

Tymoteusz Piwowarski, Krzysztof Muszyński, Stipe Rimac

1. Overview & Architecture

This project implements an on-chain quadratic voting system for DAO proposals, fulfilling all requirements:

1. VotingToken.sol

- ERC-20 token (OpenZeppelin) extended with owner-only mint and participant-only burn.
- Manages the supply of voting tokens, each costing a fixed tokenPrice in wei.

2. IExecutableProposal.sol

- Interface for external “action” contracts.
- Defines executeProposal(uint proposalId, uint numVotes, uint numTokens) to be called when a proposal passes.

3. QuadraticVoting.sol

- Core governance contract (inherits OpenZeppelin’s ReentrancyGuard).
- Manages participants, token sales/refunds, proposal lifecycle, vote staking, auto-execution, and final closing.

Key Data Structures & Flows

- **Participants** register at any time by sending Ether; they receive tokens proportional to msg.value / tokenPrice (with excess refunded).
- **Proposals** are created by participants while voting is open. They can be:
 - **Funding** (budget > 0): auto-approve when votes ≥ threshold and budget is available.
 - **Signaling** (budget = 0): tallied but executed only upon closeVoting().
- **Quadratic staking**: each additional vote on a proposal costs (totalVotes)² – (prevVotes)² tokens. Tokens are locked until votes are withdrawn or consumed by execution.
- **Dynamic threshold** for funding proposals:

$$threshold_i = (0,2 + \frac{budget_i}{totalbudget}) \cdot numParticipants + numPendingProposals$$

computed on each stake.

2. Detailed Design Decisions

2.1. Token Economics & ERC-20 Extension

- **OpenZeppelin's ERC-20** imported for battle-tested correctness.
- **Owner-only mint, participant-only burn**, to maintain supply and allow participants to exit.
- **Excess Ether refunds** in addParticipant/buyTokens avoid “dust” accumulation.

2.2. Proposal Representation

- A lightweight struct Proposal holds metadata and vote tallies within QuadraticVoting, optimizing gas over deploying a full contract per proposal.
- **External payloads** (the actual business logic when a proposal passes) live in separate contracts implementing IExecutableProposal, keeping governance data and execution logic decoupled.

2.3. Voting & Budget Accounting

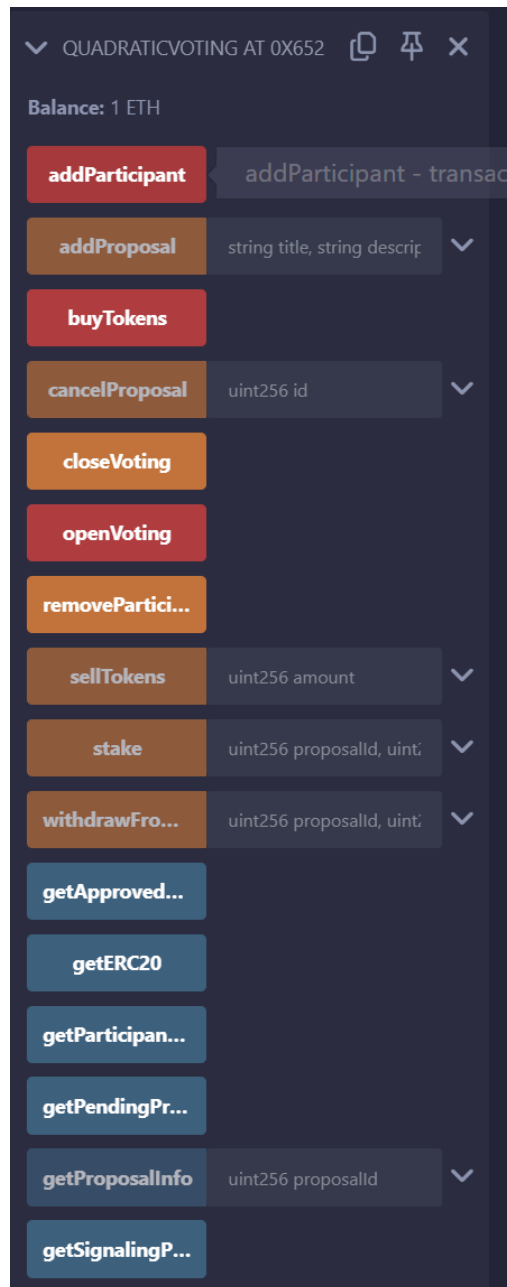
- **lockedTokens mapping** tracks tokens staked in votes, preventing double-spend via sellTokens or removeParticipant.
- On **auto-approval**:
 1. Release locked tokens back to the budget (update lockedTokens and votingBudget).
 2. Transfer p.budget to external executeProposal call with a 100 000 gas cap.
 3. Update remaining votingBudget.

