance() public view returns (uint) {
        return address(this).balance;
    }

    fallback() payable external {}
```

```
    receive () payable external {}

}

// 所有众筹
contract AllFundings {
    address payable[] fundings;
    // 创建众筹项目
    function newFunding(string calldata title, uint goal, string calldata info) public{
        CrowdFunding funding = new CrowdFunding(title, msg.sender, goal, info);
        fundings.push(funding.getAddress());
    }

    // 给一个众筹项目投钱
    function contribute(uint i) payable public {
        address payable fundingAddr = fundings[i];
        CrowdFunding(fundingAddr).contribute(uint(msg.value), msg.sender);
        fundingAddr.transfer(msg.value);
        // 众筹成功自动执行空投
        // 这里不判断众筹成功，而是放在 airdrop 里判断
        // if(CrowdFunding(fundingAddr).isSuccessful() == true) {
            CrowdFunding(fundingAddr).airdrop(getGoal(i) / 10);
        // }
    }

    // 退钱
    function returnMoney(uint i) payable public {
        address payable fundingAddr = fundings[i];
        CrowdFunding(fundingAddr).returnMoney(msg.sender);
    }

    // 创建使用请求
    function newUse(uint i, string calldata content, uint money) public {
        address payable fundingAddr = fundings[i];
        CrowdFunding(fundingAddr).newUse(msg.sender, content, money);
    }

    // 给使用请求投票
    function voteForUse(uint i, uint useId, bool isAgree) public {
        address payable fundingAddr = fundings[i];
        CrowdFunding(fundingAddr).voteForUse(msg.sender, useId, isAgree);
    }

    // 获取众筹总数
```

```
    function getTotalNum() public view returns(uint){
        return fundings.length;
    }

    // 获取某众筹地址
    function getAddress(uint i) public view returns(address payable){
        return fundings[i];
    }

    // 获取某众筹标题
    function getTitle(uint i) public view returns(string memory){
        return CrowdFunding(fundings[i]).title();
    }

    // 获取某众筹发起人
    function getInitiator(uint i) public view returns(address payable){
        return CrowdFunding(fundings[i]).initiator();
    }

    // 获取某众筹目标金额
    function getGoal(uint i) public view returns(uint){
        return CrowdFunding(fundings[i]).goal();
    }

    // 获取某众筹当前已筹金额
    function getCurrent(uint i) public view returns(uint){
        return CrowdFunding(fundings[i]).current();
    }

    // 获取某众筹剩余可用金额
    function getRemainder(uint i) public view returns(uint){
        // return CrowdFunding(fundings[i]).remainder();
        return  CrowdFunding(fundings[i]).getContractBalance();
    }

    // 获取某众筹详细信息
    function getInfo(uint i) public view returns(string memory){
        return CrowdFunding(fundings[i]).info();
    }

    // 获取某参与者在某众筹中投的钱
    function getFunderMoney(uint i, address funder) public view returns(uint){
        return CrowdFunding(fundings[i]).getFunderMoney(funder);
    }
```

```
    // 获取某众筹的使用请求总数
    function getUseNum(uint i) public view returns(uint){
        return CrowdFunding(fundings[i]).getUseNum();
    }

    // 获取某众筹的某使用请求内容
    function getUseContent(uint i, uint j) public view returns(string memory){
        return CrowdFunding(fundings[i]).getUseContent(j);
    }

    // 获取某众筹的某使用请求金额
    function getUseMoney(uint i, uint j) public view returns(uint){
        return CrowdFunding(fundings[i]).getUseMoney(j);
    }

    // 获取某众筹的某使用请求同意数量
    function getUseAgreeNum(uint i, uint j) public view returns(uint){
        return CrowdFunding(fundings[i]).getUseAgreeNum(j);
    }

    // 获取某众筹的某使用请求不同意数量
    function getUseDisagreeNum(uint i, uint j) public view returns(uint){
        return CrowdFunding(fundings[i]).getUseDisagreeNum(j);
    }

