



Monopoly GO!

Economy PM Simulation Test

James Boyle



Contents

1. Model Explanation
 - a. Setup, execution, assumptions
2. Model Analysis
 - a. Distribution of card rarities
 - b. Flaws in game design and monetization strategy
3. Recommendations
 - a. Retune album design or parameters for better balance, player satisfaction, retention, and monetization

Model Explanation



Model Explanation - Setup

Setup Overview

- Used Python to generate 10,000 player sample based on parameters given
 - Chose python because I wanted to track specific card duplicates over the event duration
 - Reference file: james_boyle_monopoly_go_simulation.py
- Created a .csv that I uploaded as a table to BigQuery
 - Output from python (file available upon request): james_boyle_monopoly_db.csv
- Allowed me to run SQL for analysis purposes, which was useful for making tables and graphs
 - Reference [link](#)



Model Explanation - Execution

- Import csv library to create .csv for SQL database, and numpy library for random choice function
- Define dictionaries for the main parameters of the simulation that will be used to generate sample card packs for each player
 - Will be useful levers for retuning the model

```
import csv
import numpy as np

duration = 50 # days

daily_pack_acquisition_rates = {
    "common_pack": 4,
    "uncommon_pack": 2,
    "rare_pack": 1,
}

cards_per_pack = {
    "common_pack": 2,
    "uncommon_pack": 3,
    "rare_pack": 4,
}

pack_odds = {
    "common_pack": [0.70, 0.25, 0.05],
    "uncommon_pack": [0.55, 0.35, 0.10],
    "rare_pack": [0.30, 0.40, 0.30],
}

card_rarities = {
    "common_cards": 100,
    "uncommon_cards": 60,
    "rare_cards": 25,
}
```



Model Explanation - Execution

- Uses previously defined dictionaries to create list of cards and their rarities
- Then creates functions to get the rarity of the card based on the card's number
 - 0 to 99 = common
 - 100 to 159 = uncommon
 - 160 to 184 = rare

```
cards = list(range(sum(card_rarities.values())))

def rarity(card_number):
    if 0 <= card_number < 100:
        return 0
    elif 100 <= card_number < 160:
        return 1
    elif card_number < 185:
        return 2
    else:
        raise ValueError()

def card_range(rarity):
    if rarity == 0:
        return cards[:100]
    elif rarity == 1:
        return cards[100:160]
    elif rarity == 2:
        return cards[160:185]
    else:
        raise ValueError()

choose_with_replacement = True
```



Model Explanation - Execution

- Generate_pack function creates packs with correct #cards and randomly chooses a card based on the probability of getting a common/uncommon/rare
- One_day function combines the arrays from opening each card pack into one array for that day
- Card_pack_type uses the fact that we know the index of the card in the pack so we can determine the rarity of the pack that it came from

```
def generate_pack(pack_type):  
    packs_to_choose_from = np.random.choice(  
        [0, 1, 2], # Possible card types to choose from  
        size=cards_per_pack[pack_type], # Number of cards to choose  
        replace=choose_with_replacement, # Choose with or without replacement  
        p=pack_odds[pack_type] # Odds of choosing the various types of cards  
    )  
    return [  
        np.random.choice(card_range(card_type))  
        for card_type in packs_to_choose_from  
    ]  
  
def one_day():  
    """Return a set of cards for one player over the course of one day"""  
    return np.concatenate(  
        tuple(  
            generate_pack(pack_type)  
            for pack_type, daily_pack_acquisition_rate in daily_pack_acquisition_rates.items()  
            for _ in range(daily_pack_acquisition_rate)  
        )  
    )  
  
def card_pack_type(card_id):  
    if 0 <= card_id < 8:  
        return "common_pack"  
    elif card_id < 14:  
        return "uncommon_pack"  
    elif card_id < 18:  
        return "rare_pack"  
    else:  
        raise ValueError()
```



