Progetto Ingegneria della Conoscenza - MTG

Autore - Damato Luigi Lele
MAT #743476
l.damato15@studenti.uniba.it
axten.dev@github.com

Sommario

1. Introduzione

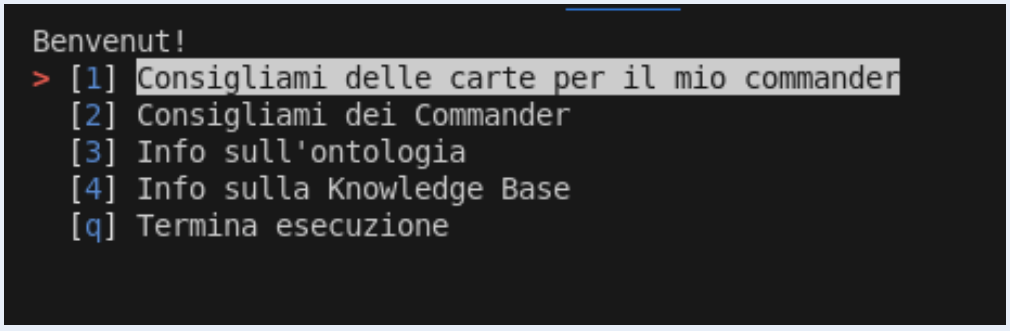
Da qualche anno molti più appassionati del gioco di carte di Magic: the Gathering si sono approcciati al formato di gioco 'Commander'. L'obiettivo principale di questo progetto è semplificare il processo di creazione di mazzi (Deckmaking) per gli utenti meno esperti. Per questo motivo sono stati realizzati:

- Un Recommender System capace di consigliare nuovi commander sulla base delle preferenze dell'utente e di utenti simili
- Un' Ontologia che descrive il dominio d'interesse specificando il significato dei simboli nel sistema
- Una Knowledge Base che permette all'utente di interrogare il dominio di interesse tramite interazione 'domanda-risposta' per determinare cosa sia vero, permettendo all'utente di acquisire informazioni sulle carte e sul risultato degli scontri.

1.1 Interfaccia

Una volta avviato il programma, l'utente interagisce da terminale attraverso la libreria Simple Term Menu (Fig 1.1) per l'interezza della sua esecuzione. Può interagire usando le frecce direzionali o direttamente dal tasto a fianco.

Figura 1.1



1.2 Dataset Utilizzati

Il programma utilizza il dataset di Scryfall per regole, diciture e contenuto delle carte (più di 30000 oggetti), aggiornandosi automaticamente quando ne vengono aggiunte di nuove, tramite espansione o rilasci speciali.
Il sistema si avvale inoltre di query mirate verso Archidekt, una piattaforma di gestione di mazzi di MTG, per ottenere le liste degli utenti e dei loro mazzi. È stato necessario

applicare operazioni di cleaning sui dati ottenuti da questo portale, oltre ad una formalizzazione degli oggetti ottenuti, per rimuovere elementi compromessi, ordinare e controllarne la qualità.

1.2 Attributi della classe Card, estraibile dal dataset Scryfall

Per una lista completa delle oltre 90 colonne del dataset di Scryfall: https://scryfall.com/docs/api/cards, di seguito l'elenco di tutte quelle utili al modello

- **oracle_id** - Identità della carta, rimane la stessa dopo le ristampe. Le carte sono aggiornate all'ultima ristampa
- **mana_value** - Solitamente il 'costo in mana' in alto a destra della carta, distribuito tra i vari colori.
- **mana_cost** - Somma dei valori assegnati ad ogni colore di mana_value.
- **color_identity** - Identità di colore di una carta, se è presente un simbolo di mana colorato in qualsiasi parte della carta, fa parte della sua identità di colore
- **colors** - Colori di una carta, se un simbolo di mana colorato è presente nel mana_value, questo è parte dei suoi colori.
- **rarity** - Rarità di una carta nei termini di comune, non comune, rara e mitica rara
- **name** - nome della carta
- **power** - Potenza della carta, se è una Creatura o un Veicolo
- **toughness** - Resistenza della carta, se è una Creatura o un Veicolo
- **text** - lista delle abilità della carta
- **types** - Tipi della carta
- **super_types** - Supertipi della carta (Leggendaria, Base, altro)
- **sub_types** - Sottotipi della carta
- **keywords** - presenti in text, indicano abilità comuni abbreviate in una sola parola

Quelle invece ottenute da text tramite algoritmo sono:

- **mana_production** - un dizionario con la produzione di mana della carta
- **default_category** - Il modo più comune per cui questa carta viene utilizzata

```python
class Card():
    oracle_id : int
    mana_value : dict[str:int]
    mana_cost : int
    color_identity : dict[str:bool]
    colors : dict[str:bool]
    rarity : Rarity
    name : str
    power : None|int
    toughness : None|int
    text : None|str
    types : list[Type]
    super_types : list[SuperType]
    sub_types : list[str]
    keywords : list[str]
    mana_production : dict[str:int|bool|str]
    default_category : list[str]
```

1.3 Librerie e Linguaggi

Il sistema è interamente scritto in linguaggio **Python**, tramite editor VS Code, sfurttando una combinazione di librerie per trattamento e analisi dei dati:

- **Pandas** per gestione del dataset e l'elaborazione dei dati strutturati
- **Numpy** per le operazioni matematiche e manipolazioni di matrici
- **Pickle** per la serializzazione dei risultati
- **Simple Term Menu** per delle interfacce utente intuitive
- **OWL** e **PySWIP** per interagire con la Knowledge Base, per aggiungerne fatti, regole e fornendo capacità di interrogazione e inferenza semantica sui dati

1.4 Preprocessing dei dati

Il dataset di Scryfall con le carte ordinate per Oracle_id contiene più di 90 colonne, per ottenere un formato utilizzabile per il machine learning è stato necessario trasformare quelle utili in dati utilizzabili, e scartare quelle inutilizzate (molti utenti compongono mazzi basandosi anche solo sugli artisti, ma non saranno utili al nostro caso questi esempi).

