LandFill

Leah Liddle

August 23, 2025

Contents

1 Introduction

1.1 Overview

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" - copying a decklist with a history of competitive success - 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 to cast spells. 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 manabase is 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 spell cards, select some broad preferences for their manabase, and are provided with a manabase that has been optimized within those preferences to facilitate 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 [2]; consequently, there is no way to judge which colours are necessary to play a deck's most decisive spells. Nevertheless, the heuristic that a winning MTG player is typically the one who spends the most mana over the course of the game [6] 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. 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.

1.2 Thesis Layout

2 Problems and Proposed Solutions to MTG Manabase Optimization

This section will provide an overview of MTG's rules and design practices insofar as they relate to manabase optimization, and the challenges which emerge form this. It then outlines the algorithmic responses to these challenges that will be used by LandFill. It will then touch on how incorporation of this into a deployable consumer product will be tested and developed. Finally, it will situate LandFill within existing scholarship on MTG automation and optimization.

2.1 Gameplay Concepts and Terminology

2.1.1 Overview

MTG can be played in several "Formats", with different deckbulding stipulations but largely identical rules, an overview of which is provided here.

At the start of an MTG game, each player shuffles their deck and draws a "hand" of seven cards. A player may "mulligan" a poor hand, shuffling back into the deck and drawing a replacement one, and typically incurring an increasing penalty for each mulligan performed (with exact details varying). One additional card is drawn at the start of each turn. Cards in hand may be played onto the "battlefield". Each turn, a player may play one land card, which they may use once per turn by "tapping" it (turning it 90°). All lands untap at the start of each turn; contingent on drawing sufficient land cards, a player should therefore have access to one mana on their first turn, two mana on their second turn, and so forth. Spell cards which supplement mana at a higher rate than this are called "ramp" spells. The spread of CMCs across a deck's spell cards is called its "curve".

Figure 1: A spell card

The "mana cost" of most spells includes a generic cost, payable by any mana, and any number of "pips", representing a single required colour. Figure 1 shows a card requiring two black mana, one red mana, one blue mana, and four generic mana. It would thus be said to have a "cost" of UBBR4, and a "converted mana cost" (CMC) of eight.

Each colour is produced by a "basic land: Plains (W), Island (U), Swamp (B), Mountain (R), Forest (G). Whereas no deck may contain more than four copies of the same card (sometimes one copy, in "singleton" formats), any deck may contain any number of basic lands. In addition to the type "land", a land card may have subtypes, providing opportunities for synergy. Confusingly within the context of mana discussion, the five basic lands, in addition to being named cards, are also card subtypes – collectively called the five "basic subtypes". Tropical Island, for example, is a non-basic land which has the subtypes Island and Forest. Any card, therefore, which references "an island" or "a forest" could have that criteria met by Tropical Island, by a Forest or by an Island. Within MTG player parlance, which will be used in this text, saying "a Forest" may refer to any card with the Forest subtype, while saying "a Basic Forest" refers to the specific card named Forest (Basic Forest, helpfully, also has the Forest subtype).

Any card with one of these subtypes taps for the corresponding colour of mana by default. However, not every card that taps for that colour of mana has that subtype. Tropical Island, for example, taps for both G and U, as does Hinterland Harbour, which is neither a Forest nor an Island.

Although a small minority of lands produce more than one mana when tapped, this is exceptionally rare. A land which "taps for BUG", taps for Black, Blue *or* Green. Lands that produce two colours are called Dual Lands, while lands that produce three are called Tri Lands; the small subsect of lands that can produce all five colours are called WUBRG lands. Some lands can

produce colourless mana, “C”, which is only useful in generic costs. Since 2015, WOTC have printed some spells which require specifically colourless mana, but this is rare.

Lands which provide an ability outside mana production are called ”utility lands,” and are irrelevant to LandFill’s calculations.

2.1.2 Land Balncing and Cycles

If a card is superior to another in any context, it is considered ”strictly better”. Tropical Island, tapping for UG, is strictly better than both Basic Island and Basic Forest. This reflects a bygone design philosophy: WOTC have, since early sets, generally avoided printing land cards which are strictly better than basic land cards. Virtually all land cards which produce > 1 colours of mana are either ”balanced” (given a downside), or produce mana via a more complex mechanism. Breeding Pool, for example, which is identical to Tropical Island save that it enters already tapped (and thus unusable on the turn it is played) unless the player pays 2 life when playing it. Lands are typically printed in cycles, which share a common balancing mechanism but refer to different colours. Stomping Ground, for example, has the same stipulation as Breeding Pool, but produces RG instead of UG. Cycles usually carry informal names within the community: Breeding Pool and Stomping Ground are both “shock lands.” The prevalence of cycles such as Check and Fetch lands (see Figure 2), additionally means that a cycle with subtypes may be considered strictly better than an identical cycle without.

To illustrate the complexity of optimization in this context, consider as an example the lands Prairie Stream and Deserted Beach, both referenced in 2. Since any situation in which Prairie Stream would enter untapped would also allow Deserted Beach to enter untapped, but the same is not true vice-versa, Deserted Beach is, taken in isolation, a stronger land. Consider, however, the following situation (in which all land cards are depicted in 2). Two players play two identical UW decks with identical UW manabases consting of Flooded Strand, Hallowed Fountain, and multipled Basic Plains and Basic Island cards. The only difference is that one includes a Prairie Stream, and the other includes a Deserted Beach.

Both decks draw the below opening hand:

$$\text{Hand} = \big[\text{Spell}(UWW) \quad \text{Spell}(UUW) \quad \text{Spell}(UU) \quad \text{Land(Basic Plains)} \quad \text{Land(Flooded Strand)}\big]$$

Since the player needs copious amounts of both U and W mana, it behoves them to use the Flooded Strand to search their library for a non-basic Plains or Island capable of producing UW. Since this hand contains no spells of CMC=1, there is no disadvantage to playing a tapped land on their first turn. The Prairie Stream player may then go and fetch the Prairie Stream at no downside. However, the Deserted Beach player has to fetch the Hallowed Fountain, leaving Deserted Beach in their library. Consider, then, that when each player shuffles

Figure 2: UW and RG lands of different cycles. Left to right: "Battle", "Shock", "Check", "Fetch", "Filter" and "Slow" Lands. Notice the "hybrid" mana cost of the filter land's second ability, meaning that requires a mana of either of its colours to activate, and the diamond marker in the output of its first ability, meaning that the first only produces colorless mana

for the Flooded Strand's effect, the UW land remaining in their respective library (Deserted Beach for the Deserted Beach player, and Hallowed Fountain for the Prairie Stream player) is placed on top. Since the Hallowed Fountain can come in untapped for a trivial life point investment, the Prarie Stream player therefore has the option of playing the spell that costs UU, whereas the Deserted Beach player - able to play only a tapped Deserted Beach or a Basic Plains, which produces only W - cannot do so.

Therefore, the appropriateness of Prairie Stream vs Deserted Beach to a given manabase depends on, among other factors:

- The probability of a player beginning a turn with N lands on the battlefield in which N is greater than 1 and at least two lands are basic.
- The probability of a player beginning a turn with N lands on the battlefield, a fetch land in hand, and no possible set of spells to play with a mana cost of N+1

This means that manbases can only be assessed in toto, and thus any lands's exclusion based on its price and availability may impact the performance of other otherwise high-performing lands in the manabase. Optimization, therefore, is not simply a matter of choosing competitively-storied lands for a player's deck. Instead, it can be thought of as a simultaneous two-part process:

- Replacing Basic Land cards with multicolor lands that improve colour access, until a point at which the accumulated balancing downsides of

those lands start to outweigh the diminishing returns from that improved access...

- ...while choosing the multicolor lands whose downsides are the most significantly ameliorated by the specific spread of CMC and cost values of the spells in the deck, and by the interactions between those lands and other lands in the manabase.

2.2 Simulation and Optimization

LandFill approaches this problem via the implementation of two algorithms: an internal algorithm, which simulates MTG games, and an external one, which provides the internal one with a series of increasingly optimized decks to test. These will be referred to as the "simulator" and the "optimizer", and the broad structure of both, and the relationship between them, will be introduced below.

2.2.1 LandFill Simulation

The difficulty of creating an automated MTG player, as will be discussed further in section 2.7, rests on MTG's status as a game of imperfect information, with this imperfection, as highlighted by Ward and Cowling, stemming from two sources: first, the random shuffle of the deck, and second, the unknowable contents of the opposing players' hands [10]. Handily, both of these are irrelevant to automated play insofar as the goal of the player is to maximize mana expenditure, as doing so concerns only the cards already drawn into the player's own hand.

This brings the Simulator into the realm of "goldfishing", an informal term for the practice of testing out a deck by taking repeated turns against no opponent, where the issue under test is the ability of the deck to deploy its relevant cards [8]. While a comprehensive goldfishing simulator would mandate the ability to effectively encode any deck's winning strategy – a NP-complete problem well beyond the scope of this work – encoding greater complexity in this way may incur diminishing returns regardless, as it would only reflect the deck's performance against an opponent that did not disrupt a player's ideal strategy. Given the heuristic (see Introduction) that a deck should aim to spend as much mana as possible, the simulator algorithm must do the following:

- Draw an initial hand of 7 cards.
- Mulligan as necessary.
- Identify which playable land will allow the expenditure of M mana, where M is the maximum that may be spent that turn.
- Play that land.
- Play a combination of spells with total CMC M.

While, within our heuristics, we may safely choose a combination of spells at cost M at random – since we have no way of telling which would be relevant in any given game, and can only estimate their value as an efficient use of mana – complexity is introduced by situations in which multiple lands allow for the spending of M mana, such as in the sample hand drawn in the previous section illustrating the appropriateness of Slow Lands vs Battle Lands: in the absence of any spells of CMC=1, playing either the Basic Plains or the Flooded Strand yelds M=0. The simulator, therefore, must have some capability of assessing which lands maximize expenditure on future turns.

2.2.2 LandFill Optimization

In a Monte Carlo search, solution space is explored by taking the random outputs of stochastic processes, and sampling the resulting distribution to appproximate the typical values of that process [7]. Assuming correct function of the aforementioned simulator, a Monte Carlo assessement of a manabase would be conducted by giving it a deck, running a large number of games with it, and recording the deck's average performance. From there, LandFall can generate an optimized manabase by testing many manabase combinations and returning the highest performing one.

