Contest «DGO SMV Smart Contract System»

SMV Smart Contract Description

Authors: RSquad Blockchain Lab

Table of Contents

1. About

This document represents RSquad's submission for the DGO SMV Smart Contract System contest.

The code of the developed system of smart contracts SMV can be found at the following link: https://github.com/RSquad/smv

The document contains:

- an overview of the statuses of competition requirements in relation to the solution developed by RSquad;
- description of the main smart contracts of the SMV smart contract system;
- description of third-party smart contracts used by the SMV smart contract system;
- description of smart contracts used for testing;
- DeBots description;
- description of basic user scenarios;
- description of the test environment and infrastructure;
- description of test cases.

The SMV smart contract system was developed based on the requirements provided in the contest description and in accordance with the architectural specifications developed as part of the Developers Contest: Soft Majority Voting system.

Limitations:

- in accordance with the terms of the contest, the SMV smart contract system must be deployed and tested on the DevNet network. However, the specified network has been inoperative since at least January 4, 2021, which makes it impossible to comply with this competitive requirement;
- due to the point above, Node SE was used for development and testing;
- the developed solution will be deployed by RSquad in the DevNet network as soon as possible after its return to a working state.

2. Glossary

- smc — smart-contract
- System — DGO SMV smart-contract system

3. Overview

The purpose of the System is to automate the decentralized governance for Free TON communities through voting.

The following figure shows the top-level diagram that describes the System:

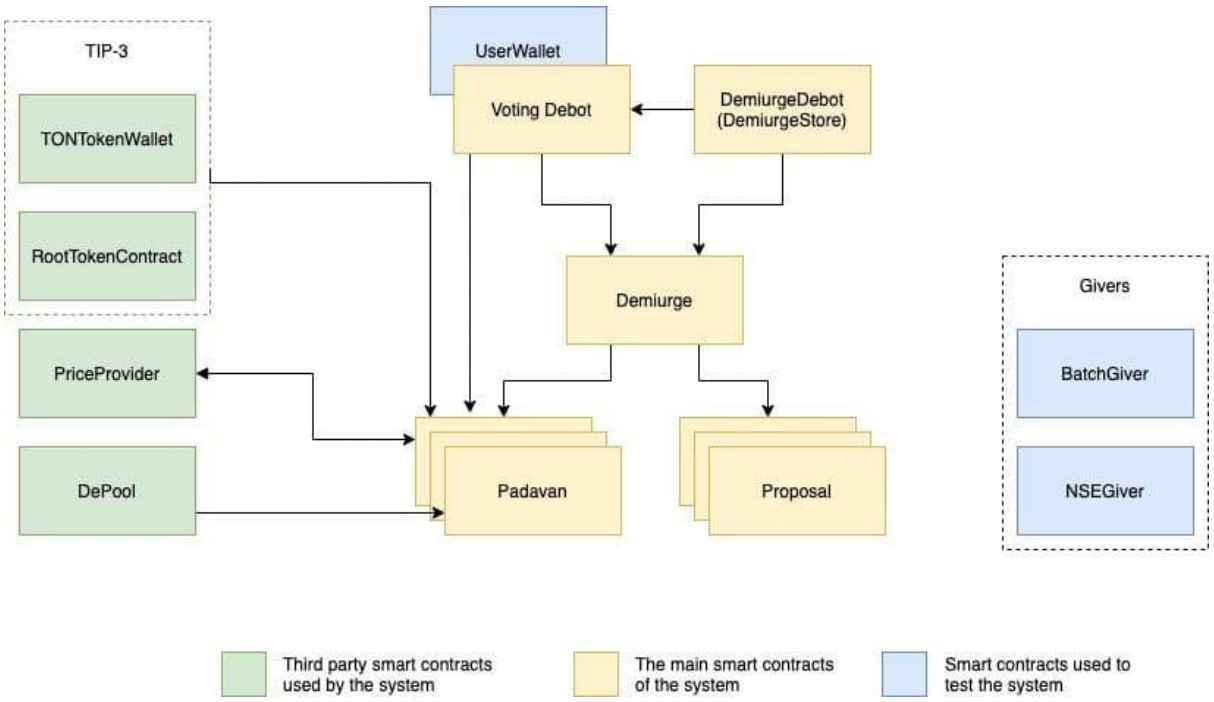


Figure 1 — High-level System Architecture

The implementation of the System requirements from the conditions of the contest is described in the table below:

Feature	Status	Comment
Should support the ability to vote with TON Crystal tokens as well as any other TIP-3 Token or DePool Stakes.	Fully supported	The system fully supports voting by TONs, all TIP-3 tokens and DePools. The scenarios are described in the section "Basic Voting Scenarios"
Should be able to grant voting rights to a subset of users identified with another token or PubKeys.	Partially supported	The System fully supports specifying the white list of voting users, but does not support token identification.
Should notify voting results by emitting both external and internal events	Fully supported	The System emits a ProposalFinalized with full info about Proposal results.
Should generate “Voting Finished” events if voting is finished early with these votes	Fully supported	The System emits a ProposalFinalized event when the timed / premature proposal ends.
Should be able to deploy Proposal Smart Contract and Collect votes for it	Fully supported	The System deploys Proposals and collects all the necessary data in the Demiurge contract.

