Gitaxian Probe: Modelling Magic the Gathering Using Artificial Intelligence

CISC 352 - Project Documentation

Nicholas Tillo & Raksha Rehal

Department of Computing, Queen's University

Professor Christian Muise

March 7, 2024

Introduction

This project is hinged on modelling various aspects of the card game known as “Magic: The Gathering” (MTG) using three different techniques central to the field of AI, i.e. planning, bayesian networks, and deep learning. We will be employing PDDL and Python to (1)

Project Setting Summary

Our project is an exploration of the techniques of artificial intelligence and its ability to be used to optimize and analyze the game of MTG, its turn structure, and its cards. The goal is to **optimize play and streamline feedback** during games to allow new players to better understand correct lines of play. MTG has over 30,000 unique cards, each with their own abilities and effects that produce entirely specialized sequences of play depending on the way that these cards are used in combination with one another. The cards demonstrate a variety of complex features that may be tricky to grasp and work with, especially for beginners to the game. Hence, this project aims to teach newcomers how to best optimize their plays using high-level abstractions of the core gameplay.

We chose this project setting for several reasons. First, there are many aspects of the game that would fit into each of the three AI approaches nicely — i.e. since (1) planning out optimal sequences of actions is one of the core gameplay features of MTG, (2) many game actions and outcomes depend on the probabilities of random variables, and (3) there is vast complexity that comes with each card, leading to an abundance of data that is able to be processed in order to produce results that — when analyzed — can help better optimize and guide gameplay. Playing MTG requires use of all sorts of cognitive architectures, and hence produces a strong setting that is able to be modeled using different AI techniques. Additionally, there are many scenarios where the application of the models developed using each approach for this setting would benefit current or even new players to MTG. There are no resources for players to learn or visualize strategies that work in the game, besides trial and error. Hence, developing models that aid players in understanding how to optimize their plays can be a useful tool to help engage players in a powerful and creative way with the game.

Approach 1: Planning, “Serum Visions”

High-Level Approach Information

The problem that we aim to solve with this approach, stemming from our problem setting, is the following: what is the most optimal sequence of moves that will result in a winning outcome, given a current board state for the opponent?

We will be modelling the scenario in this question in PDDL in order to generate an optimized sequence of actions such that each move will lead the player to winning the game (getting the opponent’s life-total to 0) in the shortest amount of turns. We are using a very simplified model of the game, which includes the following major changes to reduce the complexity of the game (each component is described in detail in the following sub-section):

- Removing the aspect of a deck of cards — i.e., players will only have access to the seven cards in their hand at the beginning of the game.
- Removing complexity from the opponent’s turn (i.e., they do not play cards or make any decisions, other than to just attack the player each turn with the predefined creatures that they start the game with).
- Removing all text from creatures, and only including the keywords “vigilance” and “haste”.
- Reducing the phases of a standard turn in MTG to have just “main, attack, block, end” and an additional phase that we defined, as “opponent”
- Removing card types other than lands, creatures, and sorceries.
- Restricting costs of spells to only consist of only 1 coloured pip and another specified amount of colourless mana (i.e. removing the concepts of dual-coloured or multi-valued colour pips).

Thus, for clarity, the following is a summary of the model that we will be representing: the game will be an adversarial card-game between a player and an opponent. Each agent will start with a predetermined “life-total” (such as 20), and the goal of the game will be for the player to get their opponent’s life total to zero, given that the player has not played any cards yet, and that the opponent already has some number of cards on the board. There are 4 phases of play for the player; i.e. “main”, “attack”, “block”, and “end”. The player can play 1 “land” on each of their turns. Lands will “tap” for 1 “mana” of the colour that they are, and the player can use their cumulative mana per turn to play “creatures” or “sorceries”. Once a card is tapped, it effectively becomes unusable for any further actions, until the beginning of the player’s next main phase; at which point everything that was tapped on their previous turn becomes untapped and usable again. The player can play creatures only on their main phase, and each creature will have 3 aspects to them: (1) their power, a number, (2) their toughness, a number, and (3) an optional keyword that has a certain effect. Creatures cannot “tap” the turn they enter as they have “summoning sickness” for that turn. Moving onto the attack phase, any creature that is not

