Zebrango Is a collection of games that is built by the community for the community.

introduction

Each month there will be an event where people can suggest or build new games that can be added to the protocol, and they can vote on each other‘s proposals. The winner will earn a prize that is collected from all the games that has been added to the protocol until the start of the events, plus the event collected participation fees, which is 0.1 Avax per participant or voter.

events

Each month, the operator will execute a function that will end the current event and start a new event.When an event ends, the winner will be chosen based on the number of votes. If multiple participants the same number of votes (highest), they all win and the prize will split evenly among them. When the event starts, the contract will claim the fees collected from all the games added to the protocol until the present time.

voters: can only vote once on the proposal
participants: can only participate once in the event

proposals

Proposals are any game (idea/ suggestion) that can be realized into a smart contract game and can be added to the protocol. Example goes from coin flip to the harder spectrum to chess and poker.

The proposals have to have
The game should be completley onChain, and decentralized.
You can use trusted Oracles like Chainlink

GitHub link: you don‘t have to code, but you need to include:

- the structure of the game
- how the game can be realized considering the gas costs
- the logic and the rewards.

adding a game to the protocol

Each winning proposal has a chance to be added to the platform, after being reviewed and developed by the Team.

contract structure: Zebrango-Gov.

Zebrango Core contract handles the Events on Zebrango, here are the most important functions:

submit proposals to the current event

```
function addProp (uint256 _eventId) external payable whenNotPaused
```

voting on submitted proposal

```
function voteOnProposal (uint256 _eventId, uint256 _propId) external payable whenNotPaused
```
// eventID = currentEvent

adding game to the platform

```
function addGame (address _game)external onlyAdmin whenNotPaused
```

executeEvent this function can only be called from the operator address
this function ends the current event, starts the next one.

```
function executeEvent()external onlyOperator whenNotPaused
```

internal functions:

start the next Event and lock the current one:
this function triggers when the operator executes the executeEvent.

```
function _safeLockEvent(uint256 _eventID) internal
```
// eventID = currentround

```
function _safeStartEvent(uint256 _eventID)internal
```
// eventID = currentround + 1

withdrawing the fees collected from all the games collection:
this function triggers when the Event starts.

```
function _withdrowGames(uint256 _eventID) internal
```

set the winners for the current Event:
this function triggers when the Event Ends.

```
function _setWinners() internal
```

what are we are working on in the future?

- second layer voting system, for developing the winning of the proposed games.
  There will be specifc game events, where developers can code, and the community can judge their code as in the first layer voting, based on satisfying the requirements, security of the code and fairness.
  After auditing the game code, it will be pushed into the game collection and the creator of the idea plus the developers can earn money from this game if the community plays it.

- making the smart contracts upgradeable

- updating the user interface