RabbitHole Take-home

Suki Yang

Problem Brief

The problem we are solving is, given all the on-chain data, what are the key insights we can derive, what are the recommendations we can draw from users' on-chain data.

The first most important thing is what are the questions to ask, it's a matter of finding what's important when the problems are ambiguous. I identified the following important questions to answer:

1. What is the distribution of users of quests? Is it more concentrated few users doing a lot of quests or distributed users doing different quests?
2. What is the total active users of quests? Daily, monthly, weekly, are they increasing or decreasing? What are the factors that contribute towards that?
3. What are the popular types of protocols? Tokens that are most frequently used by users?
4. Given that we want to measure success for all quests created, what are the metrics to measure for success of a quest? What are the factors that contribute towards the success of the quest?

Now the work is simple, write code and think creatively about factors that could contribute to the success of quests, then use rigid statistical methods to test our hypothesis.

My programming work consists of two components: SQL queries for acquiring, aggregate and analyze data on-chain; Python modeling for more advanced analysis. The coding part is under SQL Queries and Python Code.

My presentation is in the RabbitHold.pptx in the folder.

My Dune dashboard is in this link: https://dune.com/sukki/rabbithole-v2

Executive Summary

1. We have total of 239,978 participations.
2. Most popular protocol is ERC20 tokens.
3. We have total 44,569 wallets participated.
4. 80% of quest participants are on optimism.
5. We see spike of users on Feb 19th and Mar 28th, which correlated to general crypto market movement - it's worth knowing that everything is correlated to how the market moves in crypto. The spike of success rate happened when market regime turns from bear to bull.

6. What factors lead to participation ratio of quests:
    a. Conclusion: reward, duration and filled_percentage are three factors that can predict participation ratio of quests very accurately.
    b. **Linear model**: We ran linear regression on the aggregated datasets. The results are very good, with 0.013 MSE and 0.89 R square. The results show that we would the x parameters show strong linear correlation to the participation percentage, therefore if we want to increate participation percentrage, we should increate our X factors, which are: reward, duration, filled_percentage. The MSE is very low although not insignificant. We won't rune the model because for linear models there are very few parameters we can tune.
    c. **Non-linear model:** When we use reward, duration, filled_percentage to predict the participation ratio with random forest, the prediction results show amazing output. The result of random forest model is really good, although we need to be cautious because our dataset has only 100+ data and random forest can easily overfit.
    d. **Gradient Boosting Tree:** We tried another gradient boosting tree model which also returned great prediction results. We are trying this new model, because this model is not allergic to extreme values or nan value or 0s.
7. We discovered that the distribution of users claiming quests reward is more concentrated instead of distributed, to incentivized a more distributed user base, there are a few things we can do:
    a. Set customized rewards curve to incentivize new users more to participate.
    b. Apply for a grant program with the proposal of: if a quest user invite other people to join the quest, he/she gets additional rewards. We should test out that idea with a few quest, once we have data proving it's driving growth, we can toll out as a feature to acquire more users to participate.

Recommmendations

1. **Quest specific metrics**: we can collect more quest specific metric to measure if participation is linked to specific quest features, so we can change parameters to optimize for quest success.
2. **User wallet labeling**: This is a very important direction of research for us. Based on any user wallet's historical data, we are able to identify their charactoristics. Such as a wallet that uses DEX on a regular bases, is more likely to participate in DeFi related quests. Examples of labels that might be helpful includes: NFT, DeFi, CeFi, gaming, gambling, money laundering, Tornado cash, etc.
    a. We can also consider using third party application for this information so we don't need to label everything ourselves, such as clain, crystal, elliptic, I have experience building crypto compliance tool so I know how much effort it needs to have correct wallet labels.
3. **Long term monitoring of users' behavior**:
    a. A lot of times, the incentive to get users fullfil quests is to have them on the platform long-term. It's only possible to see long term effect if we monitor it.