    // 获取某众筹的某参与者是否已经针对某使用请求投票
    function getUseVote(uint i, uint j, address funder) public view returns(bool){
        return CrowdFunding(fundings[i]).getUseVote(j, funder);
    }
}
```

2. 前端搭建过程及代码

（1）搭建过程：

首先，conda 创建一个新的虚拟环境，名为 Crowding，并且下载依赖包：

conda create --name Crowding

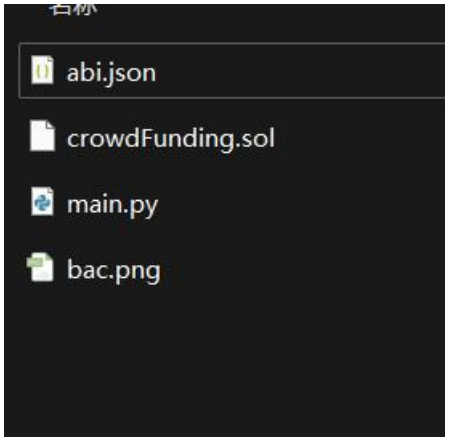
conda activate Crowding

pip install streamlit==1.2.0

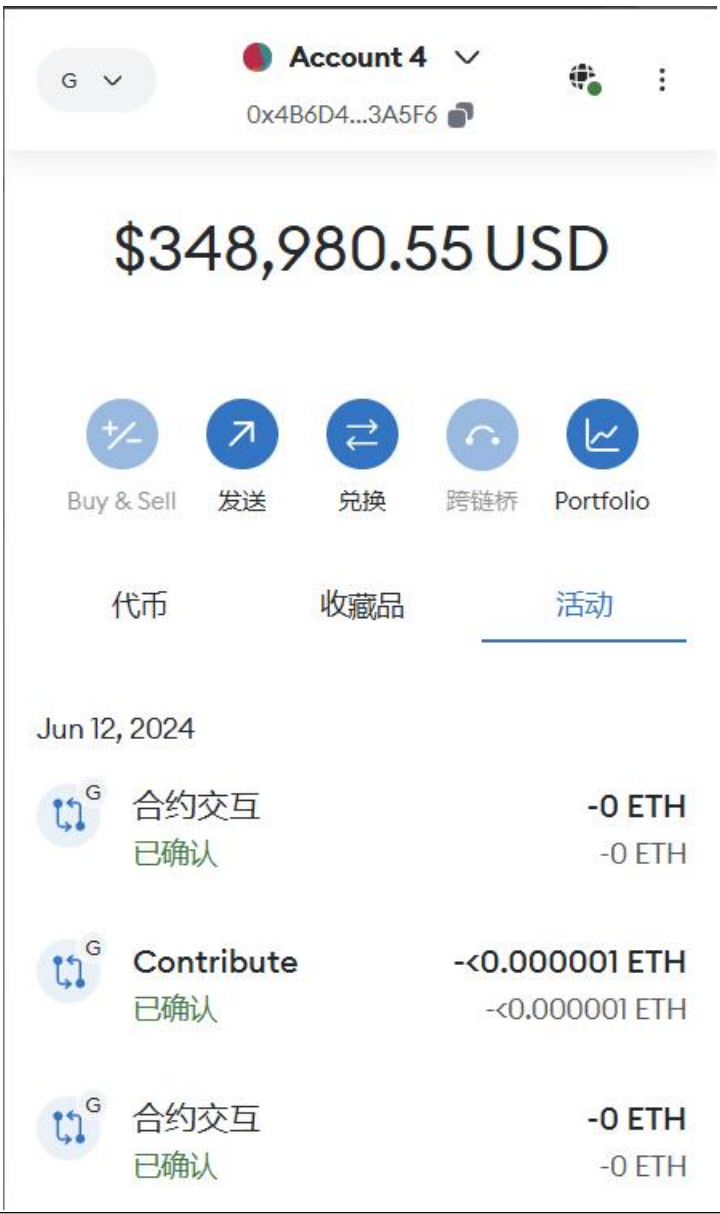
pip install web3==4.0.0

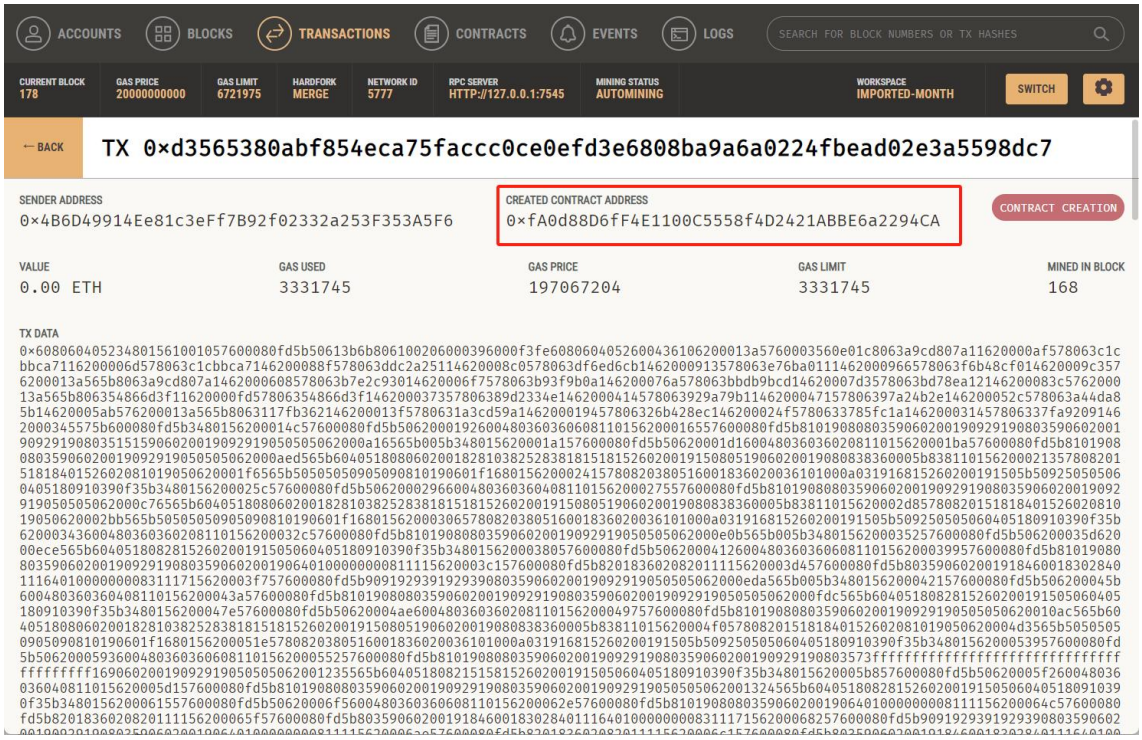
pip install protobuf==3.19.0

接着，把部署好的合约的 abi 保存到本地文件中，命名为 abi.json



然后，根据 ganache 的钱包账户复制到 python 文件中，当然，私钥和 Metamask 的配置也要提前做好。





最后，在命令行运行 python 程序：

streamlit run main.py

