True Trading Card Game (TTCG)
Official Documentation For Development

Creator: Antonius Torode

Version – 0.063
Latest update: April 13, 2025

The Original Maintainer of this work is: Antonius Torode.

The Current Maintainer of this work is: Antonius Torode.

This document is designed for the sole purpose of development of TTCG.

This document is continuously under development.
Most Current Revision Date: April 13, 2025

Torode, A.
True Trading Card Game (TTCG) –
Official Documentation For Development
2025.
Includes Source Code and References
ISBN: NONE

Contents

This page intentionally left blank.

Introduction and Concept Design

A trading card game (TCG) is a popular game format where players acquire cards primarily through purchasing packs or trading with others. Players build customized decks from their collection, aiming to create synergistic combinations of cards that offer strategic balance. These decks serve as personalized tools for competitive play, where players test their strategies against opponents in dynamic, ever-evolving matches. As the name suggests, these are card games which should inherently emphasize trading between players. However, most TCGs in practice operate as product-driven systems where cards are primarily obtained through purchasing randomized packs or individual cards from retailers. The trading aspect is often secondary, with card value dictated by artificial scarcity, power creep, and secondary market speculation which essentially shifts the focus from player-driven exchange to a collectible gambling model.

This distinction between what a trading card game should be and what they typically are game me the idea for conceiving a *true* TCG (or TTCG for short). This would be a card game where trade was emphesized, encouraged, and required in order for players to get the cards they need. To get a game and system setup for this, it's useful to analyse current games and find solutions to the problems that shift away from this philosophy.

Some of the major trade-limiting factors of modern games follow. In order to facilitate a *true* TCG, many of these need solutions. Below is a brief outline of these ideas as well as ideas to incorporate within TTCG in order to prevent these issues.

- Many common cards in TCGs have such a high abundance, that players who purchase booster packs have a large number of them, which makes trading for them unneccesary. One reason is that the booster packs are released in a way where certain cards can only be found in specific booster packs (they are printed in sets).
  - TTCG will have the feature that every card has the same number of copies in existance. The cards will all be equally abundant which prevents people from having a large accumulation of a specific card, and on the contrary, a larger array of cards to facilitate the need for trading. TTCG can also solve this by creating one 'pack' which has all possible cards in it. This would prevent getting many copies of the same card, as you have a chance to get any existing cards. It also removes the need to analyze different packs and try to get the specific ones you need.
- Key stable cards or cards of higher rarity are often limited and therefore worth a higher value than most other cards. This makes trades with them very uncommon.
  - As mentioned above, each card will have the same number of copies, eliminating the artificial limits created by reducing the number of prints of a specific card. At the same time, all cards will exist in all rarities, where each rarity is printed as often as all others - creating a flat chance for any pack to have them and providing a common limitation for everyone.
- The cards themselves are all valued based on their use in the current meta (strongest for battling) formats and decks. Certain cards are good for a short time and are therefore worth more, making people not likely to trade them. In contrast, many cards are deemed of less value when they are not in this status, making trading for them undesireable.
  - TTCG will solve this by creating only cards that work with any other cards of a type. The cards will have fixed maximum strengths and similar effects which can then be collected and combined in new and unique ways. New cards will have similar level effects that can work with all previous cards. TTCG will have creature cards that have a fixed sum of atack and defense points such that no new cards will be considered outclassing to prior cards in existance.
- Many TCGs sell pre-made decks or structure decks that remove the need to trade or collect cards organically, offering competitive-ready cards directly from the manufacturer.
  - TTCG decks (if it is decided that they will exist) will create baseline level decks and require the packs and trading to bolster with other cards. The decks will provide similar cards to

create a base for a player to start with, but will not provide a fully synergistic deck that provides stragetic advantage over other players.

- Frequent set rotations or constant new releases incentivize continuous buying rather than fostering a stable, long-term trading ecosystem.
  - The newly released cardsof TTCG will be equally mixed in with any previous cards - maintaining the philosophy that every card will have an equal amount of prints. The cards will be released in a manner where you will not know which cards are new or old, but rather have a constant stream of new cards that you come across.
- Companies intentionally design sets with a few highly desirable cards to drive pack sales, making acquiring cards through trading inefficient compared to buying.
  - All cards of TTCG will be similar, and have enough variation in their type connections, that all cards could be used with some deck structure, or strategy.
- Modern card rarities are pre-determined by the publisher and fixed across all packs, limiting the organic value assessment of cards by the player community.
  - All cards of TTCG will have a fixed number of every rarity.
- Frequent reprints of popular cards by publishers devalue previously rare or sought-after cards, discouraging long-term trading and collecting.
  - TTCG will not reprint any card. Rather, the cards will be printed to facilitate a set number of cards in existance equal to all other cards.
- High-end collector boxes with alternate art or foil versions, aimed at investors rather than players, further pushing the game toward a collectible model instead of a trading one.
  - TTCG will not have this. Instead, there might be 'lore' cards, which serve no competative advantage, but rather simply explain the lore of various cards. These cards will always be introduced and available in a setting where the normal cards are also given, which will further facilitate the trading ability.
- The randomized nature of packs mirrors gambling systems, encouraging excessive spending with little guarantee of acquiring desired cards.
  - Since there will not be a specific chance for getting currently meta or high level cards, each pack will be considered equal chance of everything and this will hopefulyl remove the aspect of gambling. Rather, by collecting packs, you simply increase your amount of cards and potential for trading to get the cards you desire.
- Tournament entry prizes or exclusive promo cards available only through organized play events, creating a two-tiered system where certain cards are locked behind pay-to-participate systems.
  - TTCG will not have promo cards.

1.1 Other Popular Trading Card Games

This is a very brief overview of some of the popular trading card games (TCG) with some pros and cons of each.

1.1.1 Yu-Gi-Oh

The Yu-Gi-Oh TCG was originally based on the anime series "Yu-Gi-Oh!", the card game was first released in 1999 by Konami in Japan. Players use a deck of cards to summon monsters, cast spells, and set traps to reduce their opponent's Life Points from 8000 to 0. The game is turn-based, with players drawing cards, summoning creatures, and activating effects to battle or control the field. The cards are split into three main types: monsters, which have attack and defense points for battling; spells, which offer various effects; and traps, which are hidden and activate under specific conditions. Summoning methods have expanded over time to include Normal Summoning, and more complex Special Summons like Fusion, Synchro, Xyz, Pendulum, and Link Summoning, each with its unique requirements. Deck construction involves a main deck with at least 40 cards, an Extra Deck for specialized monsters (up to 15 cards), and a Side Deck for between-duel adjustments (up to 15 cards). The game supports different formats like Advanced, which uses a banlist to manage overpowered cards, and Traditional, which is less restrictive. Strategy in Yu-Gi-Oh! centers around deck synergy, anticipating opponent moves, and managing resources effectively. The game has evolved through numerous card sets and mechanics, maintaining a dynamic meta-game where various deck archetypes cycle in and out of popularity, supported by a strong community and competitive scene including events like the World Championships.

Issues

Here are some commonly cited issues with the modern Yu-Gi-Oh! card game:

- Complexity of Cards: Cards have become increasingly complex with lengthy text descriptions, leading to confusion and requiring players to interpret card effects, which can vary based on punctuation or syntax.
- Turn Length: Games often involve long turns where one player performs a series of actions that can take several minutes, reducing interactivity as the other player waits.
- Power Creep: There's a significant escalation in card power over time, making older cards obsolete and necessitating frequent deck updates to remain competitive.
- Special Summoning Mechanics: The introduction of multiple new summoning methods (like Synchro, Xyz, Pendulum, and Link Summoning) adds layers of complexity that can be overwhelming for new or returning players.
- Hand Traps and Negation: The prevalence of hand traps and negation effects can make games feel one-sided, where one player might not get to play effectively if they don't have the right responses in hand.
- Game Balance: There's a perception that the game lacks balance due to the continuous introduction of new, powerful cards without effective rotation or a robust banlist that addresses all issues.
- Cost and Accessibility: The game can be expensive to stay competitive due to the rarity of key cards and the need to constantly update decks. This also affects accessibility for new players.
- Archetype Dependency: The modern game heavily revolves around archetypes or "series" of cards that have specific names or sub-types, which are designed to work synergistically with each other. This means players are often forced into investing in a narrow selection of cards to build a competitive deck, ignoring the vast majority of cards that don't fit into these specific archetypes. This can lead to:
  - Reduced Deck Diversity: Players are less inclined to experiment with a wide variety of cards since non-archetype cards might not provide the same level of strategic depth or interaction, leading to repetitive gameplay centered around a few dominant decks.
  - Barrier for New Players: For newcomers, the game can seem daunting not just because of the mechanics but also because of the need to invest in very specific cards to compete, which can be overwhelming both financially and in terms of game knowledge.
- Rarity and Price: Key cards, especially those that define or enhance archetype decks, are often released in high rarity, which significantly increases their cost. This makes keeping up with the meta-game expensive, especially as new sets introduce cards that can shift the competitive landscape.
- Lack of Official Rulings Database: Without an official database for card rulings, players and judges must interpret card text during tournaments, leading to potential inconsistencies and disputes.
- Short Game Duration with High Complexity: Matches can end quickly (sometimes in 1-2 turns), yet the complexity within these turns can be daunting, not matching the entertainment value of longer, strategic games.
- Format Issues: There's no official alternative format supported by Konami that would cater to different play styles or levels of complexity, potentially alienating players who prefer a less intense gameplay experience.

These issues are often discussed by the community, with various opinions on how much they impact the enjoyment of the game or its competitive scene. Some players appreciate the strategic depth these elements add, while others find them to be barriers to entry or fun.

Positives

Here are some of the positives of the Yu-Gi-Oh! trading card game, including aspects that were particularly beneficial before some of the more recent issues became prominent:

- Strategic Depth: Yu-Gi-Oh! offers a high level of strategic complexity. Players must think several moves ahead, manage resources, anticipate their opponent's strategies, and adapt on the fly, which can be intellectually satisfying.
- Deck Building Creativity: Before the archetype system became so central, players had more freedom to experiment with deck building. Even now, there's still room for creativity within archetypes or for players who enjoy building rogue decks.
- Community and Social Interaction: The game fosters a strong community through local events, online forums, and global tournaments. It's a social experience that brings people together, en-

couraging camaraderie and competition.
- Variety of Play Styles: There's a wide range of deck archetypes and strategies, from control to combo, from beatdown to stall, allowing players to find a style that suits their personality or play preference.
- Constant Evolution: The introduction of new cards and sets keeps the game dynamic. While this can contribute to some issues, it also means the game never gets stale, offering new strategies and mechanics to explore.
- Accessibility to Casual Play: For those not interested in the competitive scene, Yu-Gi-Oh! can still be enjoyed casually with friends or at local game stores where the pressure to keep up with the meta isn't as intense.
- Affordable Entry Point: While the competitive side can be expensive, the base game is accessible with structure decks or starter sets that provide a playable experience at a relatively low cost, especially for those just looking to enjoy the game casually.
- Cultural Impact: The game has a significant cultural footprint, with the anime, manga, and various adaptations contributing to a rich lore and fanbase, which adds to the enjoyment of playing.
- Educational Benefits: Playing Yu-Gi-Oh! can help with math skills (calculating damage, resource management), reading comprehension (understanding complex card text), and strategic thinking.
- Replayability: Due to the randomness of draws and the myriad of possible interactions between cards, no two games are exactly alike, providing high replayability.
- Global Events: The World Championships and other major tournaments give players something to strive for, offering not just competitive play but also a sense of achievement and community recognition.
- Art and Design: The artwork on Yu-Gi-Oh! cards is often praised for its quality and thematic richness, enhancing the aesthetic appeal of collecting and playing with the cards.

Before some of the modern complexities and cost issues became as pronounced, these positives made Yu-Gi-Oh! appealing to a broad audience, from casual players enjoying the thematic elements and simple battles to competitive players who thrived on its strategic depth. Even with its challenges, many of these positives still hold true for the game today.

1.1.2 Magic The Gathering

Magic: The Gathering (MTG), created by Richard Garfield and first published by Wizards of the Coast in 1993, is a pioneering trading card game where players assume the roles of wizards, casting spells, invoking creatures, and using artifacts to battle each other. In MTG, each player starts with a life total of 20 and aims to reduce their opponent's life to zero or achieve an alternative win condition like decking (making the opponent draw from an empty deck). The game uses a deck of at least 60 cards, divided into several card types: Lands for mana generation, Creatures for combat, Sorceries and Instants for one-time effects, Enchantments for ongoing effects, Artifacts for various utilities, and Planeswalkers for dynamic, player-like abilities. MTG's gameplay involves strategic deck construction and resource management, where players must balance the use of their mana to play cards at the right time. The game's depth comes from its vast card pool, multiple formats (like Standard, Modern, Legacy, and Commander), and the continuous introduction of new sets, which bring fresh mechanics and themes. Magic: The Gathering is renowned for its community, with organized play through local game stores, regional and international tournaments, and the Pro Tour. Its rich lore, intricate artwork, and the ability to blend strategy with storytelling have made MTG not only a competitive game but also a cultural phenomenon in the world of tabletop gaming.

Issues

Here are some commonly cited issues with the modern Magic: The Gathering card game:
- Complexity of Cards: Modern MTG cards often feature dense text with intricate mechanics (e.g., double-faced cards, mutate, companion rules), which can confuse new players or lead to rules disputes, especially with interactions that require precise timing or keyword understanding.
- Power Creep: Over time, card power levels have escalated (e.g., pushed rares in recent sets like Modern Horizons), rendering older cards less viable in competitive formats and pressuring players to constantly acquire new staples.
- Game Pace and Interaction: Formats like Commander can have excessively long turns due to combo-heavy decks or board states, while faster formats like Modern or Standard can end abruptly with little back-and-forth if one player draws the right answers or threats.

- Mana System Frustrations: The land-based resource system, while iconic, can lead to non-games due to mana flood (too many lands) or mana screw (too few), reducing player agency compared to games with less variance in resource access.
- Format Fragmentation: With numerous formats (Standard, Modern, Legacy, Commander, Pauper, etc.), players can feel overwhelmed choosing where to invest, and some formats (e.g., Legacy) have high barriers due to the Reserved List inflating card prices.
- Cost and Accessibility: Competitive play demands expensive decks, with staples like fetchlands, shocklands, or chase mythics (e.g., Ragavan, Nimble Pilferer) costing hundreds of dollars. Even casual Commander decks can spiral in price with desirable staples.
- Set Fatigue: The rapid release schedule (multiple sets per year, plus Secret Lairs and supplemental products) makes it hard for players to keep up financially and mentally, diluting focus and overwhelming collectors.
- Color Identity Dependency: Decks heavily rely on adhering to specific color identities or archetypes (e.g., blue for control, green for ramp), which can limit creative deck-building outside of established strategies and push players toward "solved" metas.
  - Reduced Deck Diversity: In competitive formats, the meta often coalesces around a handful of dominant decks (e.g., Izzet Murktide in Modern), discouraging experimentation with off-meta strategies.
  - Barrier for New Players: Newcomers face a steep learning curve not just with rules but also with understanding which cards are format-defining, requiring significant investment to catch up.
- Rarity and Price: Key cards are often printed at mythic rarity or in limited-run products (e.g., Secret Lairs), driving up costs and creating a pay-to-win perception, especially as reprints lag behind demand.
- Rulings Ambiguity: While MTG has a comprehensive rules database, niche interactions (especially in Commander or with older cards) can still spark debates, relying on judge calls or community consensus.
- Short Game Duration in Some Formats: Formats like Vintage or cEDH (competitive Commander) can end in 1-3 turns due to hyper-efficient combos, clashing with the expectation of interactive, strategic play.
- Lack of Casual Support: Official support leans heavily toward competitive play or Commander, leaving casual players without a clear, low-stakes format tailored to simpler gameplay or kitchen-table magic.