Model Explanation - Execution

- One_player function forms array of all the player's cards over the event duration
- Then I generate 10,000 samples with the player_experiences function
- And write that data to a .csv

```
def one_player():
    """Generate of full day's set of cards for one player"""
    return np.vstack(
        tuple(
            one_day()
            for day in range(duration)
        )
    )

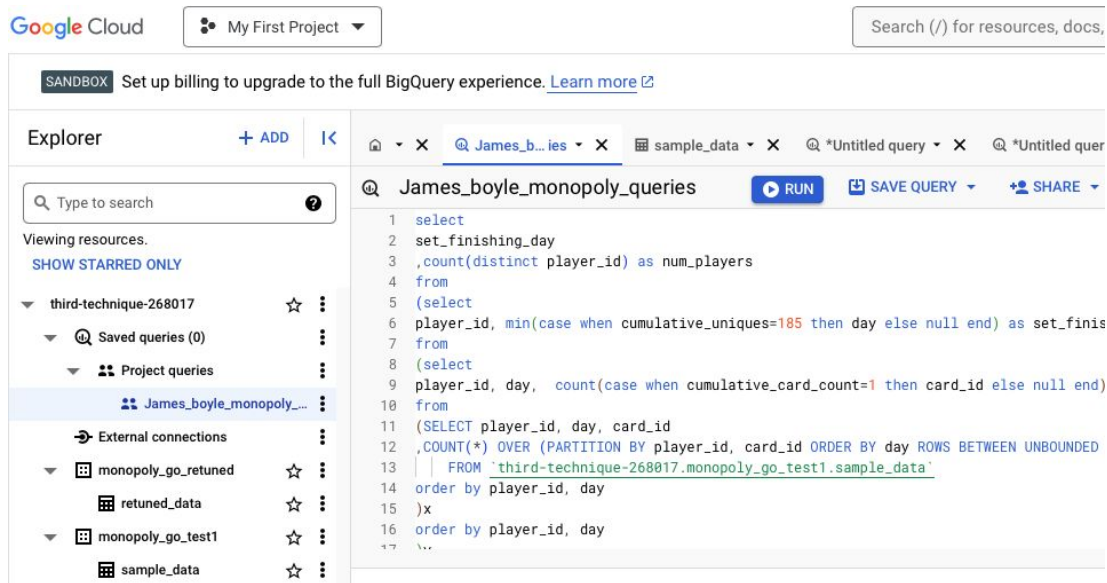
np.random.seed(1234)

n_players = 10_000
player_experiences = [one_player() for player in range(n_players)]

with open("james_boyle_monopoly_db.csv", "w", newline="") as f:
    field_names = ["player_id", "day", "card_pack_type", "card_id", "card_rarity"]
    f.write(",".join(field_names) + "\n")
    writer = csv.writer(f)
    for player_index, player_experience in enumerate(player_experiences):
        player_id = f"player_{player_index+1:06}"
        for day_index, card_set in enumerate(player_experience):
            day = day_index + 1
            for card_index, card_id in enumerate(card_set):
                writer.writerow([player_id, day, card_pack_type(card_index), card_id, rarity(card_id)])
```

Model Explanation - Execution

- I uploaded the .csv to Google Cloud Console
- Then created a database in Google BigQuery



The screenshot displays the Google Cloud BigQuery console interface. At the top, the Google Cloud logo and 'My First Project' are visible. A search bar on the right prompts 'Search (/) for resources, docs, ...'. Below the header, a 'SANDBOX' banner indicates the current environment and offers a link to 'Set up billing to upgrade to the full BigQuery experience. [Learn more](#)'. The main interface is divided into two panels. The left panel, titled 'Explorer', contains a search bar and a list of resources. Under 'third-technique-268017', there are 'Saved queries (0)', 'Project queries', and a highlighted query named 'James_boyle_monopoly_queries'. Below these are 'External connections' and a list of datasets: 'monopoly_go_returned', 'returned_data', 'monopoly_go_test1', and 'sample_data'. The right panel shows the SQL editor for the selected query. The query is a complex SELECT statement with multiple subqueries and a window function. It includes a 'RUN' button, 'SAVE QUERY' option, and a 'SHARE' button. The query text is as follows:

```
1 select
2 set_finishing_day
3 ,count(distinct player_id) as num_players
4 from
5 (select
6 player_id, min(case when cumulative_uniques=185 then day else null end) as set_finis
7 from
8 (select
9 player_id, day, count(case when cumulative_card_count=1 then card_id else null end)
10 from
11 (SELECT player_id, day, card_id
12 ,COUNT(*) OVER (PARTITION BY player_id, card_id ORDER BY day ROWS BETWEEN UNBOUNDED
13 | FROM `third-technique-268017.monopoly_go_test1.sample_data`
14 order by player_id, day
15 )x
16 order by player_id, day
17 ..
```