    b. Keep track of daily/weekly/monthly metrics of:
        i. Percentage of users who claim the rewards using the product (frequency, amount, -> engagement metrics)
4. Data shows that only 3% of total participations are on Ethereum mainnet, although after looking into mainnet transaction data, the reason for that is gas fee being too high that it doesn't make sense to fullfil the quest. One suggestion is for quests on Mainnet, we run experiments to see how much more rewards we need to give out to see a significant increase in participation. Experiments could be designed like:
    a. We launch three quests at same time of different week while we control other factors such as sentiment, how many total spots, duration, type of tasks. The only differentiation is the token reward.
    b. We want to test out the bliss spot of where the effect of claim ratio reach a plateau as rewards go up. We are optimizing for higher claim ratio with ideally minimal cost of the quest.
5. For more popular protocols, launch more quest but with less rewards (they have a larger user base); for smaller protocols, launch quests with more rewards(protocol needs to compensate for their lack of users).
6. Increase quest duration. The longer a quest last, the higher percent it gets completed.

Future Plans

There are some potential hypothesis and experiments we can run to help boost success of quests:

1. Correlation to crypto market sentiment. Is success rate of quests correlated to the general crypto market sentiment? This would mean when market is hot, price goes up, more users participate in quests. This insight will enable us to l**aunch more quests** when the price is up and market is hot. And when market is down, we can test if giving out more rewards would compensate for that.
2. Gather more geographical data of quest participants.
3. Gather more data on clustering users' wallets to decide if we have the problem of multiple wallets being the same person.

Presentation

See the attached presentation slides.

Questions

If you have any questions, feel free to reach out to Suki at suki@sukiyang.com

Appendix

SQL Queries

The following secion is SQL codes that help answer the questions, and explanations of the code. I ran all SQL code on Dune analytics, and the final dashboards and analysis are shown here:
https://dune.com/sukki/rabbithole-v2

Each code block has its own title and explanation of what the code does.

Token most frequently used by users

```
WITH quests AS (
    SELECT
        blockchain,
        contractAddress AS quest_address,
        questId AS quest_id,
        rewardTokenAddress AS reward_address
    FROM (
        SELECT 'ethereum' AS blockchain, * FROM
rabbithole_ethereum.QuestFactory_evt_QuestCreated
        UNION ALL
        SELECT 'optimism' AS blockchain, * FROM
rabbithole_optimism.QuestFactory_evt_QuestCreated
        UNION ALL
        SELECT 'polygon' AS blockchain, * FROM
rabbithole_polygon.QuestFactory_evt_QuestCreated
    )
),
receipts AS (
    SELECT
        blockchain,
        questId AS quest_id,
        questAddress AS quest_address,
        COUNT(DISTINCT recipient) AS participants
    FROM (
        SELECT 'ethereum' AS blockchain, * FROM
rabbithole_ethereum.QuestFactory_evt_ReceiptMinted
        UNION ALL
        SELECT 'optimism' AS blockchain, * FROM
rabbithole_optimism.QuestFactory_evt_ReceiptMinted
        UNION ALL
```

```
        SELECT 'polygon' AS blockchain, * FROM
rabbithole_polygon.QuestFactory_evt_ReceiptMinted
    )
    GROUP BY 1,2,3
),
popular_tokens AS (
    SELECT
        q.reward_address AS token_address,
        SUM(r.participants) AS total_participants
    FROM quests q
    JOIN receipts r ON q.blockchain = r.blockchain AND q.quest_address = r.quest_address
    GROUP BY q.reward_address
)
SELECT token_address, total_participants
FROM popular_tokens
ORDER BY total_participants DESC;
```