Feature	Status	Comment
Should support a voting for Multiple Proposals using same TON Crystal or other TIP-3 Tokens	Fully supported	The System allows you to vote with TONs, a DePool stake and any TIP-3 token for all Proposals within the System an unlimited number of times
Should support Soft Majority, Super Majority settings of the SMV	Fully supported	The System supports Soft Majority, Super Majority and Majority voting models.
Should include Group membership smart contract with user rights	Fully supported	The System has a group contract that allows you to use them as sets of whitelists
Should support the ability to Add, Exclude New member, change rights of an existing member by Proposal result event	Full support	The System allows you to create new groups, add/remove members from groups by Proposal result.
Should be able to Deploy a Contest from the Proposal if the Proposal is approved	Partially supported	There is some way to give Proposal a message, which will be executed after Proposal is accepted, but, to do this requirement in the rightest way need more time for tests.
Should be able to change parameters of the Contest (such as Voting period, Jury Groups etc.)	Fully supported	As previous point, there are many ways to change the parameters of Contests.
Contest should include: Start of the contest time; End of the contest time; Time for jury voting; Set of jurors or Juror Groups pubkeys and addresses	Fully supported	As previous point, there are many ways to change the parameters of Contests.
Should include a Contract that can store a list of some SMV Proposals, Contests and their voting results	Partially supported	All data stored on the System, except Contests results data.
Should include DeBots for all system user interfaces	Fully supported	All System functionality can be controlled by written DeBots
Should include auto-tests	Fully	The System main functionality is fully tested.

Feature	Status	Comment
designed as a smart contract or a script to test scenarios	supported	See more at the Testing section.
A solution should have a Free Software license	Fully supported	There is Apache-2.0 License
A system should be deployed and tested on the DevNet and Jury should be able to access it for testing	Temporary not supported	See Limitations at the About section

Some additional features that are not described in the contest conditions are shown in the table below:

Additional Features	Status
The System supports multiple voting by one user for one message, in different ways.	Fully supported
The System supports voting with a part of users' votes.	Fully supported
The System supports a third-party PriceProvider for converting TIP-3 tokens into votes.	Fully supported
The architecture of the deposits of the System allows the withdrawal of Type-3 tokens at the rate that is current at the time of entering the TIP-3 tokens into the system.	Fully supported
The System implements mechanisms for integration with different third-party smart contracts	Fully supported
The system mainly works on internal messages.	Fully supported
DePools in the System are set by an array, which allows accepting stakes not from one previously specified DePool during initialization, but from any DePools of the network.	Fully supported

4. System's Smart contracts

4.1. The main smart contracts of the System

4.1.1. Demiurge

It is a central smart contract in a voting system. It is a ledger that creates and stores proposals and user padavan addresses. After deployment demiurge requests Demiurge Store smart contract to gain proposal and padavan images (tvc), list of depool addresses and address of vote price provider.

Demiurge starts in preworking mode in which it has several checks that must be passed before it will accept requests to deploy proposals and padavans. Checks contains the following:

- Check that the demiurge contains a proposal image.
- Check that the demiurge contains a padavan image.
- Check that demiurge contains a list of depools.
- Check that demiurge contains the address of a price provider.

When the check mask becomes equal to 0 Demiurge is ready to work.

PUBLIC API

constructor(address store)

store - address of Demiurge Store that stores all ABIs and TVCs of the voting system.

Remark: Demiurge Debot is a Demiurge Store as well.

Called on demiurge deployment. Calls Demiurge Store to acquire necessary parameters.

deployPadavan(uint userKey)

Called by internal message only and paid by caller. Allows to deploy Padavan smart contract.

userKey - public key sent by user that is inserted into an instance of Padavan before deployment.

deployProposal(uint32 totalVotes, uint32 start, uint32 end, string description, string text, VoteCountModel model)

Called by internal message only and paid by caller. Allows to create and deploy Proposal smart contract.

totalVotes - total number of votes for proposal.

start - unixtime when proposal starts accepting votes.

end - unixtime when proposal finishes accepting votes.

description - short name of the proposal.

text - any information about the proposal.

model - voting model, can be soft majority, super majority or simple majority.

deployProposalWithWhitelist(uint32 totalVotes, uint32 start, uint32 end, string description, string text, VoteCountModel model, address[] voters)

Called by internal message only and paid by caller. Allows to create and deploy Proposal that accepts votes only from Padavans from **voters** list.

All parameters are the same as in **deployProposal.**

voters - white list of Padavan addresses that can vote for proposal.

function onStateUpdate(ProposalState state)

Called by any proposal (created previously by this demiurge) to notify about his new status.

4.1.2. Padavan

Padavan smart contract is a user ballot that allows users to vote for proposals. Padavan accepts deposits of different types (tons, tip3 tokens, depool stakes), converts them to votes and sends votes to proposals. Votes cannot be converted into deposits and received back until all the proposals that the Padavan voted for are completed. The User can vote for different proposals with a different number of votes, but a number of locked votes in padavan is always the maximum number of votes spent for one proposal.

At any time a user can ask to reclaim some deposits equivalent to a number of votes. When it happens Padavan starts to query the status of all voted proposals. If any of them is already completed Padavan removes it from the active proposals list and updates the value of locked votes. If the required number of votes becomes less or equal to unlocked votes then Padavan converts the requested number of votes into a deposit (tons, tokens or stake) and sends it back to the user.

Padavan is controlled by a user contract that requested deployPadavan from Demiurge.