In particolare dobbiamo tradurre *mana_value*, *color_identity* e *colors* da stringhe a dizionari:

```python
def mana_value(mv:str) ->dict:
    mana_value = {color:mv.count(color)  for color in "WURBGC"}
    if len(mv)>0 and mv[1].isdigit():
        mana_value['C']+= int(mv[1])
    return mana_value


def colors(ci:list) -> dict:
    translator = {'W':'White','U':'Blue','B':'Black','R':'Red','G':'Green'}
    return {k: v in ci for k,v in translator.items()}
```

dividere la typeline in supertipo, tipo e sottotipi:

```python
def split_typeline(typeline:str) -> tuple:
    supertypes = [type.name for type in SuperType if type.name in typeline]
    types = [type.name for type in Type if type.name in typeline]
    try:
        subtypes = typeline.split('—')[1].split(" ")[1:]
    except:
        subtypes = None
    return (supertypes, types, subtypes)
```

Rendere keywords una lista effettiva, e rimuovere le keywords dal testo

```python
def keywords(kw:str) -> list:
    return kw[1:-1].replace("'", '').replace(" ", '').split(',')

def text(keywords:list, t:str) -> str:
    for kw in keywords:
        t = t[len(kw)+1:]
    return t
```

E dobbiamo estrarre la feature mana_production, questa implementazione è parzialmente esatta, omettendo casi particolari (carte con produzioni diverse di mana), ma per un uso generico è quella più efficace

```python
def get_mana_production(self)->dict:
    match = re.search(r'\.?\s*(.*?)\s*Add\s+(.*)\.', self.text)
    if not match:
        return None

    #suppongo ci sia un costo di attivazione
    cost, mana_produced = self.text.split(':').strip()
    mana_dict = {"W": 0, "U": 0, "R": 0, "B": 0, "G": 0, "C":0, "Or": True, "Note": "", "cost":cost}

    # trova i simboli tra {}
    mana_symbols = re.findall(r'\{([WUBRGC])\}', mana_produced)

    if mana_symbols:
        # Add {*} for each [...]
        if "for each" in mana_produced:
            quality_match = re.search(r'for each (\w+)', mana_produced)
            if quality_match:
                x = quality_match.group(1)
                for symbol in mana_symbols:
                    mana_dict[symbol] = "X"
                mana_dict["Note"] = f"for each {x}"
        else:
            for symbol in mana_symbols:
                mana_dict[symbol] += 1
                mana_dict["Or"] = False

    # Add X mana of any color
    QUANTITY = {"one":1, "two":2, "three":3, "four":4, "five":5, "X":"X"}
    quantity_match = re.search(r'Add (\w+) mana of any color', mana_produced)
    if quantity_match:
        x = quality_match.group(1)
        for symbol in "WURBG":
            mana_dict[symbol] = QUANTITY[x]
    return  mana_dict
```

Infine ogni carta è stata vettorializzata, utilizzando *OneHotEncoder* come Encoder per sub_types e keywords, e associando una classe Enum ai valori di Tipi e Supertipi

2. Recommender System

Un sistema (o motore) di raccomandazione è un software di filtraggio dei contenuti che crea delle raccomandazioni personalizzate specifiche per l'utente che lo utilizza, così da aiutarlo nelle sue scelte. È stato scelto un **approccio collaborativo** nella scelta dei commander, e un **approccio ibrido** nella scelta delle carte.

2.1 Commander Recommender

Questo recommender sfrutta un approccio di collaborative filtering.
Insieme ai mazzi $m \in M_A$ dell'utente A da Archidekt, otteniamo il vettore dei colori v di ciascuno di esso.

```
Benvenut!
Inserisci il tuo nome utente su Archidekt: Axten
Ho trovato 7 mazzi per Axten, 2 Commander:
Salvati 2 mazzi in owner/Axten da Archidekt- 1.67kB
Salvate 78 carte in grimoire/Axten da Archidekt- 2.91kB

Commander più utilizzati da Axten:
- Rionya, Fire Dancer
- Reyhan, Last of the Abzan
- Silas Renn, Seeker Adept

Salvati 189 mazzi per Rionya, Fire Dancer
Salvati 120 mazzi per Reyhan, Last of the Abzan
Salvati 104 mazzi per Silas Renn, Seeker Adept
```

Sapendo che i colori all'interno vengono ordinati come 'WURBG' (White, blUe, Red, Black, Green) possiamo avere una stima generale del mana_value medio, i colori più presenti, altre informazioni base come presenza di permanenti senza colori.

Analiziamo le carte all'interno di $m \in M_A$ per ottenere i commander di A, effettuiamo quindi una ricerca dei mazzi che li contengono su Archidekt e otteniamo una lista di utenti con gusti simili ad A, solitamente sull'ordine di 10^5.