Most Popular Protocol

```
WITH quests AS (
    SELECT
        blockchain,
        contractAddress AS quest_address,
        questId AS quest_id,
        contractType AS quest_type
    FROM (
        SELECT 'ethereum' AS blockchain, * FROM
rabbithole_ethereum.QuestFactory_evt_QuestCreated
        UNION ALL
        SELECT 'optimism' AS blockchain, * FROM
rabbithole_optimism.QuestFactory_evt_QuestCreated
        UNION ALL
        SELECT 'polygon' AS blockchain, * FROM
rabbithole_polygon.QuestFactory_evt_QuestCreated
    )
),
receipts AS (
    SELECT
        blockchain,
        questId AS quest_id,
        questAddress AS quest_address,
        COUNT(DISTINCT recipient) AS participants
```

```
    FROM (
        SELECT 'ethereum' AS blockchain, * FROM
rabbithole_ethereum.QuestFactory_evt_ReceiptMinted
        UNION ALL
        SELECT 'optimism' AS blockchain, * FROM
rabbithole_optimism.QuestFactory_evt_ReceiptMinted
        UNION ALL
        SELECT 'polygon' AS blockchain, * FROM
rabbithole_polygon.QuestFactory_evt_ReceiptMinted
    )
    GROUP BY 1,2,3
),

popular_protocols AS (
    SELECT
        q.quest_type AS protocol,
        SUM(r.participants) AS total_participants
    FROM quests q
    JOIN receipts r ON q.blockchain = r.blockchain AND q.quest_address = r.quest_address
    GROUP BY q.quest_type
)
SELECT protocol, total_participants
FROM popular_protocols
ORDER BY total_participants DESC;
```

Total Wallets Participated

```
select
  count(distinct to_) as total_participate_wallet
from
  rabbithole_optimism.RabbitHoleReceipt_call_mint
where
  call_success = true
```

Total quests created

```
SELECT 'optimism' as blockchain, *
FROM rabbithole_optimism.QuestFactory_evt_QuestCreated qcr
```

Quest participants breakdown

```
WITH quests AS (
    SELECT
        blockchain,
        contractAddress as quest_address,
        questId as quest_id,
        contractType as quest_type,
        from_unixtime(cast(startTime as decimal)) as start_time,
        from_unixtime(cast(endTime as decimal)) as end_time,
        cast(rewardAmountOrTokenId as double) as reward_amount_or_token_id,
        rewardTokenAddress as reward_address,
        cast(totalParticipants as bigint) as max_participants
    FROM (
        SELECT 'ethereum' as blockchain, * FROM
rabbithole_ethereum.QuestFactory_evt_QuestCreated
        UNION ALL
        SELECT 'optimism' as blockchain, * FROM
rabbithole_optimism.QuestFactory_evt_QuestCreated
        UNION ALL
        SELECT 'polygon' as blockchain, * FROM
rabbithole_polygon.QuestFactory_evt_QuestCreated
    )
),
receipts AS (
    SELECT
        blockchain,
        questId as quest_id,
        questAddress as quest_address,
        count(distinct recipient) as participants
    FROM (
        SELECT 'ethereum' as blockchain, * FROM
rabbithole_ethereum.QuestFactory_evt_ReceiptMinted
        UNION ALL
        SELECT 'optimism' as blockchain, * FROM
rabbithole_optimism.QuestFactory_evt_ReceiptMinted
        UNION ALL
        SELECT 'polygon' as blockchain, * FROM
rabbithole_polygon.QuestFactory_evt_ReceiptMinted
    )
    GROUP BY 1,2,3
),
quest_participation AS (
    SELECT
```

```
        q.blockchain,
        q.quest_type,
        q.quest_id,
        q.quest_address,
        r.participants,
        q.max_participants,
        r.participants / q.max_participants * 100 AS participation_rate
    FROM quests q
    JOIN receipts r ON q.blockchain = r.blockchain AND q.quest_address = r.quest_address
AND q.quest_id = r.quest_id
)
SELECT
    blockchain,
    quest_type,
    COUNT(*) AS num_quests,
    SUM(participants) AS total_participants,
    AVG(participants) AS avg_participants,
    AVG(participation_rate) AS avg_participation_rate
FROM quest_participation
GROUP BY blockchain, quest_type
ORDER BY blockchain, avg_participation_rate DESC;
```