As is typical of combinatorial optimization, the search space here is vast. As will be discussed later, it was decided early in development to restrict LandFill to the singleton Commander Format, meaning that no manabases containing duplicate non-basic lands ever needs be examined; even then, however, possible combinations for a manabase of 20-40 land cards (typical for the format) number in the trillions.

Fortunately, approaches abound to narrow such a search space. LandFill's search space is L dimensional, where L is the number of lands required by the deck, in which each dimension represents the quantity of a given land in the manabase (limited, in the Commander format, to values 0 or 1 for all non-basic lands). As will be detailed in later sections, LandFill uses a variant of the Hill Climbing Technique, also known as Neighbourhood Search. In Hill Climbing optimization, once an initial solution $\underline{x}_c$ is determined, all solutions in its neighbourhood $N(\underline{x}_c)$ - ie, all solutions obtainable by some simple transformation, such as, in this case, the substition of one card for another - are examined, where the first higher performing value is adopted as the new $\underline{x}_c$ [9]. In the context of manabase optimization, the iterations are as follows:

- Generate an initial manabase, $\underline{x}_c$, for the input deck.
- Conduct many simulations and determine the average performance, $F(\underline{x}_c)$, of the deck with this manabase.
- Exchange a land in the deck for a different one, creating a new manabase, $\underline{x}_t$.
- If $F(\underline{x}_t) > F(\underline{x}_c)$, adopt $\underline{x}_t$ as the new value of $\underline{x}_c$, and return to step 2.

- If not, return the original land to the manabase and return to step 2, this time making a different, previously unexplored substitution.
- If all lands in the deck have been systematically replaced with every candidate land that could replace them, and no value has exceeded $F(\underline{x}_c)$, return $\underline{x}_c$ as an optimized manabase.

A key weakness of the hill climbing approach is that it is vulnerable to local maxima. While approaches that are less susceptible to local maxima, such as genetic algorithms and simulated annealing, do exist, they are not appropriate in this context for reasons that will be discussed in later sections. However, as LandFill is designed for casual use, this is a forgivable flaw. In practice, a truly optimized manabase would account for the interactions of spell cards with the manabase, and the relative importance of particular spells at particular moments, which as stated is a NP-complete problem. Within that restriction, LandFill needs only to be able to create a more reliable manabase than a human player could in a comparable length of time to add value.

2.3 Development and Testing Methodologies

That LandFill can offer only rough approximations can be forgiven only insofar as it offers a useful service. To that end, the wider development project exists within wider Human Computer Interaction (HCI) constraints: LandFill must be flexible enough to fit easily into a range of different deck construction strategies and accommodating of non-gameplay manabase restrictions such as price, availabilty and personal preference. Indeed, the quality of LandFill's output relates to the ease of input. Any opportunity a player has to specify a personal preference narrows the search space.

Moreover, LandFall is necessarily an incomplete project. While updating its databases with new cards is trivial via Cron, new cycles of dual and tri lands are printed regularly, and each one, to be incorporated in the Simulator, must be individually coded. This means that LandFall lends itself naturally to an Iterative Development approach, as continued Iteration after launch is inevitable. Accepting this means that the launch date of LandFill does not mark the end of development, and, therefore, pre-launch development must, in addition to focussing on development of the Optimizer and the Simulator as a minimum viable product, prioritize user features so as to ensure that, on launch, it has included the ones most likely to attract repeat customers.

User testing of LandFill has therefore been divided into two stages, pre-development and pre-launch, with the former identifying unavoidable features, and the second gauging consumer satisfaction with the product. The pre-development stage consists of the following:

- A series of Semi Structured Interviews, which gauge the way prospective users create MTG decks and the manabases thereof.
- A series of Think-Aloud evaluations of a mockup, in order to observe the patterns a user falls into when using an app to this purpose.

- A Kano Questionnaire, the questions of which are based on results of the previous evaluations, in order to guide development priorities.

The post-development stage consists of the following:

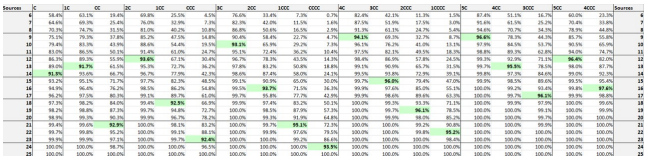
- A TLX questionnaire assessing ease of use.
- An additional set of semi-structured interviews, to determine whether the manabase proposed by the software is one the player will use.

2.4 Relevance to Existing Scholarship

Much modern academic interest in MTG, from a computer science perspective, rests on Ward and Cowling's landmark 2009 paper, "Monte Carlo Search Applied to Card Selection in Magic: The Gathering", which hypothesizes that methods used to automate other imperfect information games, such as Bridge and Poker, may be applied to MTG [10][3][1]. A decade hence, interest in this problem has resurged thanks to the rollout of WOTC's virtual MTG venue, MTG:Arena, which, although primarily a player-vs-player (PvP) engine, features an AI opponent, nicknamed Sparky, who, although useful in gameplay tutorials, presents no challenge to experienced players[1].

As mentioned, the disengagement of LandFill's simulator component with MTG's status as a game of imperfect information puts it somewhat outside the realm of scholarship inaugurated by Ward and Cowling. Insofar as MTG is studied from a computer science perspective, analyses are chiefely concerned with how to encode strategic thinking in MTG AI insofar as that pertains to relationships between sequences of castable spells. Ward and Cowling's research focusses on algorithimic approaches to initiating combat between creature spells once cast [10], while Alvin *et al*, for example, explore graphical representation of card synergies [1]; in both cases, effective use of lands to play spells is trivial, as the deciding factor in which spell to cast is based on the exigencies of the game state. Indeed, Esche's 2018 research into simulation eschews casting any multicolour spells, testing his virtual player only on a mono-red deck [3]. Since the question of whether a given set of spells comprising between them pips of multiple colours can be played the mana proeduce by a set of multicolour lands proved nontrivial, LandFill helps fill a small gap in the existing literature.

Much more significant, however, is LandFill's extension of the often non-academic scholarship conducted around MTG deck optimization by players and writers. Guides exist on how to write basic scripts in order to use montecarlo techniques to analyse a given deck [12]. A central figure in this practice is Frank Karsten, who has written extensively on how to determine a manabase for a multicoloured deck. In his seminal series of articles, *How Many Sources Do You Need to Consistently Cast Your Spells?* Karsten eschews analysis of individual spells in favour of probabalistic analysis of hypothetical spells of a given mana cost (including both generic and coloured pips), using the hypergeometric formula to assess how many lands capable of producing color C are required in a deck in order to guarantee a roughly ninety percent change that a



Sources	C	2C	CC	2C	2CC	CCC	3C	2CC	1CCC	CCCC	4C	3CC	2CCC	1CCCC	5C	4CC	3CCC	2CCCC	CCCCC	Sources
6	58.4%	63.1%	19.4%	69.8%	25.5%	4.5%	76.6%	33.4%	7.3%	0.7%	82.4%	42.1%	11.3%	1.5%	87.4%	51.1%	16.7%	60.0%	23.3%	6
7	64.6%	69.1%	25.4%	76.0%	32.9%	7.3%	82.3%	42.0%	11.5%	1.6%	87.5%	51.9%	17.5%	3.0%	91.6%	61.5%	25.2%	70.4%	33.8%	7
8	70.1%	74.7%	31.5%	81.0%	40.2%	10.8%	86.8%	50.6%	16.5%	2.9%	91.3%	61.1%	24.7%	5.4%	94.6%	70.7%	34.3%	78.9%	44.8%	8
9	75.1%	79.3%	37.8%	85.2%	47.5%	14.8%	90.4%	58.4%	22.7%	4.7%	**94.1%**	69.3%	32.7%	8.7%	**96.8%**	78.3%	44.3%	85.7%	55.8%	9
10	79.4%	83.3%	43.9%	88.6%	54.4%	19.1%	**93.1%**	65.9%	29.2%	7.3%	96.1%	76.2%	41.0%	13.1%	97.9%	84.5%	53.7%	90.5%	65.9%	10
11	83.0%	86.5%	50.1%	91.4%	61.0%	24.7%	95.1%	72.4%	36.2%	10.4%	97.5%	82.1%	49.5%	18.3%	98.8%	89.3%	62.8%	94.0%	74.7%	11
12	86.3%	89.3%	55.9%	**93.6%**	67.1%	30.4%	96.7%	78.3%	43.5%	14.5%	98.4%	86.9%	57.8%	24.5%	99.3%	92.9%	71.1%	**96.4%**	82.0%	12
13	89.0%	**91.7%**	61.5%	95.3%	72.7%	36.2%	97.8%	83.2%	50.8%	18.8%	99.1%	90.9%	65.7%	31.5%	99.7%	**95.9%**	78.5%	98.0%	87.7%	13
14	**91.3%**	93.6%	66.7%	96.7%	77.9%	42.7%	98.6%	87.4%	58.0%	24.1%	99.5%	93.8%	72.9%	39.1%	99.8%	97.3%	84.6%	99.0%	92.3%	14
15	93.2%	95.1%	71.7%	97.7%	82.3%	48.5%	99.1%	90.9%	65.0%	30.0%	99.7%	**96.0%**	79.4%	47.0%	99.9%	98.5%	89.6%	99.5%	95.4%	15
16	94.9%	96.4%	76.2%	98.5%	86.2%	54.8%	99.5%	**93.7%**	71.5%	36.1%	99.9%	97.6%	85.0%	55.1%	100.0%	99.2%	93.4%	99.8%	**97.6%**	16
17	96.2%	97.5%	80.3%	99.1%	89.7%	61.0%	99.7%	95.8%	77.7%	42.9%	99.9%	98.6%	89.6%	63.3%	100.0%	99.7%	**96.1%**	99.9%	98.8%	17
18	97.3%	98.2%	84.0%	99.4%	**92.5%**	66.9%	99.9%	97.4%	83.2%	50.1%	100.0%	99.2%	93.3%	71.1%	100.0%	99.9%	97.9%	100.0%	99.6%	18
19	98.2%	98.8%	87.3%	99.7%	94.8%	72.7%	100.0%	98.5%	87.9%	57.3%	100.0%	99.7%	**96.1%**	78.5%	100.0%	100.0%	99.1%	100.0%	99.9%	19
20	98.9%	99.3%	90.3%	99.9%	96.7%	78.2%	100.0%	99.3%	91.9%	64.8%	100.0%	99.9%	98.0%	85.2%	100.0%	100.0%	99.7%	100.0%	100.0%	20
21	99.4%	99.6%	**92.9%**	100.0%	98.1%	83.2%	100.0%	99.7%	**95.1%**	72.3%	100.0%	100.0%	99.2%	90.8%	100.0%	100.0%	99.9%	100.0%	100.0%	21
22	99.7%	99.8%	95.2%	100.0%	99.1%	88.1%	100.0%	99.9%	97.0%	78.5%	100.0%	100.0%	99.8%	**95.2%**	100.0%	100.0%	100.0%	100.0%	100.0%	22
23	99.9%	99.9%	97.1%	100.0%	99.7%	**92.4%**	100.0%	100.0%	99.2%	86.6%	100.0%	100.0%	100.0%	98.6%	100.0%	100.0%	100.0%	100.0%	100.0%	23
24	100.0%	100.0%	98.7%	100.0%	100.0%	96.5%	100.0%	100.0%	100.0%	**93.9%**	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	24
25	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	25