Poiché, diversamente dal vettore dei colori, ottenere un commander da un mazzo non è una procedura immediata (deve essere effettuata una query per mazzo, e cercare tra le 100 carte quella nella categoria 'commander', diversamente dal vettore dei colori, che viene restituito direttamente nella ricerca dei mazzi), per ottenere più efficaciemente risultati comunque accurati, applichiamo un filtro parziale per similarità con v, usandolo come feature principale.

Ricaviamo il centroide $\overrightarrow{r_A}$ dei colori nei mazzi in M_A.

```
Vettore media dei colori per Axten:
[W:5, U:12, R:25, B:13, G:0]
```

Ricaviamo quindi per ogni utente U, il centroide $\overrightarrow{r_U}$ dei colori nei mazzi di U.
Possiamo poi ordinare la lista dei $k = 30$ utenti più vicini in base alla **distanza euclidea** tra i centroidi, che si è dimostrato più efficiente in tempi di esecuzione rispetto alla *similarità del coseno* e al *coefficiente di correlazione di Pearson*.

Da questi utenti ricaviamo l'analisi completa dei loro mazzi e delle carte al loro interno.

```
3049 Utenti con commander simili, ordino per similarità e ne esamino i primi 30
Salvate 93 carte in Eggs Discount.pkl da Archidekt - 31.18 kB
Salvate 93 carte in Zer1.pkl da Archidekt - 31.59 kB
Salvate 84 carte in Red Thief.pkl da Archidekt - 31.72 kB
Non riesco ad aggiungere altre carte, procedo oltre
Salvate 84 carte in Imoteck trial.pkl da Archidekt - 28.33 kB
Salvate 102 carte in Bully Maguire.pkl da Archidekt - 34.97 kB
Salvate 93 carte in Piu Pia.pkl da Archidekt - 34.53 kB
Salvate 90 carte in Nachoni.pkl da Archidekt - 30.97 kB
```

```python
def analyze(self, pooling:Pooling):
    subtypes = self.all_subtypes()  # ottieni tutti i sottotipi delle carte
    deck_aggregates = {}
    deck_grimoires = split(self) # ottieni un dict[deck_id,grimoire]

    for deck_id, grimoire in deck_grimoires.items():
        card_vectors = [card.vectorize(subtypes) for card in grimoire]
        deck_vector = pooling(card_vectors)
        deck_aggregates[deck_id] = deck_vector

    return (deck_aggregates.keys(), np.unique(deck_aggregates.values()))
```

Come già detto in 1.4, ogni carta è stata già vettorializzata, ma è in questo punto che, avendo ottenuto il sottodominio dei sottotipi, supertipi e keywords che verranno utilizzati che possiamo applicare l'**Analisi delle Componenti Principali (PCA)** per ridurre la dimensionalità dei vettori-deck. Questa tecnica consente di condensare le caratteristiche principali dei mazzi (come la distribuzione dei colori, il tipo di carta, le meccaniche chiave, ecc.) in un insieme di componenti che spiegano la maggior parte della varianza. Riducendo la dimensionalità, è possibile gestire dati complessi mantenendo le informazioni più rilevanti.

Dopo aver applicato la PCA, vengono esclusi i mazzi già presenti in M_A, la similarità tra i mazzi viene calcolata utilizzando nuovamente la **distanza euclidea** tra i vettori trasformati, infine vengono mostrati i primi (5) con distanza minore

```
I 5 commander più vicini a quelli di Axten sono:
commander                 distanza
Zaffai, Thunder Conductor   0.0971
Melek, Izzet Paragon        0.0988
Kresh, the Bloodbraided     0.1327
Marchesa, the Black Rose    0.1852
Meren of Clan Nel Toth      0.1914
```

2.2 Cards Recommender

Ho avuto solo modo di ideare questo sistema, non di implementarlo.
Idealmente questo Recommender System, dopo aver ottenuto una (o più) carta da usare come commander, suggerirebbe almeno 20 carte sinergiche ad essa, sulla base delle regole e altri parametri, come la curva del mana, strategia da implementare, budget massimo, eccetera.
Per questo è evidente la necessità di un *reasoner*, un motore di inferenza, che elabori *automaticamente* le regole salvate nella Knowledge Base e ne ottenga deduzioni, che ci permettano di clusterizzare l'intero dataset di carte, raggruppando oggetti in base allo stile di gioco per i quali possono essere utili.

Questo reasoner attuerebbe regole di inferenza basate su proposizioni logiche **if-then** proprie del **Semantic Web Rule Language** (SWRL):
`hasParent(?x1,?x2) ∧ hasBrother(?x2,?x3) ⇒ hasUncle(?x1,?x3)`[1], o più semplicemente *condizioni → conseguenza*.

È possibile aggiungere regole **SWRL** alla Knowledge Base tramite la libreria **OWLready2**. Inoltre un sistema del genere permetterebbe un'alta personalizzabilità ai giocatori, che potrebbero aggiungere, modificare o rimuovere regole dal sistema; feature **unica** per un recommender del genere.

Esempi in termini di gioco

Ad esempio, Teysa Karlov è un commander molto popolare, per le molteplici strategie per cui è utile:
- creazione di token
- guadagno di vita
- sacrificare creature

il motore di inferenza permetterebbe di capire quali carte sfrutterebbero i 'trigger', quali creature conviene di più sacrificare, quali carte creano token più forti e in maggior numero o comunque sinergizzino bene con altre creature di sottotipo Umano o Consigliere.

Però Magic è un gioco molto (a volte anche troppo) complesso, con un compendio di regole lungo 295 pagine, e costantemente aggiornato all'uscita di nuove espansioni.

