LandFill

Leah Liddle

August 12, 2025

Contents

1 Introduction

Magic: The Gathering (MTG) is a Trading Card Game (TCG) designed by Wizards of the Coast (WOTC). Players take the role of a wizard, whose deck is a library of spells with which they battle one or more similarly equipped opponents. Pursuit of the hobby thus involves mastery of both gameplay and deck construction. While tournament-level participation relies on "netdecking" - using a decklist with proven competitivenesss, rather than assembling one oneself - the hobby is intended to have a signifciant creative component, with the unranked Commander format gaining popularity in recent years through its focus on unique and personal decks.

In addition to spell cards, decks also contain "land" cards, which generate "mana", a resource expended bo cast spell. Mana comes in five colours: White, Blue, Black, Red and Green (abbreviated respectively to W, U, B, R and G, or WUBRG collectively) - with spells requring specific colours, and lands producing one or more colours. A deck whose lands - the "manabase" of the deck - produces more colours gives the deck access to more spells, but also runs a higher risk of "color-screwing" the deck, in which a player, needing a certain colour, only draws lands of a different colour. Whereas selecting a list of spells is a stimulating ceative pursuit, choosing a deck's manabasei s less so: within a set budget, and notwithstanding the minority of lands which come with effects outside mana provision, there are objectively lists of lands that will maximize a player's chances of being able to play their cards.

LandFill is a webapp that automates this aspect of deck building. Users input a list of spellcards, select some broad preferences for their manabase, and are provided with a manabase that has been optimized within those preferences to facilitae reliable casting of their spells. This optimization is necessarily approximate: Churchill *et al* have demonstrated that, as it is possible to construct within an MTG game a Turing Machine whose halting is the necessary condition for a player's victory, a deck's winning strategy is undecidable [**churchill2019magic**]. Nevertheless, the heuristic that a winning MTG player is typically the one who spends the most mana over the course of the game

[**KarstenCurve**] provides an opening for the issue to be appraoched via Monte Carlo methods. Using a stripped-down MTG simulator, in which the simulated player aims only to spend as much mana as possible each turn, LandFill estimates over iterated games how effective a given manabase is at allowing a player to do so. It then produces successively optimized manabases via a Hill Climbing Algorithm. While LandFill may never rival the experienced eye of a seasoned competitive deckbuilder, it is nevertheless my contention that an app that provides a list of lands of demonstrated efficacy, via a click of a button, would pay dividends for casual players in ease of deckbuilding and satisfaction of games.

2 MTG and Mana Optimization Problems Therein

2.1 Gameplay Concepts

At the start of an MTG game, each player shuffles their deck and draws seven cards. One additional card is drawn at the start of each turn.

2.2 Land Balancing

"Balancing" refers to the aspect of card design in which a useful card is given drawbacks to prevent it from being strictly better than many other cards. In spells, this is typically enforced via mana cost: a stronger spell costs more mana.

WOTC generally avoids making lands that are strictly better than basic lands; cards that are, such as Underground Sea, typically reflect an out of fashion design philosophy, and due to age and power are thus rare and expensive. Because of this, most lands that tap for two or more colors of mana are given some sort of drawback. Players will therefore assemble the bulk of their "manabase"(a collective noun for the land cards chosen for their deck) out of a mixture of basic lands, which tap for one colour with no drawbacks, and multicolor lands that feature some drawback, choosing based on personal experience, cards in their collection, and price.

The most common drawback for a multicolor land is to make it enter tapped, and thus cannot be used on the turn they are played. While there are several cycles of two or three color lands that simply enter tapped, many can enter untapped if the player meets a specific criteria, or pays a specific resource, when they are played. Since many cycles enter tapped or untapped depending on what other lands are in play, this gives manabases a self referential character.

There are also supermechanical factors that balance lands: more powerful ones are rarer, making them more expensive, less likely to show up in a player's existing collection, and sometimes bringing a social stigma.

Lands with basic land types are generally considered better than lands that produce the same colors without basic land types. This is due to their synergy with cards that can search your library for basic land types, notably including

the "Fetch Lands", a highly-regarded land cycle that, rather than producing any mana, searches your library for a land with one of two basic land types.

Most non-basic non-utility lands - which will be a central focus of this document, and referred to as "nonbasics" going forward - tap for two colors of mana, although some tap for three and some for all five.

2.3 Overview of Manabase Optimization

Most strategies to prevent mana screw focus on choosing appropriate spells: a central concept is a deck's "mana curve", referring to the number of spells it has at each mana cost. Choosing appropriate lands is comparatively understudied, as it is a much more numerically complex process - requiring a count of the total number of pips in your deck, and how they are spread across the cards - but also one that is readily addressed by heuristics:

Multicolor lands can broadly be ranked into three categories:

Optimum lands - cycles that are strictly better than Basic Lands (such as Shock Lands or, in some context, Bond Lands), or that have repeatedly proved their efficacy in tournament play (Shock Lands and Fetch Lands).

Strong lands - multicolor lands that enter tapped unless a specific condition is met, or are balanced by another non-negligable factor that keeps them outside the above category. This may also include lands that enter tapped but tap for three or more colours, or which have basic land types and thus may be searched by fetch lands.