PUBLIC API

function voteFor(address proposal, bool choice, uint32 votes)

Called by internal message only and paid by caller. Allows to vote for the proposal with certain votes (yes or no).

proposal - address of proposal to vote for.

choice - ‘yes’ or ‘no’.

votes - number of votes to send for proposal.

function depositTons(uint32 tons)

Called by internal message only and paid by caller. Allows to deposit tons into Padavan. Deposit is converted to votes using the vote price requested from PriceProvider smart contract.

tons - number of tons to deposit and lock in Padavan.

function depositTokens(address returnTo, uint256 tokenId, uint64 tokens)

Called by internal message only and paid by caller. Allows to deposit and lock tip3 tokens into Padavan. Tokens must be already transferred to Padavan’s tip3 token account (wallet) before this function is called. This function checks that the balance of Padavan’s token wallet must be bigger then **tokens** argument. If so then the deposit is accepted and locked in a token account and converted into votes.

returnTo - address of user token wallet to which return tokens when they will be unlocked.

tokenId - ID of tip3 token. It is an address of the root token wallet without workchain id.

tokens - number tip3 tokens to deposit into Padavan.

function reclaimDeposit(uint32 deposit)

Called by internal message only and paid by caller. Allows to return deposits (tons, tip3 tokens, depool stake) back to the user.

deposit - number of votes that must be converted to deposits and returned to the user.

function confirmVote(uint64 pid, uint32 deposit)

Called by Proposal to notify Padavan that votes are accepted.

pid - proposal id.

deposit - number of accepted votes.

function rejectVote(uint64 pid, uint32 deposit, uint16 ec)

Called by Proposal to notify Padavan that votes are rejected.

pid - proposal id.

deposit - number of rejected votes.

ec - reason of rejection (exit code).

function updateStatus(uint64 pid, ProposalState state)

Called by Proposal to response on Padavan's **queryStatus** request.

state - proposal current state (can be New, onVoting, FInalized, Ended, Passed, Failed).

function createTokenAccount(address tokenRoot)

Allows user to create tip3 token wallet controlled by Padavan. Created token wallet can be used to deposit tip3 tokens to it.

tokenRoot - address of token root smart contract that emits tip3 tokens.

function onTransfer(address source, uint128 amount)

Called by DePool to transfer ownership of user stake to Padavan.

source - address of user wallet that transfers ownership to Padavan.

amount - number of transferred nanotons.

4.1.3. Proposal

Smart contract that accumulates votes from Padavans. Deployed by Demiurge by user request (**deployProposal**) Notifies about its state to Demiurge.

Can be optionally instantiated with a white list of Padavan addresses. In that case Proposal accepts votes only from addressed from this list.

PUBLIC API

voteFor(uint256 key, bool choice, uint32 deposit)

Called by Padavans to vote for the proposal. Proposal makes a verification check (see TIP3 spec) to be sure that the sender is a Padavan smc.

key - Padavan public key. Used in a verification check.

choice - "yes" or "no".

deposit - number of votes.

queryStatus()

Called by Padavan to query Proposal status.

wrapUp()

Can be called by any smart contract by internal message. Asks Proposal to update its status.

4.1.4. Demiurge Debot

An entry point to an onchain voting system. Allows to deploy new Demiurge to blockchain or to attach to existing Demiurge. Also deploys Voting Debot for users.

Debot implements the interface of Demiurge Store and stores all images (tvc) and ABIs of voting system contracts.

4.1.5. Voting Debot

Debot that works on behalf of the user. Deploys Padavan and allows to create new proposals, deposit tons, convert them to votes and vote for existing proposals.

4.2. External smart contracts used by voting system

4.2.1. NSEGiver

Builtin giver of NodeSE. Used to deploy contracts in local node tests.

4.2.2. RootTokenContract & TONTokenWallet

TIP3 smart contracts. Can be found here:

https://github.com/tonlabs/ton-labs-contracts/tree/master/cpp/tokens-fungible

4.2.3. DePool

DePool smart contract. Used to transfer ownership of user stake to Padavan. Can be found here:

https://github.com/tonlabs/ton-labs-contracts/tree/master/cpp/solidity/depool

4.2.4. PriceProvider

Simple smart contract that implements an interface of converting tons and tokens to votes.

4.3. Smart contracts used by test system

4.3.1. UserWallet

Test user wallet used to send requests to Demiurge and control Padavan.

4.3.2. BatchGiver

Giver smart contract that allows to make several transfers in one transaction. Used to increase speed of contracts deployment.

5. How to use DeBots

1. Run nodeSE locally.
2. Put tonos-cli (ver >= 0.2.0) into directory ./bin in the root of repo.
3. Deploy and init Demiurge Debot. Go to the root of project repo. Then change the current directory to sbin and run bash script deploy_debot.sh.
4. Important: set variable price_provider in script to actual price provider.
5. After Debot starts you will see the main menu.

```
Demiurge DeBot, version 0.1.0
Hello, user, i'm a Demiurge Debot! What do you want to do?
Current Demiurge: 0:0000000000000000000000000000000000000000000000000000000000000000
1) Attach another demiurge
2) Deploy new demiurge - start new voting platform onchain
3) Deploy user voting debot
4) Quit

debash$
```

