CMPE483 HW1 Documentation

This is the documentation of the first homework of CMPE483.

Content:

1. Description

This project implements an autonomous decentralized governance token contract called MyGov using *OpenZeppelin ERC20* token contract. This token allows crowds to collectively manage a decentralized autonomous organization (DAO).

This governance token contract will provide the following functionality:

1. All standard ERC20 functions are provided by MyGov with some additional functions.

2. MyGov provides these transactional activities: donation to MyGov, survey service, project proposal, voting, and funding.

3. Anyone who owns at least 1 MyGov token is a MyGov member. Anyone can donate to MyGov; however, only members can carry out the other two activities provided by the contract.

4. Anyone can call the get information functions. View functions are open to everyone, i.e. you do not need to be a member to get information.

5. Submitting a Project Proposal costs 5 MyGov tokens and 0.1 Ether.

6. Submitting a Survey costs 2 MyGov tokens and 0.04 Ether.

7. MyGov token supply is 10 million (fixed).

8. Donations can be accepted in ethers and MyGov tokens only. Ethers can be granted to winning Project proposals. MyGov tokens are given away via faucet function.

9. MyGov tokens are distributed via a faucet function. The faucet gives 1 token to an address. If the address obtained a token before, it cannot get a token from the faucet any longer.

10. Each member can give 1 vote to each proposal. Members can also delegate their votes. Members who voted or delegated vote cannot reduce their MyGov balance to zero until the voting deadlines.

11. In order to get a Project proposal funded, at least 1/10 of the members must vote yes AND there should be sufficient ether amount in the MyGov contract. Project proposers must call the reserveProjectGrant function in order to reserve the funding by the proposal deadline. If the project proposer does not reserve by the deadline, funding is lost. Also, if there is not sufficient ether in MyGov contract when trying to reserve, the fund is lost. In order to receive payments according to the schedule, at least 1/100 of the members should vote yes. If at least 1/100 do not vote yes, then the current as well as the remaining payments are terminated, i.e. the project is no longer funded.

2. Function Definitions

- ✓ **constructor(uint tokensupply)**

  This is the constructor of our contract. It initializes the token supply to 10 million.

- ✓ **function delegateVoteTo(address memberaddr, uint projectid) public**

  When it's called, this function delegates vote to the specified member for the project proposal or the project payment.

- ✓ **function donateEther( ) public payable**

  This function donates ether to the smart contract. The amount equals to the "msg.value".

- ✓ **function donateMyGovToken(uint amount) public**

  This function donates MyGov token to the smart contract. The amount is specified in the function input.

- ✓ **function voteForProjectProposal(uint projectid, bool choice) public**

  Clients can vote for a project proposal using this function. The project that they voted for and their choices are taken as parameters.

- ✓ **function voteForProjectPayment(uint projectid, bool choice) public**

  This function can be used to vote for the approval of a payment. Keep in mind that after a project proposal passed, it has to pass another voting to approve its payments.

  Clients can vote for a project payment decision using this function. Which project they voted to and their choices are taken as parameters.

- ✓ **function submitProjectProposal(string ipfshash, uint votedeadline, uint [ ] paymentamounts, uint [ ] payschedule) public payable returns (uint projectid)**

  Using this function, clients can submit their project proposals. This function takes four arguments: the IPFS hash of the project file, the specified deadline of the voting, the planned payment amounts, and the planned payment schedule, in order. Then, it returns the unique projectid of the proposal. This id can later be used to access the required information about the project.

  However, this function has a cost. You need to pay 0.1 ethers and 5 MyGov tokens to use this.

  For more information about the IPFS system, visit the following page.
  https://developers.cloudflare.com/web3/ipfs-gateway/concepts/ipfs/

- ✓ **function submitSurvey(string ipfshash, uint surveydeadline, uint numchoices, uint atmostchoice) public payable returns (uint surveyid)**

  Using this function, clients can submit surveys in order to gain information about voters' decisions. It takes four arguments: the IPFS hash of the project file, the specified deadline of the voting, the planned payment amounts, and the planned payment schedule, in order. Then, it returns a unique survey id, which can later be used to access information about the survey.