Model Explanation - Assumptions

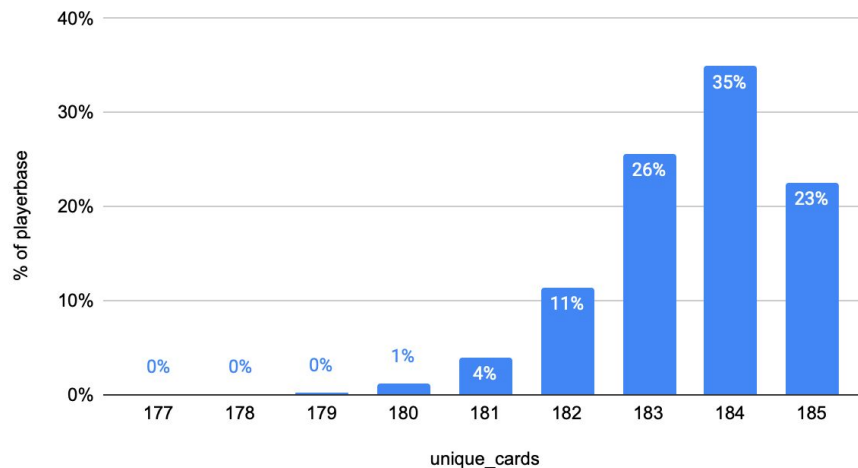
- Explicitly not modeling real-world behavior:
 - Entire playerbase is getting all the packs per day
 - Not modeling typical player earn rates based on their sessions per day or engagement with faucet features
 - This would wildly affect card acquisition variance, and therefore, tuning

Model Analysis

Model Analysis - Simulation Outcomes

- First looked into how many of the event cards players were getting by the end
- Very tight distribution of cards, but only 23% completing the entire set
- Opportunity: players who are missing a few cards could be monetized to complete the collection

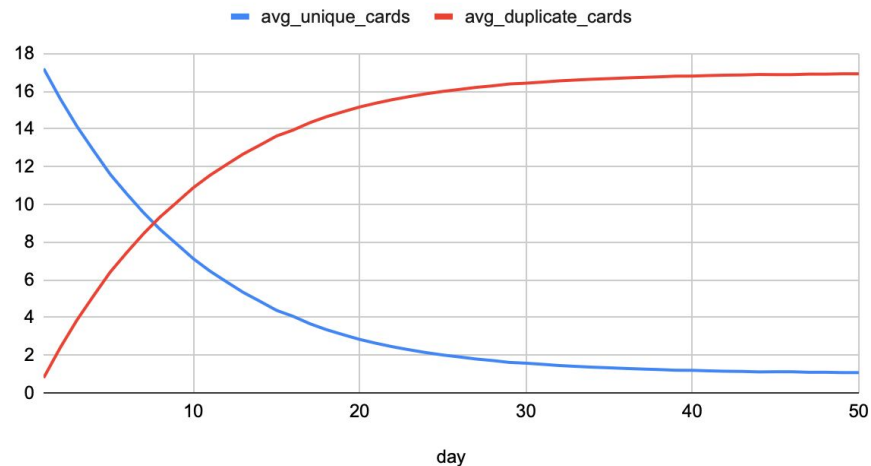
% of Players with Unique Cards Obtained by End of Event



Model Analysis - Simulation Outcomes

- Since there are 900 cards earned by each player over the event, and most are getting close to the complete set of 185, there are a lot of duplicates
- Around day 7, players are already getting as many duplicates and unique cards
- Around day 25, they are getting only 2 unique cards on average
- Represents a problem for retention as players likely to disengage with event if mostly earning duplicates
- Difficult to solve with current event structure

Average Unique and Duplicate Cards per Day





Model Analysis - Simulation Outcomes

- Looked into if players were completing sets of different rarities
- Disturbingly, the rarer the set, the more likely players are to complete it
- Low set completion rates across the board led to very low completion of the entire set of 185 cards