6. Choose item 2. Generate a new seed phrase and generate a public key from the phrase. Enter the public key to debot.

```
Deploy new demiurge:
Please, generate seed phrase for new demiurge.
Enter public key generated from this phrase:
 > 0xd16572a3d454113535a5f29ef8b70539fcee26b3b157ee3321efcaff4cfa7728
Generated demiurge address: 0:eae5546502e3137fdc0199a1e498d1598e4bc948c9f4b3d9ec0c78e80
3bdf856
Now, sign message with keypair used to deploy me.
1) Sign and deploy
2) Exit

debash$
```

7. Choose “sign and deploy”. Then enter the path to DemiurgeDebot.keys.json as you see in the screenshot.

```
enter seed phrase or path to keypair file > ../src/DemiurgeDebot.keys.json
Sending message 6c82d099e4eaf38ae90e9442fb250dbd4bdcd21c8fdbc9aa00eeeaab5f44d676
Transaction succeeded.
Succeeded!
Please, restart debot.
1) Exit

debash$
```

8. Demiurge is deployed. Restart the debot.

```
Demiurge DeBot, version 0.1.0
Hello, user, i'm a Demiurge Debot! What do you want to do?
Current Demiurge: 0:EAE5546502E3137FDC0199A1E498D1598E4BC948C9F4B3D9EC0C78E803BDF856
1) Attach another demiurge
2) Deploy new demiurge - start new voting platform onchain
3) Deploy user voting debot
4) Quit

debash$
```

9. Now the debot attached to deployed Demiurge. Choose item 3 to deploy the Voting Debot for the user. Follow the instructions of debot.

```
Deploy User Voting Debot:
I will guide you step by step to deploy your personal debot for voting.
This debot will help you create proposals, vote for proposals and also deposit and recl
aim funds for voting
1) Ok, continue
2) Quit

debash$ 1
Step 1 of 4:
Important: generate master keypair for user debot. Keep it in secret because you will u
se it to control debot.
For security reasons debot cannot do it for you. Please, use external tool for this and
 come back with generated keypair.
1) I already generated keypair. Continue
2) I will generate keypair and come back later

debash$ 1
Step 2 of 4:
Ok, now you have to enter public key from generated keypair.
1) Enter public key

debash$
```

10. Then you will see the generated address of your future Voting Debot. Send some tons to this address. Use bash script ./sbin/nsegiver.sh <address> <nanotons>. Then continue with item 1.

```
 > 0x801fe857364d946a0b6b7a88510aa49feb5341e1080c10986367876475b1bf10
Step 3 of 4:
Voting Debot address: 0:e85282970fcab3f15e5b17796e0e96f9b8bd340729481f71c25356b667e79f0
4
Voting Debot public key: 0x801fe857364d946a0b6b7a88510aa49feb5341e1080c10986367876475b1
bf10
Please, send at least 10 tons to debot address before i will be able to deploy voting d
ebot at this address.
1) I have sent tons to this address. Continue
2) Transfer tons from user multisig
3) Quit

debash$
```

11. Sign deploy message with seed phrase of Voting Debot.

```
debash$ 1
Step 4 of 4:
I'm ready for deploy. Please, sign deploy message with generated keypair (at step 1).
1) Sign and deploy
2) Quit

debash$
```

12. Your Voting Debot is deployed.

```
enter seed phrase or path to keypair file > pluck attitude fashion earn fashion shoulde
r solid basic rebel jewel tenant spirit
Sending message 7ba2a683256ca7fe08576050f9b1c79528bf3eda18ff9faf02b645bba09aff36
Transaction succeeded.
```

13. Exit Demiurge DEbot and start Voting Debot.
14. ../bin/tonos-cli debot fetch <address_of_voting_debot> You don't have a Padavan yet. Choose item 1 to deploy it.

```
Voting DeBot, version 0.1.2
Hello, i'm your personal Voting Debot!
You don't have a padavan contract yet. Ready to deploy?
1) Yes, deploy
2) Quit

debash$
```

15. Enter the seed phrase of Voting Debot to sign deploy request.

```
Deploy Padavan:
Deploy fee is 3.50000000 tons
1) Deploy

debash$ 1
enter seed phrase or path to keypair file > pluck attitude fashion earn fashion shoulde
r solid basic rebel jewel tenant spirit
Sending message a5cae2b6b5771d846616f240c99ed25f0b1cf85473d8faafd137ce30841bf52c
Transaction succeeded.
Deploy succeeded!
Please, restart debot.
1) Exit

debash$
```

16. Padavan deployed. Restart the debot and you will see the extended menu of Voting Debot.

```
Voting DeBot, version 0.1.2
Hello, i'm your personal Voting Debot!
Active proposals: 0
Total votes: 0
Locked votes: 0
Available votes: 0
Ballot address: 0:3ce1ba707be780722d7cdeff9f55edeb22f2e3d67b4d089b0beef95299984483
1) Acquire votes
2) Reclaim votes
3) Vote for proposal
4) View my proposals
5) View all proposals
6) Create new proposal
7) Quit

debash$
```

17. Choose item 6 to create a new proposal. Follow the instructions to set proposal parameters. Sign request with Voting Debot.

```
debash$ 6
Create Proposal:
Enter total votes:
 > 20
Enter start time (unixtime):
 > 1610143714
Enter end time (unixtime):
 > 1610144700
Enter description:
 > very important proposal 1
Enter text:
 > some comments about this proposal
Choose voting model:
1) Super majority
2) Soft majority

debash$ 2

1) Sign and create

debash$
```