  The logic behind a survey is that it allows you to learn about the approval chance of a project with a low cost. Submitting proposal costs 0.1 ethers and 5 MyGov tokens, whereas submitting a survey costs 0.04 Ether and 2 MyGov tokens.

- ✓ **function takeSurvey(uint surveyid, uint [ ] choices) public**

  This function can be used to participate in the survey. It takes the surveyid and the array of choices as parameters.

- ✓ **function reserveProjectGrant(uint projectid) public**

  This function can be used to reserve the total amount of the Project Grant, so that this amount cannot be used by any other project. It takes the projectid as parameter.

- ✓ **function faucet( )**

  This function provides faucet functionality. When called, it grants 1 MyGov Token to the caller if they have not used this faucet before.

- ✓ **function withdrawProjectPayment(uint projectid) public**

  This function provides the transfer of the required money to get the project done. It takes the projectid as parameter.

****** The rest of the functions are getter functions. They do not change the state of the blockchain. They are rather used to obtain information about the state. Each of them uses*** view ***keyword.***

- ✓ **function getSurveyResults(uint surveyid) public view returns(uint numtaken, uint [ ] results)**

  After the survey has finished, this function can be called to see the result of the survey. The function takes the unique surveyid as parameter and returns the total number of participants and survey results.

- ✓ **function getSurveyInfo(uint surveyid) public view returns(string ipfshash, uint surveydeadline, uint numchoices, uint atmostchoice)**

  This function can be used to get the general information about the survey. It takes the unique surveyid as parameter and returns the IPFS hash of the submitted file, the survey deadline, the number of choices, and the maximum number of choices of a user.

- ✓ **function getSurveyOwner(uint surveyid) public view returns(address surveyowner)**

  This function can be used to learn the owner of the survey. It takes the unique surveyid as parameter and returns the address of the survey owner.

- ✓ **function getIsProjectFunded(uint projectid) public view returns(bool funded)**

  This function can be used to learn if the project has been funded at the moment. It returns a boolean value; it returns true if the project has been funded, else false. It is public, so everyone can see if it has been funded or not.

- ✓ **function getProjectNextPayment(uint projectid) public view returns(int next)**

  This function can be used to learn the due date of the next payment. It returns the time of the next date. Only the projectid is needed to call this function.

- ✓ **function getProjectOwner(uint projectid) public view returns(address projectowner)**

  This function can be used to get the project owner of a project. It takes projectid as parameter and returns the address of the project owner.

- ✓ **function getProjectInfo(uint activityid) public view returns(string ipfshash, uint votedeadline, uint [ ] paymentamounts, uint [ ] payschedule)**

  This function can be used to get the general information about a project. It takes the unique projectid as parameter and returns the IPFS hash of the submitted file, the project deadline, the payment array, and the payment schedule.

- ✓ **function getNoOfProjectProposals() public view returns(uint numproposals)**

  This function can be used to get the number of the funded projects in the smart contract. It returns the mentioned number.

- ✓ **function getNoOfFundedProjects () public view returns(uint numfunded)**

  This function can be used to get the number of the funded projects in the smart contract. It returns the mentioned number.

- ✓ **function getEtherReceivedByProject (uint projectid) public view returns(uint amount)**

  This function can be used to get the amount of Ether received by a specific project. The function takes the projectid as input and it returns the received amount of Ether.

- ✓ **function getNoOfSurveys( ) public view returns(uint numsurveys)**

  This function can be used to get the number of surveys done in the smart contract. It returns the number of surveys.

3. Test Cases / Average Gas Usages

All testing is conducted using Brownie python interface and Ganache local blockchain.

<u>Unit Tests</u>

Unit Tests are done with 10 accounts provided by the Ganache CLI.
Test scripts are provided in the zip file.

faucet()
Two tests have been used for the faucet function.
• test_faucet_gives_coin calls the faucet() function from 10 accounts and checks if their balance is increased by 1.
• test_faucet_works_only_once calls the faucet() function thrice with the same accounts and checks if the second and third times result in a revert. Then checks the balance of the caller account.

constructor()
constructor() function is tested in the sense of total supply.
• test_total_supply deploys the contract and checks if the MyGov balance of the contract is 10^7.