3. Ontologia

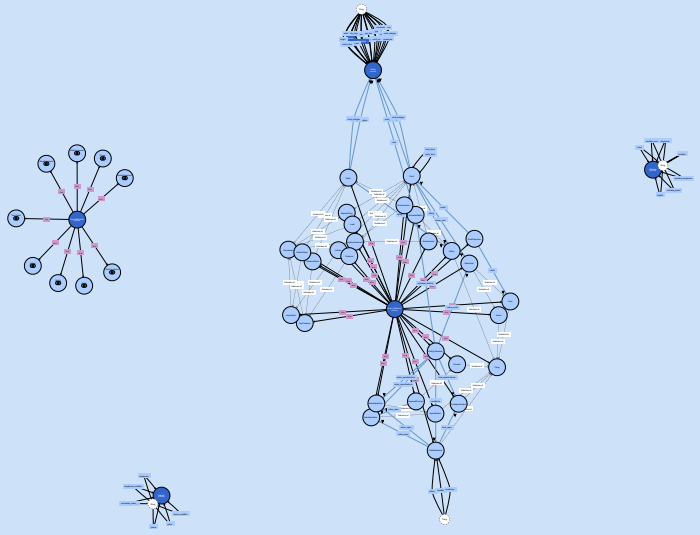
L'ontologia, in informatica, è un modello di rappresentazione formale della realtà e della conoscenza. È una struttura di dati che consente di descrivere le entità (oggetti, concetti, ecc.) e le loro relazioni in un determinato dominio di conoscenza.

Per questo progetto, è stato deciso di espandere un'ontologia già presente, creata dall'utente Cyril Matthey-Doret (cmdoret) e disponibile pubblicamente su Github, alle quali mancano entità significative:

- Tipi di carta come Planeswalker, Battaglie, Dungeon;
- Tipi di entità come Emblemi, Designazioni, Azioni di gioco;
- Molteplici Keywords (sono presenti 17 sulle 181 totali), e le loro descrizioni.

Visualizzazione grafica dell'ontologia creata da cmdoret

È consigliato https://service.tib.eu/webvowl/ per visualizzare il file `data/Ontology/mtg_ontology.owl.tll`, di seguito un SVG



3.1 Aggiunte

Riporto i prefissi dello schema, presenti nel file .ttl

```
@prefix IAO: <http://purl.obolibrary.org/obo/IAO_> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix linkml: <https://w3id.org/linkml/> .
@prefix mtg: <https://mtg.fandom.com/wiki/> .
@prefix mtgo: <https://cmdoret.net/mtg_ontology/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
```

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <http://schema.org/> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

3.1.1 Planeswalker

Mantenendo una sintassi omogenea alle altre dichiarazioni, definisco le classi **Planeswalker AnyPlaneswalker** e **PlaneswalkerToken**

```
mtgo:Planeswalker a owl:Class;
    rdfs:label "Planeswalker" ;
    rdfs:seeAlso "mtg:Planeswalker" ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onClass linkml:Integer ;
            owl:onProperty mtgo:loyalty ;
            owl:qualifiedCardinality 1 ],
        mtgo:AnyPlaneswalker,
        mtgo:Card ;
    skos:definition "A card that represents a planeswalker that can be
cast and put onto the battlefield." ;
    skos:exactMatch mtgo:Planeswalker .

mtgo:PlaneswalkerToken a owl:Class;
    rdfs:label "PlaneswalkerToken" ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onClass linkml:Integer ;
            owl:onProperty mtgo:loyalty ;
            owl:qualifiedCardinality 1 ],
        mtgo:AnyPlaneswalker,
        mtgo:Token ;
    skos:definition "A token that represents a planeswalker that can be
placed onto the battlefield by other effects." ;
    skos:exactMatch mtgo:PlaneswalkerToken .

mtgo:AnyPlaneswalker a owl:Class;
    rdfs:seeAlso "mtg:Planeswalker" ;
    rdfs:label "Planeswalker" ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onClass linkml:Integer ;
            owl:onProperty mtgo:loyalty ;
            owl:qualifiedCardinality 1 ],
        mtgo:AnyPlaneswalker,
        mtgo:Permanent ;
    skos:definition "A planeswalker that can be summoned to the
```

```
battlefield." ;
    skos:exactMatch mtgo:AnyPlaneswalker .
```

Perché è necessario definire tre elementi differenti? Sono tre identità della stessa carta, ma in fasi diverse della partita:
Una carta *Planeswalker* è tale quando non è sul campo di combattimento, se invece viene evocata può diventare un permanente *AnyPlaneswalker*, ma appena è fuori dal campo torna ad essere una carta. Ci sono alcuni effetti che permettono di creare delle copie dello stesso permanente, creando dei token *PlaneswalkerToken*, che sono dei permanenti a tutti gli effetti, ma appena escono dal campo smettono di esistere, non potendo tornare ad essere carta.
Definire le sottoclassi *Any* e *Token* è necessario solo in caso di Carte che quando lanciate diventano *permanenti*. Tutto questo è già ben definito nella classe Card di cmdoret