18. You will return to the main menu. Choose item 5 to view a list of all proposals.

```
Sending message 1a4669c4f5c59663d82a282700d8a982ef44a71c8433e0c8f9457304cae007e1
Transaction succeeded.
Hello, i'm your personal Voting Debot!
Active proposals: 0
Total votes: 0
Locked votes: 0
Available votes: 0
Ballot address: 0:3ce1ba707be780722d7cdeff9f55edeb22f2e3d67b4d089b0beef95299984483
1) Acquire votes
2) Reclaim votes
3) Vote for proposal
4) View my proposals
5) View all proposals
6) Create new proposal
7) Quit

debash$
```

19. You will see your created proposal. Return to main menu.

```
View proposals:
List of proposals
ID 0 very important proposal 1
Start: Sat, 09 Jan 2021 01:08:34 +0300, End: Sat, 09 Jan 2021 01:25:00 +0300
State: New, Total votes: 20, options: "soft majority", Address: 0:03c1274a6e78e32690043
438057331d15c1448e93dc04228b6874020030df4f2, creator: 0:e85282970fcab3f15e5b17796e0e96f
9b8bd340729481f71c25356b667e79f04

1) Return

debash$
```

20. Choose item 1 to deposit tons into Padavan and receive votes.

```
Deposit menu:
1) Deposit tons
2) Return

debash$ 1
Deposit tons:
Enter an integer number of tons:
 > 20
1) Sign and deposit
2) Exit

debash$ 1
```

21. After returning to the main menu you will see updated voting stats. You have 20 available votes.

```
Hello, i'm your personal Voting Debot!
Active proposals: 0
Total votes: 20
Locked votes: 0
Available votes: 20
Ballot address: 0:3ce1ba707be780722d7cdeff9f55edeb22f2e3d67b4d089b0beef95299984483
1) Acquire votes
2) Reclaim votes
3) Vote for proposal
4) View my proposals
5) View all proposals
6) Create new proposal
7) Quit

debash$
```

22. Choose item 3 to vote for the proposal. Enter the number of votes to send. Sign request with seed phrase of Voting Debot .

```
Follow the instruction:
Enter proposal id:
 > 0
Enter votes count:
 > 15
1) Vote Yes
2) Vote No

debash$ 1
enter seed phrase or path to keypair file > pluck attitude fashion earn fashion shoulde
r solid basic rebel jewel tenant spirit
Sending message df7b1c4ac574ce5d192b5c70e9e933b619759698105af88436250af0bc92b40f
Transaction succeeded.
```

23. Return to the main menu. Now you have 1 active proposal and 15 locked votes.

```
Hello, i'm your personal Voting Debot!
Active proposals: 1
Total votes: 20
Locked votes: 15
Available votes: 5
Ballot address: 0:3ce1ba707be780722d7cdeff9f55edeb22f2e3d67b4d089b0beef95299984483
1) Acquire votes
2) Reclaim votes
3) Vote for proposal
4) View my proposals
5) View all proposals
6) Create new proposal
7) Quit

debash$
```

24. Choose 4 to see a list of active proposals.

```
List of voted proposals:
ID 0 very important proposal 1
Start: Sat, 09 Jan 2021 01:08:34 +0300, End: Sat, 09 Jan 2021 01:25:00 +0300
State: Passed, Total votes: 20, options: "soft majority", Address: 0:03c1274a6e78e32690
043438057331d15c1448e93dc04228b6874020030df4f2, creator: 0:e85282970fcab3f15e5b17796e0e
96f9b8bd340729481f71c25356b667e79f04

Sent votes: 15
1) Return

debash$
```

25. Return to main and choose 2 to reclaim some votes.

```
Enter number of votes:
 > 10
1) Sign and reclaim

debash$ 1
enter seed phrase or path to keypair file > pluck attitude fashion earn fashion shoulde
r solid basic rebel jewel tenant spirit
Sending message 94b2a9bc31474a33304addfd5a146f1df46a32ff1bd0f6069306c87f82959fb8
Transaction succeeded.
```

26. Restart debot to update vote statistics. Now you have only 10 votes.

```
Hello, i'm your personal Voting Debot!
Active proposals: 0
Total votes: 10
Locked votes: 0
Available votes: 10
Ballot address: 0:3ce1ba707be780722d7cdeff9f55edeb22f2e3d67b4d089b0beef95299984483
1) Acquire votes
2) Reclaim votes
3) Vote for proposal
4) View my proposals
5) View all proposals
6) Create new proposal
7) Quit

debash$
```

Experiment more, for example, you can deposit more tons or reclaim all available votes.

6. Deploy and initialize the System

There are two ways to initialize the system - manual and through DeBot.

6.1. Manual system start

To start the system manually, you need to prepare:

- Demiurge contract
- Padavan contract
- PriceProvider contract
- Proposal

System deployment script:

- Deploy Demiurge
- Deploy PriceProvider
- Point Demiurge Proposal's tvc
- Point Demiurge Padavan's tvc
- Point Demiurge address of PriceProvider
- Point Demiurge addresses of DePools
- The System is ready to use

A more detailed example can be found at ./tests/parts/deploy-system.ts

6.2. System start using DeBots

The voting system can be configured and used with debots. There are 2 debots:

1. Demiurge Debot - central debot. One for the whole voting system. Deploys Demiurge and Voting Debot for each user.
2. Voting Debot. One debot per user. Deploy Padavan and allow users to vote.