donateEther()
donateEther() function is tested simply by donating ether and checking the balance of the contract
• test_donate_ether calls the donateEther() function with msg.value of 10 wei, then checks if the contract value has increased.

donateMyGovToken()
donateMyGovToken() function is tested in a similar fashion to the above.
• test_donate_mygov_token transfers 1000 MyGov to an account and then that account calls the donateMyGovToken() function to donate 500 MyGov to the contract. Then we compare the MyGov balance of the contract before and after the donation.

submitProjectProposal(), getProjectOwner(), getProjectInfo(), getNoOfProjectProposals()
These four functions are tested together.
• test_can_submit_proposal_and_get_info transfers 1000 MyGov to an account and then submits a proposal via the submitProjectProposal() function. We check if the getProjectInfo(), getProjectOwner() and getNoOfProjectProposals() returns the relevant information.

delegateVoteTo()
• In the test_delegate_vote, a proposal gets submitted and two members delegates their vote to two other members. Then we check the voteWeight (delegatedVotes + 1) and isVoted values.

reserveProjectGrant()
• In the test_reserve_project_grant, a proposal is submitted and voted true more than 10% of the members. Then reserveProjectGrant() function is called. We compare the total funding of the project and the reservedAmount variable. In this case the funding required is less than the proposal value, so no ether donation was required.

voteForProjectProposal()
Voting function for proposals is checked by submitting a proposal, voting by some members and then checking the votecount.
• In the test_vote_for_project_proposal, we transfer some MyGov to accounts[0] and call the faucet function from all 10 accounts. Accounts[0] submits a project proposal and the proposal is voted true four times by accounts[0], … , accounts[4] and voted false two times by accounts[6] and accounts[7]. Then we compare the value of voteCount with the number of true voters.

voteForProjectPayment()
This function is tested implicitly in the tests of below functions:
withdrawProjectPayment(), getIsProjectFunded(), getNoOfFundedProjects(), getEtherReceivedByProject(), getProjectNextPayment()

Withdraw function is the most complex of the unit tests.
It is almost an integration test because of the number steps required for the withdrawal action.
• In test_withdraw_project_payment 1000 MyGov is transferred to accounts[0]. We donate 100 ETH to the contract from accounts[9] and call the faucet from all 10 accounts. Then a proposal with two payments is submitted via the submitProjectProposal() function. We set the initial_balance variable to the ETH balance of the project owner. Also, we set the next_payment_before_1st_period variable to the getProjectNextPayment().
• After 4 "true" votes (more than 10%), reserveProjectGrant(projected) is called and voteForProjectPayment for the first payment is called. Then with built-in chain.mine(timestamp=*) function is called to time-travel and mine a block with timestamp 1 second after the first payment date. withdrawProjectPayment(0) is called and we set the intermediary_balance variable to the ETH balance of project owner. Also, we set the next_payment_after _1st_period variable to the getProjectNextPayment().
• voteForProjectPayment is called by more than 1% of the members and again using the built-in chain.mine(timestamp=*) function we time travel and mine a block with timestamp 1 second after the second payment date. withdrawProjectPayment(1) is called and we set the final_balance variable to the ETH balance of the project owner.
• Finally, we compare the initial_balance, intermediary_balance and final_balance values.
• We check if the get functions listed above return the relevant information.

Since there are not many survey functions, we decided to test all of them at once as an integration test.

Integration Tests

We tested surveys and proposals in two separate tests because they are independent.

a) Projects

For doing an integration test on projects part of the contract, we decided to use the following scenario to combine the functions:
- Contract is deployed, all accounts use the faucet, we transfer 1000 MyGov to the 0^{th} account for submitting cost and donate 50 ETH to the contract from 10^{th} account.
- A project is submitted by account 0.
- Project deadline is 1 day after the submission and the payment amounts are 1 ETH each, which are scheduled to be paid 1 week and 2 weeks after the proposal voting deadline.
- Half of the accounts delegates their votes to the first other half of the accounts.
- While first half of the non-delegating accounts vote yes, other half votes no. Hence project proposal is accepted.
- Enough accounts vote yes for the project payments.
- Funding is withdrawn.
- Finally, we check whether the balance of the proposal owner and the values returned by the get functions associated with projects.