These issues are debated within the MTG community, with some players embracing the complexity and others feeling it alienates newcomers or shifts focus too heavily toward profit-driven design.

Positives

Here are some of the positives of Magic: The Gathering, including aspects that shine in its current state and historically:

- Strategic Depth: MTG's blend of resource management, timing, and bluffing creates a rich strategic experience. Players must balance offense, defense, and deck synergy, rewarding skill and foresight.
- Deck-Building Freedom: While archetypes exist, MTG's color pie and mana system allow for diverse deck construction. From janky brews to tier-one lists, players can express creativity across formats.
- Community and Social Bonds: MTG thrives on its global community, with local game store events (e.g., Friday Night Magic), online play via MTG Arena, and massive tournaments fostering connection and rivalry.
- Variety of Play Styles: Formats cater to different preferences—Standard for a rotating meta, Commander for multiplayer chaos, Legacy for eternal power, Pauper for budget brews—ensuring something for everyone.
- Constant Evolution: New sets introduce fresh mechanics (e.g., Duskmourn's eerie themes or Bloomburrow's creature focus), keeping the game dynamic while building on its 30+ year history.
- Casual Accessibility: MTG Arena offers a free-to-play entry point, and physical starter decks or preconstructed Commander decks provide affordable ways to dip into the game casually.
- Affordable Entry Point: While competitive play is costly, casual options like Jumpstart packs or precons keep the barrier low for new or returning players looking to have fun without breaking the bank.

- Cultural Legacy: As the first trading card game, MTG's lore—spanning planes like Ravnica, Innistrad, and Zendikar—adds depth, enhanced by novels, art, and a growing media presence (e.g., Netflix series rumors).
- Educational Value: MTG sharpens critical thinking (e.g., sequencing plays), math skills (e.g., combat calculations), and adaptability, often praised as a mental workout disguised as fun.
- Replayability: Randomized draws, diverse matchups, and format-specific metas ensure games rarely feel identical, offering endless replay value.
- Global Events: Pro Tours, MagicCons, and Commander-focused gatherings give players aspirational goals, blending competition with celebration of the game's culture.
- Art and Design: MTG's card art is a hallmark, with stunning illustrations (e.g., Rebecca Guay's classics or John Avon's lands) and thematic cohesion that elevate collecting and gameplay.

Historically, MTG's elegant core design and slower pace made it a standout, and even with modern challenges, its adaptability and community passion keep it a titan in the TCG world.

1.1.3 Pokemon

The Pokémon Trading Card Game (Pokémon TCG), launched in 1996 by Creatures Inc. and managed by The Pokémon Company, is a collectible card game tied to the iconic Pokémon franchise, blending simple mechanics with nostalgic appeal and strategic depth. Players build decks from a vast pool of Pokémon, Trainer, and Energy cards, aiming to knock out opponents' Pokémon and claim 6 Prize cards to win, with gameplay centered on attaching energy to power attacks and evolving Pokémon through stages. Its accessibility—rooted in straightforward rules and affordable starter decks—makes it a gateway for younger players and casual fans, while modern expansions like Scarlet & Violet introduce mechanics (e.g., Tera Pokémon, VMAX) that cater to competitive scenes. Backed by stunning card art, a global community, and events like the Pokémon World Championships, the TCG thrives on its franchise synergy, though it faces critique for power creep, archetype dominance, and high costs for meta cards, balancing simplicity with a dynamic, evolving meta as of February 21, 2025.

Issues

Here are some commonly cited issues with the modern Pokémon Trading Card Game:

- Simplistic Core Mechanics: While accessible, the game's reliance on basic attack-damage mechanics and energy attachment can feel less strategic compared to the layered complexity of competitors like Yu-Gi-Oh! or MTG, limiting depth for advanced players.
- Power Creep: Newer Pokémon (e.g., Pokémon V, VMAX, and ex cards) consistently outclass older ones with higher HP and damage output, making older decks obsolete and requiring frequent updates to stay competitive.
- Turn Length and Passivity: Early turns can be slow as players build up energy and evolve Pokémon, leading to passive gameplay where one player waits for the other to "set up," especially in mirror matches or against stall decks.
- Prize Card System: The prize card mechanic (winning by taking 6 prizes) can lead to swingy games where drawing a key card early or late disproportionately decides the outcome, reducing skill expression in some cases.
- Item and Supporter Dependency: Competitive decks heavily rely on specific Trainer cards (e.g., Boss's Orders, Professor's Research) for consistency and disruption, creating a bottleneck where games hinge on drawing these staples rather than Pokémon interactions.
- Cost and Accessibility: Meta-defining cards (e.g., rare Alternate Arts or Secret Rares) are expensive due to collector demand, and the lack of affordable reprints makes building top-tier decks costly, despite the game's kid-friendly image.
- Archetype Dominance: The game pushes specific Pokémon types or strategies (e.g., Charizard, Gardevoir) through synergistic support cards, limiting deck diversity and pressuring players to invest in a narrow pool of viable archetypes.
  - Reduced Deck Diversity: Tournaments often feature a handful of dominant decks (e.g., Lugia VSTAR, Miraidon ex), discouraging experimentation with rogue strategies or less-supported Pokémon.
  - Barrier for New Players: Newcomers face not only rule learning but also the need to acquire specific, often pricey cards to compete, overwhelming those without prior knowledge or budget.
- Rarity and Price: Chase cards like full-art Trainers or hyper-rare Pokémon are printed at low rates, inflating secondary market prices and tying competitive success to financial investment.

- Lack of Official Rulings Clarity: While simpler than some TCGs, edge-case interactions (e.g., Ability timing) lack a robust, centralized rulings database, leading to occasional confusion at events.
- Short Competitive Games: Top-tier decks can close out games in 3-5 turns, especially with fast setups like one-Prize attackers (e.g., Radiant Charizard), clashing with the expectation of evolving Pokémon over time.
- Format Rotation Issues: Standard rotation refreshes the meta but can alienate players who lose access to favorite cards, and the lack of a widely supported eternal format limits options for those wanting to use older collections.

These issues are often debated within the Pokémon TCG community, with some enjoying its simplicity and others wishing for more strategic nuance or affordability.

Positives

Here are some of the positives of the Pokémon Trading Card Game, including aspects that shine today and historically:

- Accessibility and Simplicity: The game's straightforward rules—attach energy, evolve Pokémon, attack—make it easy for beginners, kids, and casual players to pick up and enjoy without a steep learning curve.
- Deck-Building Creativity: While archetypes dominate competitive play, casual players can experiment with a wide range of Pokémon, Trainers, and strategies, especially with the variety of Pokémon types and evolution lines.
- Community and Social Appeal: Local leagues, online play via Pokémon TCG Live, and global events foster a welcoming community, blending nostalgia with competition for players of all ages.
- Variety of Play Styles: Decks range from aggressive single-Prize attackers to tanky VMAX stall builds to combo-heavy evolution strategies, offering options for different preferences.
- Constant Evolution: New sets (e.g., Scarlet & Violet expansions) introduce fresh mechanics like Tera Pokémon or reworked ex cards, keeping the game dynamic while staying true to its roots.
- Casual Accessibility: Preconstructed decks (e.g., Battle Decks, League Battle Decks) provide an affordable, ready-to-play entry point, ideal for casual fun or learning the ropes.
- Affordable Entry Point: Compared to MTG or Yu-Gi-Oh!, casual Pokémon play is budget-friendly with starter products, and Pokémon TCG Live offers a free digital option (though with in-game grind).
- Cultural Impact: Tied to the Pokémon franchise's massive popularity—spanning games, anime, and merchandise—the TCG taps into a nostalgic and vibrant universe, enhancing its appeal through familiar characters and lore.
- Educational Benefits: The game teaches basic math (damage calculation), planning (energy management), and reading skills (card text), making it a subtle learning tool for younger players.
- Replayability: Randomized draws, evolving board states, and matchup variety ensure games feel fresh, especially in casual settings with diverse Pokémon options.
- Global Events: The Pokémon World Championships and regional tournaments offer a prestigious stage for competitive players, blending prizes with community celebration.
- Art and Design: Pokémon cards are renowned for their stunning artwork—full-art cards, Secret Rares, and nostalgic reprints—making collecting as rewarding as playing.

Historically, the Pokémon TCG's simplicity and Pokémon brand loyalty made it a gateway TCG, and today, it balances kid-friendly charm with enough depth to sustain a competitive scene.

Basic Rules

2.1 Relevant Terms and Definitions

CARD Any valid card from the game.
DECK The collection of cards a player uses in the game.
HAND The cards a player holds from their deck.
DRAW The act of taking a card from the deck and adding it to the hand.
SHUFFLE The act of mixing a pile of cards.
DISCARD The act of moving a card from the hand to the discard pile.
DISCARD PILE The area where discarded, destroyed, or milled cards are placed.
SEARCH The ability to look through a deck or discard pile for a specific card.
MILL Sending cards from the top of the deck to the discard pile.
RECOVER The ability to retrieve a card from the discard pile or another out-of-play area.
FIELD The two-row area where cards are played by both players.
UNIT ZONE The upper row of a player's field, where unit cards are played.
SPELL ZONE The lower row of a player's field, where spell cards are played.
PLAY The action of placing a card on the field in an active position.
ACTIVE The top card of a stack on the field, considered in play and able to act.
TAP Indicate a card has been used or activated by switching its position from vertical to horizontal.
UNTAP Switching a cards position from horizontal to vertical (the opposing of 'tap').
RANK UP The act of placing a card of one rank higher on top of another card (also called Level Up or Promote).
UNDER Cards physically stacked beneath another card of the same type (e.g., unit under unit, spell under spell), often for layered effects.
BELOW Spell cards in the lower row supporting a unit in the upper row of the two-row field.
DESTROY The act of sending an active card to the discard pile.
TURN The period when one player can play cards and take actions.
POINT The unit tracking a player's score, starting at 20.
UNIT A card with Attack, Defense, and a non-spell type; rank determines its base stats (e.g., Rank 1: 500 total).
SPELL CARD A card of type "spell," typically placed below units to provide effects and stat boosts.
TYPE A primary card category: Water, Fire, Earth, Air, Nature, Electric, Light, Dark.
SUBTYPE A card subcategory: Dragon, Warrior, Beast, Spell-caster, Flying, Insect, Reptile, Humanoid, Machine, Fairy, Ghost, Aquatic, etc.
RANK The number of stars on a card, indicating its power level (also called Level).
ATTACK The offensive stat of a unit card.
DEFENSE The defensive stat of a unit card.
EFFECT Text on a card that performs an action within the game.
ACTION The act of performing a card's effect.
EQUIP The act of attaching a card to another for added effects.
COUNTER The act of negating or modifying another effect or action.
IMMUNE A status where a card is unaffected by specific effects.
TRIGGER A condition that automatically activates an effect.
CHAIN A method of resolving multiple effects in order.
INDESTRUCTIBLE A status where a card cannot be destroyed (also called Immortal).
EXACTLY A term requiring a condition to be precisely met for an effect to activate.
OWNING The player whose deck a card originated from.

GAIN The act of increasing a player's points or a card's stats.
LOSE The act of decreasing a player's points or a card's stats.
MODIFY The act of altering a card's stats, rank, or type.
PHASE A distinct segment of a turn (e.g., draw phase, play phase).
PLACE The act of positioning a card in a specific location (e.g., under another card).
PREVENT The act of stopping a specified action or effect from occurring.
ROW One of the two horizontal areas on the field (upper for units, lower for spells).
STACK A group of cards physically layered under another card of the same type (including the top card).
SWAP The act of exchanging the positions or stats of two cards.
DORMATN A card is considered dormant when it is physically under another card (not the top card of a stack).

2.2 Basic Rules

2.2.1 Deck Construction

- Each player must construct a deck with a minimum of 50 cards and maximum of 70 cards.
- A player can have no more than 2 copies of a card with the same name (across all rarities) in their deck.
- A deck may contain any ratio of unit and spell cards.

2.2.2 Game Start

- To decide who goes first, any random method can be used (e.g., coin toss, age, agreement). The default method for official play should be the roll of a die (assign one player as even, and another as odd).
- Each player begins the game by drawing 7 cards.
- If a player does not like their starting hand, they may shuffle it back into the deck and draw 6 new cards (mulligan). This can only be done once per game.
- The first player to play cannot attack on their first turn nor draw at the start of their turn.

2.2.3 Turn Structure

- Each player's turn consists of three phases:
  1. **Draw Phase**: At the start of every turn (except the first player's first turn), the player draws a card.
  2. **Main Phase**:
     - A player may play up to two cards to their field (unless a card specifies otherwise).
     - Cards played or ranked up by effects must respect the unit-stack and spell-stack field limits.
     - A card effect that plays a card to the field does not count toward this limit.
  3. **Battle Phase**: Combat and attacks occur.

2.2.4 Playing Cards

- A card can only be played to the field in face up vertical position.
- When a card on the field activates an effect, the card is tapped (swapped to horizontal position).
- Tapping indicates one of the two effects has been used, preventing the other until untapped.
- **Spell Cards:**
  - A Spell card effect can be used immediately after the card is played.
  - Spell card effects can be activated at any point during either players turn once the card is on the field.
  - If a spell card is activated in response to another action, a chain of effects occur.
- **Unit Cards:**
  - A unit card effect can be used immediately after the card is played.
  - Unit effects can only be activated on the owning players turn.

2.2.5 Card Effects

- Every card has two effects.
- All rules and effects are able to be overruled if a card effect explicitly states that it does so.
- Most effects can be activated at any time while the card is face-up on the field.
- If multiple effects trigger at the same time, they activate in reverse order of activation. If an effect is no longer valid because of this order, the effect simply does nothing (but still counts as activated).
- Tapped units cards cannot activate effects unless they specifically state otherwise.
- **Unit Cards:**
  - When a unit effect is activated, that card is tapped immediately, then the effect activates.
  - Tapped cards cannot activate effects.
  - A unit card can attack whether tapped or not.
- **Spell Cards:**
  - When a spell card effect is activated (even if it's is a continuous effect), that card is tapped immediately, then the effect activates.
- **Effect Conditions:**
  - If a card effect activates when the card is sent to the discard pile or deck, it can only be activated if the card was initially untapped.
  - Cards in the hand, deck, underneath other cards (dormant), and in the discard pile are considered untapped.
  - A card effect cannot be activated when (in response to) it is destroyed (unless a card states otherwise).
- **Effect Types:**
  - Various effects have different types which determine subtle differences in when and how they can be used.
  - A list of the various effect types can be found in section 3.3.1.

2.2.6 Leveling Up

- Any card with a rank higher than one can only be played to the field by ranking up another card of the same type.
- To rank up a card, place a card with the same type and one rank higher on top of that card.
- All cards under the card played are considered underneath that card and are immediately untapped. These are commonly called dormant cards.
- All dormant card remain on the field.
- Dormant cards cannot be tapped or have their effects activated, unless an effect states otherwise.
- dormant cards do not count towards the field unit limit.

2.2.7 Battle Phase

- The last part of a players turn is the battle phase.
- Cards cannot be played to the field during the battle phase.
- Every unit a player owns can attack once during the owner players Battle Phase.
- If the opponent controls any units, the attacker must target one of them.
- If an opponent has no units remaining, a unit can attack the opponent directly.
- When a unit attacks, the attack value is used to determine its strength.
- When a unit is defending, the defense value is used to determine its strength.
- During an attack, whichever unit has the higher attack prevails, and the other is destroyed (unless a card states otherwise) and sent to the discard pile.
- When a card is destroyed in battle or by an effect, the dormant card directly underneath it becomes an active card.

2.2.8 Gaining Points

- Every player starts the game with 20 points.
- To gain points, you must destroy units on your opponents side of the field.
- When a unit is destroyed, the owner of that unit loses points equal to the rank of that unit.
- During a players battle phase, if there are no units on the opponents side of the field, a unit can attack directly and the opposing player loses points equal to the rank of the attacking unit.

