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

April 7, 2024

Contents

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”

Applying the Problem Setting

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 creatures that the player has of a specific name.

- 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:

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

e.g. "`S_LightningBolt1`"

  - The "S" indicates to users of this program that this object refers to a spell.
  - The suffix of an integer n represents the index of lands that the player has of a specific type.

  - Information about the spells are captured in the following **predicates**:
    - `(spell_effect ?effect - spell_effect ?spell - spell)`
      - ?effect will refer to either "heal" or "deal", which will be an indicator that the spell of name ?spell will either (1) heal the player for a certain amount or (2) deal damage to a certain creature.
    - `(is_spell_colour ?spell - spell ?colour - colour)`

  - The **functions** associated with spells are represented as the following numeric fluents:
    - `(spell_value ?spell - spell)`
      - Indicates a value for healing/dealing to a life total or toughness.
    - `(spell_converted_mana_cost ?spell - spell)`

Model Description

The opponent will be represented as having no cards in hand, no lands played, and starting with all their creatures on their board (with summoning sickness). The opponent is not able to affect the game state at all, outside of performing their routine attacks and blocks during their delegated phases. It will be a very one sided fight since 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 for the player from the given game state.

- We will be trying to run the model in order to get the `current_life_total_enemy` numeric fluent to 0 or less for the first three problem files. This will be done via setting goals accordingly.
- Since the aim 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 (i.e. we do not want the opponent to purposely 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 way we advance the game is by defining game phases where certain actions can be executed. These actions must be performed in a certain order, which will be defined by the preconditions for each action. For example, the player can win by attacking the opponent over and over again with a creature, doing damage to them. But in order to get this creature to a point where it is able to attack, the other rules of the game must be met, for example, first a land must be played, the creature must not have summoning sickness, etc. Each of these preconditions can be met via other actions (e.g. "play_land"), and all of these actions in combination with each other have their own preconditions and effects. Brought together, PDDL will output a legal sequence of actions representing the turn- and game-structure of MTG.
- We will be using the online PDDL editor for code, and the 2020 ENHSP planner to output the resulting plan.

Cards Used

The following cards are used in our state-space representation.

Table 1.1: *Each card as a visual reference for later use*

Creatures

Grizzly Bear | Raging Goblin | Dross Crocodile | Knight | Aegis Turtle

Spells

Lightning Bolt | Chaplain's Blessing

Lands

Plain | Island | Swamp | Mountain | Forest

Plans Generated

We explored four different problem settings for the domain file that we defined. In this section, you will first find the starting state, as defined by each problem file. These will be represented in a visual setting of the board in its entirety. You will then find the resulting plan for each problem file, as well as annotated comments along the right hand-side, stepping through each action and describing what is happening in terms of MTG gameplay. You can use the hyperlinks attached to each file name in order to skip to the analysis section for the corresponding problem file.

`Starting State for ProblemFile1.pddl`

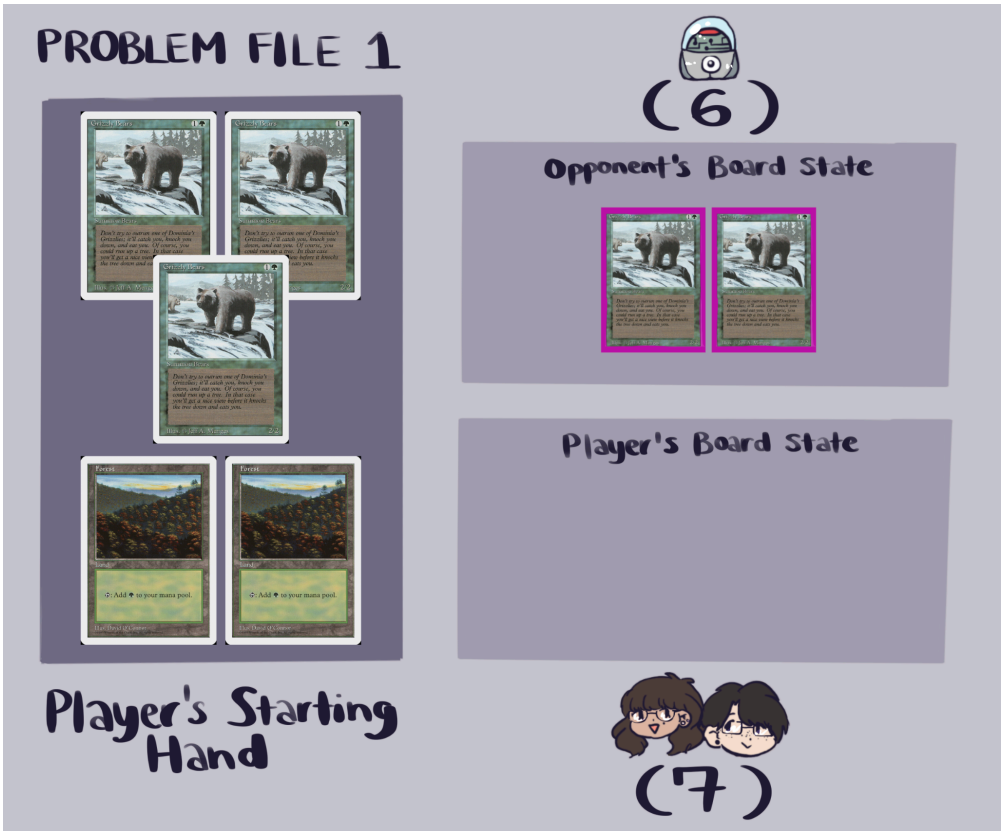


Figure 1.1: *The starting state of the first problem file, showing hand, board state, and healths.*

`ProblemFile1.ppdl`

Plan	Explanation
`problem solved`	
`0.0: (play_land l_forest1)`	Player's **"main"** phase: play one forest.
`1.0: (pass)`	Pass from "main" to **"attack"** phase.
`2.0: (pass)`	Pass from "attack" to **"block"** phase; We have no creatures to attack, so nothing happens in this phase yet.
`3.0: (all_opp_creatures_block)`	Indication that all the opponent's creatures are blocking.

```
4.0: (pass)                                  Pass from "block" to "end" phase.
5.0: (clear)                                 Process any damage taken.

6.0: (untap_opponent_cards                   Untap opponent's creatures.
c_grizzlybear2)
7.0: (untap_opponent_cards
c_grizzlybear4)
8.0: (pass)                                  Pass from player's "end" to opponent's
                                                                    "attack".
9.0: (enemy_attack c_grizzlybear2)           Declare what creatures the opponent is
10.0: (enemy_attack c_grizzlybear4)                          attacking us with.
11.0: (all_opp_creatures_attack)                Indication that all the opponent's
                                                          creatures are attacking.
12.0: (pass)                                 Pass from opponent's "attack" to "block";
                                                 We do not have any creatures, thus we
                                                                    cannot block.
13.0: (pass)                                 Pass from opponent's "block" to "end".
14.0: (opponent_clean_up c_grizzlybear2)     Because it was not blocked, we take the
15.0: (opponent_clean_up c_grizzlybear4)                damage during this clean-up.
16.0: (opponent_clear)                          Indication that all creatures' statues
                                                          and health have been reset.
17.0: (pass)                                 Pass from opponent's "end" to player's
                                                                    "main".
18.0: (play_land l_forest2)                                  Play a second forest.
19.0: (tap_land l_forest1 green)               Tap both forests, to produce 2 green
20.0: (tap_land l_forest2 green)                                            mana.
21.0: (play_creature c_grizzlybear1          Play a Grizzly Bear 1 with our available
green)                                                                      mana.
22.0: (pass)                                          Pass from "main" to "attack";
                                             Our Grizzly Bear 1 has summoning sickness
                                                       so he cannot attack this turn.
23.0: (pass)                                 Pass from "attack" to "block" phase;
24.0: (all_opp_creatures_block)                 Indication that all the opponent's
                                                          creatures are blocking.
25.0: (pass)                                  Pass from "block" to "end" phase.
26.0: (untap_opponent_cards                     Untap the opponent's creatures.
c_grizzlybear2)
27.0: (untap_opponent_cards
c_grizzlybear4)
28.0: (clear)
                                              Process any damage taken, reset creature
29.0: (pass)                                                              statuses.
30.0: (enemy_attack c_grizzlybear2)            Pass from player's "end" to opponent's
31.0: (enemy_attack c_grizzlybear4)                                       "attack";
32.0: (all_opp_creatures_attack)             They will attack with all their creatures
33.0: (pass)                                                                again.
34.0: (player_block_creature                 Pass from opponent's "attack" to "block".
c_grizzlybear4 c_grizzlybear1)                 Choose to block the opponent's Grizzly
35.0: (kill_creature c_grizzlybear1)                    Bear 4 with our Grizzly Bear 1
36.0: (kill_creature c_grizzlybear4)           Both creatures have the same power and
37.0: (can_pass_from_block)                        toughness of 2; so, they both die.
                                              Indication that damage between creatures
                                              has been exchanged, and that we can pass
38.0: (pass)                                                  from the block phase.
```

```
39.0: (opponent_clean_up c_grizzlybear2)
40.0: (untap_player_cards c_grizzlybear1)
41.0: (opponent_clean_up c_grizzlybear1)
42.0: (opponent_clean_up c_grizzlybear4)
43.0: (opponent_clear)
44.0: (untap_player_cards l_forest1)
45.0: (untap_player_cards l_forest2)
46.0: (pass)
47.0: (tap_land l_forest1 green)
48.0: (tap_land l_forest2 green)
49.0: (play_creature c_grizzlybear5
green)
50.0: (pass)
51.0: (pass)
52.0: (all_opp_creatures_block)
53.0: (pass)
54.0: (untap_opponent_cards
c_grizzlybear4)
55.0: (untap_opponent_cards
c_grizzlybear2)
56.0: (clear)
57.0: (pass)
58.0: (enemy_attack c_grizzlybear2)
59.0: (all_opp_creatures_attack)
60.0: (pass)
61.0: (player_block_creature
c_grizzlybear2 c_grizzlybear5)
62.0: (kill_creature c_grizzlybear2)
63.0: (kill_creature c_grizzlybear5)
64.0: (can_pass_from_block)
65.0: (pass)
66.0: (untap_player_cards c_grizzlybear5)
67.0: (opponent_clean_up c_grizzlybear5)
68.0: (opponent_clean_up c_grizzlybear2)
69.0: (opponent_clear)
70.0: (untap_player_cards l_forest1)
71.0: (untap_player_cards l_forest2)
72.0: (pass)
73.0: (tap_land l_forest1 green)
74.0: (tap_land l_forest2 green)
75.0: (play_creature c_grizzlybear3
green)
76.0: (pass)
77.0: (pass)
78.0: (all_opp_creatures_block)
79.0: (pass)
80.0: (untap_opponent_cards
c_grizzlybear2)
```