Figure 3: Frank Karsten's recommendations for how many lands a deck should include capable of producing colour C; he recommends adopting the highest Y-axis score corresponding to a card whose mana cost appears in your deck [6]

player is able to cast the most color-intensive spell in their deck of CMC M on the turn that their Mth land is played. Karsten's results are plotted in a grid, shown in 3.

While this article engages broadly with the issue of how many multicolour lands are necessary in a deck before they incur diminishing returns in deck playability, the second key optimization question, concerning the severity of the downsides of different land cards in the context of different decks and manabases, is beyond this. While it is possible to use hypergeometric analysis to determine the probability of drawing a colored land at a helpful time, supplementing this by analysis of whether that land will be able to use its full productive capacities on the turn it is drawn requires an understanding of lands have already been drawn, and how they have been played, which depends on the cards drawn in subsequent hands before the one where the spell of cost M is played. This is where Monte Carlo methods prove their utility. Karsten has used Monte Carlo methods to analyse manabase choices in the context of the Limited format (which features small decks and a heavily restricted set of lands to choose from)[5] and the Standard format around the release of the Ixalan Set[4], using a similar methodology to LandFill's: creating a basic mana-spending gameplay algorithm and using it to run a high number of simulations. Since, however, these simulations only cover the lands avalable in these formats, data returned only hints at deckbuilding principles, and cannot spontaneously generate a manabase by application of those principles. LandFill, therefore, may be seen as an attempt to take the methodologies used by Karsten for these specific contexts, and generalize them into a widely applicable deckbuilding tool in the vein of his grid.

It is also worth comparing LandFill's approach to those used by existing deck construction webapps, of which there are three that have comparable functionality.

The first is MTG: Arena, which is capable of automatically filling a deck with an appropriate proportion of each basic land in accordance with the deck's colours. While this is relevant for the Standard and Limited formats largely played on Arena, the deckbuilding restrictions of which only offer few nonbasic lands, it is less appropriate for other formats, in which a sizeable proportion of a manabase will be nonbasic; it is also only accessable to decks constructable via the limited set of cards available on MTG Arena.

The second is ManaGathering, a database of nonbasic lands sorted by colour. Players input the colours of their deck and are given a list of nonbasics within those colours, sorted by cycle. Although ManaGathering has no optimization or manabase generation facility, it fulfills a similar role in the deck building process as LandFill, facilitating the choice of generically strong mana-producing lands once utility lands or lands fulfilling a niche strategic concern of the deck have been chosen. A broad success metric for LandFill is that it should return a better result than a player using ManaGathering could in a comparable time.

Most sophisticated of these is Archidekt, which combines a basic-land allocator *a la* MTG: Arena with a communal "Package" system. Players create "packages" of lands which are saved publically on the site, and may be imported by other players. For example, a player may get a list of colour-appropriate lands of a reasonable price by importing a given manabase package and then automatically generating a list of basic lands. The question of whether LandFill or ArchiDekt are capable of producing more reliable manabases is unlikely to be answered in the timeframe of this investigation, but LandFill represents an alternative approach; morever, as Archidekt's functionality is limited to decks that are stored within its database, it offers a more flexible service to deckbuilders who may prefer other collection-trackingd databases such as TappedOut or Moxfield.

2.5 Library/Language Choices

My choices of language 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, LandFill 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.

The use of Object Relational Mappers (ORMs) is common in web design, usually as a component of a CRUD (Create, Read, Delete, Update) interface taking user data. Although LandFill makes heavy use of Object Relational Map-

ping to store a database of MTG cards, the user needs to only read the database. For this reason, SQLAlchemy, an ORM esteemed for rapid performance at the cost of easy data amendment**??**, was the obvious choice.

2.6 LandFill Structure

With both the inherent challenges of manabase optimization and the set of potential user needs thus established, LandFill will be built as four interacting components, outlined below.

- The Database - LandFill's database of MTG card objects, stored in the backend as mtg.db.
- The Simulator - see 2.2.1
- The Optimizer - see 2.2.2
- The Web Interface - a user-friendly web interface designed according to the priorities set out in 3

3 Pre-Development User Testing Output

3.1 Overview

I began development by holding one-on-one user testing sessions with eight MTG players. The sessions were divided into two parts, a Semi-Structured Interview and a Think-Aloud Mockup Test, with an additional Kano Questionnaire circulated to users afterwards.

3.1.1 Semi-Structured Interview

As LandFill needs to be incorporable into a players' deck construction processes, I was interested in expanding my understanding of how players went about creating manabases moreso than I was interested in specific feedback about potential features of the product. In recognition of this, I chose to lead with a semi-structured interview, a data-gathering method in which the interviewer treats a series of pre-prepared questions as broad discussion prompts. My questions were as follows:

- How do you approach selecting lands for a deck, and how does this vary across formats you play?
- How do you approach acquiring lands for a deck (eg, do you assemble a list of cards to purchase, do you assemble a list of cards you already have - and if so, do you have a good knowledge of what lands you own)?
- What role does existing deckbuilding support apps, such as Moxfield and Tappedout, play in your process?

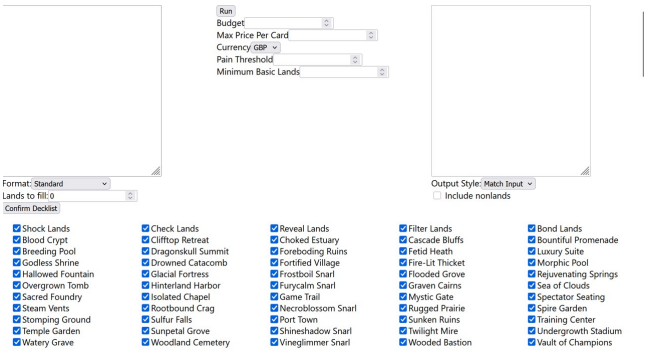


Figure 4: Draft front-end used in the mockup testing.

- Do you factor the strategy of your deck into land choices in terms of pure mana production (ie, not including utility lands).
- How do you mulligan? How does this varya cross formats that you play?

3.1.2 Think-Aloud Mockup Testing

In a "Think-Aloud Evaluation", users are asked to narrate aloud their thoughts and opinions while attempting to use a system [11]. This is an appropriate method for early development since it can be conducted on a "mockup", ie, an aesthetically versimillitudinous but non-functional representation of the planned interface. Users are occasionally prompted for input, and may be given solutions to problems if necessary, but are largely expected to use the product unassisted.

The LandFill mockup presented to users is displayed in figure 4. Since the mockup did not include contextual hints, users were told when they moved their mouse over the Pain Threshold input that this represented the amount of life users were willing to lose to lands such as Shock Lands which require a life point investment.

3.1.3 Kano Questionnaire

Kano Analysis is an approach to user evaluation that questions the emotioanl response of a prospective user to the presence or absence of a given feature. Questionnaire questions were developed after analysis of the intial user testing results, so as to prioritize which features, suggested by individual testers, were reflexive of widespead demand and warrented focus within LandFill's limited development timeframe. A generic Kano template is displayed in 5. This approach was applied to the following proposed features for LandFill:

- The option to exclude from consideration all lands which always enter tapped.
- The option to exclude from consideration all lands above a certain price.

Figure 5: A question in a generic Kano questionnaire

- The option to exclude any individual land or cycle from consideration via the player's own preference.
- The option to mark some lands as mandatory for LandFill to include.
- The option to "weight" lands, so that LandFill prioritizes a player's preferred cycles in its evaluation.
- The option to input a list of lands as well as a list of nonlands and have LandFill choose the best of these.
- The option to tell LandFill not to recommend "Off-Color Fetches" - ie, a Fetch Land that can search for an Island or Plains in a UR deck, as fetching Islands is still useful for that deck.
- The abiilty to see an image and description of any suggested land/cycle.
- The ability to view performance metrics from a deck with a given manabase.
- A FAQ explaining how LandFill determines lands.
- The ability to specify how much life a player is comfortable to lose to land cycles that require a life point investment, ie, Shock Lands.
- The ability to view low-performing lands in a given deckilst, to inform a player's choice of supplemental mana generation spells, or what to replace on the release of new cycles.
- The ability to generate manabases for, respectively, the Commander, Modern, Legacy, Pauper and Limited formats.
- The ability to copy a decklist into and out of LandFill with minimal reformatting from, respectively, the following Deck database apps: TappedOut, Archidekt, Moxfield, Deckbox.

