🌟

Angry Teenagers DAO contracts

Created	@July 11, 2022 3:19 PM
Lead	Christophe
Status	Completed
Type	Technical Spec
Related to Back End Work (Specification)	
Related to Front End Work (1) (Specification)	
Property	

Background

The goal of this document is to describe the Angry Teenager DAO contracts as well as the way to deploy them on the Tezos blockchain.

Contents

Technical overview of the Angry Teenager DAO

Code can be found in our GitHub

Requirements

1. The DAO shall initially implement two voting strategies:

- A “majority voting” strategy with a dynamic quorum and a supermajority threshold of 80%. Quorum is re-evaluated after each vote by adding 80% of the last quorum with 20% of the current total votes.
- An “opt-out strategy”: The opt-out strategy defines two phases. In the first phase, the proposal is injected. People can vote only to reject the proposal. If 10% or more of the total voting power of the FA2 contracts at the time the vote is opened rejects the proposal then phase 2 is entered, else the vote is marked as passed and closed. The phase 2 uses a “majority voting” with a fixed quorum of 25% of the voting power of the FA2 contract and a supermajority contract of 80%.

2. A proposal shall contain:
    - A *string* title
    - A link to the proposal text stored in a decentralised storage medium
    - A hash of this proposal text
    - A Tezos address of the author
    - An optional lambda code which executes if and only if the proposal is passed. This lambda contains a list of operations that are executed if the proposal is passed. The lambda will be fund transfer (to send fund from the DAO to the project wallet) and will look like this:

    ```
    { { DROP ;
    NIL operation ;
    PUSH address  PROJECT_WALLET_ADDRESS;
    CONTRACT unit ;
    IF_NONE { PUSH int 7 ; FAILWITH } {} ;
    PUSH mutez AMOUNT_TEZ_TO_TRANSFER ;
    UNIT ;
    TRANSFER_TOKENS ;
    CONS } }
    ```

    - The voting strategy used (see previous req (1))
    - In AngryTeenagers, only the EcoMint admin can inject a proposal.
    - Everybody can call the end entry-point to end a vote. This is to avoid a vote to be blocked.
3. Each voting strategy shall contain governance parameters. This governance parameters shall be chosen carefully as they cannot be changed. However new votin strategies can be added by the DAO itself or by the EcoMint admin.

1. The DAO shall use the FA2 contracts holding the AngryTeenagers NFTs. A snapshot of the voting power of a user is taken when the proposal is injected. When a vote is introduced the voting power of a user is computed as it was when the proposal was injected (and not when the vote entry-point is called).
2. If no FA2 contract is registered, all actions on the DAO are rejected. The FA2 contract holding the AngryTeenagers can only be registered by the EcoMint admin and only one time.
3. New voting strategies can be introduced by the DAO using a proposal with a lambda or by the EcoMint admin.
4. All historical data of passed votes are recorded and can be retrieved via off-chain views.

5. Data of vote in progress can be retrieved via off-chain views.

Architecture

The deployed DAO contains 4 contracts linked together:

1. The main DAO contract. This is the main set of entry-points for the user. It contains a link to the FA2 AngryTeenagers NFTs collection and two voting strategies by defaults (each of this voting strategies is implemented in a separate contract).
2. The “majority” voting strategy contract.
3. The “opt-out” voting strategy contract.
4. Another “majority” voting strategy contract with different governance parameters and used by the “opt-out” contract to manage the phase 2.

