POLITECNICO DI TORINO

Dipartimento di Ingegneria Gestionale e della Produzione

Corso di Laurea in Ingegneria Gestionale
Classe L-8

Tesi di Laurea

Applicazione per la simulazione dei formati limited di “magic: the gathering” e conseguente ottimizzazione del mazzo

Relatore
Prof. Fulvio Corno

Candidato
Leonardo Priami

AA. 2022/2023

Indice

1 Proposta di progetto

1.1 Titolo della proposta

Applicazione per la simulazione dei formati limited di magic: the gathering e conseguente ottimizzazione del mazzo.

1.2 Descrizione del problema proposto

Magic: the gathering è il gioco di carte collezionabili più conosciuto e giocato al mondo; la sua popolarità deriva dal fatto che può essere giocato in più modalità (formati) differenti, attirando quindi un maggior numero di giocatori rispetto ad altri suoi competitors.

Essendo un gioco di carte collezionabili, ogni giocatore utilizza un mazzo proprio, composto da carte appartenenti alla sua collezione personale rispettando dei vincoli che possono variare a seconda del formato scelto; in alcuni di questi formati però, denominati appunto "limited", la scelta delle carte da includere nel mazzo è limitata al contenuto di 6 pacchetti da 15 carte ciascuno aperti poco prima dell'inizio della partita o del torneo.

I giocatori dovranno quindi creare un mazzo di 40 carte a partire dalle 90 aperte in questo modo e da altre carte "base", messe a disposizione gratuitamente dall'organizzatore dell'evento e uguali per tutti i giocatori.

Questa applicazione si pone l'obiettivo di simulare l'apertura dei pacchetti e di creare il mazzo più performante possibile tramite l'utilizzo di un algoritmo ricorsivo.

1.3 Descrizione della rilevanza gestionale del problema

Nonostante per la maggior parte dei giocatori sia solo un passatempo, c'è chi ha fatto di magic la sua professione e chi aspira a farlo.

A tale scopo, fare le scelte giuste durante la costruzione del mazzo è importante tanto quanto fare le giocate migliori durante una partita, e questo richiede una particolare abilità nel calcolare a mente numerose probabilità con poco tempo a disposizione.

Questa applicazione risolverebbe tale problema e sarebbe di aiuto a tutti i giocatori che vogliono migliorare la propria performance senza il bisogno di acquisire le conoscenze statistiche necessarie.

Inoltre, il modello utilizzato per ottimizzare il mazzo potrebbe essere applicato anche alla risoluzione di problemi in ambito industriale o aziendale.

1.4 Descrizione dei data-set per la valutazione

I dati utilizzati per la risoluzione del problema verranno tratti dal database al link seguente:

https://mtgjson.com

Tutti gli attributi necessari alla risoluzione del problema sono contenuti nel file denominato AllPrintings (esportabile in SQL) o sono da questi calcolabili.

Le tabelle del database utilizzate saranno Set, che rappresenta le diverse espansioni del gioco, e Card, che rappresenta singolarmente ciascuna carta con i propri parametri di gioco necessari all'implementazione dell'algoritmo.

1.5 Descrizione preliminare degli algoritmi coinvolti

Il primo passo per la risoluzione del problema consiste nel simulare l'apertura di 6 pacchetti da 15 carte ciascuno.

Per fare ciò, verranno selezionate in maniera casuale 15 carte tra quelle contenute nell'espansione selezionata dall'utente per 6 volte (la stessa carta non può essere presente in più di una copia nello stesso pacchetto, ma potrebbe essere contenuta in più pacchetti).

Una volta determinate le 90 carte disponibili, a ciascuna di esse verrà attribuito un valore che ne indica la forza, ovvero il vantaggio che fornisce a chi la gioca nel corso di una partita, basandosi su parametri di gioco già contenuti all'interno della tabella Card presente nel database.

Le carte di magic: the gathering si suddividono in 5 colori (bianco, blu, nero, rosso, verde) e in 2 tipi (terra e magia).

Le terre avranno un valore di forza nullo, poiché il loro unico scopo all'interno del gioco è quello di consentire al giocatore di giocare le magie: ogni magia ha un costo in termini di terre necessarie per poter essere giocata, che ne specifica il numero e il colore.

Le magie avranno invece un valore di forza proporzionale al loro costo e alla loro rarità.