3. Solidity Best Practices & Resources

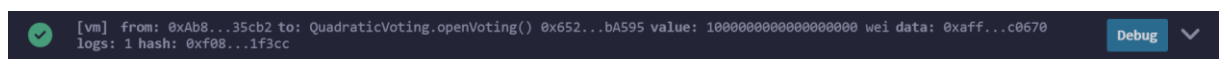
- **Modifiers** (onlyOwner, onlyParticipant, votingOpen) encapsulate common checks.
- **Inheritance** from OpenZeppelin's ReentrancyGuard for nonReentrant on all Ether-transferring functions.
- **Minimal external state**: only three contracts, each with a single well-defined responsibility.
- **Inline comments** document critical reasoning and guard conditions, aiding maintainability.

4. Security & Vulnerability Mitigation

Threat	Mitigation
Reentrancy attacks	All functions that transfer Ether or burn tokens (e.g. removeParticipant, sellTokens, _checkAndExecuteProposal) are protected with the nonReentrant modifier from OpenZeppelin's ReentrancyGuard.
Unexpected callbacks	External calls to executeProposal are restricted to 100,000 gas, minimizing the chance for state manipulation via callbacks.

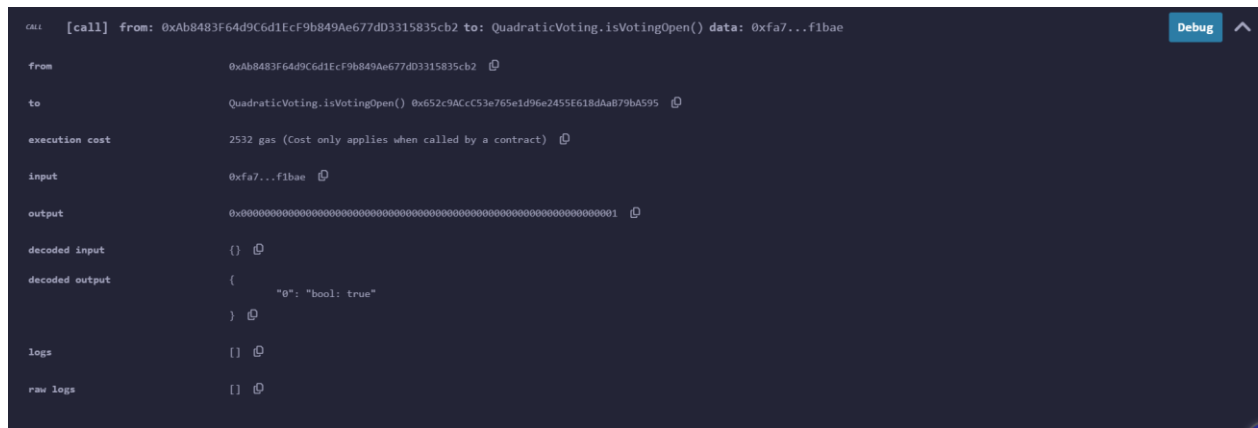


Check if openVoting works



Voting is now opened

Let's check isVotingOpened()



CALL [call] from: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 to: QuadraticVoting.isVotingOpen() data: 0xfa7...f1bae

from: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2

to: QuadraticVoting.isVotingOpen() 0x652c9ACc53e765e1d96e2455E618dAaB79bA595

execution cost: 2532 gas (Cost only applies when called by a contract)

input: 0xfa7...f1bae

output: 0x0001

decoded input: {}

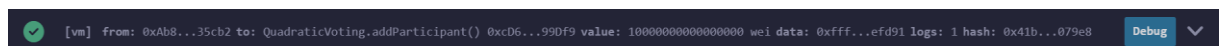
decoded output: { "0": "bool: true" }

logs: []

raw logs: []

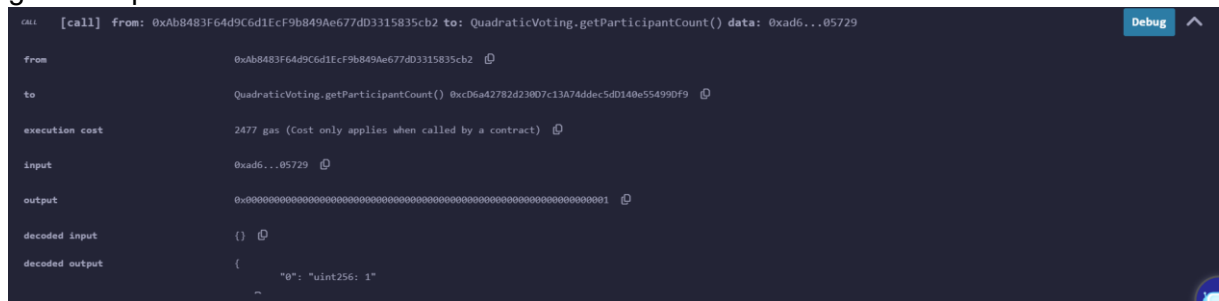
Bool: 'true' then voting is opened.