Una caratteristica intrinseca dei planeswalker è la lealtà, per cui prendendo spunto dalla forza di una creatura (sua caratteristica intrinseca), dalla definizione di segnalino (counter) e parafrasando il regolamento definisco:

```
mtgo:loyalty a owl:ObjectProperty;
    rdfs:label "loyalty" ;
    rdfs:range linkml:Integer ;
    skos:definition "The amount of loyalty counters on a planeswalker when
it enters the battlefield. This is the number in the lower right corner of
a planeswalker card." ;
    skos:exactMatch mtgo:loyalty .

mtgo:variable_loyalty a owl:ObjectProperty;
    rdfs:label "variable_loyalty" ;
    rdfs:range linkml:Boolean ;
    rdfs:subPropertyOf mtgo:variable ;
    skos:definition "Whether a planeswalker's loyalty is variable. Denoted
as X on cards." ;
    skos:exactMatch mtgo:variable_loyalty .

mtgo:loyalty_counter a mtgo:Counter;
    rdfs:label "loyalty_counter" ;
    rdfs:subClassOf [a owl:Restriction;
        owl:maxQualifiedCardinality 1;
        owl:onClass linkml:Integer;
        owl:onProperty mtgo:loyalty ],
    skos:definition "A counter that tracks the loyalty of a
planeswalker.";
    skos:exactMatch mtgo:loyalty_counter

mtgo:loyalty_cost a owl:ObjectProperty;
    rdfs:label "loyalty_cost" ;
```

```
    rdfs:range linkml:Integer ;
    skos:definition "The number of loyalty counters added or removed upon
activating a loyalty ability" ;
    skos:exactMatch mtgo:loyalty_cost .

mtgo:loyalty_ability a owl:Class;
    rdfs:label "LoyaltyAbility" ;
    rdfs:seeAlso "mtg:Loyalty_ability" ;
    rdfs:subClassOf mtgo:ActivatedAbility;
    skos:definition "A planeswalker ability which can be activated once a
turn, during that turn's player main phase" ;
    skos:exactMatch mtgo:loyalty_ability.
```

Planeswalkers nel gioco

Un planeswalker è una potente entità all'interno della storia e lore di Magic the Gathering, una creatura che, trascendendo la sua natura, è capace di viaggare attraverso i piani di esistenza.
Diversamente dalle creature, i planeswalker possono essere bersaglio di attacco, e hanno delle abilità basate sulla "lealtà", lasciando intendere che quando non ci sono più segnalini lealtà su un planeswalker, non è morto, ma semplicemente non intende più prestare aiuto al giocatore (che all'interno del gioco, è un planeswalker a sua volta).

3.1.2 Battaglia

Una Battaglia è un permanente che entra con dei segnalini difesa, può essere bersagliata tramite attacchi, e quando scende a zero segnalini difesa va nel cimitero. Essenzialmente una Battaglia è un Planeswalker senza abilità basate sulla lealtà.

```
mtgo:Battle a owl:Class;
    rdfs:label "Battle" ;
    rdfs:seeAlso "mtg:Battle" ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onClass linkml:Integer ;
            owl:onProperty mtgo:defense ;
            owl:qualifiedCardinality 1 ],
        mtgo:AnyBattle,
        mtgo:Card ;
    skos:definition "A card that represents a battle that can be cast and
put onto the battlefield." ;
    skos:exactMatch mtgo:Battle .

mtgo:BattleToken a owl:Class;
    rdfs:label "BattleToken" ;
```

```
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onClass linkml:Integer ;
            owl:onProperty mtgo:defense ;
            owl:qualifiedCardinality 1 ],
        mtgo:AnyBattle,
        mtgo:Token ;
    skos:definition "A token that represents a battle that can be placed
onto the battlefield by other effects." ;
    skos:exactMatch mtgo:BattleToken .

mtgo:AnyBattle a owl:Class;
    rdfs:seeAlso "mtg:Battle" ;
    rdfs:label "Battle" ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onClass linkml:Integer ;
            owl:onProperty mtgo:defense ;
            owl:qualifiedCardinality 1 ],
        mtgo:AnyBattle,
        mtgo:Permanent ;
    skos:definition "A Battle that can be summoned to the battlefield." ;
    skos:exactMatch mtgo:AnyBattle .
```

La caratteristica intrinseca delle battaglie è la difesa.

```
mtgo:defense a owl:ObjectProperty;
    rdfs:label "loyalty" ;
    rdfs:range linkml:defense ;
    skos:definition "The amount of defense counters on a battle when it
enters the battlefield. This is the number in the lower right corner of a
Battle card.";
    skos:exactMatch mtgo:defense .

mtgo:defense_counter a owl:Class,
        linkml:ClassDefinition ;
    rdfs:label "defense_counter" ;
    rdfs:subClassOf [a owl:Restriction;
        owl:maxQualifiedCardinality 1;
        owl:onClass linkml:Integer;
        owl:onProperty mtgo:defense ],
            mtgo:Counter;
    skos:definition "A counter that tracks the defense of a battle.";
    skos:exactMatch mtgo:defense_counter
```

Attualmente esiste un solo sottotipo di battaglia attualmente '*Siege*', che secondo le regole permette a chi la lancia di scegliere un avversario che la protegga, e una volta 'buttata giù' il suo proprietario la esilia e la lancia gratuitamente 'trasformata' (dall'altro lato), ottenendo effetti aggiuntivi.

Non andremo a definire questi criteri di trigger legati unicamente a questo tipo di carta, ma possiamo definire un permanente Trasformato.