Main DAO contract

State machine

The state of the contract can be retrieved using the off-chain view get_contract_state

```
#### NONE (0)
No vote are in progress
Transition:
propose -> State STARTING_VOTE (1)

#### STARTING_VOTE (1)
A valid proposal has been injected. The voting strategies contract is requested to open a vote.
propose_callback -> State VOTE_ONGOING (2)

#### VOTE_ONGOING (2)
The voing strategies opened a vote succesffully. A vote is in progress and is waiting for the starting block to be r
eached to allow vote to be sent.
end -> State ENDING_VOTE (3)
end_with_malformed_lambda -> State ENDING_WITH_MALFORMED_LAMBDA (4)

#### ENDING_VOTE (3)
The end entrypoint has been called and the voting strategy contract is requested to close the vote.
end_callback -> NONE (0)
next_voting_phase_callback -> State VOTE_ONGOING (2)

#### ENDING_VOTE_WITH_MALFORMED_LAMBDA (4)
In case the lambda is malformed, the end entrypoint will fail making the contract to be locked. We can then call wit
h the admin right the function end_with_malformed_lambda.
end_callback -> NONE (0)
next_voting_phase_callback -> State VOTE_ONGOING (2)
```

Interface

Contract entrypoints:

- `add_voting_strategy`: (who: DAO i.e the contract itself with a lambda). Add a new voting strategies.

  ```
  sp.TRecord(id=sp.TNat, name=sp.TString, address=sp.TAddress)
  ```

- `register_angry_teenager_fa2`: (who: admin) The admin registers the FA2 contract holding the AngryTeenagers collection. This function can only be called one time.

  ```
  sp.TAddress
  ```

- `propose`: (who: admin) A proposal is injected for a new vote. In the AngryTeenagers project only the admin can call this function.

  ```
  sp.TRecord(
    title = sp.TString,
    description_link = sp.TString,
    description_hash = sp.TString,
    proposal_lambda = sp.TOption(sp.TLambda(sp.TUnit, sp.TList(sp.TOperation))),
    voting_strategy = sp.TNat
  ```

- `propose_callback`: (who: the voting strategy contract chosen by the proposal) the voting strategy callback this contract to acknowledge the creation of a new vote.

  ```
  sp.TRecord(id=sp.TNat)
  ```

- `vote`: (who: AngryTeenagers owner) AngryTeenagers owners sending their vote.

  ```
  sp.TRecord(proposal_id=sp.TNat, vote_value=sp.TNat)
  ```

- `end`: (who: all) Ending the vote. Everybody can call this function to avoid a vote to be blocked.

  ```
  sp.TNat
  ```

- `end_with_malformed_lambda` (who: admin): End the vote without executing the lambda of the proposal. This is useful to unlock the contract when the lambda of a proposal is malformed and fails to execute. This function, if the vote is passed and if the proposal attached to the vote contains a lambda will:

  1. mark the vote as failed
  2. add a information about the failure (given by parameters)
  3. store everything in the contract storage for transparency reasons.

  ```
  sp.TRecord(proposal_id=sp.TNat, lambda_error=sp.TRecord(description_link=sp.TString, hash_description=sp.TString))
  ```

- `end_callback`: (who: the voting strategy contract chosen by the proposal) The voting strategy contract chosen by the proposal callback this contract to close the vote.

  ```
  sp.TRecord(voting_id=sp.TNat, voting_outcome=sp.TNat)
  ```

- `next_voting_phase_callback`: (who: the voting strategy contract chosen by the proposal) In case the voting strategy has several phases, it will call this function instead of `end_callback` and let the vote open for this next phase.

  ```
  sp.TNat
  ```

- `mutez_transfer`: (DAO) Allow the DAO to remove XTZ fund from the contract.

  ```
  sp.TRecord(destination=sp.TAddress, amount=sp.TMutez)
  ```

- `default`: Allow to send XTZ to the contract

  ```
  sp.TUnit
  ```

- `delegate`: (DAO) Allow the DAO to delegate the XTZ on the contract

  ```
  sp.TOption(sp.TAddress)
  ```

- `set_next_admin`: (who: admin) Set the next admin. The next admin is not admin yet. It will be come admin by calling the entrypoint `validate_new_admin`.

  ```
  sp.TAddress
  ```