3.2 Emerging Themes

While initial user-testing yielded multiple small design considerations, and will be cited as various development decisions are outlined throughout this writeup, key themes are outlined below:

3.2.1 Commander/Casual Preference

Commander, Limited and Pauper were the three most popular formats among interviewees. Of these three, a plurality of respondents openly acknowledged that Commander would be the only format for which they would consider using LandFill. Limited players felt that, since the format is played using cards assigned to the player at random immediately before a game, deck construction is restricted largely to basics, and there is rarely time to upload a decklist to an online service, especially on a mobile device. In Pauper, which restricts decks to only common and cheap cards, use of utility lands, or lands with specific synergistic qualities, is so normalized that manabase strategies prioritize this over consistent mana generation. While fewer interviewees had much experience with more competitive formats such as Standard, Legacy and Modern, they raised doubts as to the ability of any spontaneous manabase generator to gain traction within those scenes, as players in those formats habitually use existing deck archetypes, and attach to them manabases with proven competitive credentials. In responses to the Kano Questionnaire, most respondents expressed that they would like/be netural on the support for non-Commander formats, but only one respondent identified any format other than Commander as a minimum expectation.

It was decided, therefore, to make the initial launch of LandFill solely a Commander product. Commander's deckbuilding restrictions are markedly different from othe formats, and will be discussed at length in my outline of the Simulator; given this distinctiveness, I deemed it more important to make a product that worked seamlessly for Commander decks, rather than one that attempted to accommodate potential decks across a wider range of formats.

3.2.2 Preference over Strategy

When asked about how the strategy of a given deck informed the choice of lands, nearly all players said that it did so, but almost solely in regards to the ancillary effects of certain cycles – ie, Gain Lands (tapped dual lands that give one life on entry) being popular in decks that trigger from gaining life. Only one said that they would actively choose slower but more colour-diverse lands for slower decks. However, when presented with the Mockup outlined in 4, all users swiftly devoted themselves to using the tickboxes to remove lands or cycles that they did not want to have included in the deck, despite having been told to imagine this mockup returning a fully optimized list of lands. Several users also expressed a desire for a weighting functionality, so that they may specify lands which they prefer.

Players also expressed a dislike of certain categories of lands, including dual lands which invariably enter tapped and off-color fetches (see 3.2.3).

The lack of interest in strategic decisions in manabase assembly validates LandFill's use-case, as it suggests that, by testing lands against the needs of the deck, LandFill is definitionally putting more consideration into land selection than a player. However, player input needs to include both a positive component

- in which players insist on the inclusion of suboptimal lands such as Gain Lands because of the strategies they facilitate - and a negative component, in which players are given extensive leeway to remove lands from consideration.

Additionally, the minority of players who did factor strategy into their manabase choices did so by accepting a greater risk of lacking the necessary colors to play spells in exchange for a minimal risk of nonbasic lands entering tapped in decks that wanted to win on earlier turns, reflecting a high-risk/high-reward strategy. For this reason, it is important that LandFill allow players to specify how many turns they wanted a simulated game to run for.

3.2.3 Deckbuilder Personas

Interviewees broadly designed manabases in one of two ways, embodied in the below personas:

- Persona A, who possesses a large collection of land cards and, on creation of a new deck, selects from this collection a selection of lands they like in these colours.
- Persona B, who develops a new deck including lands and then orders these land cards for it.

This prompted me to ask interviewees which of the following two models of LandFill they would prefer:

- Model 1 - LandFill, in addition to taking a decklist of nonland cards, also takes a list of land cards that the player might have found in their collection, and returns the optimum subsection of these.
- Model 2 - LandFill takes a decklist of nonland cards and asks the user only to specify what lands they do not want, generating an optimized list from the remaining options.

I expected a preference towards Model 1 to be strongly associated with Persona A deckbuilders and vice versa, but to my surprise, a majority of deckbuilders across the personae preferred Model 2. Several Persona A deckbuiders preferred it because they were enthused by the prospect of being recommended lands they had not heard of before.

While this was a majority consensus, and all Kano respondents said they would be able tolerate the absence of Model 1 functionality, some users did express a desire for Model 1 functionality, making it a viable route for future development of LandFill. However, the initial design covered in this writeup will adhere to Model 2.

3.2.4 Flexible Input Parsing

All users polled made use of at least one online database (a plurality used TappedOut, with Moxfield and Deckbox also being poopular). In the Think-Aloud evaluation, they copy pasted decklists from these sources into LandFill,

and said they would return the list of outputted lands there. Interestingly, many copied not from the inbuilt export feature of these sites, but instead by simply copying the homepage.

This suggests LandFill would benefit from a flexible input parsing device that can accept a decklist presented in the formats corresponding to both exported lists and copy+pasted front pages of these sites. Moreover, TappedOut allows for categorization of cards into custom types, which are included in the decklist when copied. The parser, therefore, must be able to recognise these custom types and parse them not as cards but as keys to a dict object which contains the list of cards corresponding to that category, so that, on output, the deck can be re-formatted and pasted back in.

3.2.5 Mulligans

Players were not able to describe any consistent rules on which they based their decision to mulligan; all subjects said the decision would be based not just on the castability of spells in the hand with the lands drawn, but also the strategy enactable via those cards. It is not, therefore, my priority to provide a means to assess a given LandFill user's mulligan preferences. For the initial design, the mulligan heuristic will be built into the simulator.

3.2.6 Life Loss, Cost, and the Knapsack Problem

Because user testing was conducted before the development of the Hill Climbing algorithm, it included two proposed features which provied unimplementable: total cost, referring to the total cost of the manabase, and pain threshold (see 3.1.2). In theory, these could be adhered to simply by discounting any manabase that exceeded these values. In the context of Hill Climbing, however, this runs afoul of the NP complete "Knapsack Problem" inasmuch as no algorithm exists that can restrict the search space to only these combinations while providing a neighbourhood to any input value that is also guaranteed to fit within this criteria; the algorithm may, for example, refuse to accept a manabase of a set price and thus never become the neighbour of a cheaper maximum.

In the absence of these features as defined, it is worth considering how the principles they embody may be embedded in LandFill's design. The "Max Price per Card" input in Figure 4 serves as a proxy for "price" and allows players to request a cheaper overall deck. Since, however, the accumulated price of the deck will not be known until completion of the simulation, this highlights the necessity of a post-completion "re-run" feature, by which players may remove individual expensive cards from consideration and then re-run the optimizer without having to return to an earlier stage in the input process.

Life Loss, interestingly, proved to be a negligible concern among Kano respondents, and playtesters during the think aloud did not cite the pain threshold feature as important to them. This surprised me, as life loss is a common way of balancing otherwise strictly superior lands. Rather than a player input, during development, the life penalty of a given land was considered instead a tie-breaker

development - ie, a land requiring life investment would not be favored over a comparably designed land without a life penalty. The implementation of this is discussed in 6.1.2.

4 The Database

4.1 Database Requirements

The database in which LandFill stores card objects will need regular updating by the site owner. This is for two reasons. First WOTC regularly releases sets of new MTG cards. Second, since LandFill allows users to request cards below a certain price, the database must keep track of the fluctuating price of MTG cards.

In a given session, the database will be queried at two points: first, to determine the mana costs of the input cards, and second, to identfiy a list of candidate lands for the manabase. Since the first of these points necessarily must be a series of individual queries, response time is a factor. This means that the database must be stored locally: an early prototype used the Scrython API (to be discussed in 4.2) to fetch requested cards from an existing online MTG database, and took several minutes to retrieve a list of 50 nonland cards due to a combination of connection latency and the need to avoid overwhelming the server.

To accommodate both constraints, the database therefore must be able to regularly update itself from an online database while storing the information locally.

4.2 Choice of Online Database - ScryFall/Scrython

Although several projects exist that offer MTG card data for downloading, these are maintained by for the comparatively small and undemanding market of MTG programmers, and not the much larger community of MTG players. As a result, they suffer from slow updates– mtgio API, for example, is several sets out of date at time of writing. Others, like mtg.json, include large amounts of extraneous data on card reprints to parse through.

Fortunately, there is a comprehensive and player-focussed database of MTG cards, “Scryfall”, which offers an API for code integration, for which a python library, “Scrython” has already been developed. Scrython accepts card queries using the format of ScryFall’s advanced search option, and returns a list of dict objects listing card attributes, referred to within LandFill’s code and this writeup as SCOs (Scrython Card Objects). LandFill’s database, thererefore, is a translation of these SCOs to SQLAlchemy’s ORM format. Allowing for small pauses to prevent server overuse, downloading all 30,000 cards (at time of writing) takes around one hour, and could easily be run weekly, or in response to new sets and notable fluctuations (typically prompted by the announcement of card reprints or the banning/unbanning of cards in certain formats).

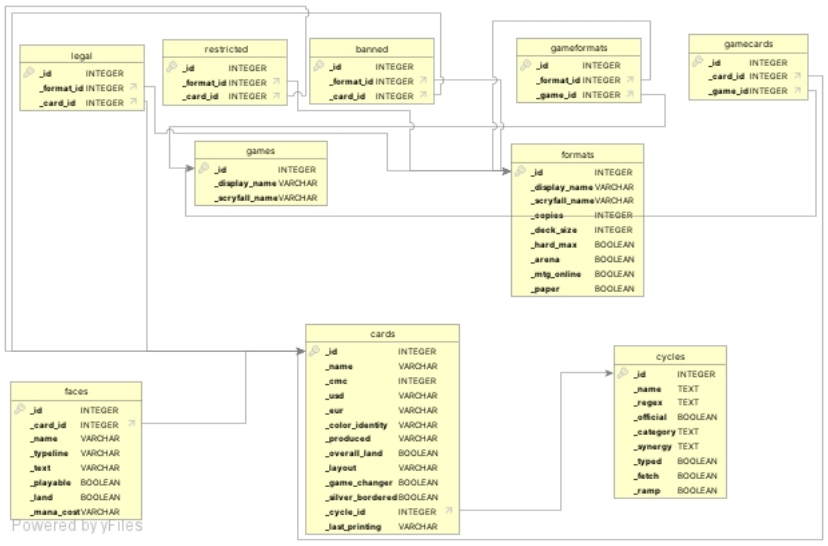


Figure 6: Layout of mtg.db

In addition to ease of access ScryFall has the advantage of listing the colours produced by a given land card, eliminating the need to parse this from the card text. Its most significant shortcomings pertain to price data: it only offers a price in EUR and USD and not GBP, and it fails to list the price for a small minority of cards. As a simple fix for the former, LandFill uses the CurrencyConverter library to determine the price in GBP from the price in EUR. The latter would require a more complex code fix to scrape an alternate database on download if a price is not found on ScryFall, which is considered out of scope for the initial release. Given the small number of cards affected, it is acceptable at the current stage to enter these prices manually.