3.1.3 Permanente Trasformato

Creiamo la proprietà *isTransformed* applicabile unicamente a *Permanent*

```
mtgo:isTransformed a owl:DatatypeProperty ;
    rdfs:domain mtgo:Permanent ;
    rdfs:range xsd:boolean ;
    rdfs:label "is transformed" ;
    skos:definition "Indicates whether the permanent is transformed." .
```

Definiamo l'azione transform come classe, e poi la aggiungiamo alla lista di Azioni già presenti.

```
mtg:transform a mtgo:Action ;
        skos:definition "To turn over a permanent so that its other face
is up";
    rdfs:label "transform" .
```

3.1.4 Dungeon

I dungeon sono carte che non vanno mischiate nel mazzo, infatti hanno un dorso diverso dalle altre, ma fanno parte della sideboard. Non vengono evocate sul campo come tutte le altre (ci sono altri casi, come *Piani*, *Schemi* e *Cospirazioni*), e non c'è bisogno di più carte raffiguranti lo stesso dungeon in un mazzo.

```
mtgo:Dungeon a owl:Class ;
    rdfs:subClassOf
            [ a owl:Restriction ;
                    owl:onProperty mtgo:startingRoom ;
                    owl:maxQualifiedCardinality 1 ;
                    owl:onClass mtgo:Room ],
            [ a owl:Restriction ;
                    owl:onProperty mtgo:startingRoom ;
                    owl:minQualifiedCardinality 1 ;
                    owl:onClass mtgo:Room ],
            mtgo:Card,
            mtgo:NamedThing ;
    rdfs:label "Dungeon" ;
    skos:definition "A card that represents a Dungeon that can be
explored." ;
    owl:hasPart mtgo:Room ;
```

```
    rdfs:comment "A dungeon consists of interconnected rooms, each with a
triggered ability." .
```

I dungeon vengono esplorati tramite effetto di incantesimi, abilità attivate o innescate che citano "*esplori il dungeon*" o "*prendi l'iniziativa*". Se non hai già iniziato ad esplorare un dungeon, entri nella prima stanza, altrimenti prosegui nelle stanze adiacenti secondo le frecce.

```
mtgo:Room a owl:Class ;
    rdfs:subClassOf mtgo:NamedThing ;
    owl:hasPart mtgo:RoomAbility ;
    rdfs:label "Room" ;
    skos:definition "A room within a dungeon." ;
    rdfs:comment "Each room contains one room ability that activates upon
exploration." .

mtgo:startingRoom a owl:ObjectProperty ;
    rdfs:domain mtgo:Dungeon ;
    rdfs:range mtgo:Room ;
    rdfs:label "starting room" ;
    skos:definition "Indicates the room where the dungeon exploration
starts." .

mtgo:leadsTo a owl:ObjectProperty ;
    rdfs:domain mtgo:Room ;
    rdfs:range mtgo:Room ;
    rdfs:label "adjacent to" ;
    skos:definition "Indicates that a room is adjacent to another room." .
```

I dungeon sono una serie di stanze ognuna con la sua abilità, che viene innescata quando ci entri per la prima volta, e una volta che viene risolto l'effetto dell'ultima stanza, il giocatore ha completato il dungeon, risultato che può attivare determinati effetti.

```
mtgo:RoomAbility a owl:Class ;
        rdfs:subClassOf mtgo:TriggeredAbility ;
        rdfs:label "Room Ability" ;
        skos:definition "An ability that triggers when you enter a room."
.
```

3.1.5 Keywords

Sono state aggiunte tutte le keywords mancanti, con una definizione tratta automaticamente da https://mtg.fandom.com/wiki/ tramite lo script presente in

`keywords/estrai_keywords.ipynb`. Ecco le prime tre in ordine alfabetico:

```
mtg:Absorb a owl:Class ;
    rdfs:label "Absorb" ;
    rdfs:subClassOf mtgo:StaticAbility ;
    skos:definition "If a source would deal damage to this creature,
prevent N of that damage." ;
    mtgo:writtenAs "Absorb N" ;
    skos:exactMatch mtg:absorb .

mtg:Affinity a owl:Class ;
    rdfs:label "Affinity" ;
    rdfs:subClassOf mtgo:StaticAbility ;
    skos:definition "This spell costs less to cast for each [text] you
control." ;
    mtgo:writtenAs "Affinity for [text]" ;
    skos:exactMatch mtg:affinity .

mtg:Afflict a owl:Class ;
    rdfs:label "Afflict" ;
    rdfs:subClassOf mtgo:TriggeredAbility ;
    skos:definition "Whenever this creature becomes blocked, defending
player loses N life." ;
    mtgo:writtenAs "Afflict N" ;
    skos:exactMatch mtg:afflict .
```

3.2 Modifiche

Il tipo di ciascun (o quasi) Individuo era ambiguamente specificato tramite una doppia dichiarazione:

```
mtg:main_phase a owl:Class,
                mtgo:TurnPhase ;
        rdfs:label
        "main_phase",
        "post_combat_main_phase",
        "pre_combat_main_phase" .
```

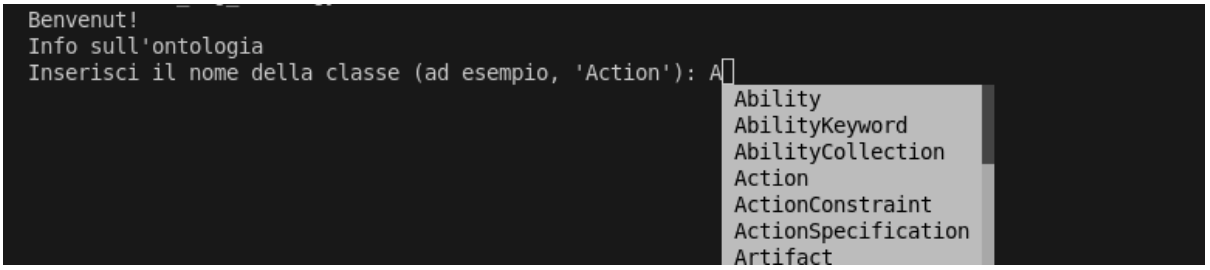
In questo esempio in particolare, la fase principale `main_phase` è specificata sia come una *owl:Class*, che come *mtgo:TurnPhase*. Questa entità in particolare può essere invece trattata come un individuo, con due pseudonimi che indicano comunque la fase principale del gioco.