Set Type	% Finishing Set	% Unfinished Set	Average Set Completion Day
Common Set	52%	48%	43
Uncommon Set	60%	40%	41
Rare Set	74%	26%	37
Complete Set	23%	77%	46



Model Analysis - Flaws in game design and monetization strategy

What stood out to me before analyzing performance:

- Strange that common packs give 2 cards while uncommon gives 3 and rare gives 4
 - Would expect rarer packs to contain same # or fewer cards
- 18 cards per day seems like a lot of cards, especially because 900 cards over the event means lots of duplicates
 - Lower numbers of cards in general might give them more value
- Using same rarity names for cards and packs could cause problems
 - Team/config confusion
 - Player frustration
 - Players could be upset that rare packs gave them common cards
 - Current game gives guaranteed rarer card in rarer packs
 - Player confusion
 - Perhaps have different names for packs with common/uncommon/rare cards inside to solve this problem
 - Current game uses colors and stars to get around this issue



Model Analysis - Flaws in game design and monetization strategy

What stood out after analyzing performance:

- Very early in the event, players will be getting more duplicates than new cards
 - Harms event retention and monetization potential if players expect mostly duplicates
- Rarer sets are being completed at a higher rate than more common sets
 - Rarer cards should have higher perceived value, and thus, be harder to get
 - Players less likely to play or pay to complete common sets
- Low numbers of players are completing sets of any rarity, which could cause dissatisfaction
 - Players would expect to at least come close to completing common sets, and have smaller chance of completing rarer sets

Recommendations



Recommendations - Outlining Goals

First, I would define the goals of the event

- Since it's a branded event, we might want to drive social media posts and virality over monetization
 - This would impact our target for how many players we want to complete the set
 - Might be fine with almost everyone completing the set, as early set completion could drive more social posts versus a world where few players are ever completing a set
 - Don't want to drive negative social media posts for the IP owners where the event was "too hard"
- I'm not modeling any social impact to KPIs here, so I will opt for tuning to drive retention and monetization
- Goals:
 - Rarer sets should be harder to complete than common sets
 - Overall set completion should remain relatively low



Recommendations - Retuning

- Goal: increase common set completion, decrease rarer set completion
 - Boosts retention because they feel satisfaction in completing common sets
 - Increases monetization potential because rarer sets are harder to complete
- Method:
 - Make rarer packs give fewer cards
 - Packs per day
 - Common: 4 -> 3
 - Cards/pack
 - Uncommon: 3 -> 2
 - Rare: 4 -> 1
 - Adjust probabilities for pack types
 - Common: +20%c, -16%u, -4%r
 - Uncommon: -20%c, +25%u, -5%r
 - Rare: -15%c, -15%u, +30%r
 - Increase proportion of rare cards
 - Common: 100 -> 50
 - Uncommon: 60 -> 30
 - Rare: 25 -> 15
 - Relevant file:
james_boyle_monopoly_go_retuning.py



Recommendations - Retuning Outcomes

Original Tuning

Set Type	% Finishing Set	% Unfinished Set	Average Set Completion Day
Common Set	52%	48%	43
Uncommon Set	60%	40%	41
Rare Set	74%	26%	37
Complete Set	23%	77%	46

New Tuning

Set Type	% Finishing Set	% Unfinished Set	Average Set Completion Day
Common Set	96%	4%	32
Uncommon Set	87%	13%	35
Rare Set	42%	58%	40
Complete Set	35%	65%	43

- Players more likely to complete common sets , far less likely to complete rare sets
 - Good for retention and monetization
- Players more like to complete entire set
 - Making more common sets easier to complete wasn't completely offset by more difficult rare sets (could tune for that)
 - Higher completion not necessarily bad thing (23% could be considered low), as it can yield higher satisfaction and retention



Recommendations - Some Other Ideas to Explore

- Stagger card pack release
 - Everyone starts earning common packs
 - Rarer or just *new* packs are released in later event weeks
 - This would allow for more fewer duplicates overall, more generous tuning, and shifting player focus which could drive instant gratification monetization
- Guaranteed unique card pack
 - Could be item in the shop so players can pay to get missing card to complete their collection
 - Sale is rate-limited so users can't just complete their collection on day one, but still need to log in