ScryFall also has the advantage of listing, as an attribute for a given land card, what colors of mana that land is capable of producing.

4.3 Database Layout

Since LandFill uses Flask, it is built on the slightly more feature-heavy Flask-SQLAlchemy extension, which means that each mapped class extends the Flask-Sqlalchemy.model ancestor class, rather than the declarative-base class used in pure SQLAlchemy. The relational mapping between models in mtg.db is shown in 6.

The two upper rows of tables in 6 map the many-to-many relationships which denote the availability of a card across formats and "games" (eg, physical MTG games, MTG:Arena); one card is legal in many formats, and one format has many legal cards. Since the initial deployment of LandFill focusses on the Commander format, this is largely irrelevant, but remains in the schema for use

in future expansion. Discussion henceforth will focus solely on the models Card (table: cards), Cycle (table: cycles) and Face (table:faces).

Although not every attribute of a SCO is embodied in a model, in this initial development stage, LandFill errs in favour of storing too much data rather than too little, meaning that some card attributes, including `card._silver_bordered` and `card._last_printing`, are not relevant to this writeup. Attributes will be explained as they become relevant.

4.4 Representing Multiple-Faced Cards

Some cards have two faces, each of which may be considered two different cards with their own text and mana cost. Although most cards have only one face, to keep within the strictures of 1st Normal Form, faces are considered to have a many-to-one relationship with cards. Broadly, the Face object has attributes used to determine its interaction with a given game state: ie, its casting cost and potential Basic Land types (stored in the `faces._typeline` attribute), while the Card object holds attributes relating to the viability of a card within a deck and a given user's preferences (IE, what colours of mana it produces, the cost of the card).

The rules on how to pplay a card's different face varies between cards: some maybe played with either face, while others have a single playable face and swap to the other under certaing ame conditions. This is determined by the `cards._layout` attribute, which may be one of a series of strings denoting a different way of formatting face relationships (ie, cards with the "transform" layout have one playable face; cards with the "adventure" and "mdfc" layout have two). The playabiltiy of a given face is stored in the `faces._playable attribute.`

Some cards are a spell card on one side and a land card on another. A card is considered a land if it has at least one playable face that is a land, represented by the `card._overall_land` attribute.

4.5 Cycles

Recall that land cards within the same cycle are mechanically identical but refer to different colours in their text. For this reason, each cycle has a string attribute, `cycle._regex`, consisting of a Regular Expression. If DBManager matches the regex of a cycle to a card, that card is connected to that cycle via Foreign Key. Since some cycles are mechanically identical to each other but are distinguished by the presence of basic land types or whether the constituent cards have the "snow" card type, these are both also stored as Boolean cycle attributes, to distinguish matching cards.

Although some spell cards are arranged into cycles - mechanically similar cards where each costs a different colour or set thereof - this is irrelevant to LandFill, and the cycles table therefore stores only land cycles.

5 The Simulator

5.1 The Commander Format

In the Commander Format, deckbuilders choose a creature to be the "Commander" of the deck. Rather than being included in the deck, the Commander is placed in the "Command Zone", allowing them to be reliably cast in every game. Some cards are marked to allow a "Partner", a secondary commander also placed in the command zone. Because of this, commander decks are generally designed to execute a strategy that benefits greatly from having the commander in play.

As mentioned in 3.2.2, it is necessary for users to be able to set how many turns a game can run. 10 turns has been adopted as an defualt game length, based on analysis conducted by the Command Zone podcast [**CommanderMythBusters**].

The Commander Format makes widespread use of the "London Mulligan", in which, after mulliganing N times, a player puts N-1 cards from their hand onto the bottom of their library; the -1 allows players to take a free mulligan.

5.2 Classes

5.2.1 GameCard and subclasses

During simulation, a land must exhibit the unique behaviour of its cycle. LandFill, in order to execute the method overriding required here, requires a class hierarchy beginning with a general Card object, of which spells and lands are behaviourally distinct child classes, and each cycle represents a further child class of Land. While SQLAlchemy does support Single Table Inheritance - storing distinct subclasses, with distinct methods, in the same shared table belonging to the ancestor class - and temporary storage of game state information would be storable via cached properties without impacting the contents of the .db file, I opted instead to avoid using model objects in the simulation, instead using a new abstract class, GameCard, to avoid the risk of accidentally persisting data between game, as well as to facilitate complex interactions such as sublands (see **??**). All GameCards possess two boolean attributes, "mandatory" and "permitted", denoting cards which must be included in a deck (ie, nonland cards input by the player) and lands marked for consideration (ie, excluding cards instantiated from the ORM to be submitted for user approval, but deemed by the player to be unsuitable for the deck).

The simulation makes heavy use of two methods belonging to the Land subclass: `land.live_prod(game)` and `land.enters_untapped(game)`, each of which take a Game as an argument, and determine, based on the conditions of that game, respectively, what colours of mana a land can produce, and whether a land enters untapped (see 7 and 8). The overriding of these methods is the core difference between most cycle subclasses, although some cycles are more complex as will be illustrated. Note that while most classes base the output of `self.live_prod(game)` on the "produced" attribute taken from the instantiating ORM model, this is more complicated for SearchLands, which determine which lands

Figure 7: Examle of a land belonging to the Verge cycle and the implementation of `land.live_prod(game)` in that cycle

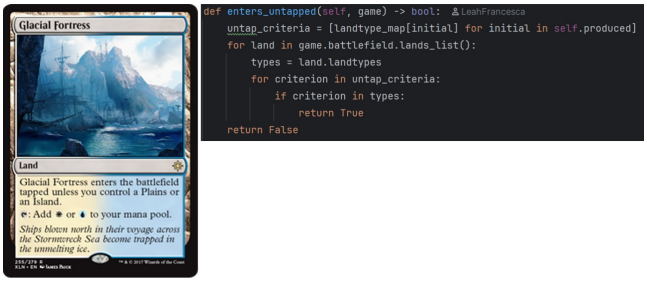


Figure 8: Examle of a land belonging to the Check cycle and the implementation of `land.enters_untapped(game)` in that cycle

are searchable by a given card based on some simple parsing logic performed on the instantiating card's text.

5.2.2 CardCollection and Subclasses

In an MTG game, cards move between the Deck, the Hand, the Battlefield and the Graveyard. All of these are modelled as descendants of a common abstract class, CardCollection, which allows for easy modelling of transfer between one CardCollection and the other.

MonteCarlo, the object which runs tests with different decks and makes adjustments according to the hillclimbing algorithm, was also modelled as a CardCollection – it may be thought of as a virtual "player", who has their own collection of cards which they swap into and out of the deck before each game.

5.2.3 Simulation and Subclasses

The Simulation abstract class covers objects that perform actions with a specific deck, and thus take a deck as an attribute. A Game and a Trial are both subclasses of simulation: a single Game, when run, simulates a game of MTG with its deck and updates its own attributes with relevant performance data,

while a Trial, when run, instantiates many Game objects with its deck, runs them, and updates its own attributes with aggregated performance data.

5.2.4 "Lump"

Consider the below game state:

Battlefield: [Basic Forest, Basic Forest, Basic Island] Hand: [G, GG].

Although any individual spell in the player's hand is castable, including the largest spell, the current selection of lands is suboptimal, as if the player had access to three green mana, they could cast two spells this turn. For this reason, rather than assessing the castability of spells, LandFill assesses the castability of "lumps". A lump is an aggregation of Spells, which unlike a CardCollection are not removed from the previous CardCollection: on a given turn, a game will have a Hand, a CardCollection, and a series of Lumps, which represent different permutations of cards in that hand.

5.3 Initiating a Game

At the start of each game, the deck is shuffled. The first seven cards from the Deck are moved to the Hand. Because a full implementation of the London Mulligan (see **??**) would require an understanding of deck strategy beyond the scope of this work (see **??**), the simulator mulligans any hand that contains fewer than three lands, and does not take any non-free mulligans to avoid making decisions about which cards to move to the bottom of the library.

To avoid having to model the Command Zone, the Commander and Partner are added to the player's hand after mulligans.

5.4 Running a Turn

The below process is repeated several times in a game depending on the game's anticipated length as entered by the user.

Many decisions made in running an individual term refer to the number L, which is the number of lands on the battlefield at the start of the turn. The most mana accessible on any given turn, then, is L+1.

5.4.1 Untap and Draw

At the start of every turn, all lands tapped status is set to False. In practice, since all spells are cast simultaneously each turn as part of a lump and tapped lands are simply treated as lands that cannot be used the turn they enter; nonetheless, adhering to an "untap step" at the start of every turn ensures a common design pattern, and facilitates modelling new land designs as they are printed. After this, the first card in the Deck is moved to the Hand.

5.4.2 Determining Lumps

Recall that a Lump is a combinationof spells in the hand. LandFill here uses the itertools module's combinations function to get possible spell combinations. During development, this was identified as an potential runtime bottleneck. A non-mulliganned initial opening hand containing 2 lands and 6 spells (after the draw during the first turn) produces $2^6 - 1 = 63$ combinations; while the only relevant combinations are those with combined CMC $< L + 1$, this cannot be calculated in advance due to the knapsack problem. The simulator therefore uses the following heuristics:

- There is no need to search for any combinations of more than L+1 spells, as (notwithstanding spells of CMC zero, which are irrelevant for manabase optimization), there is no way to play more than this number of spells in a single turn using only lands.
- Any spells of $CMC > L + 1$ are ineligible, as they would not be castable in the first place.

Code profiling, carried out throughout development via the `line_profiler` library, shows that, with these heuristics in place, playing lumps takes up around 1/3rd of the runtime of `game.run_turn()`, making it an area for further improvement. In the meantime, it is worth noting that the immediate solution of using "lazy" combination generation - in this context, ceasing generation after a playable combination has been found - is not applicable here. Consider the following opening hand ($L = 0$):

[U, Basic Island, G, Basic Forest, Basic Swamp, GB]

Were combinations to be generated lazily, the simulator would identify the card of cost U as being playable, and play the Basic Island as the land for the turn. This would be a suboptimal play, however, as playing the Basic Forest would allow the same mana expenditure while also allowing the playing of a spell on turn 2 (if the Swamp is then played).