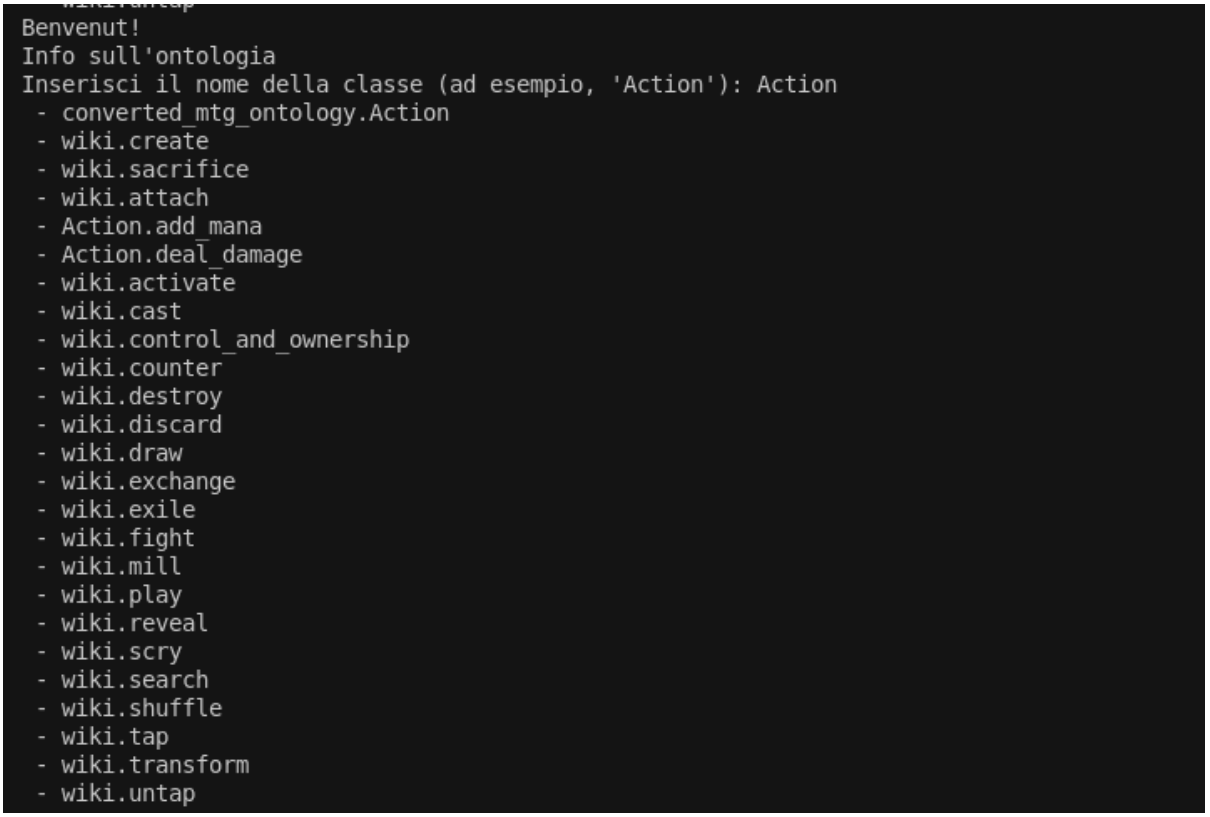
```
mtg:main_phase mtgo:TurnPhase ;
    rdfs:label "main_phase" ;
    skos:altLabel "post_combat_main_phase",
        "pre_combat_main_phase" .
```

3.3 Interazione

Il menù *Simple Term Menu* e la libreria *prompt_toolkit* si sono rivelati essere degli strumenti perfetti per garantire un'esperienza lineare all'utente, che non necessita di alcuna esperienza pregressa ma è guidato ad ogni passo.

Tutte le query di richiesta utente sono effettuate tramite owlready2





4. Knowledge Base

Una knowledge base è una banca dati che racchiude fatti sul mondo di interesse, ragionamenti su quei fatti, e regole logiche che permettono di ottenere (deducendo o inducendo) nuovi fatti o evidenziare incongruenze.

Una KB è quindi definibile come un insieme di assiomi, proposizioni asserite come vere che permettono di raccogliere, organizzare e distribuire la conoscenza.
In questo caso, un utente può porre domande inerenti al dominio approfondito e ottenere risposta dalla KB.

4.1 Gestione

Il popolamento dei fatti nella KB avviene automaticamente tramite **PySWIP**, che in linguaggio Prolog preleva i fatti da ogni carta presente nel file .csv, aggiornato all'avvio, rendendo di fatto la KB sempre aggiornata.
Alcune regole sono applicate poi, per una dimostrazione della funzionalità del sistema.