Adding participant



✓ [vm] from: 0xAb8...35cb2 to: QuadraticVoting.addParticipant() 0xcD6...99DF9 value: 10000000000000000 wei data: 0xffff...efd91 logs: 1 hash: 0x41b...079e8

getParticipantCount works



CALL [call] from: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 to: QuadraticVoting.getParticipantCount() data: 0xad6...05729

from: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2

to: QuadraticVoting.getParticipantCount() 0xcD6a42782d23807c13A74ddec5dD140e55499DF9

execution cost: 2477 gas (Cost only applies when called by a contract)

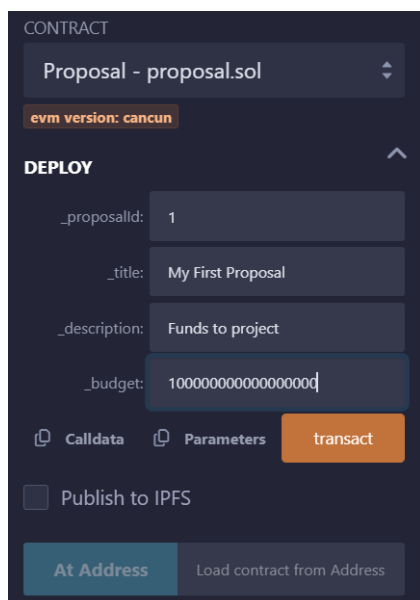
input: 0xad6...05729

output: 0x0001

decoded input: {}

decoded output: { "0": "uint256: 1" }

Deploying proposal



CONTRACT

Proposal - proposal.sol

evm version: cancun

DEPLOY

_proposalId: 1

_title: My First Proposal

_description: Funds to project

_budget: 10000000000000000

Calldata Parameters transact

☐ Publish to IPFS

At Address Load contract from Address



✓ [vm] from: 0x4B2...C02db to: Proposal.constructor() value: 0 wei data: 0x608...00000 logs: 0 hash: 0xc1c...f2a19

Adding deployed proposal to quadratic voting contract

✓

QUADRATICVOTING AT 0XCD€

Balance: 5.01 ETH

addParticipant

ADDPROPOSAL

title:

My First Proposal

description:

This funds the project

budget:

10000000000000000

proposalAddress:

0xa9d281dA3B02DF2ffc8A1955c

Calldata

Parameters

transact

✓ [vm] from: 0xAb8...35cb2 to: QuadraticVoting.addProposal(string,string,uint256,address) 0xcD6...99Df9 value: 0 wei data: 0x9ab...00000 logs: 0 hash: 0xe2f...07bfc Debug

Proposal count

CALL [call] from: 0xAb8483F64d9C6d1EcF9b849Ae677d03315835cb2 to: QuadraticVoting.proposalCount() data: 0xda3...5c664 Debug

from

0xAb8483F64d9C6d1EcF9b849Ae677d03315835cb2

to

QuadraticVoting.proposalCount() 0xcD6a42782d230D7c13A74ddc5dD140e55499Df9

execution cost

2492 gas (Cost only applies when called by a contract)

input

0xda3...5c664

output

0x0001

decoded input

()

decoded output

{ "0": "uint256: 1"

Proposal was added successfully

Approving proposal in voting token:

✓

VOTINGTOKEN AT 0XB27...07C

Balance: 0 ETH

APPROVE

spender:

0xcD6a42782d230D7c13A74dd€

amount:

1

Calldata

Parameters

transact

✓ [vm] from: 0xAb8...35cb2 to: VotingToken.approve(address,uint256) 0xb27...07c2c value: 0 wei data: 0x095...00001 logs: 1 hash: 0xc09...1f7cc Debug

Now call stake function

STAKE

proposalId: 1

newVotes: 1

Calldata Parameters **transact**

[vm] from: 0xAb8...35cb2 to: QuadraticVoting.stake(uint256,uint256) 0xcD6...99Df9 value: 0 wei data: 0x7b0...00001 logs: 3 hash: 0x7e6...e20f7 **Debug**

getProposalInfo with proposal id = 1

getProposalInfo 1 getProposalInfo - call

0: string: title My First Proposal

1: string: description This funds the project

2: uint256: budget 1000000000000000000

3: address: creator 0xAb8483F64d9C6d1Ec F9b849Ae677dD3315835cb2

4: uint8: status 3

5: bool: isSignaling false

6: uint256: totalVotes 1

Now let's try to check other type of proposal

QUADRATICVOTING AT 0XCDE

Balance: 4.91 ETH

addParticipant

ADDPROPOSAL

title: "Signal Proposal"

description: "Expressing preference"

budget: 0

proposalAddress: "0xa9d281dA3B02DF2fc8A1955"

Calldata Parameters **transact**

getProposalInfo 2

0: string: title Signal Proposal

1: string: description Expressing preference

2: uint256: budget 0

3: address: creator 0xAb8483F64d9C6d1Ec F9b849Ae677dD3315835cb2

4: uint8: status 0

5: bool: isSignaling true

6: uint256: totalVotes 0

I cancelled this proposal

[vm] from: 0xAb8...35cb2 to: QuadraticVoting.cancelProposal(uint256) 0xcD6...99Df9 value: 0 wei data: 0xe0a...00002 logs: 0 hash: 0xae8...2db2d **Debug**

This project successfully implements a secure, gas-efficient, and modular on-chain quadratic voting system for DAOs. Through careful use of OpenZeppelin libraries, explicit Ether/token management, and dynamic proposal handling, it ensures both fairness and safety in decentralized governance.