- `validate_next_admin` (who: “next_admin”) Validate the new admin. `set_next_admin` shall be called by the old admin prior to this function.

  ```
  sp.TUnit
  ```

- `unlock_contract` (who: “admin”) Unlock the contract when it is stuck in the state STARTING_VOTE or ENDING_VOTE. This could be useful if the voting strategy of the proposal is registered but not reachable anymore. Then the admin can unlock the contract by setting it back to state NONE. This can only be done 10 blocks after the proposal is injected.

  ```
  sp.TUnit
  ```

▼ Off-chain view

- `get_number_of_historical_outcomes`: Return the number of historical votes outcome (vote in progress is not counted).

  ```
  sp.TUnit
  ```

Return:

```
sp.TNat
```

- `get_historical_outcome_data`: Get the historical data of one particular vote.

  ```
  sp.TNat
  ```

  Return:

  ```
  sp.TBigMap(sp.TNat, sp.TRecord(
    outcome = sp.TNat,
    poll_data = PollType.POLL_TYPE
  ))
  ```

- `is_poll_in_progress`: Return whether a vote is in progress or not

  ```
  sp.TUnit
  ```

  Return:

  ```
  sp.TBool
  ```

- `get_current_poll_data`: Get the current data of the vote that is in progress if it exists.

  ```
  sp.TUnit
  ```

  Return:

  ```
  sp.TRecord(
                  proposal=Proposal.PROPOSAL_TYPE,
                  proposal_id=sp.TNat,
                  author=sp.TAddress,
                  voting_strategy_address=sp.TAddress,
                  voting_id=sp.TNat,
                  snapshot_block=sp.TNat
  )
  ```

- `get_contract_state`: Get the state of the contract.

  ```
  sp.TUnit
  ```

  Return:

  ```
  sp.TNat
  ```

Voting strategies

Voting strategies are generic contract that can be injected in the main DAO contract (by the DAO itself). They must fulfill the following interface:

Interface

- `start`: (who: main DAO contract) Request a new vote to start.

  ```
  sp.TNat (total available voters right now in the FA2 contract)
  ```

- `vote`: (who: main DAO contract) Send the vote of a AngryTeenagers owners

  ```
  sp.TRecord(votes=sp.TNat, address=sp.TAddress, vote_value=sp.TNat, vote_id=sp.TNat)
  ```

- `end`: (who: main DAO contract) End a vote

  ```
  sp.TNat (vote id)
  ```

Off-chain views

- `get_number_of_historical_outcomes:` Return the number of historical votes outcome (vote in progress is not counted).

  ```
  sp.TUnit
  ```

  Return:

  ```
  sp.TNat
  ```

- `get_historical_outcome_data`: Get the historical data of one particular vote.

  ```
  sp.TNat
  ```

  Return:

  ```
  sp.TBigMap(sp.TNat, sp.TRecord(
    outcome = sp.TNat,
    poll_data = PollType.POLL_TYPE
  ))
  ```

- `get_current_poll_data`: Get the current data of the vote that is in progress if it exists.

  ```
  sp.TUnit
  ```

  Return:

  ```
  sp.TRecord(
                  proposal=Proposal.PROPOSAL_TYPE,
                  proposal_id=sp.TNat,
                  author=sp.TAddress,
                  voting_strategy_address=sp.TAddress,
                  voting_id=sp.TNat,
                  snapshot_block=sp.TNat
  )
  ```

- `get_contract_state`: Get the state of the contract.

  ```
  sp.TUnit
  ```

  Return:

  ```
  sp.TNat
  ```

- `get_voter_history`: Get the voter info (vote value, level of the vote and number of votes) per voters per vote id

  ```
  sp.TRecord(address=sp.TAddress, vote_id=spTNat)
  ```

  Return:

  ```
  sp.TRecord(
    vote_value=sp.TNat,
    level=sp.TNat,
    votes=sp.TNat,
  )
  ```

Default Voting Strategies

By default, the main DAO component comes with two voting strategies. These two voting strategies implement the generic interface describe above plus some other functionality describes above.