4.2 Fatti

Per introdurre le carte come **facts** all'interno della KB, è bastato implementare la funzione `Card.to_facts()`, che traduce l'oggetto carta in una lista di stringhe leggibili da Prolog, come ad esempio

```
from src.card import Card
wurmcoil = Card(name="Wurmcoil Larva")
facts = wurmcoil.to_facts()
print(facts)
```

Output

```
cost('Wurmcoil Larva',5)
mana_value('Wurmcoil Larva','B',2)
mana_value('Wurmcoil Larva','C',3)
rarity('Wurmcoil Larva','uncommon')
power('Wurmcoil Larva',3)
toughness('Wurmcoil Larva',3)
text('Wurmcoil Larva','When Wurmcoil Larva dies, create a 1/2
black Phyrexian Wurm artifact creature token with deathtouch and a
2/1 black Phyrexian Wurm artifact creature token with lifelink.')
type('Wurmcoil Larva','Artifact')
```

```
type('Wurmcoil Larva','Creature')
sub_type('Wurmcoil Larva','Phyrexian')
sub_type('Wurmcoil Larva','Wurm')
keyword('Wurmcoil Larva','Lifelink')
keyword('Wurmcoil Larva','Deathtouch')
```

Questi fatti vengono quindi asseriti nella Knowledge Base tramite libreria **pyswip**, permettendoci di creare facilmente delle semplici query come quali sono le keywords presenti in 'Wurmcoil Larva', o qual è la sua forza o costo.

```
from pyswip import Prolog
prolog = Prolog()
for fact in facts:
        prolog.assertz(fact)
```

L'ideale sarebbe inserire tutte le carte possibili all'interno della Knowledge Base:

```
#g contiene tutte le carte
g = Grimoire(omnicomprehensive=True)

for i, card in enumerate(g):
        print(f'asseriti fatti su {i} carte su {len(g)}', end='\r')
        facts = card.to_facts()
        for i, fact in enumerate(facts):
                prolog.assertz(fact)
```

4.3 Regole

Possiamo quindi lavorare sulle Regole all'interno della Knowledge Base.
Per una semplice dimostrazione, implementiamo le regole basilari del combattimento in magic, nella fase dello scambio di danni di una creatura attaccante A e una bloccante B.
Generalmente, se una creatura ha più forza della tempra della creatura bloccante, questa è abbastanza per distruggerla

```
overpowers(A, B) :-
        power(A, Power),
        toughness(B, Toughness),
        Power > Toughness.
```

Ma considereremo due parole chiave determinanti per il risultato e tra le più comuni:

- **Deathtouch** - Qualsiasi danno inferto da questa creatura è abbastanza per distruggere la creatura ricevente
- **Indestructible** - Questa creatura non può essere distrutta
  Per capire se una creatura dispone di una determinata keyword, non c'è bisogno di alcuna regola, basta una query generale come:

  Esempio

  ```
  ?- keyword('Kodama of the West Tree', Keyword).
  ```

Quindi possiamo arricchire la nostra regola aggiungendo la postilla di 'se A ha deathtouch, non serve fare più danni della tempra di B'

```
overpowers(A, B) :-
        power(A, Power),
        toughness(B, Toughness),
        (Power > Toughness; keyword(A, 'Deathtouch')).
```

Allo stesso tempo però, se B ha indistruttibile, non può essere distrutta indipendentemente dal danno ricevuto. Quindi definiamo **would_kill**

```
would_kill(A, B) :-
        (overpowers(A, B) \+ keyword(B, 'Indestructible')).
```

Esempio

```
?- would_kill('Giant', 'Bear Cub').
```

Il gioco si fa interessante quando aggiungiamo una terza (e ancora più comune) keyword:

- **Trample** - I danni inferti in eccesso ad una creatura, vengono fatti al bersaglio originale dell'attacco (player, planeswalker, battaglia)
  Normalmente una creatura bloccata non infliggerebbe danni al bersaglio originale, con trample si può infliggere la differenza dei danni necessari a distruggere una creatura al bersaglio originale.
  Per cui se il danno necessario per distruggerla è 1 grazie a deathtouch i danni inferti al giocatore saranno direttamente Forza della creatura -1, anche se la

creatura è indistruttibile (per il funzionamento non banale dei livelli di regole nel gioco).

per cui possiamo definire come regola **damage_to_player**

```
damage_to_player(A, B, Value) :-
        power(A, Power),
        toughness(B, Toughness),
        (keyword(A, 'Trample'), keyword(A, 'Deathtouch')
                -> Value is Power - 1 ;
        keyword(A, 'Trample')
                -> Value is Power - Toughness ;
        Value is 0 ).
```

Esempio

```
?- damage_to_player('Giant', 'Bear Cub', Value).
```

5. Conclusioni

Il progetto si è rivelato più arduo del previsto da affrontare da solo, soprattutto per aver sottovalutato un database già fatto sui mazzi degli utenti.

Obiettivi futuri del progetto

- Come già detto in 2.2 Cards Recommender, un motore di inferenza renderebbe questo progetto unico, richiederebbe però una fitta rete di regole in Prolog, al pari del regolamento disponibile online.
  - Potrebbe rendere possibili simulazioni intere di partite, e rivelare carte che la community non ha mai considerato.
- L'aggiunta di regole custom creerebbe una community non da sottovalutare dietro un tool del genere.
- La possibilità di ottenere mazzi da altre piattaforme come Archidekt potrebbe essere utile per raccomandazioni ancora più precise e varie.

Damato Luigi Lele