```
(CrowdFunding) C:\Users\20848\Desktop\区块链小组作业>streamlit run main.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://10.81.149.192:8501
```

（2）前端代码：

```
import streamlit as st
from web3 import Web3
import json
import pandas as pd
import base64


# 初始化 Web3 连接
w3 = Web3(Web3.HTTPProvider("HTTP://127.0.0.1:7545"))  # 例如 Infura 或 Alchemy 的 URL


# 加载智能合约 ABI
with open("abi.json") as f:
    crowd_funding_abi = json.load(f)


# 设置智能合约地址
crowd_funding_address = "0xfA0d88D6fF4E1100C5558f4D2421ABBE6a2294CA"
```

```
# 实例化智能合约
crowd_funding_contract = w3.eth.contract(address=crowd_funding_address, abi=crowd_funding_abi)


def main_bg(main_bg):
    main_bg_ext = "png"
    st.markdown(
        f"""
         <style>
         .stApp {{
             background: url({base64.b64encode(open(main_bg,
"rb").read()).decode()});
             background-size: cover
         }}
         </style>
         """,
        unsafe_allow_html=True
    )


def main():
    menu = ["全部众筹", "创建众筹", "使用申请", "使用申请投票"]
    main_bg('bac.png')
    choice = st.sidebar.selectbox("菜单", menu)

    # 根据选择显示不同的页面
    if choice == "全部众筹":
        all_crowdfundings()
    elif choice == "创建众筹":
        my_creations()
    elif choice == "使用申请":
        use4apply()
    elif choice == "使用申请投票":
        vote()

# 全部众筹开始
def fetch_crowdfunding_data():
    total_fundings = crowd_funding_contract.functions.getTotalNum().call()
    crowdfunding_data = []

    for i in range(total_fundings):
        title = crowd_funding_contract.functions.getTitle(i).call()
        initiator = crowd_funding_contract.functions.getInitiator(i).call()
        goal = crowd_funding_contract.functions.getGoal(i).call()
        current = crowd_funding_contract.functions.getCurrent(i).call()
        is_success = "成功" if goal == current else "进行中"
```