tapped or summoning sick may attack the opponent, with the intention of dealing damage equal to its power to the opponent's life-total. The opponent will also have creatures, and these creatures can defend them if they are untapped. To defend oneself, the opponent may assign an untapped creature to "block" one other attacking creature (from the opposing agent's end). The result of a creature who blocks an attacking creature is the following: the attacking creature assigns damage equal to its power to the blocking creature's toughness, and vice-versa. This way, no damage carries through to the opposing player's life-total; however, if either creature's power is equal to or greater than that of the opposing (could be either the blocking or attacking) creature's toughness, then that (or those) creatures will "die" and be removed from the board. After these attacking and blocking phases, the player will progress to the end phase, where the turn will be "cleaned up" for the player in order to prepare for their next main phase. The following phase is called "opponentAttack", where the opponent will simply declare their attackers to be all their current creatures. The "opponentBlock" phase takes place, where the player will have to determine the most effective blocks, and damage will be exchanged as a result of this phase occurring. After it is complete, the turn passes back to "main" where the player will take their next turn in a similar fashion, once again. Both the goal and the metric towards the goal will remain the same.

Domain Description - The Board State

The game starts at an arbitrary state for the opponent, and a completely empty board for the player. The player has a hand of x cards that they are able to play out in order to defend themselves, and chip down their opponent's health at the same time. The player knows the following facts about the game (represented in the PDDL domain file for this approach's implementation):

Basic Types & Constants

- The defined types are $colour$, $land$, $creature$, $spell\_effect$, $player$, and $phase$
- $colour$, $spell\_effect$, and $phase$ will have associated constants.

Cards Representations

- **Lands** are represented as **objects** which will have the following naming convention:

$$L\_(land\ name)(count)$$

  e.g. "`L_Mountain1`"
  - The "L" is simply for accessibility for those unfamiliar with the game to recall that this object is representing one of the following *land names*:
    - Plains
    - Island
    - Swamp
    - Mountain
    - Forest
  - The suffix of an integer n represents the index of lands that the player has of a specific type. E.g., if we have 3 Forests, our objects will be:
    - `L_Forest1`
    - `L_Forest2`
    - `L_Forest3`
  - The status of the lands are represented by the **predicates**:
    - `is_tapped ?card`
      Indicates whether or not a land is tapped; if it is, it cannot be used to produce mana until it next becomes untapped.
    - `is_in_hand ?card`
      Represents whether a certain card has been played or not, being true for in hand, and false for on the battlefield.
    - `is_land_colour ?land ?colour`

Representing which of the 5 colours the land is (with Mountain = red, Swamp = black, Forest = green, Plains = white, Island = blue).

- The numeric fluents, i.e. **functions** associated with lands are the following:
  - `(available_mana ?colour)`
    Variable to keep track of how much mana for each colour there is.

  - `(available_mana_total)`
    Variable to keep track of how much total mana there is, regardless of colour.

- The **actions** relating to lands are the following:
  - `play_land`
    This action will allow the player to play their land, i.e. move a hand card from their hand onto the battlefield, in order to prep it for use.

  - `tap_land`
    By tapping their lands (only once a turn per land), players can actually use them to gather "mana", which is essentially the currency that can be used to pay for the costs of creatures and sorceries.

- **Creatures** are represented by the **objects** which will have the following naming convention:

$$C\_(card\ name)(count)$$

e.g. "`C_GrizzlyBear1`"

- The "C" is for accessibility reasons, indicating to users of this program that this object refers to a creature.
- The suffix of an integer n represents the index of lands that the player has of a specific type.