See section 'How to use Debots' that describes how these debots are working.

7. User Scenarios

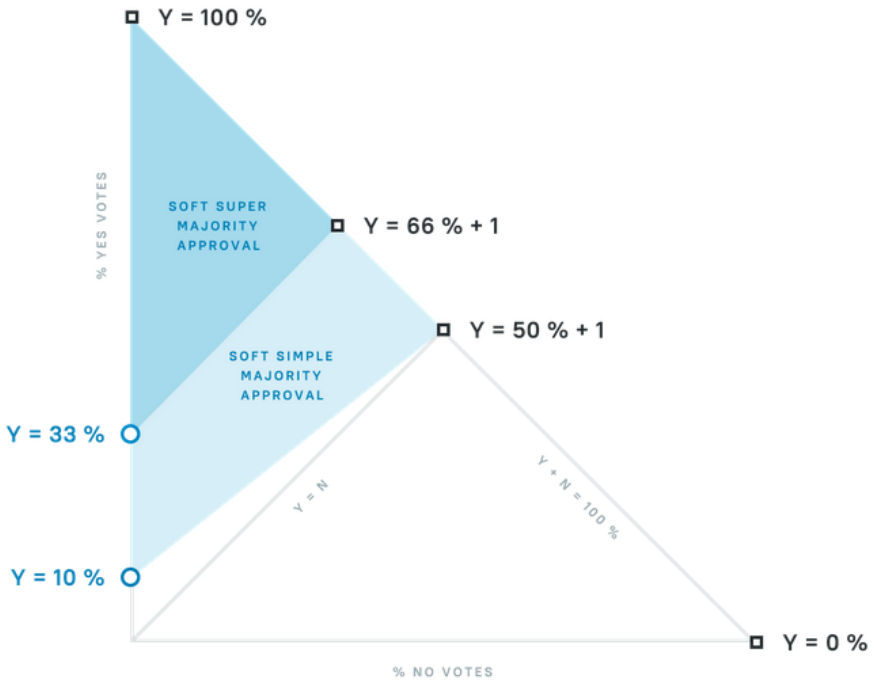
7.1. Proposal creation scenarios

Proposal can be created by any user after deploying and initializing the system. Importantly, Proposal accepts only internal messages, therefore, to deploy and work with Proposal, you should use Multisig, UserWallet from the example or analogs.

The main parameters and functions of the Proposal are described in System's Smart Contracts paragraph.

To create a Proposal, you need to specify:

- Voting period — the start and end time of voting after which the results are summed up
- The voting model is Majority, Soft Majotiry or Super Majority, below are the formulas for calculating the model, where y - votes for, n - votes against, t - total votes according to the picture:



  - Majority (y > n)
  - Soft Majority (y * t * 10 >= t * t + n * (8 * t + 20))
  - Super Majority (y * t * 3 >= t * t + n * (t + 6))

More details can be found here — https://forum.freeton.org/t/developers-contest-soft-majority-voting-system-finished/65

An example of using all models can be found in the majorities test

- Description
- Accompanying text
- White sheet of voters:
    - Not specified, in which case all Padavan owners can vote
    - Specified, in this case, only those Padavans whose identifier is indicated in the sheet vote
    - A link to the group is specified - in this case, only Padavans members of the specified group vote
- Appointment proclaimed
    - No final result handler
    - To create a contest
    - To add to the group
    - To remove from the group
    - To create a group

Please, see proposal creation examples here ./tests/parts/deploy-proposal.ts

7.2. Group scenarios

1. Adding a new member to the group
    a. invoke *applyFor(string name)* function of the *Group* contract, as specified in the *IGroup* interface. Address of the sender is considered to be applying for the group membership. NB: *Padavan* contract can be efficiently used for the submission process. The corresponding function is *applyToGroup(address group, string name)* from the *IPadavan* interface.
    b. Provided the input data is valid, a proposal to include a new member to the group is automatically created and put to voting.
    c. Upon voting completion, the results are evaluated.
    d. If the proposal passes, the applicant is added to the list of group members, and becomes eligible (and responsible) to vote for the proposals in scope of this group from now on.
2. Removing a member from the group
    e. invoke *unseat(uint32 id, address addr)* function of the *Group* contract, as specified in the *IGroup* interface. The respective helper in the *IPadavan* interface is *removeFromGroup(address group, uint32 id, address addr)*.
    f. Provided the input data is valid, a proposal to remove the specified member from the group is automatically created and put to voting.
    g. Upon voting completion, the results are evaluated.
    h. If the proposal passes, the specified member is removed from the group, thus revoking voting rights for the proposals deployed subsequently.
3. Voting using groups (whitelist)

i. proposals deployed by a group are put to voting in a very special fashion, enabling only a selected list of individual contracts to vote for them. This voting model is sometimes referred to as “whitelist”. Proposals with this feature disregard any votes cast from the addresses not on the list.