- Players can also gain or lose points based on various card effects.

2.2.9 Field Limitations

- A players field is split into two parts (upper for units, and lower for spell cards).
- Dormant cards (cards under other cards within stacks) don't count toward unit or spell limits.
- A players upper field can support a maximum of 5 unit card stacks.
- A players lower field can support a maximum of 5 spell card stacks.

2.2.10 Hand Size Limit

- Players may hold a maximum of 10 cards in their hand.
- At the end of their turn, if a player has more than 10 cards, they must discard down to 10. This occurs after Battle Phase, before the next turn begins.

2.2.11 Deck Exhaustion Rule

- If a player cannot draw a card because their deck is empty, they take 5 points of damage instead.
- This damage occurs at the start of their Draw Phase each turn that they are unable to draw.

Card Logistics

3.1 Card Types

There are two primary categories of cards - unit cards (or units for short) and spell cards (or spells for short). Every card has a card 'type' associated with it. Spell cards are differentiated in their function and purpose, but are simply a card with the type of 'spell'. The unit cards encompass all of the remaining card types. Every unit card also has one or more sub-type(s) (also called secondary type(s)), which is a more focused type specific to the card itself (primarily the art/lore of the unit).

Types

The primary types, of which each card is assigned one, are listed below with a brief description.

Earth: Units aligned with nature, resilience, and the physical world. Earth units often have high defense and abilities that enhance survivability.

Water: Units embodying fluidity, adaptability, and the power of the seas. Water units typically specialize in control effects and healing.

Fire: Units representing destruction, passion, and chaotic power. Fire units are known for high attack values and aggressive effects.

Air: Units characterized by speed, agility, and freedom. Air units often have evasion abilities and excel at quick, tactical strikes.

Electric: Units aligned with energy, and the power of electricity. Electrical units often have abilities focused on stunning enemies, or interupting other cards.

Light: Units symbolizing purity, order, and restoration. Light units frequently provide healing, support, and protective effects.

Nature: Units embodying the wild, growth, and the harmony of living things. Nature units often excel in versatility, with abilities that promote regeneration, summon allies, or manipulate the battlefield with natural forces.

Dark: Units associated with corruption, death, and forbidden power. Dark units excel at destruction, disruption, and sacrificial effects.

Spell: Non-unit cards that represent magical powers, blessings, equipment, spell effects, artifacts, curses, or more. Spells provide a wide range of one-time or ongoing effects to influence the game.

Sub-Types

The secondary types, of which each unit card is assigned one or more, are listed below with a brief description.

Avian: Winged creatures of the sky, known for their agility, keen senses, and mastery of aerial maneuvers.

Dragon: Powerful, mythical beings with immense strength and destructive capabilities.

Beast: Natural creatures with raw physical power, often found in forests and plains.

Elemental: Manifestations of elemental forces, bound to the magic of nature.

Aquatic: Sea-dwelling creatures adept at controlling water and adapting to fluid environments.

Warrior: Skilled fighters trained in physical combat, often acting as frontline attackers.

Spellcaster: Mystics and mages who wield magical powers to cast spells and manipulate the battlefield.

Machine: Artificial constructs built for labor, warfare, or magical purposes.

Ghost: Ethereal spirits bound to the mortal realm, often with abilities to haunt or possess.

Insect: Swarming, persistent creatures that thrive in large numbers and rapid reproduction.

Reptile: Cold-blooded creatures known for their resilience and venomous strikes.

Fairy: Small, magical beings capable of blessings, mischief, and subtle manipulations.
Undead: Reanimated corpses and skeletal beings driven by dark magic or lingering souls.
Botanic: Living plants and fungal creatures that can entangle foes, spread toxins, or regenerate.

3.2 Cards

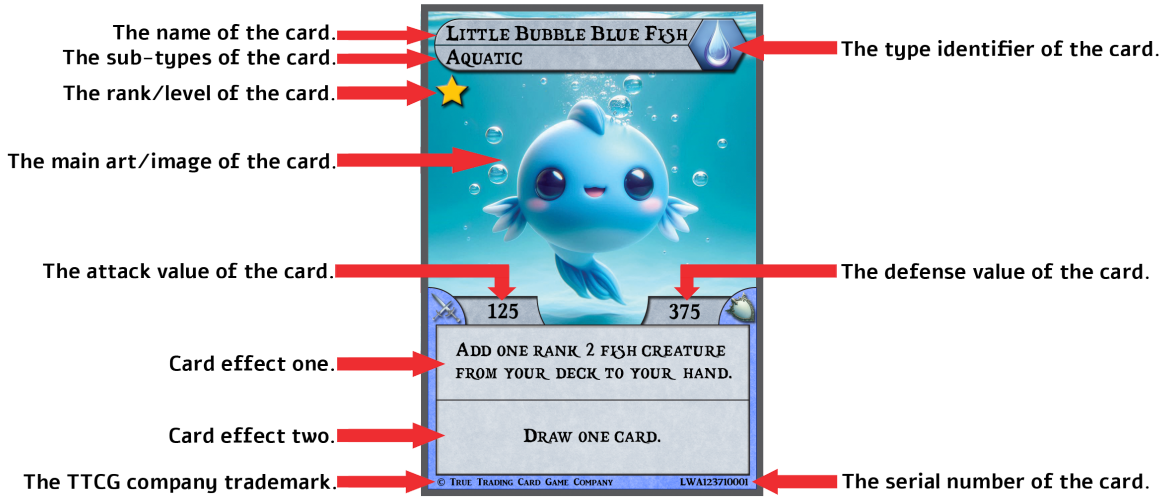

Figure 3.1: This is a sample card with a brief explanation for what represents what on the card. This is a rank 1 water unit with a subtype of aquatic.

3.2.1 Unit Cards

Unit Cards: Unit cards, or simply units, are one of the two primary card categories in the game, distinct from spell cards. Each unit card is assigned a single primary type - Earth, Water, Fire, etc - which defines its thematic alignment, such as high defense creatures for Earth or aggressive effects for Fire. Additionally, every unit possesses one or more subtypes (e.g., Dragon, Warrior, Beast), reflecting its specific identity, lore, and artwork, which often influence its effects. Units can be rank 1 to 5, and can be ranked up using any unit of the same type (exceptions apply when effects state otherwise) with a unit of one rank higher. Units feature attack and defense stats determined by their rank, starting at a base of 500 total points for Rank 1 and increasing by 500 points for each rank. They can be played to the upper row of the field (unit zone). Each unit card has two effects, activated by tapping the card, with tapping locking out the second effect until untapped by another card's effect. Units can attack during the Battle Phase, regardless of their tapped state, targeting opposing units or the opponent directly if no units remain.

3.2.2 Spell Cards

Spell Cards: Spell cards, or simply spells, are one of the two primary card categories in the game, distinct from unit cards. Each spell card is assigned the primary type 'Spell,' distinguishing it from unit types, and represents magical powers, blessings, equipment, artifacts, curses, or other effects. Unlike units, spells do not have subtypes, attack, or defense stats, focusing instead on their functional purpose. They can be played to the lower row of the field (spell zone). Spells range from rank 1 to rank 3 and can be ranked up just like any other card using a spell card of a higher rank. Each spell card has two effects, activated by tapping the card, with tapping locking out the second effect until untapped by another card's effect. Spell effects can be activated immediately upon playing and at any point during either player's turn, offering a wide range of one-time or ongoing influences, such as stat boosts, control, or disruption, often supporting units in the upper row. Rather than an attack and defense value, spells all

include a stat bonus (e.g., +0, +5, -10) which is applied to the active unit card in the field zone above the spell. These stat bonuses vary heavily depending on the spell, but typically range from -10 to 10 for rank 1 spells, increasing or decreasing by 10 for each rank higher.

3.2.3 Lore Cards

// TODO - these are still in development

3.3 Effect Variations

Effects are the core mechanics that drive strategic interaction in the game, modifying the state of play through targeted actions and conditional triggers. Each effect is composed of three key elements: *targets*, which define what the effect applies to; *actions*, which specify what happens to those targets; and *conditions*, which determine when or how the effect can be activated. Targets range from specific cards (e.g., "rank 2 units" or "card(s) under this card") to broad categories (e.g., "all cards" or "opponent's cards"), allowing for precise or sweeping impacts. Actions encompass a variety of outcomes, such as destroying cards, drawing resources, or modifying stats, shaping the game's flow and player decisions. Conditions impose requirements or timing, like discarding a card, losing points, or waiting for an attack, adding depth and tactical nuance. Together, these elements create a flexible system where effects can be combined to produce diverse and dynamic gameplay scenarios.

3.3.1 Effect Types

- **Normal**: A Normal effect is one that can be activated during your turn by tapping the card.
- **Continuous**: This effect is activated like a normal effect, but stays active while the card is on the field.
  - Continuous effects remain active after tapping; tapping only prevents the second effect's activation.
  - When a continuous effect is untapped, its effect is no longer active.
  - Re-tapping can reactivate the continuous effect if chosen.
- **Counter**: This effect can only be activated in response to a specific opponents action (determined by the card effect).
- **Equip**: This card is 'equip' to another card and remains active until the other card or this card is destroyed.
  - When a card is destroyed, all equip cards connected to that card are destroyed with it.
  - If an equip card is ranked up, it is no longer considered active or equip to what it was before ranking up.
  - You can equip a card to either players card, but the equip card always goes to the discard pile of the players who had the card in their deck at the start of the game.
- Latent: This effect is triggered only when this card is currently untapped and meets some specific trigger condition.
- **Dormant**: This is an effect that can only be activated when the card is tapped and can only be activated on a turn after the card was tapped.
- **Passive**: This effect is always active and is triggered whenever the effect conditions are met, whether tapped or untapped.
- **Overload:** This effect can be activated by tapping the card and discarding another card from your hand, offering a powerful one-time effect that exceeds typical rank limitations.
- **Echo:** This effect activates once when the card is played, then activates again the next time the card is untapped, tapping it only on the second activation.
- **Pulse**: This effect can be activated by tapping the card during your turn or your opponent's turn whenever a game state changes or is about to change (e.g., a card is played, destroyed, tapped, untapped, a point is lost, etc), offering a quick, situational benefit (e.g., stat boost, draw a card, deal minor damage).

3.3.2 Unit Effect Variations

// TODO

3.3.3 Spell Effect Variations

// TODO

3.3.4 Effect Conditions

Many effects have a condition that must be met in order to use the effect or as part of the effect's actions. A list of these conditions follows with a brief description of what they mean.

"Discard one card to...": The player must choose and discard one card from their hand to activate the effect of the card. Sometimes this condition is further specific, in which case the player must choose and discard one card matching the exact condition to activate the effect of the card. The follow-up effect is considered activated after the action of discarding the card.

"Discard this card to...": The player must discard the specific card to trigger its effect, often a one-time cost for a powerful result. The follow-up effect is considered activated after the action of discarding the card.

"If you have...": A condition based on whether the player has a specific card, type, or number of cards in play or in hand (e.g., "If you have a Fire unit on the field...").

"Let your opponent...": The player allows the opponent to perform a specific action or gain some benefit, often with a trade-off for the player. The follow-up effect is considered activated after the action done by the opponent and can only be activated if the opponent successfully takes that action.

"Lose X point(s) to...": The player must sacrifice a certain amount of points (e.g., life points, mana) to trigger the effect. The follow-up effect is considered activated after the player loses the point(s).

"This turn...": The effect lasts or can only be used within the current turn, typically providing a temporary advantage or ability.

"While this card remains on the field...": The effect remains active as long as the card stays in play, often providing continuous benefits or penalties.

"When this card is send to the discard pile...": This effect is triggered when the card is discarded or destroyed, usually offering a secondary benefit upon leaving the field. A card must be considered untapped to activate this effect. The follow-up effect is considered activated after the card is sent to the discard pile.

"When another card is destroyed...": This condition activates the effect when a different card is destroyed, potentially creating a chain reaction. The follow-up effect is considered activated after the other card is destroyed.

"When an enemy card attacks...": The effect activates when an enemy unit declares an attack, providing defensive actions, counterattacks, or interrupts. The follow-up effect is activated before the attack.

"If you control...": A condition based on controlling a specific unit or type of unit, or having a certain number of cards in play.

"At the start of your turn...": The effect triggers automatically at the beginning of the player's turn, typically with a passive or setup ability.

"After this card attacks...": The effect occurs after the card has attacked, often used for follow-up actions or penalties. The follow-up effect is activated after the attack is finished.

"When this card attacks...": The effect occurs after the card has attacked, often used for follow-up actions or penalties. The follow-up effect is activated before the attack.

"During your opponent's phase..." The effect activates during the opponent's phase (the exact phase will be specified by the card), often to disrupt or delay their strategy.

"When a card effect activates..." The effect activates when another card effect activates. The follow-up effect occurs before the card effect that triggered this card. This is typically known as a counter effect.

"Destroy one card to..." The player must destroy a card they control to activate the effect. The follow-up effect activates after the destruction.

"Destroy this card to..." The player must destroy this card from the field to trigger its effect, often a one-time cost. The follow-up effect activates after destruction.

"Skip your next turn to..." The player must skip their next turn to activate the effect, a significant cost for a powerful outcome. The follow-up effect activates after the turn is skipped.

"Tap...", "Untap..."

3.3.5 Effect Targets

Many effects have targets which the effect applies to. These targets can be broad or specific depending on the effect and all conditions of the target must be met in order for the effect to target that card. These are all pretty precise and straight forward. Regardless, a list of common target formats follows with a brief description of what they mean.

"one", "two", ... Often, an effect specifies a number of cards that its effect applies to. This is straight forward and the number must be adhered to unless another card specifies otherwise.

"rank 1", "rank 2", ... Often, a specific rank of card is specified in the target. This must be a card matching that rank value.

"rank x or lower" This represents a card matching the rank or having a rank lower than the rank specified.

"rank x or higher" This represents a card matching the rank or having a rank higher than the rank specified.

"dark", "light", "spell", ... Often, a card will specify a type of card in the target. This type must be adhered to.

"dragon", "warrior", ... Often, a card will specify a sub-type of card in the target. This type must be adhered to.

"this card" The effect targets the card it is currently associated with, often used for self-targeting abilities.

"all cards", "each" A broad target that includes all cards of a specified type or set. For example, "all Fire-type units" or "each unit on the field."

"opponent's cards" Specifies that the target is the opponent's cards, often used for disrupting or removing enemy units or spells.

"your cards" Specifies that the target is your own cards, often for protection or support abilities that affect your side of the field.

"lowest attack", "highest defense", ... A conditional target based on the card's stats. For example, "lowest attack" targets the card with the lowest attack value.

"in your hand", "in your discard pile", "in your deck", ... Specifies cards in specific locations, such as those in the player's hand or discard pile.

"equipped unit" Targets a unit that is currently equipped with an item or equipment card.

"another card on the field." This would be any card on either player's side of the field. There must be another card on the field other than the card that says this to activate this effect.

"another card on your side of the field." This would be any card on your side of the field. There must be another card on the field other than the card that says this to activate this effect.

"card(s) under this card" This is a card that is physically underneath the card in question (not to be confused with *below* the card on the field).

"card(s) under another card" Targets any card(s) physically underneath a different card on the field, distinct from the card with the effect.

"card(s) below this card" This is a card that is *below* the card in questions position on the field - such as a spell card below a unit (not to be confused with *under* a card).

The targets of a card can be any combination of the above descriptors or other possible card features.

3.3.6 Effect Actions

Effect actions define what an effect does to the targeted cards or entities. These actions are typically used in the effect's resolution and determine how the game state is modified. Below is a list of common effect actions with a brief description of each:

Destroy The target card is removed from the field and sent to the discard pile. This action usually eliminates the target from play completely.

Send The target card is moved from one place to another, such as from the field to the discard pile, from hand to deck, or other designated zones depending on the effect.