- The status of the creatures are captured in the following **predicates**:
  - `is_creature_colour ?creature ?colour`
    Indicates that a creature is one of the 5 colours, which will further imply that at least one mana of its colour must be spent to cast it.

  - `is_tapped ?card`
  - `is_owned_by_player ?creature`
  - `is_summoning_sick ?creature`
  - `is_attacking ?creature`

        - To know which creatures must be blocked,
        - And to not declare the same creature as attacking multiple times

    - `is_blocking ?creature`
      To know which creatures are blocking so that a creature is not declared as blocking multiple other creatures.

    - `has_vigilance ?creature`
    - `has_haste ?creature`

  - The **functions** associated with creatures are represented as the following numeric fluents:
    - `toughness ?creature`
    - `power ?creature`
    - `converted_mana_cost ?creature`
      Includes the one coloured pip that also must be accounted for when playing a creature of any colour.

- **Spells** are what we will call the mechanism of "sorceries" in MTG. They are represented with the naming convention:

$$S\_(spell\ name)(count)$$

Model Description

The opponent will be represented as having no cards in hand, and is not able to affect the game state at all, outside of performing their routine attacks on their turns. It will be a very one sided fight, but the goal of this approach is to model a sequence of actions on the player's end using the core gameplay features of MTG. Hence, the planner will attempt to create a path to victory from the given game state.

- We will be trying to run the model in order to maximize the `current_player_health` numeric fluent for the game problems. This will be done as we explore how to use "metrics" in PDDL…
- Since the goal of the model is for the player to win, i.e. to get the opponent's health to 0, the opponent's turns will be defined in a very rigid manner that forces them to play the game in a certain way, such that they do not contribute to that goal themselves (e.g. purposefully skip their attack and blocking phases to just allow the player to defeat them).
- Each action has in their precondition, `(>=(current_life_total_player)0)` so that we do not expand branches where our life is zero to save time. This also represents a game loss, so therefore it will not ever return a plan where we lose, ensuring correctness.

The Problem Files

Creatures

Bears	Goblins	Dross Crocodile	Knight	Aegis Turtle
				

Spells

Lightning Bolt,	Chaplain's Blessing
	

File 1 - Simple Setup

Game State:

- Enemy:
  - Creatures On Board:
    - Grizzly Bear
    - Grizzly Bear
- Player:
  - Cards In Hand:
    - Forest
    - Forest
    - Grizzly Bear
    - Grizzly Bear
    - Grizzly Bear
- Generated Action Path:

File 2 - Introducing Spells

Game State:

- Enemy:
    - Creatures On Board:
        - Grizzly Bear
        - Grizzly Bear
        - Grizzly Bear
- Player:
    - Cards In Hand:
        - Mountain
        - Forest
        - Island
        - Lightning Bolt
        - Grizzly Bear
        - Aegis Turtle
        - Lightning Bolt
- Generated Action Path:

File 3 - Complex Creatures

Game State:

- Enemy:
    - Creatures On Board:
        - Grizzly Bear
        - Goblin
        - Goblin
        - Dross Crocodile
        - Knight
- Player:
    - Cards In Hand:
        - Mountain
        - Swamp
        - Forest
        - Goblin
        - Goblin
        - Grizzly Bear
        - Dross Crocodile
    - Cards On Field
        - Mountain
        - Knight
        - Aegis Turtle

File 4 - Overall Complex Boardstate
[TBD]

Approach 2: Networks

The question we are attempting to solve with a Bayesian approach is: “What should I prioritize in this game, what resource is the most important?” We will take a look at how all the variables in the game affect each other. Which Variables we choose will be gathered from looking at previous tournament play games to find what states lead to wins. Possible variables can be, cards in hand, creatures on board, colors in deck, cards drawn in the game. All of these variables have interconnected relationships with each other. And therefore we can map it as a network of possibilities. We can eventually answer questions like: What is the most important factor to winning?