Pass from **opponent's** "block" to **"end"**.

Take the damage from Grizzly Bear 2. We can untap our Grizzly Bear 1, but notice that he will be unable to take any more game actions with him (i.e. attacking and blocking), since he is now dead.

Reset statuses on the opponent's Grizzly Bears; notice that Grizzly Bear 4 will no longer take game actions either.

Pass from opponent's "end" to player's **"main"**.

Tap both forests for enough mana to play our second Grizzly Bear, Grizzly Bear 5.

Pass from "main" to **"attack"**.

Pass from "attack" to **"block"**.

Pass from "block" to **"end"**.

Pass from "end" to **opponent's "attack"**.

They will, by default, attack with their one remaining creature.

Pass from **opponent's** "attack" to **"block"**.

Choose to block their Grizzly Bear 5 with our Grizzly Bear 2.

Both creatures die.

Pass from **opponent's** "block" to **"end"**.

Pass from opponent's "end" to **"main"**.

Tap our forests for mana, and play our last Grizzly Bear.

Pass from "main" to **"attack"**.

Pass from "attack" to **"block"**.

Pass from "block" to **"end"**.

```
81.0: (clear)
82.0: (pass)
83.0: (all_opp_creatures_attack)            Pass from "end" to opponent's "attack";
                                                We see here no indications of which
                                            creatures the opponent attacks with; and
                                            this is because they no longer have any.
                                            All they will be able to do for the rest
                                              of this game is pass their turns, with
84.0: (pass)                                                    nothing else to do.
85.0: (pass)                              Pass from opponent's "attack" to "block".
86.0: (opponent_clear)                      Pass from opponent's "block" to "end".
87.0: (untap_player_cards c_grizzlybear3)
88.0: (pass)
89.0: (pass)                                Pass from opponent's "end" to "main".
90.0: (attack_opponent c_grizzlybear3)              Pass from "main" to "attack".
                                                 Declare that our Grizzly Bear is
91.0: (pass)                                                          attacking.
92.0: (all_opp_creatures_block)                   Pass from "attack" to "block".
93.0: (pass)                                 Opponent has no creatures to block with.
94.0: (clean_up c_grizzlybear3)                         Pass from "block" to "end".
95.0: (clear)                                      Opponent takes 2 damage from this
                                               creature. Their life total is now at
96.0: (pass)                                                              6-2=4.
97.0: (all_opp_creatures_attack)            Pass from "end" to opponent's "attack".
98.0: (pass)
99.0: (pass)                              /*
100.0: (opponent_clear)                       For the rest of this game, the player
101.0: (untap_player_cards                   will simply skip their main phase (with
c_grizzlybear3)                             nothing else to play), attack with their
102.0: (pass)                                 Grizzly Bear, and pass the turn. They
103.0: (pass)                                will then wait for the opponent to pass
                                              to their end, such that the player can
                                             untap their Grizzly bear and repeat the
                                                         process by attacking again.

                                                                              */
                                                   Opponent takes 2 damage from this
104.0: (attack_opponent c_grizzlybear3)          creature. Their life total is now at
105.0: (pass)                                                             4-2=2.
106.0: (all_opp_creatures_block)
107.0: (pass)
108.0: (clean_up c_grizzlybear3)
109.0: (clear)
110.0: (pass)
111.0: (all_opp_creatures_attack)
112.0: (pass)
113.0: (pass)
114.0: (opponent_clear)
115.0: (untap_player_cards
c_grizzlybear3)
116.0: (pass)
117.0: (pass)
118.0: (attack_opponent c_grizzlybear3)
119.0: (pass)
```

120.0: (all_opp_creatures_block) 121.0: (pass) 122.0: (clean_up c_grizzlybear3)	Opponent takes 2 damage from this creature. Their life total is now at 2-2=0. Player wins the game! :)

Starting State for ProblemFile2.pddl

Figure 1.2: *The starting state of the second problem file, showing hand, board state, and healths.*

ProblemFile2.ppdl

Plan	Explanation
problem solved 0.0: (play_land l_island1) 1.0: (pass) 2.0: (pass) 3.0: (all_opp_creatures_block) 4.0: (pass) 5.0: (clear) 6.0: (untap_opponent_cards c_goblin1) 7.0: (untap_opponent_cards c_grizzlybear2) 8.0: (pass)	**Player's turn:** play an island and pass. **Opponent's turn:** attack us with their Grizzly Bear and Raging Goblin (after their status of summoning sickness has been lifted). Player has no creatures to block the

```
9.0: (enemy_attack c_grizzlybear2)
10.0: (enemy_attack c_goblin1)
11.0: (all_opp_creatures_attack)
12.0: (pass)
13.0: (pass)
14.0: (opponent_clean_up c_goblin1)
15.0: (opponent_clean_up c_grizzlybear2)
16.0: (opponent_clear)
17.0: (pass)
18.0: (play_land l_plains1)
19.0: (tap_land l_plains1 white)
20.0: (tap_land l_island1 blue)
21.0: (play_creature c_knight1 white)
22.0: (pass)
23.0: (pass)
24.0: (all_opp_creatures_block)
25.0: (pass)
26.0: (untap_opponent_cards c_goblin1)
27.0: (untap_opponent_cards
c_grizzlybear2)
28.0: (clear)
29.0: (pass)
30.0: (enemy_attack c_grizzlybear2)
31.0: (enemy_attack c_goblin1)
32.0: (all_opp_creatures_attack)
33.0: (pass)
34.0: (player_block_creature c_goblin1
c_knight1)
35.0: (kill_creature c_goblin1)
36.0: (can_pass_from_block)
37.0: (pass)
38.0: (opponent_clean_up c_goblin1)
39.0: (untap_player_cards c_knight1)
40.0: (opponent_clean_up c_knight1)
41.0: (opponent_clean_up c_grizzlybear2)
42.0: (opponent_clear)
43.0: (pass)
44.0: (pass)
45.0: (attack_opponent c_knight1)
46.0: (pass)
47.0: (all_opp_creatures_block)
48.0: (pass)
49.0: (untap_opponent_cards c_goblin1)
50.0: (untap_opponent_cards
c_grizzlybear2)
51.0: (clean_up c_knight1)
52.0: (clear)
53.0: (pass)
54.0: (enemy_attack c_grizzlybear2)
55.0: (all_opp_creatures_attack)
56.0: (pass)
57.0: (pass)
58.0: (opponent_clean_up c_grizzlybear2)
59.0: (opponent_clear)
60.0: (pass)

61.0: (pass)
62.0: (attack_opponent c_knight1)
63.0: (pass)
64.0: (all_opp_creatures_block)
65.0: (pass)
```

incoming damage with, so they take the damage. Player's life total is now 8-2-1=5.

Player's turn: play a plains. Tap both lands in order to play a knight, and pass.

Opponent's turn: attack like usual with both of their creatures. The player will choose to block the goblin with the knight. Since the knight is 2/2 and the goblin is 1/1, the goblin dies but the knight does not (he has 1 damage marked on him, but this resets before our next turn).

Player's turn: The player chooses not to play the Aegis Turtle in their hand, and simply attacks with their knight. The knight has vigilance, so it does not tap. The opponent's bear is tapped, so there is no creature on the opponent's field able to block our knight. The damage goes through and their life total becomes 4-2=2.