We created respectively 100, 200, 300, and 400 accounts and repeated the test scenario above, all passed the tests with no unexpected results.

Average gas usages:

MyGov <Contract>
├─ constructor - avg: 3606123 avg (confirmed): 3606123 low: 3606123 high: 3606123
├─ submitProjectProposal - avg: 297238 avg (confirmed): 297238 low: 297238 high: 297238
├─ faucet - avg: 106688 avg (confirmed): 106688 low: 106581 high: 136581
├─ reserveProjectGrant - avg: 105317 avg (confirmed): 105317 low: 105317 high: 105317
├─ voteForProjectProposal - avg: 89869 avg (confirmed): 89869 low: 76306 high: 118184
├─ delegateVoteTo - avg: 86564 avg (confirmed): 86564 low: 76986 high: 96173
├─ voteForProjectPayment - avg: 71762 avg (confirmed): 71762 low: 71762 high: 86762
├─ withdrawProjectPayment - avg: 68486 avg (confirmed): 68486 low: 60986 high: 75986
├─ transfer - avg: 38809 avg (confirmed): 38809 low: 38809 high: 38809
└─ donateEther - avg: 21249 avg (confirmed): 21249 low: 21249 high: 21249

b) Surveys

For testing the survey functions, we decided the use the following scenario:

- Contract is deployed, all accounts use the faucet, we transfer 1000 MyGov to the 0^{th} account for submitting cost.
- A survey with 6 choices where a member can choose at most 3 options is submitted.
- Half of the accounts choose options [1,2,3] and quarter of the accounts choose option [5]
- We mine a block with a timestamp 1 seconds bigger than the deadline.
- Finally, we check whether the info, results, owner and total number of surveys are correct.

We created respectively 100, 200, 300, and 400 accounts and repeated the test scenario above, all passed the tests with no unexpected results.

Average gas usages:

MyGov <Contract>
├─ constructor - avg: 3599659 avg (confirmed): 3599659 low: 3599659 high: 3599659
├─ submitSurvey - avg: 212709 avg (confirmed): 212709 low: 212709 high: 212709
├─ faucet - avg: 106857 avg (confirmed): 106857 low: 106581 high: 136581
├─ takeSurvey - avg: 90106 avg (confirmed): 90106 low: 77133 high: 155123
└─ transfer - avg: 38809 avg (confirmed): 38809 low: 38809 high: 38809

Task Achievement Table	Yes	Partially	No
I have prepared documentation with at least 6 pages.	✓		
I have provided average gas usages for the interface functions.	✓		
I have provided comments in my code.	✓		
I have developed test scripts, performed tests and submitted test scripts as well documented test results.	✓		
I have developed smart contract Solidity code and submitted it.	✓		
Function delegateVoteTo is implemented and works.	✓		
Function donateEther is implemented and works.	✓		
Function donateMyGovToken is implemented and works.	✓		
Function voteForProjectProposal is implemented and works.	✓		
Function voteForProjectPayment is implemented and works.	✓		
Function submitProjectProposal is implemented and works.	✓		
Function submitSurvey is implemented and works.	✓		
Function submitSurvey is implemented and works.	✓		
Function takeSurvey is implemented and works.	✓		
Function reserveProjectGrant is implemented and works.	✓		
Function withdrawProjectPayment is implemented and works.	✓		
Function getSurveyResults is implemented and works.	✓		
Function getSurveyInfo is implemented and works.	✓		
Function getSurveyOwner is implemented and works.	✓		
Function getIsProjectFunded is implemented and works.	✓		
Function getProjectNextPayment is implemented and works.	✓		
Function getProjectOwner is implemented and works.	✓		
Function getProjectInfo is implemented and works.	✓		
Function getNoOfProjectProposals is implemented and works.	✓		
Function getNoOfFundedProjects is implemented and works.	✓		
Function getEtherReceivedByProject is implemented and works.	✓		
Function getNoOfSurveys is implemented and works.	✓		
I have tested my smart contract with 100 addresses and documented the results of these tests.	✓		
I have tested my smart contract with 200 addresses and documented the results of these tests.	✓		
I have tested my smart contract with 300 addresses and documented the results of these tests.	✓		
I have tested my smart contract with more than 300 addresses and documented the results of these tests.	✓		