Mill The target card or cards are placed from the top of a deck into the discard pile, often used to deplete the opponent's deck or trigger discard-related effects.

Discard The target card is taken from the player's hand and placed in the discard pile, typically reducing resources or disrupting the opponent's strategy.

Counter The target card's effect is negated or stopped entirely. A counter might prevent the resolution of a spell, effect, or action from the opponent, nullifying its effect.

Return The target card is returned to its previous location or zone, such as returning a card from the field to the hand, deck, or another appropriate zone.
Equip A specific piece of equipment is attached to a target card, usually a unit, granting it additional abilities or buffs.
Activate The effect triggers or activates a specified ability or action on a card, often requiring specific conditions or costs to be met.
Add The effect adds a specified card or value to a target card, player, or zone. For example, adding cards to a player's hand.
Shuffle The target cards are shuffled back into the deck, discard pile, or any other designated zone. This is often used to obscure the order of cards and introduce randomness.
Reveal The target card is revealed face-up to all players, usually as a means of providing information or triggering specific effects based on visibility.
Search The player may search through their deck, hand, for a specific card, typically allowing for strategic card retrieval.
Gain The target card or player gains a benefit, such as life points, attack points, defense points, or other advantages.
Lose The target card or player loses a specified amount of points, attack, defense, or other metric.
Skip The target player or card's action is skipped, preventing them from performing a certain action, such as skipping a phase or a turn.
Draw Moves cards from the top of the deck to the hand, increasing available resources.
Play Places a card into active use on the field from a zone (e.g., hand, discard pile).
Place Positions a card in a specific location (e.g., under another card).
Swap Exchanges the position of two cards or swaps their stats (e.g., Attack and Defense).
Prevent Stops a specified action from activating or affecting a target (e.g., preventing destruction).
Modify Alters a card's stats, rank, or type (e.g., increasing rank or doubling Attack).
Rank Up Stacking a card of one higher rank on top of a card.

3.4 Serial Number Generation for Trading Cards

The serial number (SN) for each trading card is a concise, unique identifier generated from the card's attributes, designed to distinguish potentially millions of unique cards while maintaining a compact format of characters. This section details the algorithm implemented in the Python function `generate_serial_number`, which leverages card attributes such as name, type, subtypes, level, attack, defense, effects, styles, and rarity. This method is found in the `card_maker_ui.py` script (see 5.7).

The base N is a set of characters which are chosen for their unique appearance with the selected font. Some characters clash with others and it is hard to tell which ones are which. Some characters also cause the positioning of the serial number to chance (based on how the font display is programmed). For The characters chosen in N, both of these were considered and undesired characters were removed. The important part is that a unique serial number can be generated with a sufficiently large N. At the time of writing this, the selected characters are (subject to change) as follows:

0123456789AaBbCcDdEeFfGHhiKkLMmNnOoPRrSsTtUuVvWwXxYZz

3.4.1 Overview of the Serial Number Format

The serial number is constructed as a string by concatenating encoded representations of the card's attributes in a fixed order. Each component is derived as follows:

- **Initial Character**: The first letter of the card's name, capitalized (e.g., "D" for "Dragon").
- **Level**: The card's level as a single digit (e.g., "5" for level 5).
- **Type and Subtypes ID**: A base-N encoded index representing the combination of the card's type and subtypes, derived from a precomputed list of valid combinations. A maximum total of 6 total types can be input.
- **Attack and Defense IDs**: Single characters in base-N representing the attack and defense values, scaled relative to the card's level.
- **Effect IDs**: The first character of each effect text (or "0" if no effect).
- **Effect Style IDs**: Base-N encoded indices of the effect styles from a predefined list.
- **Rarity**: A single integer representing the card's rarity (e.g., "0" for base rarity).
- **Padding**: A final "0" character reserved for future expansion.

The resulting SN is a string such as "U40XY87S060900", where each segment encodes specific card data.

3.4.2 Attribute Encoding Details

Type and Subtypes Combination ID

The type and subtypes are combined into a comma-separated string (e.g., "Fire, Dragon, Warrior"), which is then matched against a precomputed list of all valid combinations generated by `get_sequence_combinations`. This function:

- Takes a list of possible types (e.g., `ALL_TYPES_LIST_LOWER`) and generates all unique combinations, ensuring at most one primary type (e.g., "Fire") per combination.
- Caches results in `ALL_SEQUENCE_BUFFER` to avoid recomputation.
- Returns a sorted list of combinations, such as ["", "Dragon", "Fire", "Fire, Dragon", ...].

The `get_combination_id` function finds the index of the input combination in this list and converts it to a fixed-length base-N string using the character set $C = \{0, 1, \ldots, 9, A, \ldots, Z\}$, where $|C| = N$. For an index i, the base-N representation is computed as:

$$\text{result} = \bigoplus_{k} C[i \mod N], \quad i = \lfloor i/N \rfloor, \tag{3.1}$$

where $\bigoplus$ denotes string concatenation from right to left, and the result is padded with leading zeros to 4 digits (e.g., index 10 becomes "000A").

Attack and Defense IDs

The attack and defense values (integers from 0 to 500 × level) are mapped to one of N segments using `get_number_id`. The maximum value is $M = 500 \times \text{level}$, and the segment size is:

$$S = \frac{M}{N}. \tag{3.2}$$

The segment index is calculated as:

$$\text{index} = \frac{N}{M} \max\left(0, \lfloor \min(\text{value}, M) \rfloor\right), \tag{3.3}$$

and mapped to a character in C (e.g., 1500 at level 5 might map to "K").

Effect and Style IDs

For effects, the first character is used (e.g., "D" for "Draw 1 card"), or "0" if the effect is empty. Effect styles are encoded via `get_index_in_baseN`, which finds the index of the style in `VALID_OVERLAY_STYLES` and converts it to base-N (e.g., index 1 becomes "1", N becomes "10").

3.4.3 Uniqueness and Conciseness

Uniqueness is achieved by:

- Using a deterministic combination of all card attributes.
- Leveraging the large combinatorial space of base-N encoding (e.g., if $N = 36$, then $36^4 = 1,679,616$ possible type/subtype IDs).
- Including level-specific scaling for attack and defense, reducing overlap across levels.

The SN length is kept concise (currently set to 15 characters) by using single characters where possible and fixed-length encoding for variable components.

3.4.4 Preprocessing and Performance

The `preprocess_all_types_list` function precomputes type combinations in a separate thread, storing them in `ALL_SEQUENCE_BUFFER`. The global flag `PREPROCESSING_FINISHED` ensures the SN generation waits for this step, returning "Loading.." if incomplete. This system balances uniqueness, readability, and efficiency for large-scale card generation.

Cardscape

The term "Cardscape" combines "card" with "landscape" for a playful yet descriptive title, keeping it light and inclusive. This chapter delves into the broader world surrounding the game, detailing the unique logistics of card production, such as innovative printing methods, as well as the digital realm of websites and online tools that enhance gameplay. It also covers the community features designed to connect players, fostering a lively and interactive environment that extends the experience far beyond the deck.

4.1 Pack Integrity Mechanism

To ensure fairness and prevent players from discerning the contents of a card pack prior to opening, each pack includes a small piece of shiny paper, cut into a random shape, placed alongside the cards. This shape varies unpredictably from pack to pack, rendering tactile inspection unreliable, as feeling the package yields no consistent clues about its contents. The weight of this paper is carefully calibrated so that, when combined with the cards, every pack maintains an identical total weight, neutralizing any advantage that might arise from subtle differences in card composition. Additionally, the paper's reflective, shiny surface is designed to thwart advanced detection methods, such as laser scanning, by diffusing light and obscuring internal details. This mechanism guarantees that each pack remains a mystery until opened, preserving the integrity and excitement of the game for all players. Potential enhancements under consideration include adding color or pattern variations to the shiny paper—such as random metallic hues or faint designs—to introduce visual noise and further deter inspection through packaging edges, as well as incorporating an anti-tamper seal that visibly alters if the pack is opened or exposed to heat or light, providing an additional safeguard. Another possibility is transforming these random shiny shapes into a collector's item, with unique or rare designs that players might trade or catalog, adding an extra layer of engagement to the unpacking experience.

4.2 Lore and Worldbuilding Tie-Ins

The Cardscape weaves a rich narrative tapestry through its integration of lore and worldbuilding, extending the game's universe beyond mere mechanics. Central to this is the inclusion of special lore cards, unique to each pack. Unlike standard cards, these lore cards offer no advantages in competitive play; instead, they serve as collectible glimpses into the stories behind the game's characters, events, or artifacts. Each lore card details the background of a randomly selected card from the game's roster, enriching the player's understanding of its place within the broader Cardscape.

To enhance their distinctiveness, every pack contains exactly one lore card, crafted as a small piece of material cut into a unique, random shape. This could potentially replace the previously considered shiny paper mechanism, shifting the approach from uniform pack weight to intentional variation. As each lore card's shape differs unpredictably, the total weight of every pack becomes similarly random, thwarting attempts to deduce contents through feel or scale. This design not only preserves fairness by obscuring pack advantages but also transforms the lore cards into tactile, one-of-a-kind keepsakes. Players may uncover tales of ancient battles, forgotten realms, or cryptic origins, with the physical uniqueness of each lore card mirroring the diversity of the stories they tell, deepening the connection to the game's evolving mythology.

4.3 Online Card Database

A cornerstone of the Cardscape is the online card database, a comprehensive, community-driven catalog of all cards in circulation. Each card in the game bears a unique serial number and a specific name, tying its physical existence to this digital platform. Initially, every card entry in the database remains hidden, its details obscured from view. To unlock a card and make it visible to the community, players must collect it and submit its serial number along with its name via the database interface.

The unlocking mechanism is designed to encourage collective effort, though its exact implementation remains under consideration. Players will submit a card's serial number and name, tied to their account, contributing to a community threshold—for example, five distinct submissions—required to reveal the card to all. To prevent a single player from unlocking cards alone, various options are being explored. One possibility is that each card's serial number is entirely unique, with only distinct submissions counting toward the threshold. Alternatively, different versions of a card—such as varying print styles, rarities, or editions—might each have distinct serial number patterns, requiring submissions of each type to unlock the card fully. Another approach could limit submissions per account or incorporate additional verification steps. Regardless of the final method, the system will ensure that unveiling the Cardscape is a shared endeavor, rewarding widespread participation and discovery.

This system not only fosters engagement but also mirrors the organic discovery of the game's universe, as players collectively illuminate the Cardscape one card at a time.

4.4 Player-Created Content Portal

The Cardscape empowers its community through the Player-Created Content Portal, an innovative online platform where players can design their own cards. Accessible via the game's website, this portal provides a comprehensive toolkit featuring all possible effects, abilities, and features available within the game's mechanics. Players can craft custom cards by selecting from predefined options—such as attack values, special abilities, or thematic elements—and pair them with original artwork or lore submissions. Once created, these designs can be shared with the community for feedback, showcased in a public gallery, or even submitted for consideration in future expansions. To tie this into the physical game, each player-crafted card is assigned a unique digital identifier, which could potentially link to limited-edition prints or database perks if selected for official release. This portal not only fosters creativity but also strengthens the Cardscape's collaborative spirit, allowing players to leave their mark on the game's ever-evolving universe.

4.5 Hidden Code Challenges

Embedded within the Cardscape are the Hidden Code Challenges, a layer of mystery designed to intrigue and unite the community. Scattered across various cards, subtle clues are concealed within the artwork—enigmatic symbols, patterns, or visual hints that, when pieced together, form a larger puzzle. These clues span multiple cards, requiring players to collaborate and combine their collections to uncover the full picture. While the exact nature of the reward remains in development, solving these challenges will trigger a significant event or unlock a unique feature within the Cardscape—be it a digital reveal, a physical bonus, or a shift in the game's narrative. Still in the planning stages, this system promises to reward keen observation and collective effort, weaving an undercurrent of discovery through the game that deepens as the community grows.

4.6 Card Rarity Tiers and Progression

The game features six rarity tiers for each card, applied to both units and spell cards, with stat boosts and visual enhancements increasing progressively from Tier 1 to Tier 6. Each card exists in all rarities with a fixed total distribution, allocated by percentage. Units have starting Attack and Defense totals based on rank (Rank 1: 500, Rank 2: 1000, Rank 3: 1500, Rank 4: 2000, Rank 5: 2500), with rarity boosts added consistently. Spell cards provide prominent effects and small Attack/Defense bonuses (or penalties) to the unit placed above them on the field, starting with a base value (positive, negative, or zero) that reflects their thematic role, plus rarity bonuses. Artwork remains identical across tiers for both card types, distinguished by layered visual effects.

- **Tier 1: Common**
  - *Unit Stat Boost*: +0 (Base: Rank 1: 500, Rank 2: 1000, Rank 3: 1500, Rank 4: 2000, Rank 5: 2500)
  - *Spell Base Boost*: Defined per card (e.g., +20/+0, +0/+20, -10/-10, +0/+0)
  - *Spell Rarity Bonus*: +0/+0
  - *Spell Total Boost*: Base value only
  - *Visuals*: Standard matte finish
  - *Distribution*: 50% of total copies
- **Tier 2: Uncommon**
  - *Unit Stat Boost*: +10 (Base +10: Rank 1: 510, Rank 2: 1010, Rank 3: 1510, Rank 4: 2010, Rank 5: 2510)
  - *Spell Base Boost*: Defined per card
  - *Spell Rarity Bonus*: +5 total (e.g., +5/+0, +0/+5)
  - *Spell Total Boost*: Base + rarity bonus (e.g., base +20/+0 becomes +25/+0)
  - *Visuals*: Shiny text (metallic foil on name and stats)
  - *Distribution*: 25% of total copies
- **Tier 3: Rare**
  - *Unit Stat Boost*: +20 (Base +20: Rank 1: 520, Rank 2: 1020, Rank 3: 1520, Rank 4: 2020, Rank 5: 2520)
  - *Spell Base Boost*: Defined per card
  - *Spell Rarity Bonus*: +10 total (e.g., +10/+0, +5/+5, +0/+10)
  - *Spell Total Boost*: Base + rarity bonus (e.g., base -10/-10 becomes -5/-5)
  - *Visuals*: Shiny text + shiny borders (foil accents around edges)
  - *Distribution*: 12.5% of total copies
- **Tier 4: Ultra Rare**
  - *Unit Stat Boost*: +30 (Base +30: Rank 1: 530, Rank 2: 1030, Rank 3: 1530, Rank 4: 2030, Rank 5: 2530)
  - *Spell Base Boost*: Defined per card
  - *Spell Rarity Bonus*: +15 total (e.g., +15/+0, +10/+5, +0/+15)
  - *Spell Total Boost*: Base + rarity bonus (e.g., base +0/+20 becomes +5/+35)
  - *Visuals*: Shiny text + shiny borders + holographic overlay
  - *Distribution*: 5% of total copies
- **Tier 5: Mythic Rare**
  - *Unit Stat Boost*: +40 (Base +40: Rank 1: 540, Rank 2: 1040, Rank 3: 1540, Rank 4: 2040, Rank 5: 2540)
  - *Spell Base Boost*: Defined per card
  - *Spell Rarity Bonus*: +20 total (e.g., +20/+0, +15/+5, +10/+10)
  - *Spell Total Boost*: Base + rarity bonus (e.g., base +20/+0 becomes +40/+0)
  - *Visuals*: Shiny text + shiny borders + holographic overlay + foil accents
  - *Distribution*: 2.5% of total copies
- **Tier 6: Primal Rare**
  - *Unit Stat Boost*: +50 (Base +50: Rank 1: 550, Rank 2: 1050, Rank 3: 1550, Rank 4: 2050, Rank 5: 2550)
  - *Spell Base Boost*: Defined per card
  - *Spell Rarity Bonus*: +25 total (e.g., +25/+0, +15/+10, +0/+25)
  - *Spell Total Boost*: Base + rarity bonus (e.g., base -10/-10 becomes +5/+20)
  - *Visuals*: Shiny text + shiny borders + holographic overlay + foil accents + prismatic effect
  - *Distribution*: 1.25% of total copies

Examples:

- *Unit: Rank 3 Warrior*
  - Base: 1500 (750 Attack / 750 Defense)
  - Common: 1500 (750/750)
  - Primal Rare: 1550 (775/775)
- *Spell: Sword of Valor* (Effect: "Target unit gains +50 Attack this turn")
  - Base: +20/+0
  - Common: +20/+0
  - Primal Rare: +45/+0
- *Spell: Cursed Chains* (Effect: "Target enemy unit loses 50 Attack this turn")

- Base: -10/-10
- Common: -10/-10
- Primal Rare: +0/+5

This system ensures a subtle, balanced progression for both card types. Units gain a 10% stat increase for Rank 1 (500 to 550), scaling down to 2% for Rank 5 (2500 to 2550), maintaining fairness across ranks. Spell cards' rarity bonuses enhance their utility to the unit above, with a maximum +25 boost (e.g., 5% of a Rank 1 unit's 500 total, 1% of a Rank 5's 2500). Positive spell base values enhance units, while negative values introduce strategic trade-offs, complementing their core effects without alteration. Visual flair escalates from matte to prismatic, rewarding higher rarities with striking aesthetics.