5.4.3 Assessing Lump Playability

The question of whether a given lump can be played with a given set of lands is non-trivial. This is partly to do with the inbuilt complexity of some lands: complex designs such as Filter Lands, Check Lands and Dual-Faced Lands are discussed shortly. However, even setting these aside, the ability of some lands to produce multiple colours of mana means that deciding which land should be tapped for what mana requires determining what is needed for other pips.

An option explored fairly comprehensively during development was to determine all combonations of mana production. The optimization bottleneck that this produces should be quickly obvious, especially as there are no comparable heuristics to those used when determining Lumps. Consider each entry in the array below to be the producable colours of a land on a given battlefield:

[BUG, BGC, UG, UG, UG, GB, GB, U, U]

Figure 9: Two potential arrangements of lands that can be canonicalized identically

In this case, thera are $3 * 3 * 2 * 2 * 2 * 2 * 2 * 1 * 1 = 288$ combinations, which must be re-calculated on every played land. While this can be done lazily, this does not save signficant time, as any half-completed lazy calculations must be needlessly re-commenced if another lump is played before a new land is played. Several methods were trialled to circumvent this problem.

One initial solution, to save recalculating combinations each turn, was to store the list of combinations as an attribute of the Battlefield Object. Whenever a new land, producing C colours, its first colour would be added to all existing copies, and then C-1 shallow copies of all pre-existing would be made. Even without new combination generation, however, simply iterating through the existing combinations to update them proved a debilitating bottleneck. Moreover, it greatly problematized removing lands from the battlefield, something necessary both due to gameplay stipulations (ie, the "Bounce Lands", which generate two mana per tap but require an already-played land to be returned to the player's hand) and computational shortcuts. Some lands, such as Verge Lands and Filter Lands (see 2 in 2.1.2), produce different colors of mana depending on what other lands on the battlefield. This means that, when determining what land to play, the simulator must account for not only what mana that land can produce, but what mana it facilites pre-played lands to produce. The easiest way to simulate this is to "play" each land, assess lump playability with it played, and then return it so that th enext land can be tested.

A more promising solution was to Memoize the land inputs. Because any two untapped UB (for example) lands are identical, the number of combinations to Memoize appears at first comparatively small, as each land can be canonicalized only as its colours; moreover, basic lands do not need to be canonicalized, and simply reattached to each permutation afterwards (see 9)

This functioned acceptably for a 3-color deck. Memoization via Python's pickle module produced a .pkl file of around 1000 megabytes. However, the trialling a 5 color deck instantly multiplied this size 100-fold, and caused hill climb increments that took upwards of 40 minutes even with a ratio of only around 1 cache miss for every 50 hits.

Ultimately, I settled on modelling the question as a Linear Assignment Prob-

lem using SciPy's `linear_assingment` function. Using the layman's terms example used by Koopmans and Beckman, [**koopmans1957assignment**], a Linear Assignment algorithm assesses how to use a set of plots of land to place a equally-sized set of factories, by mapping out the cost of each factory were it to be placed in a given plot as a 2D matrix, and identifying the set of plot-factory pairs incurring the lowest overall cost.

In order to ensure a square matrix, each Lump is assigned zero or more "None" pips to bring its total CMC up to L, while its generic cost is split into that many "Gen" pips. Each of these pips is equivalent to a plot of land in the above example, while each land is equivalen tto a factory. Tapping a Land for mana which it cannot produce incurs a cost of 9999, while tapping a land for a colour it can produce, a generic mana or none, incurs a cost of 2 (this is to allow space for some lands to set lower costs if their use is prioritized in some situations – this is outlined later when dealing with more complex lands). A tapped land produces any colour, or generic mana, for 9999, and None for 2. Assuming no board states of 9999/2 lands, any response of less than 9999 from the linear assignment therefore, denotes that the lump is playable. While assembling this matrix is a non-negligable time investment, this approach significantly reduced runtimes.

5.4.4 Playing a Land and a Lump

At this point, the logic of the turn splits depending on whether or not there is a land to play.

If there are no lands, the simulator plays the largest playable lump.

If there is at least one land, the lumps are sorted by total CMC, beginning with the largest. The simulator then sequentially plays each land, determines the first (largest) lump that can be played with it on the battlefield, and then returns it to the hand. This list of lands is then progressively refined by the below functions:

```
allows_largest = self.filter_by_largest(lands)
filtered_as_taplands = self.filter_as_taplands(allows_largest)
 filtered_by_most_produced = self.filter_by_most_produced(filtered
_as_taplands)
```

Where:

- `filter_by_largest()` returns the lands that allow for playing the largest lumps.
- `filter_by_taplands()` returns its input if none of the inputted lands would enter tapped this turn, and otherwise produces the ones that will. Placing this after `filter_by_largest()` ensures that the deck aims to play lands that enter tapped on a turn when that makes no difference to the amount of mana cast.
- `filter_by_most_produced()` assesses the non-generic combined pips of every spell in the hand, and returns the land or lands which remove the

largest number of these pips which are not already produced by a land on the battlefield. For example, if a hand's spells have the combined pips R B U G G, and the battlefield contains a single swamp, then Ketria Triome (producing RUG) would have a score of 1, while Zagoth Triome (producing BUG) would have a score of 2. Lands with the lowest score are returned, and out of these lands, lands which produce the greatest variety of mana are prioritized (ie, for a hand requiring G and U, a BUG land would be prioritized over a GU land)

A land is played from the returned lands at random, increasing the value of L by one, and a lump is played with values as close to L as possible. When the lump is played, each land used in its casting is set to to `land.tapped = True`, corresponding to mana dictated by the given mapping output of the linear assignment.

5.5 Concluding a Game

When a game is concluded, all cards owned by the Hand, Battlefield and Graveyard are returned to the Deck. Performance metrics for the game are set as attributes of the Game object, for querying by the Trial and MonteCarlo objects.

5.6 Heuristics

As illustrated, full simulation of a deck's strategy is NP complete. However, there are several more complex gameplay elements which are in theory modellable in LandFill's simulator, and would lead to simulations that are more representative of a game's actual behaviour. However, as the simulation is trying to maximize mana expenditure instead of win simulated games, its decisions are not necessarily representative of the exigencies of a real game anyway. For the initial development covered in this writeup, the below heuristics have been adopted; all offer an opportunity for future improvement of the product.

Spells are only cast via the mana cost of their first playable face, and not via any alternate mana costs - as mentioned, many spells are in fact two spells, either of which may be played; many more cards have alternate casting costs listed in their textboxes, where casting them for a different cost changes the behaviour of the card on cast. Ideally, LandFill would treat these both as seperate spells when assinging combinations in the hand, and denote a card as cast if either of its options are played. However, in addition to the difficulty of parsing the textboxes of cards with multiple costs, representing some cards in the hand as two cards which cannot both be played in the same lump adds an extra layer of complexity to Lump creation, and has not been implemented. *Spells with X in their mana cost are cast for X=1* - a spell with mana cost GX may be cast for one Green mana plus any amount of generic mana, usually acruing more value to the player for a higher value of X. This is complicated to include in lump combinations; moreso in the case of cards with multiple values of X (a card costing XX being includable in any lump that leaves an even

quantity of leftover mana); therefore, X is always assumed to be 1. *LandFill does not consider what the next playable spell with the addition of new lands will be* - even ignoring overall deck strategy, there is potentially room to expand LandFill's forethought. Consider the below hand with L=0:

[Land(BW), Basic Island, Basic Island, Spell(BBWW4), Spell(UU)]

Since the BW land removes more colours of mana from the hand, it would be played by the simulator. However, the spell requiring BW had a CMC of 8 and will not be played for some time; playing the Basic Island, however, would allow playing the UU spell on the following turn. Since it is necessary for the Simulator to assess the castability of Lumps rather than Spells, planning for future turns is difficult, and indeed situations like this may only be coverable via the progressive addition of heuristics. A future refactor may replace the hand object with a "queue" of spell combinations of progressively higher mana costs, adding each newly drawn card to this queue.

Ramp and Draw - many spells draw additional cards, while some provide additional mana. Doing so would potentially be a very fruitful area of development, and will be discussed in the conclusion. However, in addition to the difficulties of coding ram pand draw spells, introduction of these factors bring complex strategic considerations - it may, for example, be advisable in some situations to spend less than L + 1 mana on a ramp spell to allow for greater overall expenditure later; it is also typically advisable to play draw spells before playing lands if possible, to see if a superior land is added to the hand.

5.7 More Complex Lands

Some land cycles required significantly more complex simulation logic, outlined below. For sample card text for the respective cycles, see 2. HEY GIRL ADD DOUBLEFACED LANDS TO YOUR CYCLES.

5.7.1 Fetch Lands

During assessment of lump castability, a Fetch Land is considered to produce mana of color C mana if the deck currently contains a land for which it can search that can produce C, and that searchable land would return `land.enters_untapped(game) = True` for the current game state. The search itself happens is set when the fetch land is tapped for mana, at which point the land is moved to the graveyard, and calls the `game.filter_by_most_produced()` method from the current game object to narrow down the lands it is capable of fetching, adding an extra argument to specify that it the method must also account for pips in the hand currently accounted for by additional lands in the *hand*, rather than just on the battlefield. This encourages the searchland to make new colours of mana available.

While production of WUBRG by a Fetch Land has the standard cost of 2 in the Linear Assignment, production of "Gen" is weighted at cost 1, and "None" has cost 0, forcing the Lump to map its pips from toher lands if possible. This This prevents the SearchLand from being forced to search for a colour it does

not need to in order to pay a generic cost, and allows it to, as much as possible, find the best land for the current hand rather than just for the spell.

To ensure that the Fetch Land searched a land even on a turn when no lump is played, an additional "null lump", containing no spells, is created each turn, forcing the Fetch Land to provide a single "None" mana.

5.7.2 Cycles that Produce Multiple Mana Per Tap

Such lands are given, at substantiation, two "SubLand" objects, each of which is a Land child class, attributes of which are set via intiatialization parameters rather than a passed ORM model. A Land is replaced with its SubLands prior to its inclusion in a Linear Assignment calculation.