Opponent's turn: they untap their bear and attack with him. The player chooses not to block with their knight, as this would kill the knight, and this is the player's only damaging creature (keeping in mind that the goal is to get the opponent's health to 0). The player takes the damage, and their life total becomes 5-2=3.

Player's turn: attack with the knight. The opponent's bear is tapped and cannot block, so their life total goes down to 2-2=0.

66.0: (clean_up c_knight1)	Player wins the game! :)

Starting State for ProblemFile3.pddl

Figure 1.3: *The starting state of the third problem file, showing hand, board state, and healths.*

ProblemFile3.ppdl

Plan	Explanation
0.0: (play_land l_mountain1) 1.0: (tap_land l_mountain1 red) 2.0: (play_spell red s_lightning_bolt2 c_grizzlybear1) 3.0: (kill_creature c_grizzlybear1) 4.0: (can_pass_from_main) 5.0: (pass) 6.0: (pass) 7.0: (all_opp_creatures_block) 8.0: (pass) 9.0: (clear) 10.0: (untap_opponent_cards c_grizzlybear1) 11.0: (untap_opponent_cards c_grizzlybear2) 12.0: (pass) 13.0: (enemy_attack c_grizzlybear2)	**Player's turn:** play mountain, tap it for red, cast the Lightning Bolt spell, target the opponent's Grizzly Bear 1 and kill it. Note: can_pass_from_main is another "clear" action that correctly forces the system to kill the correct creatures after casting a spell **Opponent's turn:** they choose to attack with the Grizzly Bear 2. Player has no

```
14.0: (all_opp_creatures_attack)
15.0: (pass)
16.0: (pass)
17.0: (opponent_clean_up c_grizzlybear2)
18.0: (opponent_clear)
19.0: (untap_player_cards l_mountain1)
20.0: (pass)
21.0: (play_land l_plains1)
22.0: (tap_land l_mountain1 red)
23.0: (play_creature c_goblin1 red)
24.0: (tap_land l_plains1 white)
25.0: (play_spell white
s_chaplains_blessing1 c_goblin1)


26.0: (can_pass_from_main)
27.0: (pass)
28.0: (pass)
29.0: (all_opp_creatures_block)
30.0: (pass)
31.0: (untap_opponent_cards
c_grizzlybear2)
32.0: (clear)
33.0: (pass)
34.0: (enemy_attack c_grizzlybear2)
35.0: (all_opp_creatures_attack)
36.0: (pass)
37.0: (pass)
38.0: (opponent_clean_up c_grizzlybear2)
39.0: (opponent_clear)
40.0: (untap_player_cards l_mountain1)
41.0: (pass)
42.0: (tap_land l_mountain1 red)
43.0: (play_spell red s_lightning_bolt1
c_grizzlybear2)
44.0: (kill_creature c_grizzlybear2)
45.0: (can_pass_from_main)
46.0: (pass)
47.0: (attack_opponent c_goblin1)
48.0: (pass)
49.0: (all_opp_creatures_block)
50.0: (pass)
51.0: (untap_opponent_cards
c_grizzlybear2)
52.0: (clean_up c_goblin1)
53.0: (clear)
54.0: (pass)
55.0: (all_opp_creatures_attack)
56.0: (pass)
57.0: (pass)
58.0: (opponent_clear)
59.0: (untap_player_cards c_goblin1)
60.0: (pass)
61.0: (pass)
62.0: (attack_opponent c_goblin1)
63.0: (pass)
64.0: (all_opp_creatures_block)
65.0: (pass)
66.0: (clean_up c_goblin1)
67.0: (clear)
68.0: (pass)
```

creatures to block so their life total becomes 3-2=1.

Player's turn: untap their mountain, play a plains, tap the mountain for a red mana to play the Raging Goblin. Tap our plains for white mana to play another spell, i.e. The Chaplain's Blessing. This heals the Player for 5, so their life total becomes 1+5=6.

Note: Though the planner states that the goblin was healed with The Chaplain's Blessing, this is just a way to indicate that the spell was played, and a target needed to be stated. The player was not defined as an object, so it chose the goblin; but the effect of *healing the player* was still fulfilled.

Opponent's turn: attack with their bear, the Player chooses not to use their goblin to block, and takes the damage instead. Player's life total becomes 6-2=4.

Player's turn: tap mountain for a red mana to play another Lightning Bolt and kill the opponent's only other creature. We then attack the opponent with our goblin, dealing a damage to them and putting their life total at 3-1=2.

Opponent's turn: they have no creatures on the field anymore, so all they can do is pass their turn.

Player's turn: untap the goblin, and attack again for another damage, leaving the opponent's life total to be 2-1=1.

Opponent's turn: the opponent once again

69.0: (all_opp_creatures_attack) 70.0: (pass) 71.0: (pass) 72.0: (opponent_clear) 73.0: (untap_player_cards c_goblin1) 74.0: (pass) 75.0: (pass) 76.0: (attack_opponent c_goblin1) 77.0: (pass) 78.0: (all_opp_creatures_block) 79.0: (pass) 80.0: (clean_up c_goblin1)	has no creatures, so they must pass with no attackers. **Player's turn**: The goblin deals the final damage to the opponent, and their life total becomes 1-1=0. Player wins the game! :D

Starting State for ProblemFile4.pddl

Figure 1.1: *The starting state of the fourth problem file, showing hand, board state, and healths.*

ProblemFile4.ppdl

Note: the goal for this problem file is different from the previous three - as this time, we simply want to get the opponent's life total to be less than (or equal to) the player's. We set this particular goal as we wanted to observe what plan PDDL produced in order to get the player to be in an advantageous state over the opponent from a disadvantageous one.

Plan	Explanation

```
0.0: (play_land l_mountain1)
1.0: (tap_land l_mountain1 red)
2.0: (play_spell red s_lightning_bolt1
c_dross_crocodile2)
3.0: (kill_creature c_dross_crocodile2)
4.0: (can_pass_from_main)
5.0: (pass)
6.0: (pass)
7.0: (all_opp_creatures_block)
8.0: (pass)
9.0: (clear)
10.0: (untap_opponent_cards c_knight2)
11.0: (pass)
12.0: (enemy_attack c_knight2)
13.0: (all_opp_creatures_attack)
14.0: (pass)
15.0: (pass)
16.0: (opponent_clean_up c_knight2)
17.0: (opponent_clear)
18.0: (untap_player_cards l_mountain1)
19.0: (pass)
20.0: (play_land l_plains1)
21.0: (tap_land l_mountain1 red)
22.0: (tap_land l_plains1 white)
23.0: (play_creature c_goblin1 red)
24.0: (play_spell white
s_chaplains_blessing1 c_dross_crocodile2)
25.0: (can_pass_from_main)
26.0: (pass)
27.0: (attack_opponent
c_dross_crocodile1)
28.0: (attack_opponent c_goblin1)
29.0: (pass)
30.0: (opponent_block_creature c_goblin1
c_knight2)
31.0: (kill_creature c_goblin1)
32.0: (all_opp_creatures_block)
33.0: (pass)
34.0: (clean_up c_dross_crocodile1)
```

Player's turn: play a mountain and tap it for a red mana, and target the opponent's dross crocodile with a Lightning Bolt, killing it. Pass the turn from here.

Note: we start this game with a Dross Crocodile already played on the Player's field (not in their hand), since the online PDDL editor was unable to output a plan with a viable length, if we asked it to play the crocodile.

Opponent's turn: attack with their knight. The player has no creatures to block so they take the damage, and their life total goes down to 6-2=4.

Player's turn: play a plains. Tap the mountain for a red mana, play the goblin. Tap the plain for a white man, and play The Chaplain's Blessing to heal the Player for 5. Their life total becomes 4+5=11.

The goblin has haste, so we can attack with it. Our crocodile also no longer has summoning sickness, so we can attack with it as well. The opponent blocks our goblin with their knight, killing our goblin. However, the opponent does not have any other creatures to block our crocodile, so his damage goes through. The opponent's life total is brought down to 12-5=7.

The goal is met :)

Results and Observations

General Observations

- PDDL provides the most optimal plan IF the opponent plays as poorly as possible. For the scope of this project, we did not implement any actions for the opponent to take outside of simply attacking and blocking on all their turns. In the future, it would be extremely interesting to expand this idea, and implement a fully-functional opponent who is also playing to win. This would likely require use of external software outside of PDDL, but the results might provide even more interesting insights in regards to the gameplay itself.
- PDDL untapped creatures and lands on the end steps (instead of the main phases). Technically, this does not have an impact in our model since we do not allow for "instant" spell types. The two spells implemented can only be cast on each player's main phase, and hence we allowed for this rule violation to slide, since it still followed the simplified rules of the game that we had abstracted.
- We had to include a lot of "checking" and "clearing" actions to the PDDL in order to force the model to take "unwanted" actions for the player. These "unwanted" actions moved further away from the goal state, and hence PDDL would simply choose *not* to take these actions, in spite of them being required for our model.
    - For example, if we *just* included a "kill_creature" action (with the same goal of getting the opponent's life down to 0), the planner would just *choose* not to take the kill_creature action on its own creature, even IF required, as it would make it harder to kill the opponent). I.e., when one of our creatures' toughness drops to 0, PDDL simply would choose not to kill this creature, although that was one of the effects of creatures having 0 toughness. This way, PDDL tried to do things like still attack with a technically dead creature, as - of course - the fastest way to get to the goal of killing the opponent is to never kill one's creatures!
    - This really slowed down the execution and lowered the bar of complexity for the problems that we wanted to solve with the model.
    - Every time we took an action that had a chance of killing a creature, it would be an effect that the planner could not pass to the next phase unless it checked that each creature was "clear" to move to the next phase if they were alive. This was done using universal preconditions.