```
        crowdfunding_data.append({
            '名称': title,
            '发起人': initiator[:10] + '...' if len(initiator) > 10 else initiator,
            '目标金额': goal,
            '当前已筹集金额': current,
            '状态': is_success,
        })

    return pd.DataFrame(crowdfunding_data)

def display_crowdfunding_table():
    st.title("全部众筹项目")
    df = fetch_crowdfunding_data()

    # 使用 Pandas 样式设置文本居中
    styled_df = df.style.set_properties(**{
        'text-align': 'center'
    }).set_table_styles(
        [{'selector': 'th', 'props': [('text-align', 'center')]}]
    )

    # 渲染表格
    st.write(styled_df.to_html(), unsafe_allow_html=True)

def all_crowdfundings():
    display_crowdfunding_table()
# 全部众筹结束

# 创建众筹开始

def create_new_funding(title, goal, info):
    # 不知道咋用那个环境变量，我就直接传私钥了
    w3.eth.enable_unaudited_features()
    # 发送交易
    account =
w3.eth.account.privateKeyToAccount("0x38e382617e446f99e415bd8efcf5895304a04a52d1840656b58be24e03d88979")
    tx_hash = crowd_funding_contract.functions.newFunding(title, goal, info).transact({'from':
account.address})
    print("众筹项目已创建")

def my_creations():
    # 创建新的众筹项目表单
    st.title("创建新的众筹项目")
    title = st.text_input("项目标题")
```

```
    goal = st.number_input("目标金额", min_value=0)
    info = st.text_area("项目信息")
    if st.button("创建"):
        create_new_funding(title, goal, info)
# 创建众筹结束

def use4apply():
    st.title("众筹资金使用申请")
    # 展示用户参与投资的众筹项目
    Num = st.number_input("项目编号", key="project_number", min_value=0)
    context = st.text_area("使用请求信息")
    Money = st.number_input("金额", key="request_amount", min_value=0)


    if st.button("创建"):
        # 不知道咋用那个环境变量，我就直接传私钥了
        w3.eth.enable_unaudited_features()
        # 发送交易
        account =
w3.eth.account.privateKeyToAccount("0x38e382617e446f99e415bd8efcf5895304a04a52d1840656b58be24e03d88979")
        tx_hash = crowd_funding_contract.functions.newUse(Num, context, Money).transact({'from':
account.address})
        print("使用请求已创建")



def vote():
    st.title("众筹资金使用申请投票")
    total_fundings = crowd_funding_contract.functions.getTotalNum().call()
    crowdfunding_data = []

    for i in range(total_fundings):
        num = crowd_funding_contract.functions.getUseNum(i).call()
        for j in range(num):
            title = crowd_funding_contract.functions.getTitle(i).call()
            initiator = crowd_funding_contract.functions.getInitiator(i).call()
            goal = crowd_funding_contract.functions.getGoal(i).call()
            current = crowd_funding_contract.functions.getCurrent(i).call()
            useContext = crowd_funding_contract.functions.getUseContent(i, j).call()
            useMoney = crowd_funding_contract.functions.getUseMoney(i, j).call()
            piao = crowd_funding_contract.functions.getUseAgreeNum(i, j).call()