User time series

```
with source as (
    select 'ethereum' as blockchain, account, evt_block_time
    from rabbithole_ethereum.Erc20Quest_evt_Claimed
    union all
    select 'optimism' as blockchain, account, evt_block_time
    from rabbithole_optimism.Erc20Quest_evt_Claimed
    union all
    select 'polygon' as blockchain, account, evt_block_time
    from rabbithole_polygon.Erc20Quest_evt_Claimed
    union all
    select 'ethereum' as blockchain, recipient as account, evt_block_time
    from rabbithole_ethereum.QuestFactory_evt_ReceiptMinted
    union all
    select 'optimism' as blockchain, recipient as account, evt_block_time
    from rabbithole_optimism.QuestFactory_evt_ReceiptMinted
    union all
    select 'polygon' as blockchain, recipient as account, evt_block_time
    from rabbithole_polygon.QuestFactory_evt_ReceiptMinted
```

```
),

claimers as (
select
    date_trunc('day', evt_block_time) as day,
    count(distinct account) as unique_claimers
from source
group by 1
),

first_claimers as (
    select
    first_day,
    count(*) as first_claimers
from (
    select account,  date_trunc('day', min(evt_block_time)) as first_day
    from source
    group by 1
    )
group by 1
),

days as (
    select * from unnest(sequence(date('2023-02-15'), date(date_trunc('day',now())), interval '1'
day)) as foo(day)
)

select
cast(d.day as timestamp) as day
,coalesce(unique_claimers,0) as daily_claimers
,coalesce(first_claimers,0) as new_claimers
,coalesce(unique_claimers,0)  - coalesce(first_claimers,0) as recurring_claimers
,sum(coalesce(first_claimers,0)) over (order by d.day asc) as total_unique_claimers
from days d
left join claimers c
on d.day = c.day
left join first_claimers c2
on d.day = c2.first_day
order by d.day desc
```

Total claimed

```
SELECT 'optimism' as blockchain, *
FROM rabbithole_optimism.Erc20Quest_evt_Claimed qcr
```

Quest participants

```
WITH quests AS (
    SELECT
        blockchain,
        contractAddress as quest_address,
        questId as quest_id,
        contractType as quest_type,
        from_unixtime(cast(startTime as decimal)) as start_time,
        from_unixtime(cast(endTime as decimal)) as end_time,
        cast(rewardAmountOrTokenId as double) as reward_amount_or_token_id,
        rewardTokenAddress as reward_address,
        cast(totalParticipants as bigint) as max_participants
    FROM (
        SELECT 'ethereum' as blockchain, * FROM
rabbithole_ethereum.QuestFactory_evt_QuestCreated
        UNION ALL
        SELECT 'optimism' as blockchain, * FROM
rabbithole_optimism.QuestFactory_evt_QuestCreated
        UNION ALL
        SELECT 'polygon' as blockchain, * FROM
rabbithole_polygon.QuestFactory_evt_QuestCreated
    )
),
receipts AS (
    SELECT
        blockchain,
        questId as quest_id,
        questAddress as quest_address,
        count(distinct recipient) as participants
    FROM (
        SELECT 'ethereum' as blockchain, * FROM
rabbithole_ethereum.QuestFactory_evt_ReceiptMinted
        UNION ALL
        SELECT 'optimism' as blockchain, * FROM
rabbithole_optimism.QuestFactory_evt_ReceiptMinted
        UNION ALL
```

```
        SELECT 'polygon' as blockchain, * FROM
rabbithole_polygon.QuestFactory_evt_ReceiptMinted
    )
    GROUP BY 1,2,3
),
quest_participation AS (
    SELECT
        q.blockchain,
        q.quest_type,
        q.quest_id,
        q.quest_address,
        r.participants,
        q.max_participants,
        r.participants / q.max_participants * 100 AS participation_rate
    FROM quests q
    JOIN receipts r ON q.blockchain = r.blockchain AND q.quest_address = r.quest_address
AND q.quest_id = r.quest_id
)
SELECT
    blockchain,
    quest_type,
    COUNT(*) AS num_quests,
    SUM(participants) AS total_participants,
    AVG(participants) AS avg_participants,
    AVG(participation_rate) AS avg_participation_rate
FROM quest_participation
GROUP BY blockchain, quest_type
ORDER BY blockchain, avg_participation_rate DESC;
```