Observations for ProblemFile1.pddl

- Very interestingly, the main priority for the planner was to protect the player's life, when in a disadvantaged state.
    - In theory, this makes sense since there would be no way for the planner to reach its goal of getting the opponent's life to 0 if their own life is 0, and the

preconditions for every action required to whittle down the opponent's life *requires* that the player's life itself is above 0.
    - Hence, this plan also provided an accurate sequence of decisions, reflective of gameplay that one could expect to see when observing a real magic game!
- One aspect of real-life that this planner does not capture is the hesitation that players have sometimes with letting their creatures die. In the given scenario for this problem, it certainly was the most optimal solution to block and kill both of the opponent's grizzly bears (while also killing two of our own in the process). New players in real-life tend to hesitate when having to sacrifice their creature in order to protect them in this way. Since PDDL will only output the most optimal, this aspect is not captured within the model in any sense.

Observations for ProblemFile2.pddl

- This problem file posed more of a challenge to the planner, as the scenario required it to decide what creatures were best to block under different situations. For example, the planner chose to block the 1/1 goblin with its 2/2 knight, allowing a 2/2 bear to hit the player.
    - This is a realistic, and good move. It is extremely interesting that the planner was able to determine this to be the optimal move. It follows along the line of using the player's life itself as a resource, and focuses on getting the opponent's life to 0, as well as putting them at a state of disadvantage, while maintaining the neutrality/advantage state that a player may be in.
    - In other words, because of the life total advantage, we don't prioritize preserving our life total, and are willing to sacrifice it so that our creature stays alive, ensuring future damage (in this problem setting) - after all, players themselves are unable to attack each other, but the creatures can, inherently increasing their value in the game.
- To extend our analysis, we could next try to add a metric that would seek to minimize the damage taken by the player, in order to see how this may affect the order of blocking produced by the planner. This would be a fun extension to the project, and would further be reflective of playstyles by different types of players, showing that different sorts of players may have different priorities in the game, leading them to have entirely different playstyles and plans for approaching the game.

Observations for ProblemFile3.pddl

- This problem file introduced the prospect of non-creature spells that have numerical effects. That is; we implemented two spells that could directly affect the toughness of creatures or the life of the player. We wanted to observe how the planner utilized these

spells with regards to the current board state, and our findings were about what we expected.
    - When given the choice between playing creatures and spells, the planner output a plan that chose to use spells to first kill the opponent's creatures, before playing our own and healing the player.
    - Neither the player nor the opponent start the state at an advantage. Interestingly, the planner chose *not* to heal the player *first* (which would have increased their life total, but put them at an advantageous position only in terms of health and nothing else). In the long run, playing the Chaplain's Blessing was not the most optimal first play, as the player would then only be able to cast one red spell on their second turn, due to their hand consisting of 3 red spells/creatures and 1 white one.
        - PDDL was able to recognize this! The planner opted first to use Lightning Bolt on one of the opponent's creatures in order to minimize the damage taken on the first round.
- Hence, PDDL *did* in fact output the most optimal plan for the problem file, ordering the manner in which the spells were played correctly in order to ensure that the goal was met most efficiently.
    - It is very interesting that the planner produces an optimal output, due to the preconditions of most every action needed to meet the goal requiring that the player has a life total above 0.
- Analyzing the plan produced by this problem file helped us answer the question who or what we should cast our spells on.
- We lose a lot of information by implementing this simplified model, since the only types of spells that we implement are "sorceries", which can only be played on the player's main phase. In MTG, there are spells called "instants" that can be played at any time, in response to other spells and creatures being played - even on the opponent's turn! It would be really interesting to see what strategies PDDL might have implemented. It is likely that these strategies would have reflected practices in real play of MTG, for example, by holding one's "instant" spell until the end step of their opponent, so as to give them as little information as possible.

Observations for ProblemFile4.pddl

- The main purpose of this problem file was to test how different creatures and spells interacted. This problem file had a unique goal from the other, i.e. simply to get the player's life total to be equal to or greater than the opponent's. The reasoning behind this shift was: we sought to observe what an "advantageous" state meant to the planner, and analyze the steps that it took to get to such a point.

- In this scenario, we can see that the Lightning Bolt had multiple targets, but the planner chose the Dross Crocodile, even though the Knight had a keyword (vigilance) and a higher toughness stat, meaning that our goblin would die blocking it. Yet, the crocodile was still chosen to die. This makes sense, since the goal is to get the player's health to be equal to or above that of the opponent's, and taking 5 damage from the Dross Crocodile when we were only at 6 health (and the opponent, at 12) would make reaching this goal impossible.
- Our finding did show an issue with the model, in which the opponent had a favourable block for the player (in this case, they blocked the 1/1 goblin rather than the 5/1 crocodile.) As the goal is to get the opponent's life total to be less than the player's, the opponent is also trying to meet this goal!
    - This of course, is not realistic at all, and in real-world settings, each player would try to minimize each other's health, while trying to stay as far *above* 0 as they can.
    - This seems to be a limitation that the problem has within PDDL, and would be better suited for a different medium.

Approach 2: Probabilistic Inference, “Knowledge Pool”

Applying the Problem Setting

The question we are attempting to solve by performing probabilistic inference is: what should I prioritize in this game as a player; what resource is the most important? We will take a look at how all the variables in the game affect each other. The variables that we choose will be gathered from looking at previous MTG tournament games in order to note our observations of each variable with regards to their parents (i.e. the conditional probabilities that will make up our Bayesian Network). The variables we have chosen to focus on will further be expanded on below, and their corresponding ranges of values are also indicated. All of these variables have interconnected relationships with each other, which is why we can map it as a network of probabilities.

Model

Selected Random Variables & Ranges

Table 2.1*: All Nodes in the bayesian network and their respective domains, the values that they cover*

Node	**Values**	**Node**	**Values**
Player Winning	T/F	Current Opponent Health	0-5, 6-10, 11-15, 16-20
Number Player Creatures	0-2, 3-5, 6-8	Num Player Nonland Permanents	0-2, 3-5, 6-8
Number of Player’s Lands	0-2, 3-5, 6-8	Player Archetype	Aggro Control Midrange Combo
Number Of Instants and Sorceries Played	0-2, 3-5, 6-8	Has Red In Colour Identity	T/F
Player Cards In Hand	0-2,3-5,6-7	Has Blue In Colour Identity	T/F
Current Health	0-5,6-10, 11-15, 16-20	Has BlackIn Colour Identity	T/F

Opponent Cards In Hand	0-2,3-5,6-7		Has Green In Colour Identity	T/F
Creatures On Opponent's Board	0-2, 3-5, 6-8		Has White In Colour Identity	T/F
Number Of Opponent's Lands	0-2, 3-5, 6-8			