Scripts And Tools

5.1 Create Effect Combinations

5.1.1 Overview

The `create_effect_combinations.py` script is a Python 3 tool designed to generate all possible combinations of a sentence or set of sentences by replacing placeholders with values defined in external text files. It is ideal for creating permutations of text templates, such as game effects, test cases, or parameterized strings, where placeholders (e.g., `<rank>`) represent variable elements. The tool supports nested placeholders, numeric offsets (e.g., `<rank+1>`), and configurable post-processing to clean, filter, and refine the output.

This script automates the generation of exhaustive text combinations, with output that can be written to a file or displayed in the terminal. Configuration files enhance flexibility by allowing users to customize filtering and phrase replacements without modifying the source code.

5.1.2 Purpose

The `create_effect_combinations.py` script aims to streamline the creation of text variations from templates. Common use cases include:

- Generating card or spell effects for games (e.g., "Draw `<number>` cards").
- Producing test data for software validation.
- Creating parameterized strings for simulations or documentation.

By leveraging recursive placeholder resolution and user-defined configuration files, it ensures comprehensive and polished results with minimal manual effort.

5.1.3 Features

The tool provides the following key features:

- **Placeholder Substitution**: Replaces placeholders (e.g., `<rank>`) with values from text files in a specified directory (default: `placeholders/`).
- **Nested Placeholder Support**: Recursively resolves nested placeholders (e.g., `<effect>` containing `<number>`).
- **Offset Handling**: Adjusts numeric placeholders with offsets (e.g., `<rank+1>`, `<rank-1>`).
- **Input Flexibility**: Processes a single sentence (`-s`) or a file of sentences (`-f`, default: `effects/all_effect_templa`
- **Output Customization**: Writes combinations to a file (default: `effects/all_effects.txt`) or outputs to the terminal in test mode (`-t`).
- **Configurable Post-Processing**:
  - Removes duplicates, preserving the order of first appearance.
  - Filters out unwanted phrases defined in a configuration file (default: `placeholders/combinations_to_remove.`
  - Replaces phrases with designated alternatives from a configuration file (default: `placeholders/phrase_replac`
  - Alphabetizes the final list for readability.
- **Deduplication Utility**: Offers a standalone mode (`-d`) to remove duplicate lines from a file.
- **Error Handling**: Manages missing files, empty inputs, and recursive placeholder cycles (unresolved as, e.g., `<rank>`).

5.1.4 Configuration Files

The script uses two configuration files to customize post-processing, both supporting comments (lines starting with `#`) for documentation or disabling rules:

- `placeholders/combinations_to_remove.txt`:
  - *Purpose*: Specifies phrases to exclude from the generated combinations, such as invalid or undesirable outputs.
  - *Format*: One phrase per line. Empty lines and comments (`#`) are ignored.
  - *Example*:

```
1 or lower
# Exclude higher ranks for now
5 or higher
spell creature
```

  - *Effect*: Combinations containing these phrases (e.g., "Draw 1 or lower cards") are filtered out.
- `placeholders/phrase_replacements.txt`:
  - *Purpose*: Defines phrase replacements to refine grammar or correct errors in the output.
  - *Format*: Key-value pairs in the form `old phrase: new phrase`, one per line. Empty lines and comments (`#`) are ignored.
  - *Example*:

```
# Fix pluralization
one point(s): one point
two card(s): two cards
# Temporary disable this:
# three spell(s): three spells
spellscaster: spellcaster
```

  - *Effect*: Replaces matching phrases (e.g., "one point(s)" becomes "one point") in all combinations.

Both files are optional; if missing, the script proceeds without filtering or replacements, respectively, and issues a warning.

5.1.5 Usage

Invoke the script via the command line:

```
$ python3 create_effect_combinations.py -h
usage: create_effect_combinations.py [-h] [-s SENTENCE] [-f [FILE]]
                                     [-p PLACEHOLDER_DIR] [-o
                                        ↪ OUTPUT_FILE] [-t]
                                     [-v] [-d [DEDUPE]] [-c
                                        ↪ COMBINATIONS_TO_REMOVE]
                                     [-r REPLACEMENTS_FILE]

Generate all possible combinations for placeholders in a sentence.

options:
  -h, --help            show this help message and exit
  -s SENTENCE, --sentence SENTENCE
                        Sentence with placeholders enclosed in <>.
  -f [FILE], --file [FILE]
                        A file of sentences with placeholders enclosed
                           ↪ in <>.
                        Defaults to 'default.txt' if no file specified.
  -p PLACEHOLDER_DIR, --placeholder_dir PLACEHOLDER_DIR
                        Directory containing placeholder text files.
  -o OUTPUT_FILE, --output_file OUTPUT_FILE
                        The file to output the effects to.
  -t, --test_mode       Test mode will only output the combinations to
     ↪ terminal.
  -v, --verbose         Adds extra output during program processing.
```

```
  -d [DEDUPE], --dedupe [DEDUPE]
                        Remove duplicate lines from the specified file (or
                        'effects/all_effects.txt' if none given) and exit.
  -c COMBINATIONS_TO_REMOVE, --combinations_to_remove COMBINATIONS_TO_REMOVE
                        List file containing phrases to remove from resulting
                        combinations (default:
                        'placeholders/combinations_to_remove.txt').
  -r REPLACEMENTS_FILE, --replacements_file REPLACEMENTS_FILE
                        Configuration file containing phrase replacements (format:
                        'old phrase: new phrase') (default:
                        'placeholders/phrase_replacements.txt').
```

Exactly one of `-s` or `-f` must be provided.

5.1.6 Example

With `placeholders/number.txt`:

```
1
2
3
```

And `placeholders/combinations_to_remove.txt`:

```
# Exclude invalid draw
Draw 1
```

And `placeholders/phrase_replacements.txt`:

```
two card: two cards
```

Running:

```
python3 create_effect_combinations.py -s "Draw <number> card" -t
```

Outputs:

```
Draw 2 cards
Draw 3 card
Total combinations: 2
```

5.1.7 Requirements

- Python 3.x
- Placeholder files in the specified directory (e.g., `number.txt`).
- Optional: Configuration files for filtering and replacements.

5.1.8 Limitations

- Offset calculations assume numeric values; non-numeric values are unchanged.
- Recursive placeholder cycles are detected but result in unresolved placeholders (e.g., `<rank>`).
- Additional grammar fixes beyond configuration files are hardcoded in the script.

5.2 Add CSV Field

5.2.1 Overview

The `add_csv_field.py` script is a Python 3 tool designed to process text or CSV files containing effects (e.g., game card descriptions) and generate or update a CSV file with a custom column indicating

whether each effect matches a specified pattern or substring. It supports two modes: pattern matching with placeholder expansion (e.g., `<number> cards`) and exact substring matching. The script integrates with a directory of placeholder files to resolve patterns dynamically, making it versatile for analyzing structured text data.

This tool is particularly useful for annotating datasets, such as marking effects that contain specific keywords or structures, and outputs results in a semicolon-delimited CSV format for easy integration with other tools or analysis workflows.

5.2.2 Purpose

The `add_csv_field.py` script aims to automate the annotation of effects by adding or updating a boolean column in a CSV file based on user-defined criteria. Common use cases include:

- Tagging game effects that match specific patterns (e.g., "Draw `<number>` cards") for categorization.
- Validating or auditing text data by marking entries with exact substrings.
- Preparing datasets for further processing or analysis by adding metadata columns.

By leveraging placeholder resolution and flexible input handling, it simplifies the task of identifying and labeling patterns in text.

5.2.3 Features

The script offers the following key features:

- **Dual Input Support**: Processes plain text files (one effect per line) or semicolon-delimited CSV files with an `EFFECTNAME` column.
- **Pattern Matching**: Checks for patterns with placeholders (e.g., `Destroy <number> cards`), expanding them using values from placeholder files in a specified directory (default: `placeholders/`).
- **Exact Substring Matching**: Sets a column to `True` for effects containing an exact substring (e.g., `Draw two`).
- **Column Management**: Adds a new column if it doesn't exist or updates an existing column, setting `True` where matches occur (preserving `True` values in updates).
- **Placeholder Resolution**: Recursively resolves nested placeholders in patterns, using the same logic as `create_effect_combinations.py`.
- **Output Format**: Generates a semicolon-delimited CSV with at least two columns: `EFFECTNAME` and the user-specified column.
- **Error Handling**: Manages missing files, invalid columns, and recursive placeholder cycles (unresolved as, e.g., `<number>`).
- **Multiple inputs**: The `-t` and `-e` options both support multiple inputs.

5.2.4 Usage

Invoke the script via the command line:

```
$ python3 add_csv_field.py -h
usage: add_csv_field.py [-h] [-i INPUT] [-p PLACEHOLDER_DIR] [-o OUTPUT
    ↪ ] -c COLUMN
                        [-t TEXT] [-e EXACT] [-m MATCH_COLUMN] [-d
                            ↪ DELETE]

Convert effects file to CSV with custom pattern check.

options:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Input file (text or CSV) containing effects (
                            ↪ defaults to
                        'effects/effects_with_placeholders.csv').
  -p PLACEHOLDER_DIR, --placeholder_dir PLACEHOLDER_DIR
                        Directory containing placeholder files (
                            ↪ defaults to
                        'placeholders').
```

```
  -o OUTPUT, --output OUTPUT
                        Output CSV file (defaults to
                        'effects/effects_with_placeholders.csv').
  -c COLUMN, --column COLUMN
                        Name of the column to update or add (e.g., '
                           ↪ HasMyString').
  -t TEXT, --text TEXT  Pattern to search for (e.g., 'some text <
     ↪ placeholder> more
                        text').
  -e EXACT, --exact EXACT
                        Exact line to match and set the specified
                           ↪ column to True.
  -m MATCH_COLUMN, --match_column MATCH_COLUMN
                        An extra identifier for specifying a column
                           ↪ which must be
                        true to evaluate as a match.
  -d DELETE, --delete DELETE
                        Exact string to match for deleting entire lines
                           ↪ .
```

Exactly one of `-t` or `-e` must be provided.

5.2.5 Example

Pattern Matching

With `placeholders/number.txt`:

```
1
2
3
```

And input file `effects.txt`:

```
Draw 2 cards
Gain 5 life
Draw 1 creature
```

Running:

```
python3 add_csv_field.py -i effects.txt -o output.csv -c HasDraw -t "
   ↪ Draw <number>"
```

Outputs `output.csv`:

```
EFFECTNAME;HasDraw
Draw 2 cards;True
Gain 5 life;False
Draw 1 creature;True
```

Exact Match

With input CSV `effects.csv`:

```
EFFECTNAME;HasDraw
Draw two cards;False
Gain life;False
```

Running:

```
python3 add_csv_field.py -i effects.csv -o updated.csv -c HasDraw -e "
   ↪ Draw two"
```

Outputs `updated.csv`:

```
EFFECTNAME;HasDraw
Draw two cards;True
Gain life;False
```

5.2.6 Requirements

- Python 3.x
- Placeholder files in the specified directory (e.g., `number.txt`) for pattern matching with `-t`.

5.2.7 Limitations

- For `-e/-exact`, the specified column must already exist in CSV inputs; otherwise, an error occurs.
- Recursive placeholder cycles in patterns are detected but result in unresolved placeholders (e.g., `<number>`).
- Assumes semicolon (`;`) as the CSV delimiter; other delimiters are not supported.
- Non-CSV inputs are converted to CSV with a default `EFFECTNAME` column.

5.3 Alphabetize File

5.3.1 Overview

The `alphabetize_file.py` script is a lightweight Python 3 tool designed to read lines from a text file, sort them alphabetically, and write the sorted list to an output file. It provides a simple way to organize unordered text data into a consistent, alphabetically ordered format. The script operates on plain text files, ignoring empty lines, and is controlled via command-line arguments for input and output file paths.

This tool is a straightforward utility for tasks requiring sorted text, complementing more complex scripts like `create_effect_combinations.py` by offering a basic file-processing capability within the same toolchain.

5.3.2 Purpose

The `alphabetize_file.py` script serves to automate the sorting of text lines in a file, making it useful for:

- Organizing lists of items, such as effect names or keywords, for readability or further processing.
- Preparing data for manual review or input into other tools that require sorted text.
- Cleaning up unsorted outputs from scripts like `create_effect_combinations.py`.

Its simplicity ensures quick and reliable sorting without the need for external software.

5.3.3 Features

The script offers the following features:

- **Alphabetical Sorting**: Sorts all non-empty lines in the input file alphabetically.
- **Input/Output Flexibility**: Reads from a specified input file and writes to a specified output file, with defaults for convenience.
- **Empty Line Handling**: Ignores blank lines in the input, ensuring the output contains only meaningful content.
- **Error Handling**: Detects and reports file-not-found errors or other processing issues, exiting gracefully with informative messages.

5.3.4 Usage

Invoke the script via the command line:

```
$ python3 alphabetize_file.py -h
usage: alphabetize_file.py [-h] [-i INPUT] [-o OUTPUT]

Alphabetize lines in a file.

options:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Input file to alphabetize (defaults to 'input.
                           ↪ txt').
```

```
  -o OUTPUT, --output OUTPUT
                        Output file for sorted lines (defaults to '
                            ↪ output.txt').
```

Key arguments include:

- `-i/-input`: Input file to alphabetize (default: `input.txt`).
- `-o/-output`: Output file for sorted lines (default: `output.txt`).

5.3.5 Example

Given an input file `effects.txt`:

```
Draw 2 cards
Gain 5 life
Draw 1 creature

Add 3 points
```

Running:

```
python3 alphabetize_file.py -i effects.txt -o sorted_effects.txt
```

Outputs `sorted_effects.txt`:

```
Add 3 points
Draw 1 creature
Draw 2 cards
Gain 5 life
```

The script sorts the lines alphabetically, skipping the empty line, and writes the result to the output file.

5.3.6 Requirements

- Python 3.x
- A plain text input file (e.g., `.txt`).

5.3.7 Limitations

- Sorting is case-sensitive (e.g., "Zebra" comes before "apple").
- Does not preserve original line numbers or metadata; only the text content is sorted.
- Overwrites the output file if it already exists, without warning.
- No option to sort in reverse order or with custom comparators.

5.4 Generate and Order Effects

5.4.1 Overview

The `generate_and_order_effects.sh` script is a Bash utility designed to orchestrate the generation, categorization, and organization of effect descriptions, such as those used in game design or simulation contexts. It leverages companion Python tools (e.g., `create_effect_combinations.py` and `add_csv_field.py`) to create a comprehensive list of effects, annotate them with metadata, and store the results in a structured format. This script serves as a high-level workflow manager, evolving to meet the needs of effect generation and classification.

