Team: Any GrpName
Team Members: Abhishek Karnati,Lucas Foo ,Priyan Rajamohan, Samuel Suther, Umar Moiz

Introduction

Contract Address: 0x3D8fb48AC24E171F350e626646cbD866B31600C5
Frontend Address: http://ee4032.lucasfoosg.com/
Our smart contract implements a decentralized charity organization on the ethereum blockchain, whereby people in need of funding can submit requests and justification for a certain amount of funds, and donors can donate to be eligible to vote for which request would be receiving the funds from the organization in the next payout.

Current charity organizations are all centralized bodies, where the organization itself decides the allocation of donations. We believe that donors should have a say in where funds go to, and blockchain can provide an alternative solution. Blockchain for the purpose of charity can increase trust and transparency, reduce costs that traditional charity organizations have such as administrative costs, and increase the proportion of donated money that recipients receive.

Smart Contract

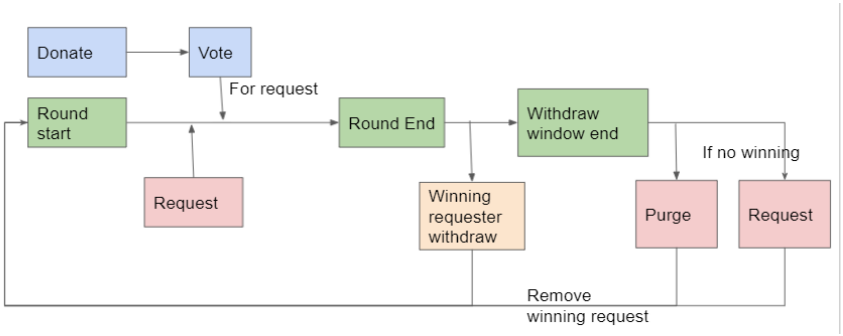

Image 1: Flowchart for smart contract

Our smart contract follows a behaviour as shown in Image 1, where at any point in time, donors would be able to donate and requesters would be able to submit new requests..

At the end of each voting round (after 24 hours current deployment, 7 days in real world deployment), a winning request is selected based on the number of votes from donors, and that request would be able to withdraw from the contract. If the winner does not withdraw after a certain period of time after the round ends (12 hours in current deployment, 1 day in real life deployment), we allow that request to be removed by others(purging). If there are no winning requests e.g when no requests at all, the round starts when someone posts a new request.

Each donor is also only allowed to vote once, i.e when a request is removed, so are their votes.

The difference in time for real world deployment and our deployed contract for this module is to encourage other people taking the module to interact with our contract and see all its features.

Design choices

The critical aim while beginning the development phase was to create a system which maintains fairness and employs trust as its core foundation. Some of the key design choices made are described below.
Selecting a charity request - With a possibly large pool of requests, the system was developed to assign priorities. This priority system helped to validate the reliability of requests and provided serious cases with an opportunity to be ahead of the queue.
Time frame for voting - We ensured that there was adequate time for the most urgent and reliable requests to gain priority from voting. In the current implementation, the voting process has been set to be 24 hrs and withdrawal as 12 hrs. Practically, we aim to set one week for voting and 24 hrs for withdrawal.
Test Driven Development - We implemented unit and functional testing to make sure that the building blocks of the application are stable and can be pieced together seamlessly. Furthemore, we followed a hybrid of agile and waterfall software development to deliver a functional product in time.
Intuitive Interface - We developed an easy to use front end for comfortable interaction with ethereum considering that the complexities of the underlying system can be difficult to understand for laymen. The restrictions were defined clearly on both the front-end and back end, to ensure that they are not bypassed while interacting with the blockchain.

Design challenges

The main challenge we faced was the inability to create a self aware finite state machine that updates with time to generate new winning requests and convert winning requests to stale requests exactly after 24 hours. This limitation arose because the code in the smart contract only gets executed when someone interacts with it. In the event where no one interacts, a winning request may indefinitely remain as a winning request until someone interacts with the smart contract to change that state. To overcome this, whenever someone withdraws or purges, we determine the time elapsed since the winning request and assign the request the appropriate state or purge it if it has exceeded 36 hours. In this way, if someone tries to donate to the contract after the current winning request passes the 24 hour mark, it will not receive the new donation. The new donations will be stored in the contract address for the next winning request. Through this we transformed our challenge to our benefit as when our application faces a lull period of no user interactions, winning requests will not get generated and get ignored. As such, all winning requests are guaranteed to get some attention. We also display the request that's next in line to win so that the requester will be incentivised to purge stale requests in order to become the next winning requester.

Possible extensions

Following are some of the extensions we were considering for this smart contract:

1. Multiple Campaigns: Currently our solution only allows for one winning request but this contract could be extended to allow for multiple winning requests at the same time. This way the decentralized system could parallelize multiple campaigns.
2. Change Vote: In the current solution, once the donor casts their vote, their vote cannot be changed. One possible extension is allowing donors to change their votes during the voting phase in case they change their mind.
3. Improved UI: In future iterations of our product, the application will be revamped to allow for a cleaner user interface based on the recommendations/feedback provided from our testers.
4. Improved proof verification system: Come up with a structured way for people to submit proofs instead of the current system so that it is easy for users to read through the proofs for multiple requests and easily verify their authenticity.

Feedback Received

Feedback	Proposed Solution
"hard to verify the proof and reason for the cause"	Verification of proof would be provided in future iterations of the product, how we intend to go about resolving this issue is by allowing a display proof feature that users may view prior to voting
"*GUI maybe you could put a seperate page for all the requests as they make page looks kinda messy right now "*	In future iterations of the product, we intend to display all requests in a table-like format to allow for better readability
" *a function can be added where people can also donate to individual requests or indicate their preference and get their money back if their preference is not the winning one as not everyone is happy to donate to everyone"*	In real life, when donations are made to a charity organization, the organization decides the request to be prioritized and funded. Similarly, in our solution, the donors themselves act as the organization and choose the request to be funded. Furthermore, by giving the power of choosing the winning request to the donors, it acts as a form of verification as the donors would reach a consensus to deprioritize the illegitimate requests based on the quality of proof provided.
"This might be a possible cyber attack that may take advantage of this system. Perhaps a way to resolve this would be to increase the cost of a vote?"	We felt 0.01 eth as the minimum donation amount per address was a nice balance between keeping the donation requirements low to prevent turning away donors and preventing spam accounts (fraud).

Change Log

[1.1.1] - 2021-09-13

Bug: Fixed order of purge request before purge donor bug

[1.1.1] - 2021-09-14

Bug: Fixed issue with wrong date format