Taplands - two-color lands that enter tapped.

when creating a manabase, players will typically start with a quantity of basic lands roughly proprotional to the pips used by their library, supplement this with what optimum lands they have access to, and then choose other nonbasic lands from among strong lands and taplands based on personal preference. While some taplands may offer additional benefits beyond mana production that make them appeal to the player (ie, the "Scry Lands", which let the player look at the top card of their library on entering), that sits outside the scope of optimization. From an optimization perspective, the question is - what strong lands are picked? Because of the aforementioned reflexive qualtity of a manabase, this question in part depends on what other lands the player has access to.

The card "Woodland Chasm" is an instructive example here. It is a tapland with two basic land types, Forest and Swamp. It would be an undesirable situation if the player draws it when having access to exactly one more land in play would allow them to cast a higher cost spell; based on the mana curve, we can assign that probability N. However, if the deck also contains the six fetch lands capable of finding a Swamp or Forest, that probability would fall significantly, as any prior turn in which the player had access to one of these fetch lands and did not need one new untapped mana woiuld have provided them with a chance to put Woodland Chasm into play with no detriment to their game.

3 Library and Language Choices

My choices of langauge and libraries were informed by two main priorities. Due to my short turnaround time, it was important that I use libraries and langauges with substantial community support for web development. Meanwhile, as a usable app, Unscrewer benefits from high performance so as to maximize the number of simulations it can run, but does not need to offer a complex user interface nor store user data, prompting me to favour high-performance tools over complex and scalable ones.

In places where these requirements are at odds, I prioritized the former: my choice of Python as a backend and Javascript as a frontend was driven largely by the popularity of these languages in web design. However, in other decisions, the two requirements informed each other constructively. I chose Flask as a backend web framework as its simplicity made it both easy to learn and reputably faster; contenders like Django are made both slower and more complex due to their abundance of features (FastAPI, potentially lighter and faster than Flask, was discarded due to its smaller userbase and thus relative paucity of learning resources). React, which I chose as my frontend framework, similarly sports a wealth of support resources, and features a Virtual DOM that lowers performance overheads when users make small input changes - a relevant concept here, as users will likely order several simulations with small preferential changes on each one.

Lacking access to the popular Django ORM, I interacted with my MTG Card database via SQLAlchemy. To fill that database, I opted to create my own script using the Scrython library that would scrape the MTG Database Scryfall, rather than relying on existing projects that compile data for download, such as mtgio API or mtg.json, as Scrython, being a widely used player resource, is kept regularly updated, and my script could be easily re-run to account for new sets or price fluctuations.

4 Database Layout

4.1 Local vs Online Storage

Since new cards are regularly released, and card-price - an important deck-building consideration - constantly fluctuates, a key feature of my database is updatability.

Since my backend has direct access to Scryfall via Scrython, I initially considered trivializing this by foregoing a locally stored database, and simply querying player inputs to Scryfall itself. Testing almost immediately showed this to be untenable: although lands could be downloaded in bulk, referencing a player's input cards *todeterminemanavalue* required an individual search for each card. Even small decklists required several minutes to parse.

Instead, I opted to create a backend object, the DBManager, called not from the server but from a seperate script, Manage_Database.py, which, on running,

would RESTfully scrape Scryfall and update all tables. Although RESTful considerations made this code slow to run - taking around ten minutes - this is fairly managable on a weekly timeframe, and could be automated in future versions of the software.

4.2 Scryfall Shortcomings

Scryfall does exclude some information which is relevant to Unscrewer. Since the end-goal of my work here would be to have a database script that would update automatically on a timescale, I needed my script to parse the following automatically:

1. Land Searching Capabilities - usefully, Scryfall lists the mana that each land produces. However, some lands produce no, or only colorless, mana, and instead sacrifice themselves to search your library for a basic land, which is not listed. Helpfully, most of these lands are sorted into cycles, which I marked in the cycles table as "fetch" lands (a standard term in the community). I then gave each land a cached property "true_produced" which combined the data from scryfall with, if the land belonged to a fetch cycle, any land types mentioned in the text.

2. Price in GBP - Scryfall lists prices in EUR and USD. Since price fluctuations would prompt most regular updates anyway, I opted to add my own GBP column and calculate it during the scrape using a conversion library.

3. Intermittent price absences - some cards do not have a listed price. I informed my database to list these with a price of minus 1, and WILL FIGURE OUT HOW TO HANDLE THESE.

4. Edge cases - MTG features "Unset" cards, which typically fall comedically outside design norms. "Little Girl", for example, costs half of one white mana $noothercardhasanon - integermanavalue$. These cards are not popular but not inconceivable to use, as players frequently make comedy decks, and although I experimented with database schema that would incorporate these - for example, storing mana value as a float rather than an int to accommodate Little Girl - I decided that warping my model around a single rarelyplayed card was a low-reward development approach. Instead, I equipped my DBManager with an array edge_cases, and gave it customized handling instructions for each one (Little Girl's mana value, for example, was rounded to one, as this much more closely fits how she would be played)

5. Cycles - Scryfall does not sort lands into cycles. I addressed this by providing each cycle with a regular expression that matched all lands within it, and sorting on download.