5.4.2 Purpose

The primary goal of `generate_and_order_effects.sh` is to automate the process of producing and organizing effect text, enabling:

- Generation of a broad set of effect combinations from predefined templates.
- Classification of effects into categories (e.g., units, spells) based on patterns or specific text.
- Preparation of structured output for further analysis or integration into larger systems.

It provides a flexible framework that can adapt as effect definitions and categorization requirements change.

5.4.3 Features

- **Effect Generation**: Produces an initial set of effects using external tools.
- **Categorization**: Adds metadata to effects based on pattern matching or exact text searches.
- **File Management**: Cleans up old files and ensures consistent output locations.
- **Extensibility**: Designed to evolve, supporting additional effect types and classification rules over time.

5.4.4 Usage

Run the script from the command line:

```
./generate_and_order_effects.sh
```

No arguments are currently required, though future iterations may introduce options for customization.

5.4.5 Requirements

- Bash shell environment.
- Python 3.x and dependent scripts (`create_effect_combinations.py`, `add_csv_field.py`).
- Placeholder files in the `placeholders/` directory.

5.4.6 Limitations

- Dependent on external Python tools and their configurations.
- Evolving nature may lead to temporary inconsistencies in output format or behavior.

5.5 Analyze Card Type and Effect Mentions

5.5.1 Overview

The Python script `parse_image_files.py` is a utility designed to analyze card data from a CSV file in the TTCG (Trading Card Template Generator) format and generate visualizations of type, subtype, and effect mention frequencies. Utilizing libraries such as `csv`, `matplotlib`, and `numpy`, the script processes card data to count occurrences of types and subtypes, tracks mentions of these terms in effect texts, and produces bar plots (and an optional 3D plot) to visualize the results (see figure 5.1 for an example plot). This tool is particularly useful for game designers seeking to understand the distribution and interactions of card attributes in a deck.

5.5.2 Purpose

The primary purpose of `parse_image_files.py` is to provide statistical insights into the card collection, enabling:

- Quantification of card types and subtypes across the dataset.
- Analysis of how often types and subtypes are referenced in card effects.
- Visualization of type-effect interactions for balancing and design evaluation.
- Optional 3D visualization of type vs. effect mention frequencies for deeper analysis.

It serves as an analytical tool to support data-driven decisions in card development.

5.5.3 Features

- **Data Reading**: Loads card data from a semicolon-delimited CSV file using `read_cards`, expecting columns like `TYPE`, `SUBTYPES`, `EFFECT1`, and `EFFECT2`.
- **Type and Subtype Counting**: Counts occurrences of predefined types (`TYPE_LIST`) and subtypes (`SUBTYPES_LIST`) using `Counter` from `collections`.
- **Effect Mention Analysis**: Tracks mentions of types and subtypes in combined `EFFECT1` and `EFFECT2` texts, case-insensitively.
- **Interaction Analysis**: Computes two-dimensional counts of types mentioning types/subtypes and subtypes mentioned per type.

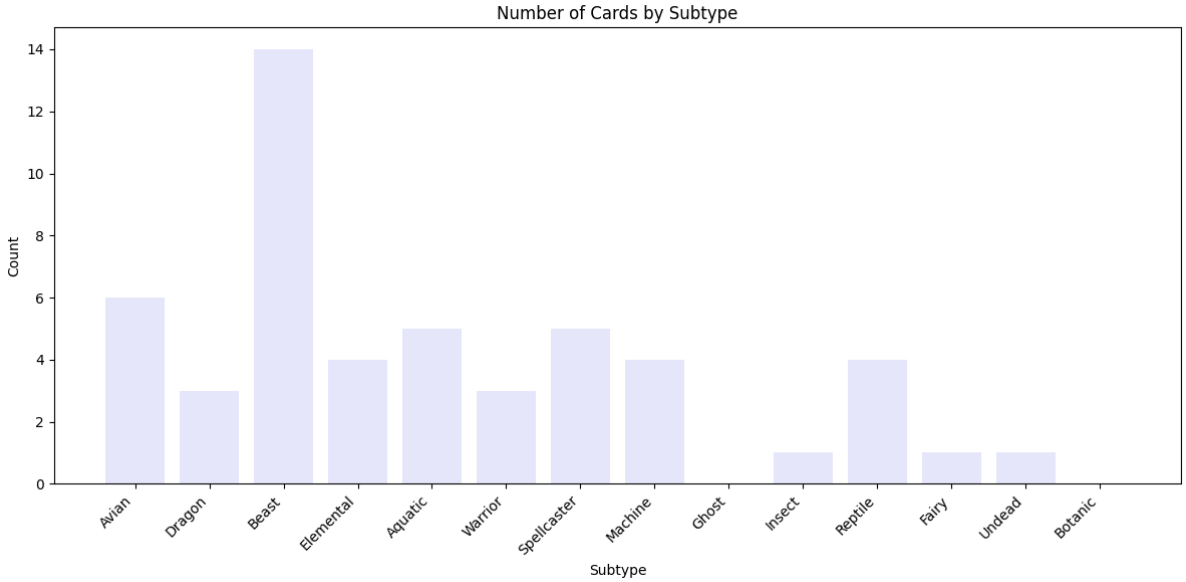

Figure 5.1: Screenshot of one plot output by the `parse_image_files.py` script.

- **Visualization**: Generates bar plots for type counts, subtype counts, effect mentions, and subtype mentions per type, with an optional 3D bar plot for type-effect interactions.

5.5.4 Usage

Run the script from the command line with an optional argument:

```
$ python3 analyze_card_stats.py -h
usage: analyze_card_stats.py [-h] [--plot-3d]

Analyze and plot card type and effect mention frequencies.

options:
  -h, --help    show this help message and exit
  --plot-3d     Generate an additional 3D plot of type vs. effect type
    ↪ frequencies.
```

By default, it reads from `DEFAULT_CARD_LIST_FILE` (a constant from `ttcg_constants`) and saves plots to the `card_list/` directory. The `-plot-3d` flag adds a 3D visualization.

5.5.5 Requirements

- Python 3.x with libraries: `matplotlib` (`pip install matplotlib`), `numpy` (`pip install numpy`).
- CSV file specified by `DEFAULT_CARD_LIST_FILE` with columns `TYPE`, `SUBTYPES`, `EFFECT1`, and `EFFECT2`.
- Constants `TYPE_LIST` and `SUBTYPES_LIST` defined in `ttcg_constants`.
- Write permissions to the `card_list/` directory for saving plots.

5.5.6 Implementation Details

The script is structured as follows:

- `read_cards(file_path)`: Reads the CSV into a list of dictionaries.
- `count_types(cards)`: Counts occurrences of each type in `TYPE_LIST`.
- `count_subtypes(cards)`: Counts subtypes from comma-separated `SUBTYPES` fields.
- `count_effect_mentions(cards)`: Counts mentions of types and subtypes in effect texts.
- `count_type_effect_interactions(cards)`: Builds a 2D dictionary of type vs. effect term mentions.

- `count_subtype_mentions_per_type(cards)`: Tracks subtype mentions per card type.
- `plot_data(...)`: Generates four bar plots and an optional 3D plot using `matplotlib`.
- `main()`: Orchestrates the analysis and visualization, printing results via `output_text`.

Plots are saved as PNG files, with bar colors distinguishing types (e.g., lightcoral) from subtypes (e.g., lightgreen) in effect mentions.

5.5.7 Limitations

- Assumes a fixed CSV format with semicolon delimiters and specific column names.
- Does not validate the existence of the `card_list/` directory before saving plots.
- The 3D plot's readability may degrade with large numbers of types or subtypes.
- Case-insensitive matching in effects may overcount terms if they appear as substrings (e.g., "fire" in "fireball").

5.6 Create Trading Card

5.6.1 Overview

The `create_card.py` script is a Python utility designed to generate trading card images in the TTCG (Trading Card Template Generator) format. It combines a type-specific background image with a level-specific star overlay and overlays customizable text fields, such as name, subtype, attack, defense, and effects. Leveraging the `Pillow` library, the script provides robust image manipulation capabilities, including text squishing for single-line fields and center-aligned text wrapping for effect descriptions, making it suitable for rapid card prototyping in game design contexts.

5.6.2 Purpose

The primary goal of `create_card.py` is to automate the creation of visually consistent trading card images, enabling:

- Customization of card attributes like name, type, level, and effects via command-line arguments.
- Dynamic rendering of text with automatic font size adjustment and horizontal squishing to fit designated boxes.
- Generation of standardized 750x1050 pixel card images with translucent overlays for seamless integration into game assets.

It serves as a flexible tool for designers to produce and iterate on card designs efficiently.

5.6.3 Features

- **Card Base Creation**: Constructs a base image using type-specific PNGs (e.g., `fire.png`), overlays level-specific star images (e.g., `1 star.png`), and overlays the optional effect style images.
- **Text Rendering**: Supports single-line text with optional squishing and centering (name, subtype, attack, defense) and wrapped, center-aligned text for effects.
- **Fallback Handling**: Uses a white background or skips overlays if image files are missing, ensuring graceful error handling.
- **Stat Generation**: Automatically calculates random attack and defense stats based on level if not provided (total stats = level $\times$ 500).

5.6.4 Usage

Run the script from the command line with optional arguments:

```
$ python3 create_card.py -h
usage: create_card.py [-h] [-l {1,2,3,4,5}] [-t TYPE] [-n NAME]
                      [-s SUBTYPE [SUBTYPE ...]] [-1 EFFECT1]
                      [--effect1_style EFFECT1_STYLE] [-2 EFFECT2]
                      [--effect2_style EFFECT2_STYLE] [-a ATTACK] [-d
                         ↪ DEFENSE]
                      [-i IMAGE] [-o OUTPUT] [--serial SERIAL] [-T
                         ↪ TRANSLUCENCY]
```

```
                      [-S SPREADSHEET]

Create a TTCG trading card.

options:
   -h, --help            show this help message and exit
   -l {1,2,3,4,5}, --level {1,2,3,4,5}
                         Card level (1-5)
   -t TYPE, --type TYPE  Card type (e.g., earth, air, fire, water, etc)
   -n NAME, --name NAME  Card name.
   -s SUBTYPE [SUBTYPE ...], --subtype SUBTYPE [SUBTYPE ...]
                         Subtypes (space-separated, e.g., Dragon Spirit)
   -1 EFFECT1, --effect1 EFFECT1
                         First effect text.
   --effect1_style EFFECT1_STYLE
                         First effect style. Valid styles are [None, 'continuous'
                            ↪ , 'counter', 'primed', 'latent',
   'passive', 'equip', 'overload', 'echo', 'surge']
   -2 EFFECT2, --effect2 EFFECT2
                         Second effect text.
   --effect2_style EFFECT2_STYLE
                         Second effect style. Valid styles same as effect1_style.
   -a ATTACK, --attack ATTACK
                         Attack value (defaults to random based on level).
   -d DEFENSE, --defense DEFENSE
                         Defense value (defaults to random based on level, sums
                            ↪ with atk to level*500)
   -i IMAGE, --image IMAGE
                         The image file for this card.
   -o OUTPUT, --output OUTPUT
                         The folder to output images to.
   --serial SERIAL       The serial number for the card.
   -T TRANSLUCENCY, --translucency TRANSLUCENCY
                         The translucency of some card art fields (valid options
                            ↪ are 50, 60, 75, and 100).
   -S SPREADSHEET, --spreadsheet SPREADSHEET
                         Create's all cards loaded from a spreadsheet.
```

Defaults include `"fire"` type, level 1, and random stats, which is primarily used for testing.

5.6.5 Requirements

- Python 3.x with the `Pillow` library (`pip install Pillow`).
- PNG images in `../images/card pngs/` (e.g., `fire.png`, `1 star.png`).
- System font `/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf` or fallback to default font.

5.6.6 Limitations

- Relies on specific file paths for background and overlay images, requiring manual setup.
- Text wrapping estimate (`0.6` factor) may need tuning for optimal line breaks.
- No built-in validation for attack/defense values exceeding level-based totals.

5.7 Card Maker UI

5.7.1 Overview

The `card_maker_ui.py` script is a Python-based graphical user interface (GUI) built with the Tkinter library, designed to create and preview trading cards in the TTCG (Trading Card Template Generator)

format. It integrates `create_card.py` for image generation and `generate_random_effects.py` for effect generation, offering an interactive platform for designing cards with real-time feedback. Utilizing a global `WIDGETS` dictionary, the script streamlines access to UI elements, enhancing code efficiency and maintainability.

5.7.2 Purpose

The primary goal of `card_maker_ui.py` is to provide an intuitive tool for card creation, enabling:

- Real-time customization of card attributes such as type, level, name, subtypes, stats, and effects through a user-friendly interface.
- Immediate visual feedback via a live card preview, updated with each input change.
- Simplified workflow for saving card data and resetting the UI to default values, supporting iterative design processes.

It serves as the central hub for TTCG card prototyping, bridging user input with automated generation features.

5.7.3 Features

- **Interactive Interface**: Three-panel layout with input fields (left), live 400×580 pixel preview (middle), and effect generator (right), all managed via a global `WIDGETS` dictionary.
- **Dynamic Updates**: Automatically refreshes the preview using `create_card.py` on input changes, with a `SKIP_PREVIEW` flag to optimize reset operations.
- **Effect Generation**: Produces up to ten random effects from a CSV file, filtered by type and subtypes, assignable to card fields with a single click.
- **Stat Randomization**: Generates ATK and DEF values (total = level × 500) via a "Randomize" button or level selection, ensuring balanced stats.
- **Auto-Filling**: The UI is intuitively designed to auto-fill some values, such as random ATK and DEF values, automatic type and name selection (when a named images are selected), and spell styles (for various keywords).
- **Loading and Editing**: The `-L` option modifies the user interface specifically for loading and modifying existing cards in the card list. This mode will override certain safeguards normally in place to allow replacing and editing existing cards (see figure **??**).

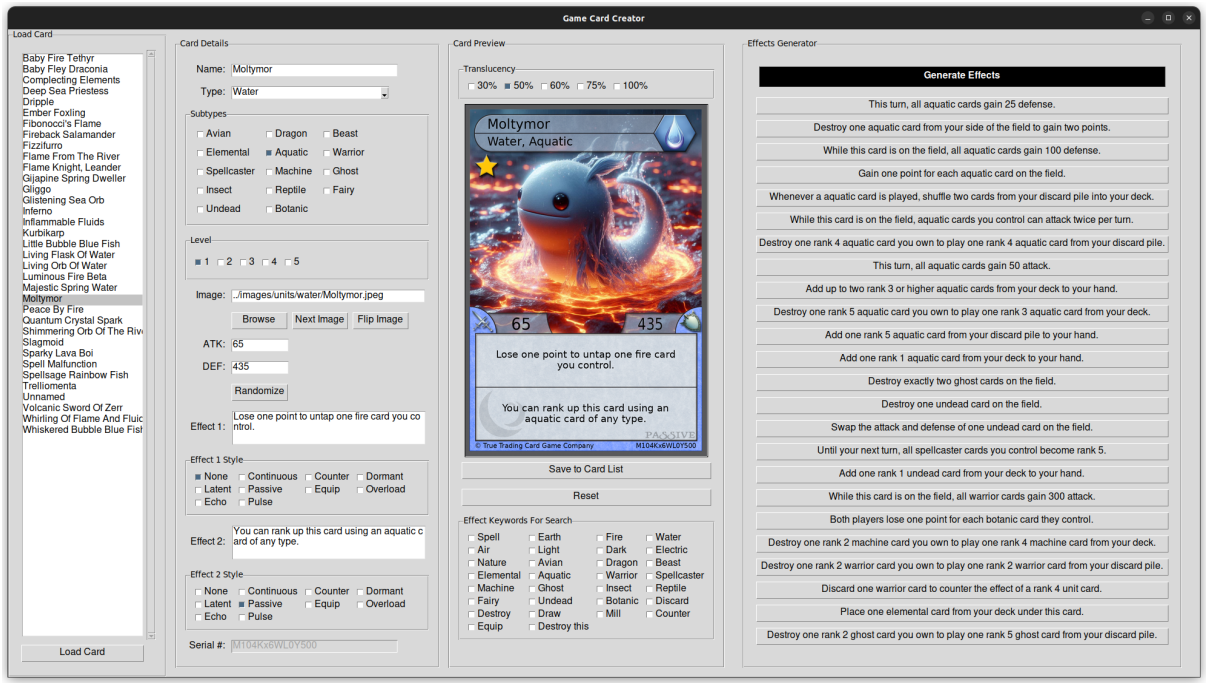

Figure 5.2: Screenshot of the `card_maker_ui.py` interface when the `-L` (load card) mode is enabled, showcasing a random card being loaded in.