7.3. Base Voting Scenarios

1. Voting with TON
    - User deploys Padavan, or requests previously deployed Padavan
    - User sends TONs to Padavan
    - Padavan "converts" TONs into voices
    - User sends votes from Padavan to Proposal
    - The volume of sent votes in tokens is frozen
    - Proposal ends on time, or prematurely, if the result is unambiguous
    - The volume of votes sent by all users is unfrozen
    - Proposal informs Demiurge of the voting result
    - The Demiurge performs an action if it was described and the result was accepted
    - User withdraws deposited TONs

Base and base-against test (to check voting for and against, respectively)

2. Voting with DePool:
    - User deploys Padavan, or requests previously deployed Padavan
    - The user transfers the stake from the DePool specified in the Demiurge to the Padavan
    - Padavan "converts" stake into votes
    - User sends votes from Padavan to Proposal
    - The volume of sent votes is frozen
    - Proposal ends on time, or prematurely, if the result is unambiguous
    - The volume of votes sent by all users is unfrozen
    - Proposal informs Demiurge of the voting result
    - The Demiurge performs an action if it was described and the result was accepted
    - User withdraws stake

3. Voting using TIP-3
    - User deploys Padavan, or requests previously deployed Padavan
    - User creates a token account for Padavan
    - User transfers tokens to Padavan
    - Padavan "converts" tokens into votes at the rate given by PriceProvider
    - User sends votes from Padavan to Proposal
    - The volume of sent votes in tokens is frozen
    - Proposal ends on time, or prematurely, if the result is unambiguous
    - The volume of votes sent by all users is unfrozen

- Proposal informs Demiurge of the voting result
- The Demiurge performs an action if it was described and the result was accepted
- User withdraws sent tokens

4. Basic voting scenario with combined votes. Combines the first three scenarios and combines ways to get votes.

8. Testing

All tests of the System are located in the tests directory.

For tests used:

```
"chai": "^4.2.0",
"mocha": "^8.2.1",
"typescript": "^4.1.3",
"@tonclient/core": "^1.5.0",
"@tonclient/lib-node": "^1.5.0"
```

To run tests, it is proposed to use Node in the package ton-dev-cli (https://github.com/tonlabs/ton-dev-cli). Important! Docker is required for correct work of tondev.

1. Install node.js (https://nodejs.org/en/)
2. Install docker (https://www.docker.com)
3. Install tondev. `npm install -g ton-dev-cli` If you encounter problems during installation, read the instructions in the official repository
4. Go to the project folder and install the dependencies `npm install`
5. Create `.env` file at the root of the project and fill it in. Available variables (this example can be used to work with Node SE):

```
NSE_GIVER_ADDRESS=0:841288ed3b55d9cdafa806807f02a0ae0c169aa5edfe88a789a6482429756a94
NETWORK=LOCAL
```

6. Run Node SE `tondev start`
7. Run tests:
   a. `npm run test` will run all available tests

b. `npm run test:TEST_NAME` will run the specified test, where the `TEST_NAME` is the name of the test (see “Description of tests”)

8.1. Infrastructure

The project infrastructure consists of the following directories:

```
├── index.ts
├── package-lock.json
├── package.json
├── src     // smart-contracts source code
├── tests   // tests dir
|   ├── base-against.test.ts  // base voting scenario #1 (against)
|   ├── base-depool.test.ts   // base voting scenario #2
|   ├── base-token.test.ts    // base voting scenario #3
|   ├── base.test.ts          // base voting scenario #1
|   ├── majorities.test.ts    // test different majorities
|   └── whitelist.test.ts     // test whitelist
|   ├── contracts             // ton contracts packages dir
|   |   ├── ton-contract.ts             // class for ton-contracts
|   |   └── ton-packages                // ton-contracts packages with tvc and abi
|   |       ├── alt-giver.package.ts
|   |       ├── batch-giver.package.ts
|   |       ├── console.package.ts
|   |       ├── demiurge.package.ts
|   |       ├── depool.package.ts
|   |       ├── dev-giver.package.ts
|   |       ├── group.package.ts
|   |       ├── nse-giver.package.ts
|   |       ├── padavan.package.ts
|   |       ├── priceprovider.package.ts
```

```
|   |       ├── proposal.package.ts
|   |       ├── roottokencontract.package.ts
|   |       ├── tontokenwallet.package.ts
|   |       └── userwallet.package.ts
|   ├── parts                // test parts dir
|   |   ├── check-proposal-results.ts
|   |   ├── deploy-padavan.ts
|   |   ├── deploy-proposal.ts
|   |   ├── deploy-system.ts
|   |   ├── deposit-to-padavan.ts
|   |   ├── reclaim.ts
|   |   ├── reclaimTokens.ts
|   |   └── vote.ts
|   ├── utils                // test utils dir
|   |   ├── code.ts
|   |   ├── common.ts
|   |   └── convert.ts
└── tsconfig.json
```

8.1.1. ton-contracts.ts

Class for working with TON contracts. It provides a convenient interface for deploying, calling, getting balance, and so on. Used in tests everywhere.

Interface:

```
export class TonContract {

 client: TonClient;

 name: string;

 tonPackage: TonPackage;

 keys?: KeyPair;

 address?: string;
```

```
 async init(params?: any): Promise<void> {}

 async callLocal({ functionName, input = {} }: { functionName: string; input?: {} }):
Promise<DecodedMessageBody> {}

 async call({ functionName, input }: { functionName: string; input?: any }):
Promise<ResultOfProcessMessage> {}

 async calcAddress({ initialData } = { initialData: {} }): Promise<string> {}

 async deploy({ initialData, input }: { initialData?: any; input?: any } = {}):
Promise<ResultOfSendMessage> {}

 async getBalance(): number {}

}
```

8.1.2. ton-packages.ts

Package which consists of ABI and tvc.