            print(useContext)
            is_success = "成功" if goal == current else "进行中"
```

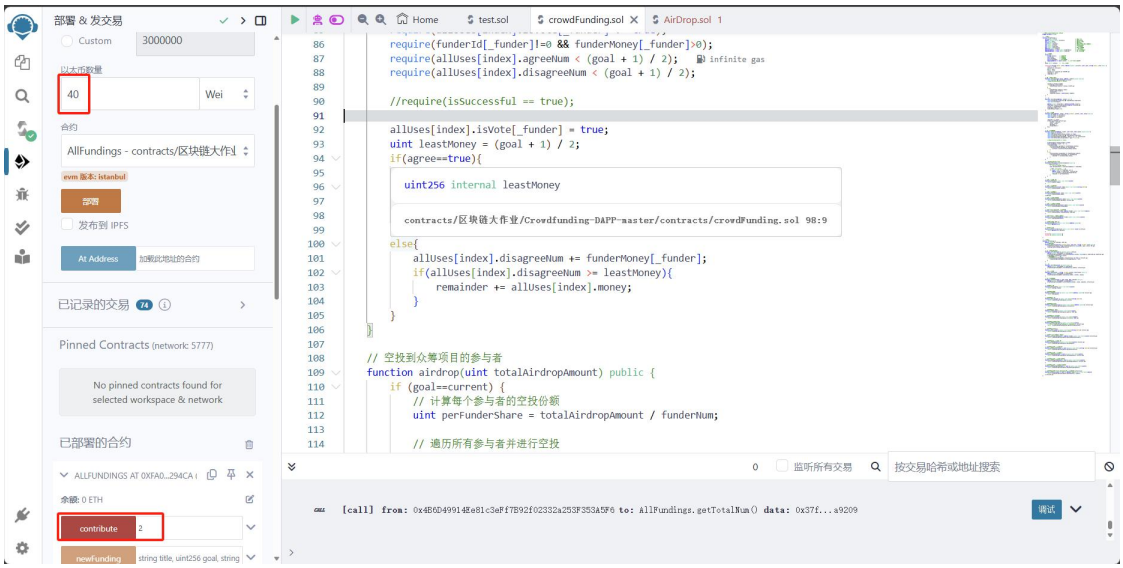
```
            crowdfunding_data.append({
                '名称': title,
                '发起人': initiator[:10] + '...' if len(initiator) > 10 else initiator,
                '目标金额': goal,
                '当前已筹集金额': current,
                '状态': is_success,
                '使用请求内容': useContext,
                '使用请求金额': useMoney,
                '使用请求票数': piao,
            })

    df = pd.DataFrame(crowdfunding_data)

    styled_df = df.style.set_properties(**{
        'text-align': 'center'
    }).set_table_styles(
        [{'selector': 'th', 'props': [('text-align', 'center')]}]
    )