5.7.4 Usage

Run the script from the command line with an optional argument:

```
$ python3 card_maker_ui.py -h
usage: card_maker_ui.py [-h] [-i INPUT_FILE] [-o OUTPUT_FILE]

User Interface for card generation.

options:
  -h, --help            show this help message and exit
  -i INPUT_FILE, --input_file INPUT_FILE
                        Path to the input CSV file to process for effects. If not
                        provided, defaults to
                        'effects/effects_with_placeholders.csv'. The file should
                        contain a header row with column names.
  -o OUTPUT_FILE, --output_file OUTPUT_FILE
                        Path to the output CSV file for saving card data. Defaults
                        to 'card_list.csv'.
  -L, --load_mode       Enable loading mode to load and overwrite existing values from the output CSV.
```

The `-i` argument specifies the effects CSV file (default: `effects/effects_with_placeholders.csv`). Default UI values include "Fire" type, level 1, "Unnamed" name, and 0 ATK/DEF.

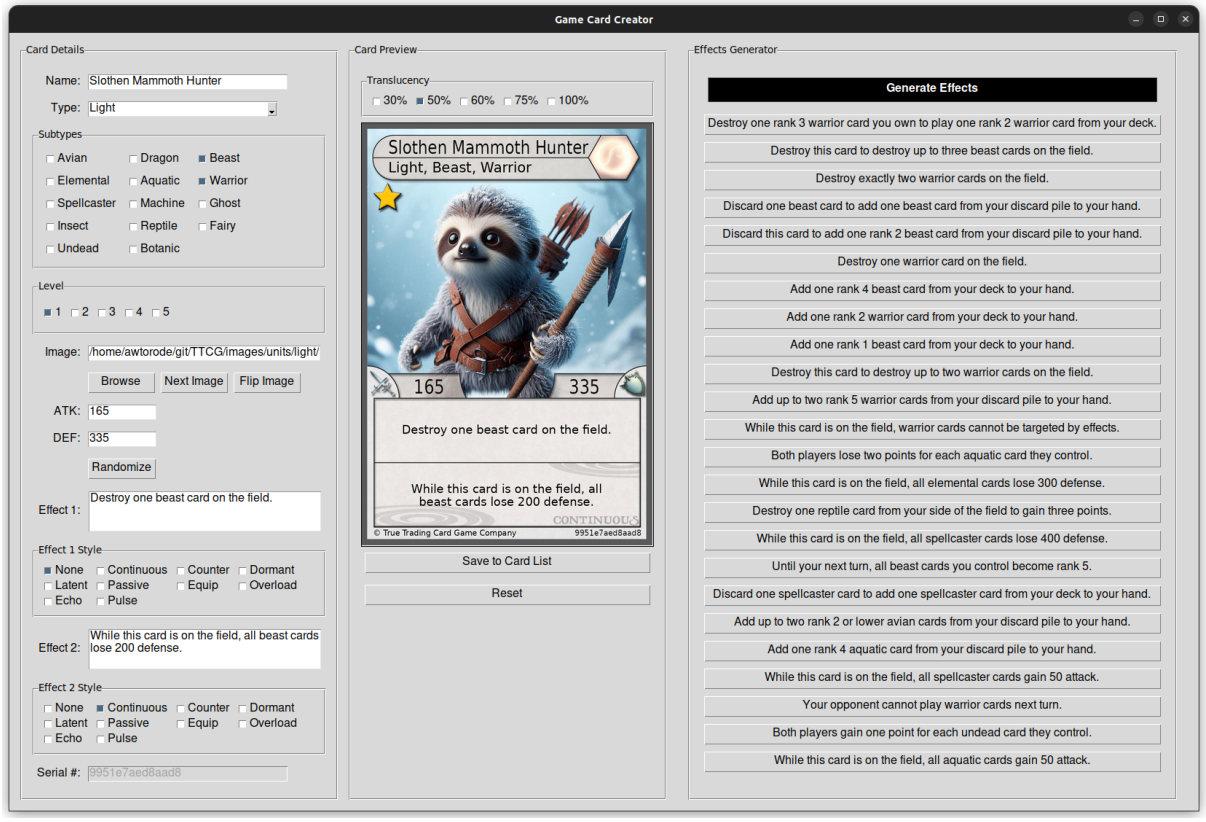

Figure 5.3: Screenshot of the `card_maker_ui.py` interface, showcasing a random card being created.

The UI consists of three main panels (see figure 5.3). The panel on the left is the primary stat panel for the card being created. This panel houses fields like the name, subtypes, level, image path, effects, effect styles, etc. These are the core elements that make up the card.

The middle panel consists of a preview window, transparency selection (for various card art features), a save button, a reset button, and a spell search field. The preview displays a continually updating preview of what the card looks like based on the core input and transparency values. The update occurs whenever a main input loses focus (i.e. after the user typed in a name and selected away from the name text box). The reset button simple resets the UI to default values. The save button saves all card input to the appropriate `csv` file for storing complete card data. The spell search features is used when a card of type spell is being generated. The values selected in this spell search are used when generating random effects for the spells. Every option that is selected will be targetted as a keyword when generating effects for a spell card.

The panel on the right consists of the effect generation feature. The main generate effects button will generate a list of random effects from the list of all effects. If a unit is being created (non-spell type card), the effects will generate based on effects that are appropriate for units. When a subtype is selected for a unit, the first half of the effects will generate using the list of subtypes as keywords for which effects should be generated. When a spell is being created, this feature instead looks at the spell search rather than subtypes (since spells don't have a subtype).

5.7.5 Requirements

- Python 3.10+ with Tkinter (standard library) and `Pillow` (`pip install Pillow`).
- Companion scripts `create_card.py` and `generate_random_effects.py` in the same directory.
- Effects CSV file (e.g., `effects/effects_with_placeholders.csv`) and PNG images in `../images/card pngs/`.

5.7.6 Limitations

- Depends on external scripts and specific file paths, requiring proper project setup.
- No current implementation for saving card data to a spreadsheet (placeholder `print` only) at the time of writing this - the tool is still being developed.
- Effect generation may produce fewer than ten unique effects if the CSV pool is limited.

How To

This chapter provides a practical guide to generating and managing effect descriptions for the project, using scripts in the `bin/` directory. Some steps may seem tedious or complicated—editing multiple files, running scripts with specific arguments, or managing directory paths—but they are essential. With thousands and thousands of effects to handle, manual processing would be impractical; this automation ensures consistency, scalability, and differentiation of metadata, making it an extremely helpful approach for large-scale effect management.

6.1 Working with This Project in Git

This project is hosted on GitHub at the following url:

`https://github.com/torodean/TTCG`

The following guidelines outline the workflow for contributing to the project using Git.

6.1.1 Repository Structure and Main Branch

The main branch serves as the primary stable branch of the project. It is configured to run automatic unit tests using GitHub Workflows. These tests are defined in the `test.yml` file located in the `.github/workflows` directory. The unit tests themselves reside in the `bin/test` directory and are executed as part of the continuous integration pipeline whenever changes are pushed to the `main` branch.

All scripts and executables in the `bin` folder are designed to be run from within that directory. Ensure your working directory is set to `bin` when executing any code located there.

6.1.2 Contributing to the Project

To make changes or contribute new features, follow these steps:

1. **Create a New Branch**: Start by creating a new branch from `main` with a descriptive name that reflects the purpose of your changes (e.g., `feature/add-new-module` or `fix/bug-issue-123`).

   ```
   git checkout main
   git pull
   git checkout -b <branch-name>
   ```

2. **Work in Your Branch**: Implement your changes or modifications in this branch. Ensure that any code added to the `bin` folder is executable from that directory.
3. **Run and Verify Tests**: Before submitting your changes, verify that all unit tests pass locally. The tests can be found in `bin/tests`. Running them locally ensures compatibility with the automated testing in the GitHub Workflow.
4. **Commit and Push**: Once your changes are complete and tests pass, commit your work and push the branch to the GitHub repository.

   ```
   git add .
   git commit -m "Descriptive commit message"
   git push origin <branch-name>
   ```

5. **Create a Merge Request**: On GitHub, create a merge request (pull request) from your branch to `main`. Provide a clear description of your changes in the merge request. The merge request will trigger the automated unit tests defined in `test.yml`. Ensure all tests pass before requesting a review.
6. **Review Process**: A project maintainer will review your merge request. Address any feedback or requested changes by pushing additional commits to your branch. Once approved, your changes will be merged into `main`.

By following this workflow, we ensure that the `main` branch remains stable and that all contributions are thoroughly tested and reviewed.

6.2 Add and Auto-Generate Effect Strings from Templates

This section guides you through creating effect descriptions from templates using scripts in the `bin/` directory. You'll combine placeholder values from `placeholders/` with templates from `effects/` to generate a list of effects.

6.2.1 What You'll Need

- A Unix-like system (e.g., Linux, macOS) with Bash and Python 3 installed or Windows with Cygwin and a compatible python setup.
- The project directory structure: `bin/`, `placeholders/`, and `effects/`. This should already exist if you cloned this project using *git*.

6.2.2 Step-by-Step Guide

Step 1: Set Up Your Placeholders and Templates

1. Open `effects/all_effect_templates.txt` in a text editor.
2. Add or edit effect templates using placeholders like `<number>` or `<type>`. For example:

```
Draw <number> cards
Gain <number> <type> points
```

3. Save the file.

Step 2: Configure Placeholders

1. Go to the `placeholders/` directory.
2. For each placeholder in your templates (e.g., `<number>`, `<type>`), create or edit a file named `<placeholder>.txt`. Examples:
   - `number.txt`: Add one number per line (e.g., `1`, `2`, `3`).
   - `type.txt`: Add one type per line (e.g., `fire`, `water`).
3. Save your changes. Keep it simple—just list the values, no comments or extra formatting.

!! Note: The placeholders correspond to file(s) in the `placeholders/` folder. The names of the placeholder files should match the names of the placeholders exactly.

Step 3: (Optional) Filter and Refine Effects

1. Open `placeholders/combinations_to_remove.txt`.
2. List phrases you don't want in your effects, one per line. For example:

```
# Skip useless effects
Draw 0 cards
```

3. Open `placeholders/phrase_replacements.txt`.
4. Add replacements in `old phrase: new phrase` format, like:

```
# Fix grammar
Gain 1 points: Gain 1 point
```

5. Save both files. Use `#` for comments if you'd like. If you include comments, the comments must be on their own line(s).

Step 4: Generate Effects

1. Navigate to the `bin/` directory in your terminal:

```
cd bin
```

2. Run the generation script:

```
python3 create_effect_combinations.py -f ../effects/
    ↪ all_effect_templates.txt
```

3. If you are using the default file paths for the effect templates, you can omit the actual file (as seen in the above command) as the script will default to it.

```
python3 create_effect_combinations.py -f
```

4. Check `../effects/all_effects.txt` for your generated effects (e.g., "Draw 1 cards", "Gain 2 fire points").

Step 5: (Optional) Automate with a Script

1. For a one-command solution, run:

```
./generate_and_order_effects.sh
```

2. This cleans up old files and generates effects automatically. It also performs other steps and processes outlined later.
3. Find the results in `../effects/all_effects.txt`.

6.2.3 Tips

- **Customize Output**: Tweak `all_effect_templates.txt` or placeholder files to change effects.
- **Fix Errors**: If a script fails, double-check that files exist and paths are correct (e.g., `../placeholders/number.txt`)
- **See Results**: Your effects land in `all_effects.txt` as plain text.

6.3 Adding Metadata to Effects and Automating the Process

This section explains how to add metadata (e.g., categories like `UNIT` or `SPELL`) to your effect descriptions using the `add_csv_field.py` script and how to automate metadata addition for all effects with `generate_and_order_effects.sh`. You'll work with files in the `bin/` and `effects/` directories, building on the effects generated earlier.

6.3.1 What You'll Need

- A Unix-like system (e.g., Linux, macOS) with Bash and Python 3 installed, or Windows with Cygwin and a compatible Python setup.
- The project directory structure: `bin/`, `placeholders/`, and `effects/`, already set up from cloning this project with *git*.
- An existing `all_effects.txt` file in `effects/` from the previous section.

6.3.2 Step-by-Step Guide

Step 1: Add Metadata with `add_csv_field.py`

1. Navigate to the `bin/` directory in your terminal:

```
cd bin
```

2. Run `add_csv_field.py` to add a metadata column to your effects. For example, to mark effects as `UNIT`:

```
python3 add_csv_field.py -i ../effects/all_effects.txt -o ../
    ↪ effects/effects_with_placeholders.csv -c UNIT -t "You can
    ↪ send up to <number> card"
```

3. Explanation:
   - `-i`: Input file (e.g., `all_effects.txt`).
   - `-o`: Output CSV file (e.g., `effects_with_placeholders.csv`).
   - `-c`: Column name to add or to set to True (if it exists) (e.g., `UNIT`).
   - `-t`: Pattern with placeholders to match (e.g., `You can send up to <number> card`).
4. Assuming all files are left in their default locations, this script can be used without an *input* or *output* arguments to get the same result

```
python3 add_csv_field.py -c UNIT -t "You can send up to <number>
    ↪ card"
```

5. Repeat for other categories. For example, to mark `SPELL` effects:

```
python3 add_csv_field.py -c SPELL -t "Your opponent loses <number>
    ↪ point"
```

6. Check `../effects/effects_with_placeholders.csv`. It will list effects with columns like `UNIT` and `SPELL`, set to "True" or "False" based on matches.

Step 2: Understand How It Works

1. The script uses placeholder files from `../placeholders/` (e.g., `number.txt`, `type.txt`) to expand patterns. For example, "Gain <number> <type> points" matches "Gain 1 fire points" if `number.txt` has "1" and `type.txt` has "fire".
2. You can use `-e` instead of `-t` for exact matches:

```
python3 add_csv_field.py -c UNIT -e "Gain 1 fire point"
```

Note: Each run adds or updates one column. Use the output CSV as input for the next category to build up metadata.