Interface:

```
type TonPackage = {

 image: string;

 abi: {};

};
```

8.2. Description of tests

8.2.1. Test “base”

It tests the Basic Voting Scenario # 1, where voting takes place by depositing TONs. The test demonstrates:

- the acceptance of the Proposal,
- the ability to vote several times using the same Padavan with a different number of votes,
- premature completion of Proposal with an obvious result,
- sending an event about the completion of voting

Test case:

- deploys and initializes the System;
- creates a Soft Majority Proposal with maximum of 10 votes;
- creates Padavan;
- deposits 10 TONs to padavan from DePool;
- sends 4 votes for Proposal;
- sends 2 more votes for Proposal to check for premature completion and the possibility of multiple sending of votes;
- checks the voting result that Proposal:
    - has been finished,
    - has been accepted,
    - tokens on Padavan are no longer frozen for reclaim;
- reclaim TONs.

8.2.2. Test “base-against”

It tests the Basic Voting Scenario # 1, where voting takes place by depositing TONs, like test “base”, but tests against scenario

Test case:

- deploys and initializes the system;
- creates a Soft Majority Proposal with maximum of 10 votes;
- creates Padavan;
- deposits 10 TONs to padavan from DePool;
- sends 6 votes against Proposal;
- checks the voting result that Proposal:
    - has been finished,
    - has been declined,
    - tokens on Padavan are no longer frozen for reclaim;
- reclaim TONs.

8.2.3. Test “base-depool”

It tests the Basic Voting Scenario # 2, where voting takes place by DePool staking. The test demonstrates:

- the acceptance of the Proposal,
- work with DePool within the system, deposit of stakes.

Test case:

- deploys and initializes the system;
- creates a Soft Majority Proposal with maximum of 10 votes;
- creates Padavan;
- transfers 10 TONs to padavan from DePool;
- sends 6 votes for Proposal;
- checks the voting result that Proposal:

    - has been finished,
    - has been passed,
    - tokens on Padavan are no longer frozen for reclaim;
- reclaim stake.

8.2.4. Test “base-token”

Testing the Basic Voting Scenario # 3.

This test verifies the correctness of voting using TIP-3 tokens. For the test, a new token is created, wallets for the user and Padavan. The user transfers tokens to the Padavan's wallet, the Padavan contacts the PriceProvider and converts the tokens into votes at the provided rate.

Test case:

- deploys and initializes the system;
- creates a Soft Majority Proposal with maximum of 10 votes;
- creates Padavan;
- creates test TIP-3 token. Deploys test RootTokenContract (rootToken) and TonTokenWallet (userToken) for user;
- creates token account for Padavan;
- deposit tokens from user account to Padavan account;
- padavan calculates votes count using PriceProvider;
- sends 10 votes for Proposal;
- checks the voting result that Proposal:
    - has been finished,
    - has been passed,
    - tokens on Padavan are no longer frozen for reclaim;
- reclaim tokens;

8.2.5. Test “majorities”

Testing different majorities Proposals.

This test verifies the correctness of the vote counting models. Six Proposals with different models are deployed and when the votes are transferred to them, they work out in different ways, according to the above formulas in the section “Proposal creation scenarios”.

Test case:

- deploys and initializes the system;
- creates a Soft Majority Proposal (proposal) with maximum of 10 votes;
- creates a Soft Majority Proposal (proposal2) with maximum of 10 votes;
- creates a Super Majority Proposal (proposal3) with maximum of 10 votes;
- creates a Super Majority Proposal (proposal4) with maximum of 10 votes;
- creates a Majority Proposal (proposal5) with maximum of 10 votes;

- creates a Majority Proposal (proposal6) with maximum of 10 votes;
- creates Padavan;
- deposit TONs to Padavan;
- sends 5 votes for proposal;
- sends 5 votes against proposal;
- checks that proposal has been finished and declined;
- sends 5 votes for proposal2;
- sends 4 votes against proposal2;
- checks that proposal2 has been finished and accepted;
- sends 7 votes for proposal3;
- sends 3 votes against proposal3;
- checks that proposal3 has been finished and accepted;
- sends 5 votes for proposal4;
- sends 4 votes against proposal4;
- checks that proposal4 has been finished and declined;
- sends 5 votes for proposal5;
- sends 5 votes against proposal5;
- checks that proposal5 has been finished and declined;
- sends 5 votes for proposal6;
- sends 4 votes against proposal6;
- checks that proposal6 has been finished and accepted;
- reclaim tokens.

8.2.6. Test “whitelist”

Testing whitelist functionalities.

This test verifies that the whitelist is working correctly. Two Padavans are created, one of which is added to the white list and checked so that the one who does not have the right to vote could not vote.

Test case:

- deploys and initializes the system;
- deploys first Padavan (padavan);
- deploys second Padavan (padavan2);
- creates a Soft Majority Proposal (proposal) with maximum of 10 votes and add padavan2 to whitelist;
- deposits 10 TONs to padavan;
- deposits 10 TONs to padavan2;
- sends 10 votes for proposal from padavan, expects error;
- sends 10 votes for proposal from padavan2, checks that the votes are counted;
- checks that proposal2 has been finished and accepted;
- padavan reclaim TONs.
- padavan2 reclaim TONs.