Initial Problem Space

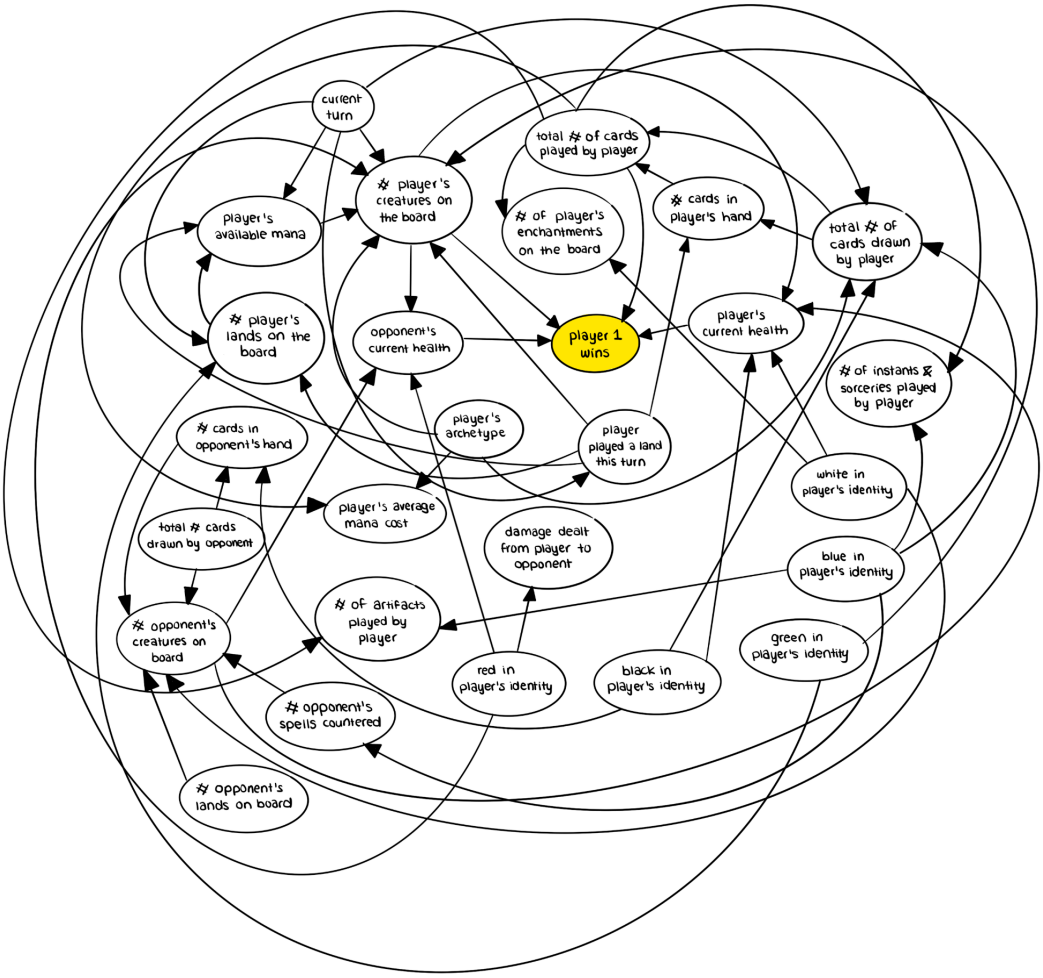


Figure 2.1*: Initial network that contains a large amount of edges and nodes, representing the effects of concepts upon each other on MTG.*

Finalized, Reduced Bayesian Network

After feedback from the Professor (and when trying to move from the brainstorming phase into implementation), we certainly realized that our initial Bayesian Network was much too large! We reduced the scope to something more viable by removing/combining nodes, and removing the weakest connections.

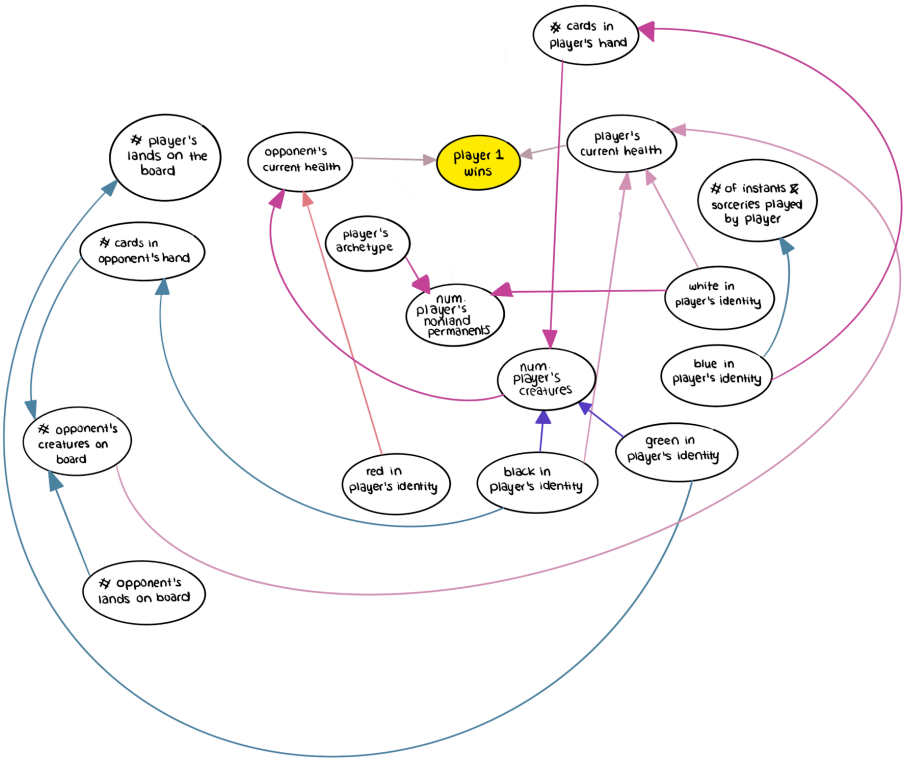


***Figure 2.2**: The simplified bayesian network with reduced amount of edges and nodes.*

Setup

Brainstorm

How are we to gather all the data about the relationships between nodes?

- Get the information of pro data and manually input snapshots of the data, basically analyzing 300 snapshots and recording the instantiation of the variables in this moment.
- 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 setting that as the evidence values.
- We will then create the tables using these data examples.

What Python libraries will we be using to aid implementation of our Bayesian Network?

- We attempted to work with the `bayespy` resource that was recommended, however we had an issue installing the module, and so we turned to `pgmpy` instead.
- `pgmpy` is a Python implementation for Baye's Network, for our project, the main groups we will be using are the `Bayesian Network`, `TabularCPD` and `VariableElimination` classes.
- `pgmpy` can be installed via "`pip install pgympy`" in the terminal.

Data Collection

To collect all the probabilities, we manually sat down and looked at over 300 screenshots (each of us studied 150) of boardstates old pro tour games of MTG. We began to mark down the factors and their corresponding values at each of these random screenshots. The videos were found from Wizards Of The Coast™ official magic competitive youtube account: Play_MTG. Each of us grabbed a random video from this channel, and selected 10-15 screenshots at different intervals of time. Most of the matches were either "best of 3" or "best of 5" - depending on the length of the match, we were able to grab more/fewer screenshots and note down our observations. We attempted to minimize bias as best we could by pulling at least one screenshot from the beginning, one from the middle and one from the end of each game, as to push the data in one direction. From this dataset, we are able to get the conditional probabilities that we need, putting the observed number of cases over the total games we observed, i.e. 300.

We realize that this is not the most reliable method for constructing a Bayesian Network, however, there was no readily made available data that gave us the measures we needed. We wanted to set up the framework for future research and data collection so that we are able to further finalize this project.

All conditional probability tables can be seen in this Google Sheet. It contains all the data we used to input into our Bayes Net, i.e. the cumulative observations made by each of us after analyzing 300 screenshots to collect data for this section.

Making The Network

The majority of the heavy lifting of this section was done by the `pgmpy` python library. Using the "`BayesianNetwork( )`" object we can initialize the framework for a Bayes Net. After that, each node can be initialized by the "`add_node( )`" function and edges with the "`add_edges_from()`" to add our master list of edges all at once. Once the shape of the bayes net is fully initialized, we must add conditional probability distributions to the data. As we had

the data handy, we manually inputted into the "`TabularCPD()`". The framework was created for automation later. Once the CPDs were inputted using "`add_cpds()`", the model was complete. Checking using the "`check_model()`" function allowed us to see if there were any errors.
The main way that data will be extracted from the bayesian network is through inference. `pgmpy` has a built-in function that allows us to do variable elimination to adjust the network to our current situation of evidence. This is done though the "`VariableElimination()`" class, with the required parameters being the required variable that we want to inspect, and the current evidence returning the augmented distribution that shows us the distribution that is desired.

Results

Queries

The following 11 queries are defined as: what is the chance of winning, given that their deck is of the following colours?

***Table 2.2**: Probabilities of Players Winning Based on their Colour Combinations*

Name	Colours	Chance Player Winning	Chance Opponent Winning
	WUBRG	0.4703	0.5297
Azorius	WU	0.5027	0.4973
Orzhov	WB	0.4616	0.5384
Boros	WR	0.5667	0.4333
Selesnya	WG	0.5121	0.4879
Dimir	UB	0.4671	0.5329
Izzet	UR	0.4830	0.5170
Simic	UG	0.4608	0.5392
Rakdos	BR	0.5276	0.4724
Golgari	BG	0.4783	0.5217
Gruul	RG	0.5123	0.4877

***Table 2.3**: Number of Nonland Permanents Based on Deck Archetype*

Archetype	0-2	3-5	6-8
Aggro	0.8172	0.1828	0.0000
Aggro + White in Colour Identity	0.7857	0.2143	0.0000
Control	0.4481	0.3977	0.1542
Midrange	0.3798	0.5180	0.1022
Combo	0.6645	0.2709	0.0646

Home Game

The main purpose of this approach is to provide a model so that players can have a metric of whether they are doing good or not. Hence, we decided to play a home game, and input screenshots of the current board state to see who had the better chance of winning at the current moment.