5.7.3 Filter Lands

A Filter Lands produces C and has two sublands, each of which may produce either of its colors. In order to account for a situation in which multiple filter lands may pay for the abilities of others, the following recursive algorithm is used:

- If a Lump is castable on a battlefield in which Filter Lands are tapped for C, it is cast.
- If not, filterlands are placed into a new list, $\underline{L}_f$. A permutation of this list is lazily generated via `itertools.permutations()`, from first Filter Land $F(1)$ to nth Filter Land $F(n)$.
- A new list, $\underline{L}_n f$, containing all non-Filter Lands is generated, including the subset of lands capable of paying $F(1)$'s cost, $P(1)$ to $P(n)$. $P(1)$ is removed from $\underline{L}_n f$, and the sublands of $F(1)$ are added.
- Repeat steps 2 and 3, starting with $\underline{L}_n f$ each time, until all Filter Lands have been either replaced with their sublands, or tap for colorless if there is no land capable of paying for them, with $F(n)$ being the final one so replaced.
- If the Lump is castable with the resulting list of lands and sublands, cast it.
- If it is not, return the land $P(1)$ that had been removed at the recursion level when the sublands of $F(n)$ had been added, and remove $P(2)$. Assess the castability of the Lump again, casting it if possible.
- If no lands $P(1)$-$P(n)$ in the list at the recursion level of $F(n)$ allow for the lump to be played, return to step six for the list at the recursion level of $F(n-1)$.
- If the recursion level of $F(1)$ is reached, generate a new permutation with new values of $F(1)$ and $F(n)$

Although this requires testing a large number of permutations if a lump is not castable, it does not provide a significant performance bottleneck.

5.7.4 Dual-Faced Lands

A Dual-Faced Land is given two sublands, and an attribute, `land.committed`, initialized to None. When tapped for "Gen" or "None", if `land.committed = None`, a subland is selected via (game.filter_by_most_produced()) in a similar manner to Fetch Lands. When tapped for colored mana, `land.committed` is set to the subland capable of producing tha tcolour. If `land.committed != None`, the land is tapped as though it were the subland assigned to that attribute. The Battlefield object sets `land.committed` to None when it gives the land to another cardCollection. Like Fetch Lands, a Double Faced Land has a lower weighting to tap for non-Generic during linear assignment.

6 The Optimizer

6.1 Constituent Classes

6.1.1 MonteCarlo and Trial

Recall that these respectively are subclasses of CardCollection and Simulation. Their relationship may be modelled as so: MonteCarlo produces a deck, creates a single Trial object for that deck, uses that Trial's `trial.run()` method to run many games, and then accesses performance data for those games via the Trial's attributes.

6.1.2 LandPrioritization

The LandPrioritization class simultaneously ameliorates three problems in the design:

- As will be discussed in FORTHCOMING SECTION, fluke results in which a land performs equivalently to a strictly better one, as long as the betterness is narrow, are not uncommon.
- As noted in 3.2.6, some lands are balanced via a life point penalty, and that this is difficult to quantify within an optimization context.
- Since LandFill uses a Hill Climbing algorithm, at each increment, a deck may be tested as many as N times, where N is the number of neighbouring decklists (ie, decklists with one card different). This is time consuming.

It is useful, therefore, to encode "strict betterness" in such a way that a land is never tested against a strict superior capable of producing its colours at any hill climb increment, and that equivalent lands that differ only in life point commitment are considered strict superiors/inferiors to each other.

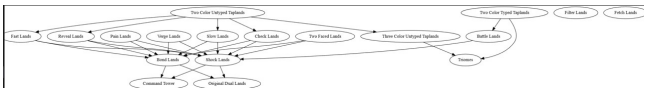


Figure 10: Digraph of strict betterness relationhips between all cycles currently handled by LandFill, where each arrow points to the superior land. Note that some labels (ie, Three Colour Untyped Taplands) refer not to single cycles but to groups of mechanically equivalent cycles which differ in ways that do not pertain to mana generation

MonteCarlo is therefore given a LandPrioritization object, which maps out strict superiority/inferiority relationship. Land A is considered is considered strictly better than Land B if it is capable of producing all colors that Land B does and either:

- Never enters tapped, while Land B sometimes does.
- Sometimes enters untapped, while Land B never does.
- Requires a life point investment that Land B does not.
- Has basic landtypes, whlle Land B does not.
- Always produces its colours, while Land B only produces some of them under certain conditions.

Strict Betterness is mapped in 10

Land objects belonging to the MonteCarlo are "registered" with the LandPrioritization. After all have been registered, each land is assigned its strict betters according to its cycle. While this was initially modelled as a simple dict object comparing a given cycle to its superiors, a more complex class was deemed necessary to accommodate situations in which a player requested a cycle with both inferiors and superiors be removed from consideration; in such cases, LandPrioritization "cascades" through, applying the superiors of the removed land to its inferiors.

6.2 Choice of Optimization Algorithm

(EG hill climb, simulated annealing) During development, the use of an Evolutionary Algorithm, a Simulated Annealing algorithm, and Steepest-Ascent Hill Climbing algorithm were all considered, with the latter being ultimately chosen. This subsection will go through each of these methods and outline reasons for their inclusion/exclusion.

6.2.1 Steepest Ascent Hill Climbing - Chosen

LandFill improves on the "Simple" Hill Climb algorithm set out in 2.2.2 by implementing instead a "Steepest Ascent" Hill Climb, also known as "method

descent." At every increment, rather than adopting as a new value for $\underline{x}_c$ the first input in its neighbourhood $N(\underline{x}_c)$ to return a higher value, as in Simple Hill Climbing,, it examines all inputs in the neighbourhood and chooses the one that offers the greatest improvement over the current value $\underline{x}_c$. In the context of LandFill, this means that at every increment, all possible combinations of lands to remove and all possible combinations of lands with which to replace it are explored.

The size of the search space at each increment is equal to

$$N \times M$$

where N = number of non-mandatory lands in the deck, and M = number of lands in the MonteCarlo with all strict superiors in the deck.

6.2.2 Evolutionary Algorithm

6.2.3 Simulated Annealing

Simulated Annealing, like Steepest Ascent Hill Climbing, is a neighbourhood

6.3 Choice of Performance Metrics

6.3.1 Overview

To test different performance metrics, four sample decks containing the same nonland cards in colours BUG were submitted to Trial objects outside the context of the Hill Climbing algorithm:

- BasicDeck - a deck containing only basic lands in the deck's colours.
- PartialDeck - a deck containing basic lands, shock lands and fetch lands, representing a deck part way through the optimization process.
- OverDeck - a deck containing no basic lands, reflecting hypothetical overapplication of the optimization principles.
- ExpectedDeck - a deck whose manabase was chosen by me, reflecting what I as a user of LandFill might expect an optimized deck to look like.

6.3.2 Use of "Wasted Mana"

In his analyses, referenced in 2.4, Karsten proposed three underlying assumptions [6] for assessing the castability of a spell of mana cost M:

- We want to cast the spell on turn M.
- We condition on drawing at least M lands by turn M.
- There are a realistic number of lands in the deck

Deck	Mean	Median	Mode	Range	Kurtosis
BasicDeck	2.696	1.0	0	21	2.589267
PartialDeck	1.479	0.0	0	18	8.076589
OverDeck	0.895	0.0	0	10	5.604503
ExpectedDeck	0.609	0.0	0	18	31.770379

Figure 11: Outputs in terms of wasted mana for each deck

To control for these factors, at the conclusion of each turn, the Simulator identifies $C(l)$, the combined CMC of the largest lump determined that turn (either cast or not) whose value is equal to or less than the number of lands on the battlefield at the turn's conclusion, and $C(c)$, the combined CMC of the lump that was cast this turn. The game tracks the accumulated value of $C(l) - C(c)$ each turn to determine W, the total wasted mana per game. This penalizes lands which enter the battlefield tapped, as they permit larger value of $C(l)$ without increasing the maximum value of $C(c)$, as well as game states in which there is an insufficient range of colours on the battlefield, without penalizing the curve of the player's deck relative to the number of lands they selected.

Focussing on mana *wasted* rather than mana *spent* is not without downsides. Bounce Lands, for example, are a cycle of lands which enter tapped and produce two mana per tap, requring in exchange the return of an already-played land to the player's hand. They perform strongly on a deck that has purposefully chosen to run fewer lands, as they effectively count for two lands in one draw, an advantage detectable only if the amount of mana *spent* per game is assessed. LandFill will not be in a position to recommend Bounce Lands for small manabases, although as Bounce Lands are fairly unpopular in the contemporary commander scene [**BounceLands**] this is considered an acceptable sacrifice. While presumably after sufficient games, mana expenditure would eventually converge on a meaningful value, tests of the aforementioned decks using mana expenditure as the metric returned minimal performance differences over multi-hour long trials.

6.3.3 Initial Analysis

Data from a trial of four decks, summarized in Figure 11, prompts several immediate observations. First, the modal wasted mana is consistently zero, while BasicDeck's median value of 1 is the highest of any deck, indicating that even at a comparatively low level of optimization, any performance considerations that are not simply the probability of wasting no mana in a game are only relevant in a minority of cases. Second, while the decline in mean wastage between PartialDeck and ExpectedDeck is comparable in size to the mean decline between BasicDeck and PartialDeck, the rate of increase of Kurtosis increases dramatically at higher levels of optimization. Third, such measures of central tendency hint that OverDeck may be a more viable deck than ExpectedDeck, having only a slightly higher mean wastage but a significantly lower range (wasting at most 10 mana per game, as opposed to ExpectedDeck's 18).

Examining the histograms of ExpectedDeck and OverDeck's mana wastage

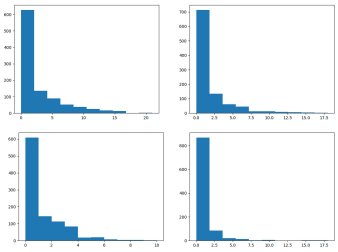


Figure 12: Histograms of the mana wasted by each deck: clockwise from top left: BasicDeck, PartialDeck, OverDeck, ExpectedDeck

clarifies this (see Figure 12). Note that between PartialDeck and OverDeck, the proportion of games with zero mana wasted declines significantly, increasing again for ExpectedDeck. This indicates that while the mean wasted mana between ExpectedDeck and OverDeck may not favour ExpectedDeck by a wide enough margin to offset the reduced risk of extreme mana wastage, OverDeck's reliability does come at the cost of a non-negligable reduction in the likelihood of a game wasting zero mana at all. Given that, as mentioned, zero mana is wasted in a plurality of games even at a low level of optimization, an amelioration of suboptimal results at the cost of optimal results is not considered a worthwhile exchange. LandFill therefore assesses a manabase based on the probability, henceforth P, of it not wasting any mana at all in a game.