Token most frequently used by users

```
WITH quests AS (
    SELECT
        blockchain,
        contractAddress AS quest_address,
        questId AS quest_id,
        rewardTokenAddress AS reward_address
    FROM (
        SELECT 'ethereum' AS blockchain, * FROM
rabbithole_ethereum.QuestFactory_evt_QuestCreated
        UNION ALL
```

```
        SELECT 'optimism' AS blockchain, * FROM
rabbithole_optimism.QuestFactory_evt_QuestCreated
        UNION ALL
        SELECT 'polygon' AS blockchain, * FROM
rabbithole_polygon.QuestFactory_evt_QuestCreated
    )
),
receipts AS (
    SELECT
        blockchain,
        questId AS quest_id,
        questAddress AS quest_address,
        COUNT(DISTINCT recipient) AS participants
    FROM (
        SELECT 'ethereum' AS blockchain, * FROM
rabbithole_ethereum.QuestFactory_evt_ReceiptMinted
        UNION ALL
        SELECT 'optimism' AS blockchain, * FROM
rabbithole_optimism.QuestFactory_evt_ReceiptMinted
        UNION ALL
        SELECT 'polygon' AS blockchain, * FROM
rabbithole_polygon.QuestFactory_evt_ReceiptMinted
    )
    GROUP BY 1,2,3
),
popular_tokens AS (
    SELECT
        q.reward_address AS token_address,
        SUM(r.participants) AS total_participants
    FROM quests q
    JOIN receipts r ON q.blockchain = r.blockchain AND q.quest_address = r.quest_address
    GROUP BY q.reward_address
)
SELECT token_address, total_participants
FROM popular_tokens
ORDER BY total_participants DESC;
```

Success Rate of Queries

```
WITH quest_claims AS (
    SELECT
        contract_address AS quest_id,
```

```
    COUNT(*) AS total_claimed
  FROM
    rabbithole_optimism.Erc20Quest_evt_Claimed
  GROUP BY
    contract_address
),

quest_success_rates AS (
  SELECT
    q.quest_id,
    p.total_participants,
    q.total_claimed,
    (q.total_claimed * 100.0) / p.total_participants AS success_rate
  FROM
    quest_claims q
  JOIN
    (SELECT contract_address, COUNT(DISTINCT account) AS total_participants
     FROM rabbithole_optimism.Erc20Quest_evt_Claimed
     GROUP BY contract_address) p
  ON q.quest_id = p.contract_address
)

SELECT
  quest_id,
  total_participants,
  total_claimed,
  success_rate
FROM
  quest_success_rates
ORDER BY
  success_rate DESC;
```

Success Rate of Queries

```
WITH quest_claims AS (
  SELECT
    contract_address AS quest_id,
    COUNT(*) AS total_claimed
  FROM
    rabbithole_optimism.Erc20Quest_evt_Claimed
  GROUP BY
    contract_address
```

```
),

quest_success_rates AS (
    SELECT
        q.quest_id,
        p.total_participants,
        q.total_claimed,
        (q.total_claimed * 100.0) / p.total_participants AS success_rate
    FROM
        quest_claims q
    JOIN
        (SELECT contract_address, COUNT(DISTINCT account) AS total_participants
         FROM rabbithole_optimism.Erc20Quest_evt_Claimed
         GROUP BY contract_address) p
    ON q.quest_id = p.contract_address
)

SELECT
    quest_id,
    total_participants,
    total_claimed,
    success_rate
FROM
    quest_success_rates
ORDER BY
    success_rate DESC;
```

Python Code

See attached RabbitHole.ipynb.