***Table 2.4**: Contains all overall information about the two decks*

	Overall Information	
Metric	Player	Opponent
Deck Archetype	Midrange	Aggro
Colours	White & Black	Red

***Table 2.5**: Showing the full game state of the first game to be testing on the network*

<table>
<tr><td colspan="4">Game 1: Player Wins!</td></tr>
<tr><td colspan="2"> 
</td><td colspan="2"> 
</td></tr>
<tr><td colspan="2">Player</td><td colspan="2">Opponent</td></tr>
<tr><td>Number Of Lands</td><td>6</td><td>Number Of Lands</td><td>3</td></tr>
<tr><td>Current Health</td><td>20</td><td>Current Health</td><td>5</td></tr>
<tr><td>Number Of Non-Land-Permanents</td><td>3</td><td colspan="2">N/A</td></tr>
<tr><td>Hand Size</td><td>1</td><td>Hand Size</td><td>5</td></tr>
<tr><td>Number Of Creatures</td><td>4</td><td>Number Of Creatures</td><td>3</td></tr>
<tr><td>Number Spells Played</td><td>0</td><td colspan="2">N/A</td></tr>
<tr><td>Probability of Player Winning</td><td>0.9231</td><td>Probability of Opponent Winning</td><td>0.0769</td></tr>
</table>

***Table 2.6**: Showing the full game state of the second game to be testing on the network*

Game 2: Opponent Wins!			
			
Player		Opponent	
Number Of Lands -	4	Number Of Lands	3
Current Health	3	Current Health	18
Number Of Non-Land-Permanents	2	N/A	
Hand Size	2	Hand Size	0
Number Of Creatures	0	Number Of Creatures	5
Number Spells Played	1	N/A	
Probability of Player Winning	**0.0000**	**Probability of Opponent Winning**	**1.0000**

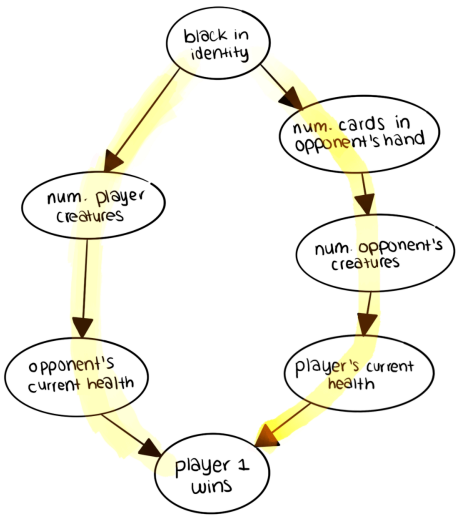
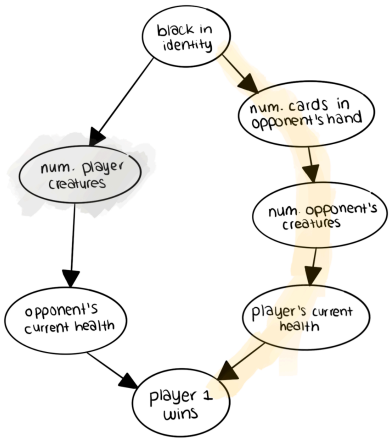
Table 2.7: *Showing the full game state of the first game to be testing on the network*

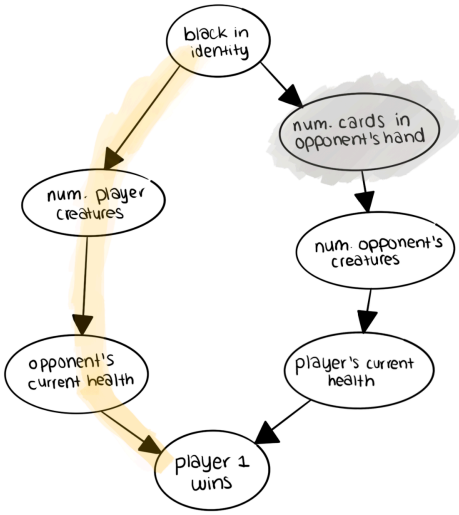
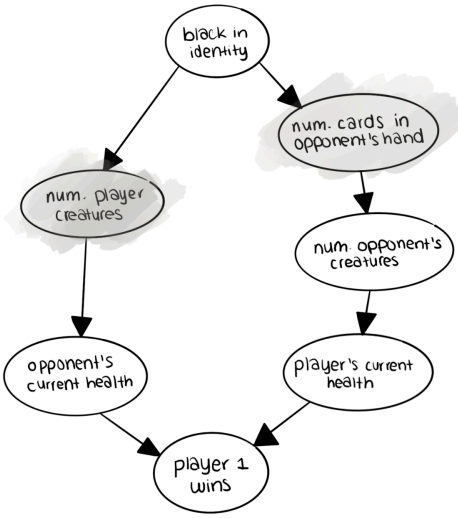
Game 3: Player Wins!			
			
Player		Opponent	
Number Of Lands -	5	Number Of Lands	4
Current Health	19	Current Health	14
Number Of Non-Land-Permanents	0	N/A	
Hand Size	1	Hand Size	1
Number Of Creatures	1	Number Of Creatures	2
Number Spells Played	2	N/A	
Probability of Player Winning	**0.2500**	**Probability of Opponent Winning**	**0.7500**

Separability

Even the fact that two nodes are independent given certain information can help to gather information about the domain, and how to play the game. The metric that we will use to consider if two nodes are independent is the d-separability metric, which will test if 2 nodes are d-separable. This will allow us to know if they are independent or not, and from there we are able to make further inferences.

Table 2.8: *All paths from black in identity to player 1 wins, gathered for d-separability metric.*

Given Evidence: None	Given Evidence: Number Of Players Creatures
	
Non-Separable	Non-Separable

Given Evidence: Cards In Opponents Hand	Given Evidence: Both Cards In Opponents Hand And Number Of Player's Creatures
	
Non-Separable	Separable

As shown here, the only time that “black in identity” and “player 1 wins” are independent is when ***both*** “number of cards in player’s hand” and “number of player’s creatures” is known to be in the current evidence. This shows the impact that the colour black has on the player winning. This shows us when or when not to take into account certain factors. For example: you only have to consider whether or not black is in the identity, only if cards in the opponent's hand are unknown, or number of creatures for the player is unknown. Otherwise, black being in a player’s identity is tied to their chance of winning, and it is not separable.

Observations

When observing a player’s chance of winning, we can find some interesting answers if we observe the colours that make up their deck’s identity. This information is printed out to the console, and is found using `pgmpy`’s variable elimination methods on our Bayes Net. We found that white and red (Boros) decks are the most likely to win, and after further introspection, we found that this fact may expose some play patterns of recent times. As the data was collected on recent years worth of games, the most dominant deck in the current metagame goes by the title “Boros Convoke”. As seen on this untapped.gg page that analyzes the current metagame, Boros Convoke has a staggering 56.4% win-rate. On the opposite side of the spectrum, the worst performing deck style was blue and green (Simic). This was not very surprising, as we found that there has not been a Simic deck that has made it to pro tours in the past few years. Thus, those players trying out the weaker strategies are going to carry a lower win-rate. Another interesting fact was that the 5 colour decks have a below 50% chance of winning. This can be surprising to some, and can help us understand how deck-building in magic works. It is not uncommon for some to think: “why not just put all the colours in my deck and play all the best cards” but this result is able to show empirical evidence that it does not work out as it is expected. The reason for this is likely due to the fact that it takes longer to setup the lands needed to play cards of all five colours - having five colours means that there is a lower chance of being able to play out the cards in one’s hand *unless* they have the correct colours of lands, which can get mismatched easily.

When looking at the relationship between deck archetypes and non-land non-creature permanents, it makes sense that the inclusion of white in one’s identity would increase the likelihood of having more non-land permanents. When inspecting popular deck lists, we found that most popular red decklists without white contain only ~8 cards that have the ability to create non-land permanents, contrasting the fact that Boros aggro decks tend to have ~13 cards that create non-land permanents.

It is very interesting that a simple Bayesian Network is able to catch the little intricacies of the bigger picture relating to MTG gameplay, colours, current trends, and their common deck lists.

Observations Regarding the Home Games

Games 1 and 2 are as expected. The screenshots were taken closer to the ends of the games. The large life total differences and stark difference in the number of creatures on each board all lead to the observed advantages. For game 1, the large health difference and the difference in lands is more than enough to cover for the fact that the opponent has more cards in hand, and thus the player is more likely to win this game. For game two, the large health difference and the amount of creatures on the opponent's board makes it certain that the player is to lose the game soon. The most interesting one is the third game, the game state looks very even when looking at the current factors. The screenshot for this game was taken about midway through. The player has slightly more health than the opponent, even cards in hand, and a similar amount of creatures. But the model seems to think that there is a massive swing in the likelihood of the opponent winning. The only explanation that we have for this is for the hyper specificity of the situation, the small amount of data that we have collected has only allowed for this case to be seen a few times.