6.3.4 Proportion Wasted vs Cumulative Distribution

One potential improvement over P as a metric would be C, the area under a Cumulative Distribution Function (CDF) of wasted mana per game. In such a metric, a decklist with a lower value of P would be penalized compared to one with a higher value, but two decks with equivalent values of P may return different scores depending on their probability of producing significantly suboptimal games.

To investigate this, two additional decks were prepared: CheckDeck and ShockDeck, both identical to BasicDeck save for the replacement of a single Forest with a UB Check Land or a UB Shock Land (see Figure 2), respectively, representing decks after a first Hill Climb increment. Wheraes OverDeck is unlikely to ever be assembled by LandFill in its normal course of operations, if one metric was better able to distinguish the slight superiority of ShockDeck (which need never have a land enter tapped) to CheckDeck (which, on occasion, will), and distinguish the slight superiority of both to BasicDeck, that would be a stronger metric to use. I conducted 100 Trials of 1000 games each for all decks, summarizing the same data first with P and second with C. Box plots of the results are shown in 13

Disappointingly, neither of these methods prove more able to capture subtle performance differences at the increment level; use of P seems slightly more

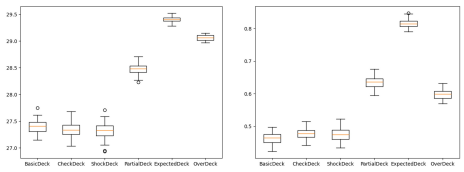


Figure 13: Box Plots representing the same Trial output data as (left to right): proportion of games in which zero mana is wasted, and area under a CMF of mana wasted.

capable, but given the overwhelming overlap between the whiskers of BasicDeck, CheckDeck and ShockDec, this does not seem to be significant. However, with the exception of OverDeck, which is significantly more harshly penalized by P, there is not a significant difference between P and C in the proximity of the whiskers between decks that have multiple cards difference between them, suggesting that the additional information provided by C does not produce a meaningfully more sensitive assessment.

Given this, it is worth noting that P presents several design advantages. First, it is faster: given that any value of wasted mana greater than 0 is equally discrediting, games can be abandoned the second any mana is wasted. Secondly, it is a simpler metric. While this will be discussed in more depth in 9.2.1, this investigation thus far has been carried out under the assumption that, as these metrics are done at a level of statistical abstraction that does not translate neatly into MTG strategy, the criterion is built into the code and not customizable by the user. Accepting this, it is important to nonetheless explain to the user the criterion, and a user should not be expected to have an understanding of Cumulative Frequency Analysis in order to understand by what metric the produced decklist is considered "optimized". That a deck has been found to have a certain probability of wasting zero mana in a game is much more comprehensible.

6.4 The Hill Climbing Algorithm

6.4.1 Setup

Once the user has selected preferences, the Deck and MonteCarlo objects are instantiated. Since both are CardCollection objects, GameCards are assigned to each accordingly:

GameCard representation of..	Assgned To	`GameCard.mandatory`
All nonland cards	Deck	True
All land cards in the player's input (ie, utility lands)	Deck	True
Any land cards that the player selected from LandFill's input screen to definitely include	Deck	True
All Land cards in a recognised cycle that are not already in the deck	MonteCarlo	False
100 basic land cards of each colour required by the deck	MonteCarlo	False

Basic Lands, rather than a random sample of nonbasic lands, are used as the starting input thanks to the principle of diminishing returns set out in 2.1.2.

6.4.2 Each Increment

6.5 Limitations

7 Frontend Design

7.0.1 Page 1

In the initial mockup, player information needed by LandFill was divided into two categories: *deck-intrinsic* information, a change in which would fundamentally re-alter the deck in question (ie, the input cards, the format, the number of lands) and *deck-extrinsic* information, which could be tweaked if a player was not happy with a provided decklist (ie, price, willingness to pay life, cost, selection of land cards). This was a poor dichotomization: with the specialization on the Commander format, most deck-intrinsic information became irrelevant (as the format is alwasy commander, and the number of lands is always 100 - the number of input cards), while many extrinsict factors proved impossible thanks to the knapsack problem. Moreover...

Operation of LandFill is sequenced as follows: a player inputs a decklist of non-land cards, a set of candidate land cards are produced, and the MonteCarlo simulation is run. In the initial mockup, in which all of LandFill was housed on one page, players typically opted to run the simulation before confirming their decklist. In the re-design, therefore

In the intial mockup drafted in 3, LandFill consisted of a single homepage, and users were generally confused

7.1 Parsing Inputs

In 3, users desired that LandFill support "round trips" (in the words of one subject), in which a list of cards was copied from a Deck database such as Tappedout or MoxField, into LandFill, and could then return the outputs without any additional formatting. For this purpose, I created the InputParser class,

of which one is created at the start of each session. Test users mentioned four such databases: TappedOut, Moxfield, Archidekt and Deckbox. since Tappedout and Moxfield both allow for lists to be copied from the deck homepage rather than from the formatted export panel, and Moxfield and Deckbox take inputted cards in the same format, this means that the InputParser has been designed to distinguish between six possible input formats, and return three possible output formats.

Since Tappedout and Archidekt both support categorization of cards (ie, draw, ramp, win strategy, enemy card removal) with custom lables, the Input Parser stores all cards in a Dict object corresponding to each category, using a default key if none are specified by the user. On completion of the Monte Carlo process, the InputParser a total of six string objects: for each database format (Deckbox and Moxfield being identical), it returns the total decklist, including categories if provided by the player, and the list of provided lands.

7.2 Exclusion/Inclusion Criteria

8 Post-Development User Tests

9 Conclusion

9.1 Findings and Product Viability

9.2 Future Development

9.2.1 User Choice

this is the HCI stuff you've asked Chatgpt about, look into the papers it's mentioned.

9.2.2 Maintenance and additional cycles

9.2.3 Relevant one-off lands

9.2.4 Ramp and Draw identification

Spells that ramp and draw are significant in Commander, and their lack of presence in LandFill represents a significant deviation between the Simulator and gameplay. The impact that this has on manabase decisions is non-trivial: many ramp spells reward disproportionate investment in Forests, while draw spells reduce the chance of drawing a suboptimal land at any time. Thoroughly modelling ramp and draw cards is essentially impossible: even aside from the number of extant spells, and the sequencing difficulties outlined in 9.2.4 , exact draw mechanisms are complex - consdier, for example, "Rhystic Study", one of the most popular draw spells in commander, which draws cards contingent on spells cast by the opponent.

One way to introduce ramp and draw would be to have the player identify ramp and draw spells in their deck, and give them a set of options, ie: "X card draws (1/2/3/...) cards (When played/at the start of turn/whenever you cast another spell). While this would not address the sequencing issues, it may lead to more deck-specific manabase assessments. Such an approach could also be a viable way to model other strategic considerations: IE, "counterspells" (cast in response to an opponent casting a spell) could be marked as such to suggest that the deck may have no reason to cast them on an early turn, when threats are small. However, this would introduce significant frontend complexity, and given that LandFill's optimization algorithm returns a viable automatic manabase rather than a consistently optimal one, I consider incorporation of strategic minutiaie like that to be non-essential for the initial launch.

9.2.5 Accounting for Ramplands/Bounce Lands

9.2.6 Simplification

This could be a more precise interface that allows players to suggest cycles they like and store them, while including a basic land optimizer; it could ALSO be used in "diminishing returns"

9.3 Evaluation

References

[1] Chris Alvin et al. "Toward a competitive agent framework for magic: The gathering". In: *The International FLAIRS Conference Proceedings*. Vol. 34. 2021.

[2] Alex Churchill, Stella Biderman, and Austin Herrick. "Magic: The gathering is Turing complete". In: *arXiv preprint arXiv:1904.09828* (2019).

[3] Alexander Esche. *Mathematical Programming and Magic: The Gathering®*. Northern Illinois University, 2018.

[4] Frank Karsten. *Mana Bases in Ixalan Standard*. 2017. URL: `https://web.archive.org/web/20200330115445/https://www.channelfireball.com/articles/mana-bases-in-ixalan-standard/` (visited on 08/13/2025).

[5] Frank Karsten. *Should You Play Tapped Duals in 2-Color Limited Decks?* 2018. URL: `https://web.archive.org/web/20201109011137/https://www.channelfireball.com/articles/should-you-play-tapped-duals-in-2-color-limited-decks/` (visited on 08/13/2025).

[6] Frank Karsten. *What's an Optimal Mana Curve and Land/Ramp Count for Commander?* 2025. URL: `https://www.tcgplayer.com/content/article/What-s-an-Optimal-Mana-Curve-and-Land-Ramp-Count-for-Commander/e22caad1-b04b-4f8a-951b-a41e9f08da14/` (visited on 08/12/2025).

[7] Nicholas Metropolis and Stanislaw Ulam. “The monte carlo method”. In: *Journal of the American statistical association* 44.247 (1949), pp. 335–341.

[8] Beth Moursund. *Playing Your Pet: Rough-Testing a Magic Deck*. 2010. URL: https://web.archive.org/web/20100902160143/http://www.wizards.com:80/Magic/Magazine/Article.aspx?x=mtg/daily/feature/106 (visited on 08/12/2025).

[9] Edward Allen Silver. “An overview of heuristic solution methods”. In: *Journal of the operational research society* 55.9 (2004), pp. 936–956.

[10] Colin D Ward and Peter I Cowling. “Monte Carlo search applied to card selection in Magic: The Gathering”. In: *2009 IEEE Symposium on Computational Intelligence and Games*. IEEE. 2009, pp. 9–16.

[11] Peter C Wright and Andrew F Monk. “The use of think-aloud evaluation methods in design”. In: *ACM SIGCHI Bulletin* 23.1 (1991), pp. 55–57.

[12] Allen Wu. *Using Monte Carlo Simulation to Improve Your Manabases*. 2018. URL: https://article.hareruyamtg.com/article/article_en_405/?lang=en (visited on 08/13/2025).