    # 渲染表格
    st.write(styled_df.to_html(), unsafe_allow_html=True)


if __name__ == "__main__":
    main()
```

四、结果与讨论

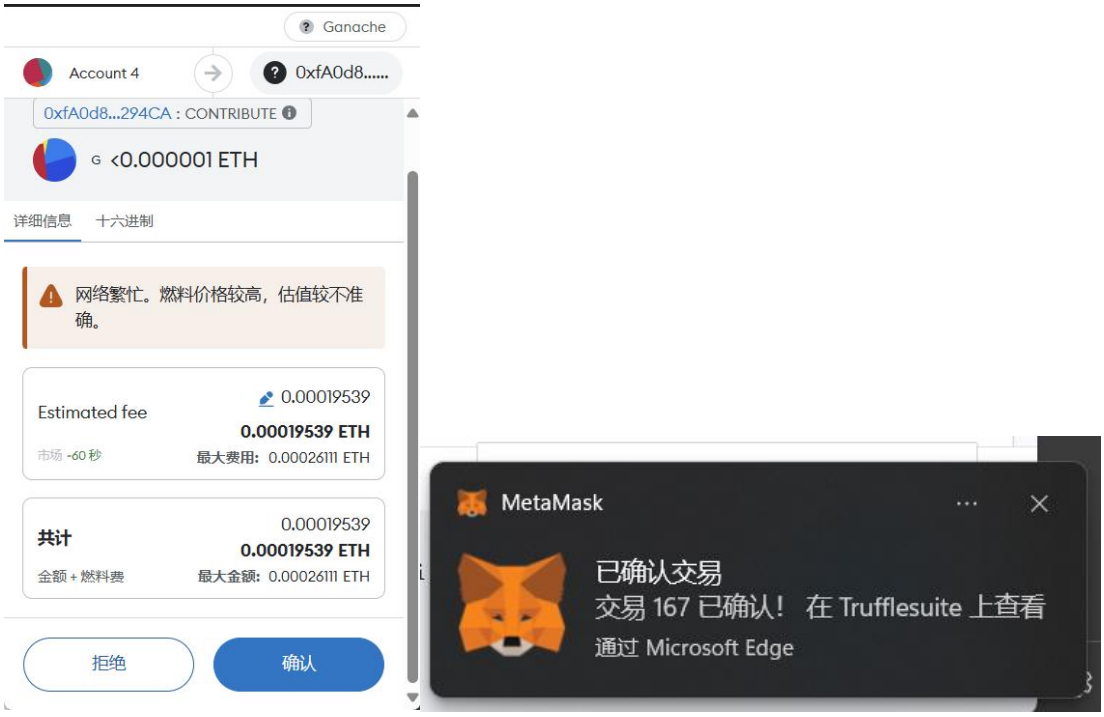
打开我们的 Dapp 后，首先创建新的众筹任务，如下图所示，在分别填好，项目标题，目标金额，项目信息后，点击创建按钮，即可完成新的众筹任务的创建过程。

与上面创建过程相同，我们继续创建了 4 个众筹项目，分别是“捐助灾区”、“救助病危老人”、“捐助残疾人”、“捐助留守儿童”。然后我们点击“全部众筹”按钮，如下图所示，此时可以看到所有的众筹信息，其中也包括了我们刚才发布的那四个众筹任务。

现在这些众筹项目的状态都是进行中，我们向“救助病危老人”项目中捐一些钱，如下图所示：



点击“contribute”按钮后，MetaMask 会跳出一个界面，点击确定后，捐助成功。



这时这个众筹项目已经成功了，如下图所示。

我们在合约中写过一个空投函数，即众筹成功时会发放奖励：

我们可以看到现在合约中仅剩下 36 了。
现在我们继续测试使用请求的功能，如下图所示，在填好项目编号、使用请求信息和金额后，继续点击创建按钮。

这时我们点击使用申请投票按钮，发现有使用申请。

然后我们进行投票，这里的众筹项目是一个人完成的，所以一个人投票后即可成功：

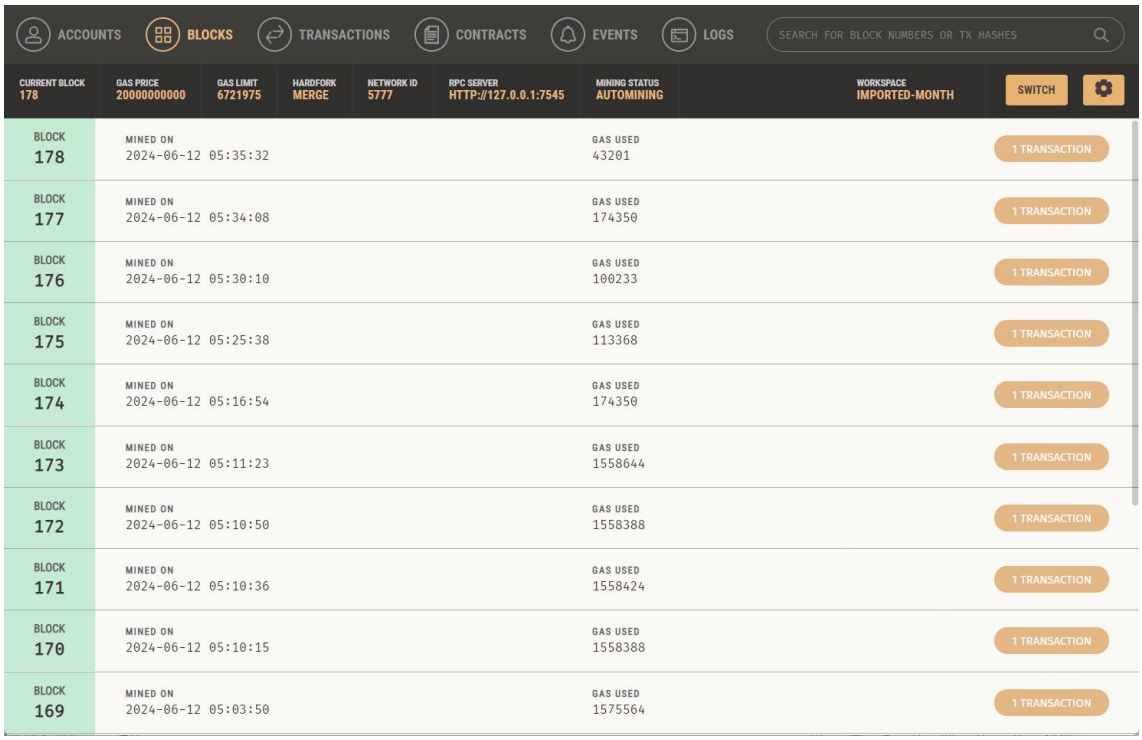
现在查看使用申请投票界面，票数由 0 变成了 40（有权重），投票成功！

最后测试退钱功能，我们先捐一点钱到第二个众筹项目中：

然后点击“returnMoney”按钮：

再次查看全部众筹页面时，如图所示，第二个众筹项目的当前已筹集金额从 10 变回了 0。

然后我们查看 Ganache，发现所有行为均在链上有保存：



实验结果评分：