As stated above, our model is most certainly skewed due to the lack of data. Only 300 data points are not enough to provide a strong metric. Thus improvements for the future should include automating the data collection process. Apart from the data collection, the model is a simple Bayesian Network, and thus has little else to be fixed. Possible improvements are the additions of edges that were not able to be added due to the increase in complexity that would come along with it. Important changes would include the following:

- Splitting the "player's non-land permanents" into 2 different nodes, i.e. "player artifacts" and "player enchantments".
- Additional edges to better describe the interactions between the board game pieces and the healths of the two players:
    - For example, one desired edge is between the "player's number of creatures" and "player's health", and the same for the opponent. This edge is quite important, as it is very true that there is a correlation between the two nodes. However, we could not include it for the reasons stated above.

Approach 3: Neural Networks, “Deep Analysis”

Applying the Problem Setting

The question we are attempting to solve with this approach is the following: can we create a neural network to output a value pertaining to how “salty” or mad a proposed card will make a player (if they were to play against said card)? Using the variety of information stored on a MTG card, we will attempt to use each of these as factors in training 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 EDHREC.

We will train this deep learning where our observations will be a subset of the MTG card, and the features of this model will be potential factors that contribute to the salt score of each card. This will include factors 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, based on the trained model. Each other subsection under this approach will detail the steps we took to implement our ANN.

Data Prepping

In order to start making the model, we must first gather the 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. The resources we need is simply a dataset containing a lot of MTG cards! In order to acquire and process such a big dataset, we will utilize the python library `requests` to get the data, others such as `flatdict` to sort and manipulate the data, and still others to create and train the model.

We first download a JSON file that contains all names of each card. They then had to be extracted into a separate file that would contain the information about each one. The .JSON was a list of dictionaries, where each entry in the list represented one singular card. This needed to be processed in order to be read into the model.

With 93,457 dictionaries in this list, and only approximately 30,000 cards in MTG, we deduced that there must have been duplicates in the data (which made sense due to the fact that cards are frequently reprinted). These absolutely had to be taken out, as replicated data in the training and the testing data would skew results heavily (keeping them in would basically be cheating)! We did this just by looping through the data, and were then left with 30,529 total card names that we were ready to proceed processing with, for our deep learning model.

Our dataset now contained each card and their respective features however, crucially, we were missing our actual data labels to classify all this data - i.e. the salt scores! These did not come with the .JSON that we downloaded, and hence we had to find the source that EDHREC turned to in order to obtain the salt score information for its cards. Eventually, we discovered that this could be achieved by pinging the same server that EDHREC pings to gather their salt scores… i.e. we discovered that the salt score for each card was available here: https://json.edhrec.com/cards/[advanced_name] where [advanced_name] is the name of any MTG card. This website was found through visual inspection of the EDHREC source code, as displayed in the following image:

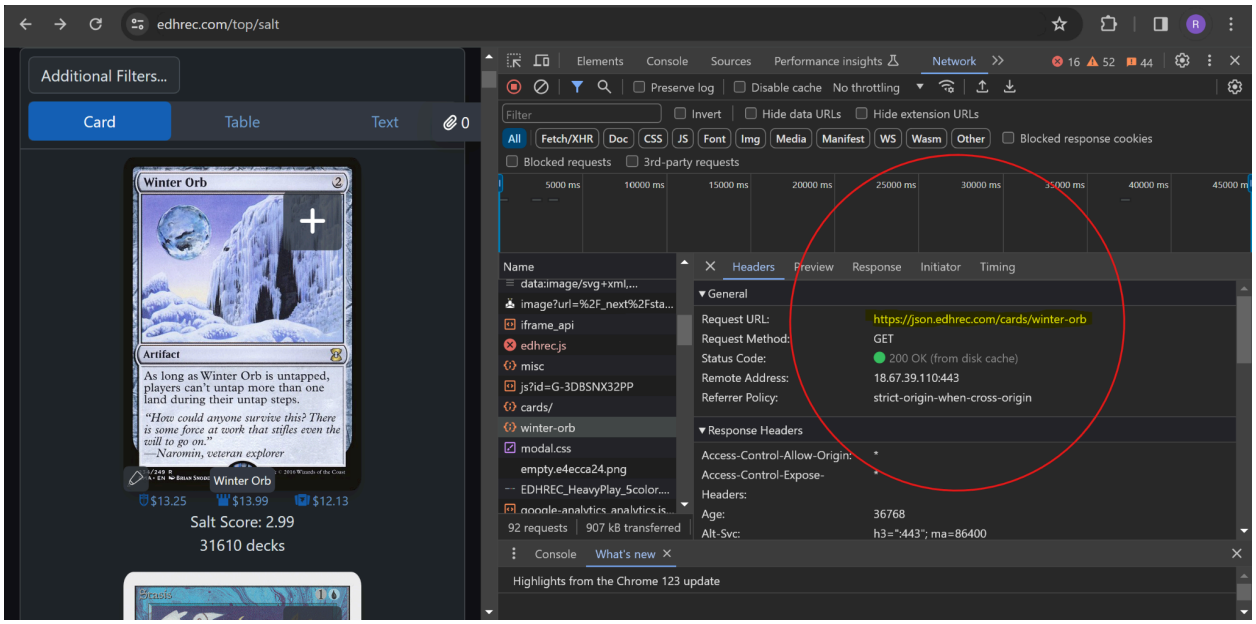


Figure 3.1: *The network tab of the EDHREC website showing the server in which it pings to gather its salt score.*

Hence, for each card, we pinged this server and gathered the salt score, storing them in our unduplicated dataset. This was the final, "sanitized" version of the data that was ready to be processed on. We extracted the salt score in a separate vector, and maintained the following features (note that these are the keys for each dictionary, where the dictionary represents a card):

```
['name', 'lang', 'released_at', 'layout', 'highres_image',
'image_status', 'mana_cost', 'cmc', 'type_line', 'oracle_text',
'power', 'toughness', 'colors', 'color_identity', 'keywords',
'legalities', 'games', 'reserved', 'finishes', 'oversized', 'promo',
'reprint', 'variation', 'set', 'set_name', 'set_type',
'collector_number', 'digital', 'rarity', 'flavor_text', 'artist',
'border_color', 'frame', 'full_art', 'textless', 'booster',
'story_spotlight', 'edhrec_rank', 'penny_rank', 'prices']
```

Some of these features are lists, e.g. the “prices” and “keywords”, of which the lengths are non-discrete. Having this inhomogeneous dataset makes it impossible for `keras` to work with, so we flatten all the data down to 1 single data structure (making it so there are no nested lists or dictionaries). This was done through a combination of `flatdicts` from the `flatdict` library, and iterative techniques to pick out and shift all data points to be in the correct shape. The full list of features can be found in the `data_keys.txt` file in the “Deep Learning” folder of the repository. After all the flattening and translating, we are left with a result in a (262,) sized vectors, where each vector contains representation of the features for each card. That is, there were 262 features for each card/observation. This influx of dimensions was due to our efforts to make the data homogenous, and mainly came from the fact that we had a boolean representing the presence of many keywords in MTG. This data is then extracted and stored in the `data.json` file to be accessed quickly later. We then have a classification vector that must be appended onto the data, corresponding to the name of the card.

We then split the data into training, validation and testing data sets at a 70-20-10 ratio. The data was split after first randomly shuffling the data in attempts to minimize introducing bias at this stage. We are now able to create the model, with our data ready. The total observations are split from a total of 30529 to 21370 instances of training vectors, 6106 instances of validation data and 3053 instances of testing data.

Making Of The Model

The making of the model itself was done with the aid of the `keras` Python library. We decided to create a simple initial model, then change loss functions and other hyper parameters (i.e. trying out different loss functions, optimizers and metrics) to fine tune, in order to attempt to determine which hyperparameters resulted in the most accurate model.

First Model

The first thing that must be done in the model is transform the data so that `keras` is able to understand it. This meant that we had to pass in *only* integer values; the strings and booleans present in our current vectors were not acceptable formats. We used a `keras` Text Vectorization layer in order to encode the strings into integers. Our first attempt was to implement a “bag of words approach” but we had an issue of the data being asynchronous. We had a (1, 262, X) Vector where X was the size of the text box of the creature. This created a vector space that was not consistent. Keras is unable to handle that type of data, so other avenues had to be researched. The way we remedied this was to apply it to the vector as a whole: i.e. to not split the words in a text box on the white spaces at all. This means that the words in a creature’s text box would be conglomerated into one. We thought of the following approach in order to get around this issue, but could not find a suitable way to implement it: splitting the data into 2 separate inputs, where the first is all of the single string characters, like date, keywords, and the inclusion of certain colours. The second is for the features that have inner-iterables (anything contained in a

dictionary, list, set, etc.) could be implemented using an RNN for the multi word inputs. We will expand on this idea a little further later on in this section.

Once the data was all set to be input into the model, we had to make the model itself. The first model we created was a baseline model to test what the other loss functions and optimizers would change, i.e. to give us a sense of what was "good" within the model.

Starting Model:

- Input Layer (Of Shape: (1,262))
- Fully Connected Neural Layer, (64 Neurons)
- Fully Connected Neural Layer, (64 Neurons)
- Output Layer (1 Value)

The Base Model

Loss Function - Mean Absolute Error

This value was initially chosen due to the output being a single numeric value, as our regression problem can use simple error calculations. This computes the mean absolute error between the labels, and the predicted values. Then it will back propagate to change the weights.