Step 3: Automate Metadata for All Effects

1. The *generate_ and_ order_ effects.sh* script handles automating calls to *create_ effect_ combinations.py* and *add_ csv_ field.py* with the proper parameters to get the desired output metadata for all possible effects that are currently configured. This file has a lot of entries as well as some loops that specify similar entries with subtle differences for certain effects.
2. Modify the automation script mentioned above appropriately for any effect additions to get the desired meta-data.
   - To add an effect which should be appropriate for any level of spell, find the appropriate for loop within the file and add the line for the spell. For example:

```
# Spell effects that are the same for each level
for level in LEVEL_{1..3}; do
    python3 add_csv_field.py -c "$level" -m SPELL -t "Your
        ↪ opponent cannot play <types> cards next turn"
    ...
    // Add Desired additions here.
done
```

   - To add an effect which should be appropriate for any specific number based on the level of spell, find the appropriate *for* loop within the file and add the line for the spell. These are spells that contain a number placeholder which scales with the level of the card. A loop exists for both spells and units but is slightly different for each. For example:

```
# Some various <number> effect combinations for spells.
NUMS=(zero one two three four) # Array for number words
for l in "LEVEL_1 1 2" "LEVEL_2 2 3" "LEVEL_3 3 4"; do
    read level min max <<< "$l"
    for ((m=min; m<=max; m++)); do
        python3 add_csv_field.py -c "$level" -m SPELL -t "
            ↪ Reveal the top ${NUMS[m]} card$( ((m>1)) && echo
            ↪ "s") of your deck and play one <type> card from
            ↪ among them"
```

```
        ...
        // Add Desired additions here.
    done
done
```

- To add an effect which should be appropriate for any specific rank based on the level of spell, find the appropriate *for* loop within the file and add the line for the spell. These are spells that contain a rank placeholder which scales with the level of the card. A loop exists for both spells and units but is slightly different for each. For example:

```
# Some various <rank> effect combinations for spells.
for l in "LEVEL_1 1 2" "LEVEL_2 2 4" "LEVEL_3 4 5"; do
    read level min max <<< "$l"
    for ((m=min; m<=max; m++)); do
        python3 add_csv_field.py -c "$level" -m SPELL -t "Add
            ↪ up to <number> rank ${m} light cards from your
            ↪ deck to your hand."
        ...
        // Add Desired additions here.
    done
done
```

- To add effects that are specific to a unit or spell, add a line with the specific unit/spell specifier as well as the effect string to search for in the appropriate section. For example:

```
# Add some unit specific effects.
python3 add_csv_field.py -c UNIT -t "You can send up to <number
    ↪ > card"
...
// Add Desired additions here.
```

- Exceptions to the above loops exist for specific effects that require specific levels which they should correspond to. These can be added in the appropriate exceptions section. For example:

```
# Exceptions to the below loop(s).
python3 add_csv_field.py -c LEVEL_1 -m SPELL -t "Destroy one <
    ↪ type> card on the field"
...
// Add Desired additions here.
```

3. Run the automation script:

```
./generate_and_order_effects.sh
```

4. This script:
   - Generates effects from `all_effect_templates.txt` (as in the previous section).
   - Adds metadata columns (e.g., `UNIT`, `SPELL`) based on predefined patterns or exact matches.
   - Cleans up old files and organizes the output.
5. Check `../effects/effects_with_placeholders.csv` for the final result, which includes all effects with metadata columns added automatically.

6.3.3 Tips

- **Customize Metadata**: Edit the patterns or exact strings in your `add_csv_field.py` commands to match your effects.
- **Fix Errors**: Ensure input files exist and placeholder files in `../placeholders/` match your patterns.
- **See Results**: Open `effects_with_placeholders.csv` in a spreadsheet or text editor to review metadata.

6.3.4 What You Get

- `effects/effects_with_placeholders.csv`: A semicolon-delimited CSV with effects and meta-data columns (e.g., `UNIT`, `SPELL`) set to "True" or "False".
- At the time of writing this, the meta-data columns programmed into this automation script(s) create columns for UNIT, SPELL, LEVEL_1, ... LEVEL_5. This allows the distinction between the two main sets of cards (units and spells) as well as different effects that should/could be valid for the varying levels of those cards.

6.4 Generating Random Effect Pairs for Cards

This section shows you how to use `generate_random_effects.py` to create specific pairs of effects for a card, such as for a trading card game. The script filters effects from a CSV based on metadata columns and generates random pairs from the filtered list, perfect for assigning unique effect combinations to a card.

6.4.1 What You'll Need

- Python 3 installed and the project structure (`bin/`, `effects/`) set up.
- `effects/effects_with_placeholders.csv` from previous sections, with metadata columns like `UNIT` or `SPELL`.

6.4.2 Steps

1. Navigate to the `bin/` directory:

```
cd bin
```

2. Run the script to generate pairs for a card. For example, to get 10 random `UNIT` effect pairs:

```
python3 generate_random_effects.py -c UNIT
```

3. For multiple filters (e.g., `UNIT` and `LEVEL_1`), use:

```
python3 generate_random_effects.py -c UNIT -c LEVEL_1
```

4. To change the number of pairs (e.g., 5 instead of 10):

```
python3 generate_random_effects.py -c UNIT -p 5
```

5. The script defaults to `../effects/effects_with_placeholders.csv`. For a custom file, add it:

```
python3 generate_random_effects.py my_effects.csv -c UNIT
```

6. Check the terminal output for numbered pairs of effects from the first column (e.g., "Gain 1 fire points" paired with "Draw 2 cards").

6.4.3 Tips

- Use metadata columns added earlier (e.g., `UNIT`, `SPELL`) to filter effects specific to your card.
- If no pairs appear, ensure your CSV has enough "True" rows for the specified columns.

6.5 Alphabetizing the Card List

To alphabetize the card list in `card_list/card_list.csv` while preserving the header row, run this command in the terminal from the directory containing `alphabetize_file.py`:

```
python3 alphabetize_file.py -i card_list/card_list.csv -o card_list/
    ↪ card_list.csv -H
```

Here, `-i` specifies the input file, `-o` the output file (overwriting the input), and `-H` keeps the header unsorted. For example, a file with "Name, Zebra, Apple, Monkey" becomes "Name, Apple, Monkey, Zebra" after sorting.

6.6 Creating a TTCG Trading Card

This section explains how to use `create_card.py` to generate a trading card image for the TTCG format, complete with effects, stats, and a custom design. It builds on effects generated earlier, turning them into a visual card.

6.6.1 What You'll Need

- Python 3 with the `Pillow` library installed (`pip install Pillow`).
- The project structure (`bin/`, `images/`) set up, with type images (e.g., `fire.png`) in `../images/card pngs/`.

6.6.2 Steps

1. Navigate to the `bin/` directory:

```
cd bin
```

2. Run the script to create a card. For example:

```
python3 create_card.py --name "Fire Dragon" --type fire --level 3
    ↪ --subtype Dragon Spirit --effect1 "Gain 2 fire points" --
    ↪ effect2 "Draw 1 cards"
```

3. Customize further if desired:
   - Add stats: `-attack 1000 -defense 500`.
   - Use a custom image: `-image ../images/custom.png`.
   - Set output folder: `-o ../my_cards`.
4. Check the output folder (defaults to `../images/generated_cards/`) for the card, e.g., `ttcg_card_Fire_Dragon.png`. It's a 750x1050 pixel image with your card details.

6.6.3 Example

Below is an example command (without an image specified) with the matching output seen in Figure 6.1.

```
./create_card.py --name "This is an even longer name for testing long
    ↪ names" --level 3 --effect1 "This is some sample text to represent
    ↪  an effect on the card." --effect2 "This is some sample text to
    ↪ represent an effect on the card. It's quite long to test longer
    ↪ effects. That way I can ensure the text fits nicely."
```

6.6.4 Tips

- Use effects from `all_effects.txt` or pairs from `generate_random_effects.py`.
- If stats are omitted, they're random but scaled to level (total = level * 500).
- Ensure type images (e.g., `fire.png`) exist in `../images/card pngs/`, or it falls back to a default.
- Use `-o` to save cards elsewhere; ensure the folder exists or the script may fail.

6.7 Creating TTCG Trading Card(s) With The UI

This section provides a step-by-step guide to creating trading cards using the TTCG Card Creator graphical user interface (GUI). The UI allows you to input card details, preview the card, and save it to a CSV file while ensuring unique card names. The interface features a dark theme for comfortable use, with all fields and previews clearly visible.

Figure 6.1: On the left is a sample card with a fake name, subtypes, and some dummy text in the effect slots. On the right is a sample level one card with translucency, and other areas filled in appropriately. This is to demonstrate the output of the *create_ card.py* script.

6.7.1 Launching the Application

1. Ensure you have Python 3 installed, along with required libraries: `tkinter`, `PIL` (Pillow), and `csv`. Install them using:

```
pip install pillow
```

2. Place your card images in a directory structure (e.g., `../images/units/fire/`, `../images/units/light/`, etc.), organized by type if desired.
3. Run the script from the command line:

```
python3 card_maker_ui.py
```

   - `-i`: Path to the effects CSV file (defaults to `effects/effects_with_placeholders.csv`).
   - `-o`: Path to the output card list CSV (defaults to `card_list/card_list.csv`).
4. The GUI will open displaying three main sections: Card Details (left), Card Preview (middle), and Effects Generator (right).

6.7.2 Creating a Card

1. **Set the Image:**
   - In the `Card Details` section, click the `Browse` button below the `Image` field to select an image file (e.g., `Ember Foxling.jpg`).
   - The image path will populate the entry field, and the card name will update to the filename (e.g., `Ember Foxling`) if the current name is `Unnamed`.
   - The type (e.g., `Fire`) will automatically set if it's in the image path and matches a predefined type list.
   - Use `Next Image` to cycle to the next image in the folder, skipping any whose names are already in `card_list/card_list.csv`.
   - Use `Flip Image` to horizontally flip the selected image if needed.
2. **Enter Card Details:**
   - `Name`: Edit the auto-filled name (e.g., change `Ember Foxling` to `Ember Foxling Alpha`) if desired.
   - `Type`: Select from the dropdown (e.g., `Fire`, `Water`) or keep the auto-detected type.

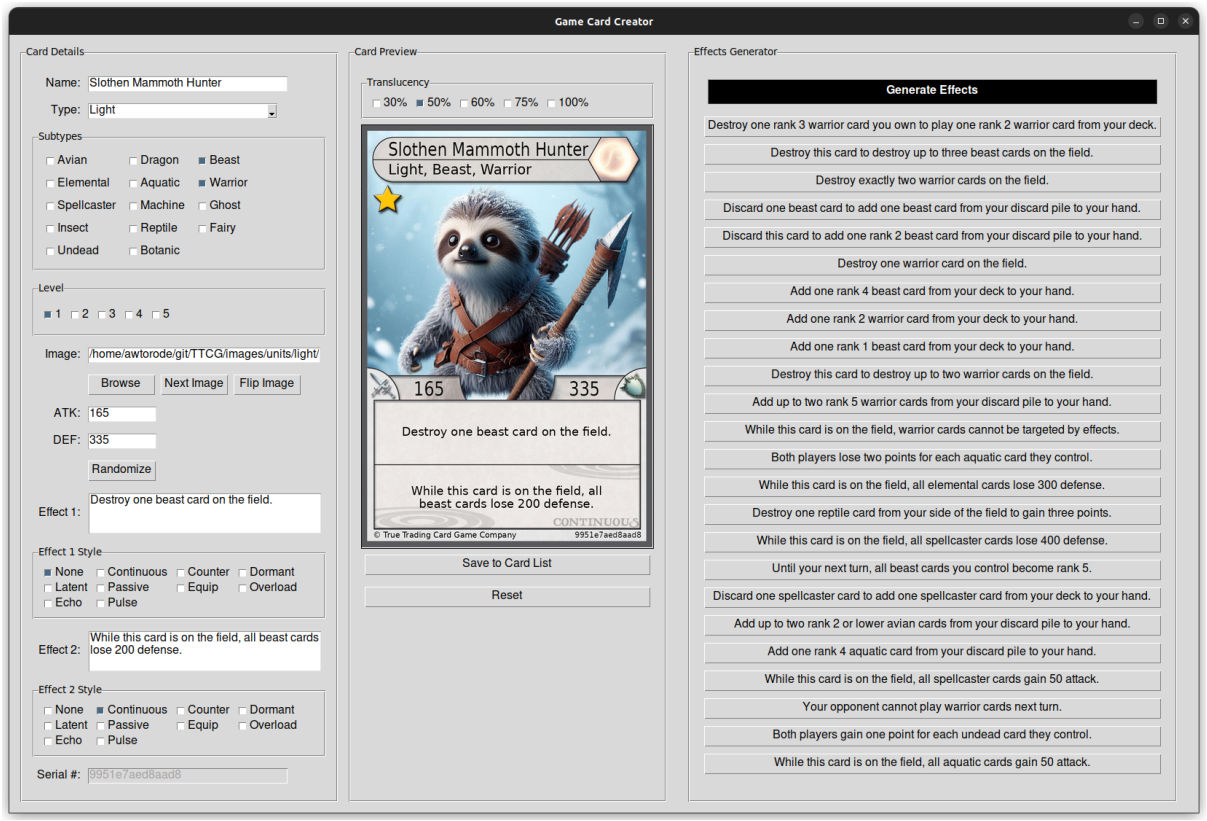

Figure 6.2: This is a sample display of the `card_maker_ui.py`

- `Subtypes`: Check boxes for applicable subtypes (e.g., `Beast`, `Elemental`). For `Spell` cards, subtypes are cleared automatically.
- `Level`: Choose a level (1–5) from the dropdown.
- `ATK` and `DEF`: Enter attack and defense values manually (e.g., `360` and `140`), or click `Randomize` to generate values based on the level.

3. **Add Effects:**
   - In the `Effects Generator` section, click `Generate Effects` to populate 18 buttons with effect suggestions from `effects/effects_with_placeholders.csv`.
   - Click an effect button to assign it to `Effect 1` or `Effect 2` (e.g., `Add up to two rank 1 beast cards from your discard pile to your hand`).
   - Manually edit `Effect 1` and `Effect 2` text boxes if needed.
4. **Adjust Transparency:**
   - In the `Card Preview` section, select a translucency level (50%, 60%, 75%, or 100%) using the checkboxes. The preview updates instantly.
5. **Preview the Card:**
   - The `Card Preview` canvas (middle section) displays the card with the selected image, details, and effects. It updates automatically as you change fields.
6. **Save the Card:**
   - Click `Save to Card List` in the `Card Preview` section.
   - The card is saved to `card_list/card_list.csv` with a unique serial number (auto-generated) if:
     - The name isn't already in the CSV.
     - The exact card data (all fields) isn't a duplicate.
     - The effect combination isn't already used.
   - If saved, the UI loads the next image with a unique name. If skipped (e.g., duplicate name), it still advances to the next image.
   - Check the console or log for messages like `Card data saved` or `Card already exists`.
   - The save feature does not generate the output card, but rather saves all of the various card data to the `card_list/card_list.csv` file for them to be generated later.

7. **Reset the UI:**
   - Click `Reset` to clear all fields and start over.
8. **Generate the card Image File(s):**
   - Once the `card_list/card_list.csv` is populated with the desired card values, the *create_ card.py* script can be used to generate the card art for each card.
   - Run the *create_ card.py* script with the spreadsheet option pointing to the `card_list/card_list.csv`.

```
python3 create_card.py -S card_list/card_list.csv
```

   - The card images will output to the `../images/generated_cards` folder.

6.7.3 Understanding the Output

- Cards are saved to `card_list/card_list.csv` in the format:

```
NAME;TYPE;SUBTYPES;LEVEL;IMAGE;ATTACK;DEFENSE;EFFECT1;EFFECT2;SERIAL;RARITY;
    TRANSLUCENCY;EFFECT1_STYLE;EFFECT2_STYLE
Ember Foxling;Fire;Beast;1;../images/units/fire/Ember Foxling.jpg;360;140;...;...;
    a7626d4dd0045a;0;50;None;None
```

- The `IMAGE` field uses a relative path from the script directory.
- `RARITY` defaults to 0 (not editable in the UI yet).

6.7.4 Tips and Troubleshooting

- **Duplicate Names:** If a name exists in `card_list/card_list.csv`, the UI skips to the next image. Ensure image filenames are unique or edit the name manually.
- **Image Not Loading:** Verify the image path is correct and the file exists. Check console logs for `Invalid image path` errors.
- **Effects Not Generating:** Ensure `effects_with_placeholders.csv` exists and contains valid effect templates.
- **UI Not Responding:** Check for errors in the console and ensure all dependencies are installed.

This interface streamlines card creation for TTCG, combining manual input with automated features like image cycling and effect generation. Experiment with different combinations to build your card collection!