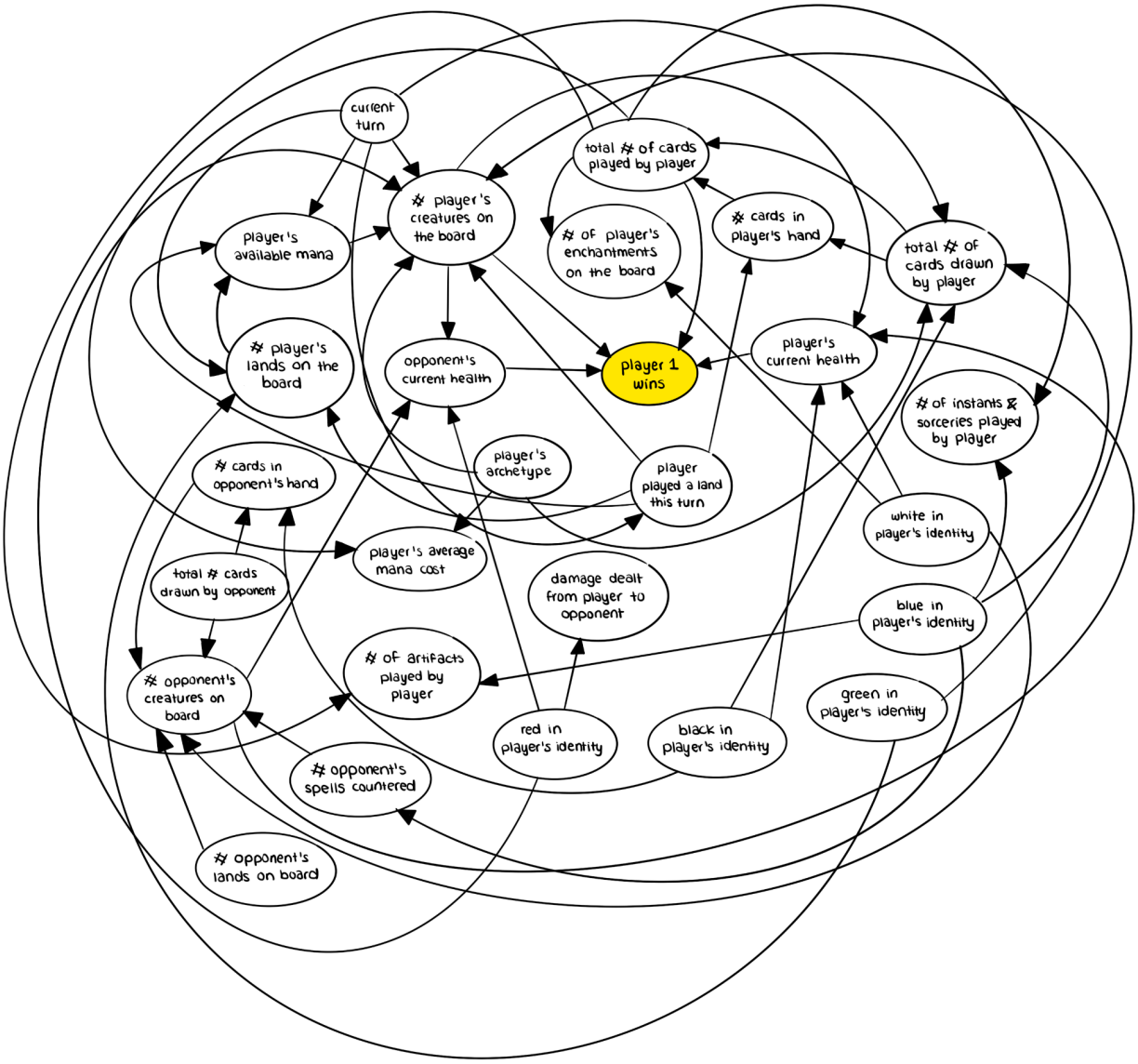
Model

Variables

Node	**Values**	**Node**	**Values**
Player Winning	T/F	Current Opponent Health	0-20
Available Mana	0-10	Creatures On Player’s Board	0-10
Have Played A Land This Turn	T/F	Creatures On Opponent’s Board	0-10
Turn Number	0-10	Has Red In Colour Identity	T/F
Player Cards In Hand	0-7	Has Blue In Colour Identity	T/F
Mulligans Taken	0-7	Has BlackIn Colour Identity	T/F
Average Mana Cost In Deck	1,1.5,2,2.5,3	Has Green In Colour Identity	T/F
Current Health	0-20	Has White In Colour Identity	T/F
Damage Dealt to Opponent	0-20	Opponent Cards In Hand	0-7
Number Of Cards Drawn Over Game	0-20	Number Opponent Spells Countered	0-5
Number Of Artifacts Played	0-20	Number Of Instants and Sorceries Played	0-20

Number Of Enchantments Played	0-20		Number Of Cards Played	0-20
Number Of Opponent's Lands	0-20		Number Opponents Cards Drawn	0-20
Deck Archetype	Aggro, Midrange, Control, Combo			

Bayesian Network



Brainstorm

- How to gather all the data about the relationships between nodes?
    - Get the information of pro data and manually input snapshots of the data.
    - Basically giving 400 snapshots.
    - Using the method in class we will just get the NAIVE approach of getting the percent of states that happened within the set of observed values and putting that as the evidence values.
    - Create the tables using these data examples.

Approach 3: Deeplearning

The question we are attempting to solve with a Deep Learning approach is: “How mad will this proposed card make people?” Using the variety of information stored on a MTG card we will attempt to train a deep-learning model to estimate the "salt score" of MTG cards. The salt score of a card is the community voted “How much do I hate playing against this card” value and can be found on the website EDHREC.

We will train this deep learning model on the values on the card, Things like mana cost, text on card, power, toughness ETC. These variables will then all be combined by our project to create a model that is capable of taking in new information, and attempting to give it a fitting salt score.