Majority voting strategy (0)

The majority voting strategy uses a voting system where the user can vote **Yay**, **Nay** or **Abstain**. For the vote to pass the number of total voters (included abstain votes) shall reached the quorum and the number of **Yay** votes shall reach the supermajority percentage defined in the governance parameters (see below).

The governance parameters can defined a fixed quorum (a fixed percentage value that is used at each vote) or a dynamic one where the quorum is updated at the end of a vote.

State Machine

```
#### NONE (0)
No vote in progress
start → IN_PROGRESS

#### IN_PROGRESS (1)
A vote is in progress
end → NONE
```

Governance parameters

Note that the governance parameters cannot be changed however a new voting strategies can be injected.

```
sp.TRecord(
  vote_delay_blocks = sp.TNat,
  vote_length_blocks = sp.TNat,
  supermajority_threshold_pertenmill = sp.TNat,
  fixed_quorum_pertenmill = sp.TNat,
  fixed_quorum = sp.TBool,
  quorum_cap_pertenmill = sp.TRecord(
    lower = sp.TNat,
    upper = sp.TNat
  )
)
```

Extended interface

- `set_poll_leader`: (who: admin) Set the main DAO contract address. This function can only be called one time when the contracts are deployed.

  ```
  sp.TAddress
  ```

- `mutez_transfer`: (who: admin) Allow the admin to remove XTZ fund from the contract if they have been sent by mistake.

  ```
  sp.TRecord(destination=sp.TAddress, amount=sp.TMutez)
  ```

- `set_next_admin`: (who: admin) Set the next admin. The next admin is not admin yet. It will be come admin by calling the entrypoint `validate_new_admin`.

  ```
  sp.TAddress
  ```

- `validate_next_admin` (who: “next_admin”) Validate the new admin. `set_next_admin` shall be called by the old admin prior to this function.

Opt out voting strategy (1)

The opt voting strategies defines a vote where the user can only vote **Nay** to reject the proposal during the *phase 1*. If the number of **Nay** votes doesn't reach a percentage defined in the governance parameters (10% for the AngryTeenagers project), the proposal is passed. If not, *phase 2* is entered and a majority voting contract is used to conduct a majority vote with a fixed quorum and a supermajority quorum (with a fixed quorum of 25% of the current total possible voters and 80% of supermajority).

State machine

```
#### NONE (0)
No vote in progress
start -> PHASE_1_OPT_OUT

#### PHASE_1_OPT_OUT (1)
The phase 1 is in progress
end -> STARTING_PHASE_2 or NONE

#### STARTING_PHASE_2 (2)
The phase 1 is rejected. Request the associated majority contract
to open a majority vote and enter phase 2
propose_callback ->

#### PHASE_2_MAJORITY (3)
The phase 2 is in progress
end -> ENDING_PHASE_2

#### ENDING_PHASE_2 (4)
The majority contract is request to end the phase 2
end_callback -> NONE
```

Governance parameters

Note that the governance parameters cannot be changed however a new voting strategies can be injected.

```
sp.TRecord(
  vote_delay_blocks = sp.TNat,
  vote_length_blocks = sp.TNat,
  objection_threshold_pertenmill = sp.TNat
)
```

Extended interface

- `set_poll_leader`: (who: admin) Set the main DAO contract address. This function can only be called one time when the contracts are deployed.

  ```
  sp.TAddress
  ```

- `mutez_transfer`: (who: admin) Allow the admin to remove XTZ fund from the contract if they have been sent by mistake.

  ```
  sp.TRecord(destination=sp.TAddress, amount=sp.TMutez)
  ```

- `set_phase_2_contract`: (who: admin) Set the majority voting strategy contract for the *phase 2*. This function can only be called one time when the contracts are deployed.