Optimizer - Adgrad

A gradient descent function that dictates the rate at which the weights of the neural network are impacted by each of the inputs. Adgrad, however, does not apply the same learning rate for each of the features, and each of the features get scaled at differing rates, with respect to the accumulated squared gradient at each iteration.

Metric - Mean Squared Error

Similar to the loss functions, we are able to use the simple regression objects, due to us having a singular number output. And thus we chose the MeanSquaredError as our metric for its simplicity and ease of understanding for us.

Results For The Base Model

Training Loss: 0.2280
Validation Loss: 0.1298
Testing Loss: 0.187891513
Testing Metric: 0.09216810
"Craterhoof" Scoring of: [-0.09712346]

Table 3.1: *Observations from testing our other metrics, loss functions, optimizers and layers*

Variant:	Training Loss	Validation Loss	Testing Loss	Testing Metric	Craterhoof Metric
None	0.2280	0.1298	0.187891513	0.09216810	-0.09712346
MeanSquaredError	0.1109	0.0859	0.08105927	0.081059	0.21022387
MeanSquaredLogarithmicError	0.0605	0.0624	0.0615904	15.43719	-0.33406422
Adam Optimizer	0.02121	0.1735	0.173770	0.087302	0.1040559
RMSprop Optimizer	0.2027	0.1738	0.173266	0.092618	0.1054273
Dropout Layer 1	0.2958	0.1758	0.17501	0.090728	N/A
Dropout Layer 2	0.56666	0.1819	0.181510999	0.0926214	N/A
Learning Rate 0.005	0.1774	0.1685	0.166844	0.08641833	N/A
Learning Rate 0.0005	0.1748	0.1679	0.16586790	0.084742	N/A

For each of the categories, we chose which metric gave the best overall result for our final product, factoring in all the columns to make our decision. Concerning the dropout layers: the first variant is located *between* the first and second neuron layer, and the second is located *after* the second layer. Concerning the learning rates, both of the tested options seem to barely affect the learning rate of the data at all, and thus we will keep it at the baseline learning rate of 0.1%. Thus with all the viable components for compiling tested, we will choose the final version of our model:

These options are:

- Loss = MeanSquared Error,
- Optimizer = Adam @ learning rate = 0.001
- 1 Dropout Layer @ 20% between the first and second layer

Giving final results of:

Training loss: 0.1190
Validation Loss:0.0828
Testing Loss: 0.08196896
Testing Metric: 0.0819689
“Craterhoof Behemoth” Scoring of: [0.20504367] compared to the actual label of [1.806943713834824].

Fitting
The training of the model is done through the built-in function for `training.fit()`. After inputting our previously computed `x_data`, `y_vec`, and `validation_data`. The hyperparameters we chose for this was a batch size of one.

Evaluating
The built-in `evaluate` function iterates over the testing data that we have created, then gives back the loss function and metric for the entirety of the testing data. For this section, the only thing we had to input was the testing data.

As an additional testing method we used the card “Craterhoof Behemoth” as a metric.

Figure 3.1*: An image of Craterhoof Behemoth.*

Craterhoof’s actual salt score is 1.806943713834824, thus ranking it as one of the top saltiest MTG cards. This should be a tough task for the model, as it is one of the “edge” cases that neural networks are not great at computing. That is, most cards in our database had a salt score of 0, due

simply to the vast amount of cards made for the game, as well as the fact that non-prominent cards are not likely to get assigned a salt score. The output of the model for this card is a good metric on how the model deals with cases like this.

Observations

The finalized model is decently accurate, with a testing loss of approximately 0.08, more than accurate enough for the simplistic model we sought out to create. We feel that due to the nature of the data, (with a lot of zeros and very few >1 scores for cards, the data is skewed. Salt score polling was also introduced around 2019, another factor that we realized impacted the data, and introduced a lot of bias, as most cards before this year would fly under the radar, rotating out of the popular decklists, and forgotten with unassigned salt scores. We feel that the model is falling into a type of mode collapse, in which it is not actually caring about any of the metrics, and only giving something close to zero due to the sheer amount of zeros that it studied - thus minimizing the loss function effectively, but giving very little information. The second largest observation is how the data is getting vectorized. We feel that we are losing a large amount of data when transferring the strings into integers. As mentioned above, we are not splitting on white-space in order to maintain the shape of the data, and input it into `keras` in a format that the library's methods accept.

After taking a look at the textVectorization depiction of the card, we see the following:

[[1 17 1 14 5 16 1 1 1 1 1 1 3 3 4 4 4 3 3 4 4 3 4 3
4 4 3 4 3 3 4 3 3 3 2 2 2 5 2 1 1 1 1 2 1 1 1 19
1 2 2 2 2 1 0 6 6 1 6 6 1 2 2 2 2 5 2 2 2 2 5 2
2 2 2 2 2 2 2 2 5 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 5 5 2 2 2]]

The entire text of Craterhoof Behemoth…

"*'Haste\nWhen Craterhoof Behemoth enters the battlefield, creatures you control gain trample and get +X/+X until end of turn, where X is the number of creatures you control.*'"

…got condensed into a single number: 1. This resulted in a MASSIVE loss of data. However the other option of splitting on spaces instead of none-splitting would have resulted in an inhomogeneous data set that could not be input into `keras` to form a neural network.

Future Advancements

In order to make the model even more accurate, there are many changes that can be made. An advancement that is either the aforementioned "bag of words" approach, or swapping over from `keras`'s provided method of textVectorization to a custom text processing data system in order to turn the long text box into homogenous data. Secondly, and more impactfully, replacing the current model with an RNN that is able to handle the inhomogeneous inputs would allow for the entire text object to be given at the same time, but individually handled to get the full scope of the information. Unfortunately, we had already begun implementation of our sequential ANN by the time we had learned about RNNs in CISC 352, and decided that it would not be efficient to redo our work at that time. A many-to-one RNN model would suit the problem perfectly, as our problem has a variety of different-length inputs (all being strings) and requires a single output (a number, representing the salt score of course)! This would complement the nature of the problem, as each input to the model would either increase or decrease the final output salt score. It also would allow us to inspect the input for each of the words to see their individual impact on the final salt score, enabling us to sort the data, and inspect even deeper into what makes a card salty or not.

Bibliography

[1] “Numeric Expressions, Conditions and Effects,” *www.cs.cmu.edu*. Available: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/fox03a-html/node3.html. [Accessed: Apr. 10, 2024]

[2] A. Green, J. Dolejsi, and M. Magnaguagno, “PDDL 2.1 Domain,” *Planning.wiki - The AI Planning & PDDL Wiki*. Available: https://planning.wiki/ref/pddl21/domain

[3] A. Green and J. Dolejsi, “PDDL Requirements,” *Planning.wiki - The AI Planning & PDDL Wiki*. Available: https://planning.wiki/ref/pddl/requirements

[4] J. Hrnčíř and Tutorial, “PDDL (Planning Domain Definition Language).” Available: https://cw.fel.cvut.cz/old/_media/courses/a4m33pah/pddl.pdf. [Accessed: Apr. 10, 2024]

[5] A. Mordoch, R. Stern, E. Scala, and B. Juba, “Safe Learning of PDDL Domains with Conditional Effects.” Available: https://icaps23.icaps-conference.org/papers/rddps/Mordoch-et-al-RDDPS23.pdf. [Accessed: Apr. 10, 2024]

[6] A. Ankan, “Bayesian Network — pgmpy 0.1.15 documentation,” *pgmpy.org*. Available: https://pgmpy.org/models/bayesiannetwork.html

[7] A. Ankan, “Discrete — pgmpy 0.1.23 documentation,” *pgmpy.org*. Available: https://pgmpy.org/factors/discrete.html. [Accessed: Apr. 10, 2024]

[8] K. Team, “Keras documentation: Model training APIs,” *keras.io*. Available: https://keras.io/api/models/model_training_apis/

[9] “optimization - Guidelines for selecting an optimizer for training neural networks,” *Data Science Stack Exchange*. Available: https://datascience.stackexchange.com/questions/10523/guidelines-for-selecting-an-optimizer-for-training-neural-networks

[10] “Keras documentation: Adam,” *keras.io*. Available: https://keras.io/api/optimizers/adam/

[11] K. Team, “Keras documentation: Regression metrics,” *keras.io*. Available: https://keras.io/api/metrics/regression_metrics/#meansquarederror-class. [Accessed: Apr. 10, 2024]

[12] “tf.keras.layers.TextVectorization | TensorFlow Core v2.7.0,” *TensorFlow*. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization

[13] “Basic text classification | TensorFlow Core,” *TensorFlow*. Available: https://www.tensorflow.org/tutorials/keras/text_classification

[14] R. Gylberth, “An Introduction to AdaGrad,” *Konvergen.AI*, May 03, 2018. Available: https://medium.com/konvergen/an-introduction-to-adagrad-f130ae871827