Our planned approach is to:

- Gather required resources from online sources in order to create a labeled/supervised dataset, we will then use this dataset in order to train a deep learning model to recognize what makes a card salty or not.
    - This will utilize python libraries to sort and manipulate the data, and a separate python library to train the model.

Model

Prepping the data:

- After downloading the JSON file that contains all the information about each card, they must be extracted into a separate
- The data is then stored into a list of dictionaries, each entry in the list represents the cards.
- With 93457 entries in the list there is more than enough data to train a deep learning model. I have a feeling however that there are not 93000 cards in magic, and thus things like art cards and promo cards are getting included.
- In order to filter these out, we will only be taking cards that are legal in 1 or more formats. (I.E can be played in at least 1 tournament somewhere in the world)
- Likely using K-Fold as a method of testing and validating the model.

- The factors (keys of the dictionary) that have been extracted about EACH card that will contribute to the training of the model will be:

```
['object', 'id', 'oracle_id', 'multiverse_ids', 'mtgo_id',
'mtgo_foil_id', 'tcgplayer_id', 'cardmarket_id', 'name', 'lang',
'released_at', 'uri', 'scryfall_uri', 'layout', 'highres_image',
'image_status', 'image_uris', 'mana_cost', 'cmc', 'type_line',
'oracle_text', 'power', 'toughness', 'colors', 'color_identity',
'keywords', 'legalities', 'games', 'reserved', 'foil', 'nonfoil',
'finishes', 'oversized', 'promo', 'reprint', 'variation', 'set_id',
'set', 'set_name', 'set_type', 'set_uri', 'set_search_uri',
'scryfall_set_uri', 'rulings_uri', 'prints_search_uri',
'collector_number', 'digital', 'rarity', 'flavor_text',
'card_back_id', 'artist', 'artist_ids', 'illustration_id',
'border_color', 'frame', 'full_art', 'textless', 'booster',
'story_spotlight', 'edhrec_rank', 'penny_rank', 'prices',
'related_uris', 'purchase_uris']
```