A questo punto verrà implementata una funzione di ottimizzazione ricorsiva che partendo dalla carta più forte costruirà un mazzo di 40 carte ottimizzandone la performance, dove per performance si intende la somma delle forze delle singole magie contenute al suo interno, ciascuna moltiplicata per la probabilità di pescare le terre necessarie a giocare la rispettiva magia.

Si noti che non sarà sufficiente selezionare le carte più forti, perché costi di gioco troppo alti o troppo differenti tra loro in termini di colore diminuiranno notevolmente la probabilità di pescare le terre necessarie per poter giocare tutte le magie all'interno del mazzo.

1.6 Descrizione preliminare delle funzionalità previste per l'applicazione software

L'utente potrà selezionare l'espansione della quale ricevere i 6 pacchetti da aprire, e cliccando un pulsante "apri pacchetti" riceverà come output una lista dei nomi delle carte aperte in questo modo.

Dopodiché, cliccando un pulsante "crea mazzo perfetto", l'utente potrà visionare la lista delle 40 carte contenute nel mazzo più performante possibile, selezionate tra le 90 aperte e quelle "base" messe a disposizione di tutti i giocatori.

Inoltre, nel caso in cui l'utente avesse trovato una o più carte che gli piacciono al punto di volerle giocare a tutti i costi, sarà disponibile una funzione in grado di determinare il mazzo migliore tra tutti quelli possibili che includono le carte selezionate.

2 Descrizione del problema affrontato

Magic: the Gathering è il gioco di carte collezionabili più popolare al mondo, con un numero di giocatori stimato tra i 30 e i 40 milioni. Ciò che lo rende differente dai giochi di carte tradizionali è la necessità per ogni giocatore di creare il proprio mazzo in maniera personalizzata, cosa che richiede una certa abilità strategica anche al di fuori della partita stessa.

Magic offre diverse modalità di gioco, che si possono però suddividere in due categorie: limited e constructed.

Nei formati constructed il giocatore costruisce il proprio mazzo avendo a disposizione tutte le carte nella propria collezione, con il vincolo di avere almeno 60 carte nel mazzo e al massimo 4 copie di ogni carta.

Nei formati limited invece, ad ogni giocatore vengono distribuiti 6 pacchetti ancora sigillati da 15 carte ciascuno, e il mazzo deve essere costruito utilizzando esclusivamente carte in essi contenute, con una dimensione minima di 40 carte.

Ciascuna delle carte all'interno di ogni pacchetto appartiene ad una di 4 rarità diverse, alle quali corrispondono un diverso numero di slot disponibili per singolo pacchetto: 11 common, 3 uncommon e 1 rare or mythic.

Nel caso dei formati constructed, l'avvento di internet ha reso la costruzione dei mazzi estremamente facile: ogni giocatore ha infatti accesso alle liste vincitrici dei più grandi tornei internazionali, che vengono semplicemente copiate e perfezionate da una community molto attiva online.

Un tale processo di condivisione e ottimizzazione non è però applicabile ai formati limited, poiché ciascun giocatore ha a disposizione un insieme di carte differente da tutti gli altri.

Proprio per questo motivo è interessante lo sviluppo di un algoritmo il cui scopo è quello di assistere i giocatori nella costruzione del proprio mazzo nei formati limited, basandosi sull'attribuzione di costi e valori alle carte trovate nei pacchetti aperti e la successiva riproduzione di un problema dello zaino, che punta alla costruzione di un mazzo con il più alto possibile valore di forza a parità di numero di carte che lo costituiscono, in questo caso 40.

Le carte di Magic si suddividono in magie e terre: le prime hanno un costo da pagare per poter essere giocate, espresso in termini di terre già in gioco da dover utilizzare, e un effetto che influenza il corso della partita, mentre le seconde hanno il solo scopo di pagare i costi delle magie.

Esistono 5 diverse tipologie di terra (pianura, isola, palude, montagna, foresta), e il costo di una magia può richiedere che alcune delle terre necessarie siano di un tipo specifico. Per esempio la carta in Fig. 1 richiede l'utilizzo di 3 terre per essere giocata, di cui un'isola e una pianura (il costo è rappresentato dai simboli in alto a destra della carta; il simbolo bianco rappresenta una pianura, quello blu un'isola).

Figura 1 – Esempio di carta multicolore.

Pertanto alle magie verrà attribuito un valore di forza proporzionale all'impatto del loro effetto sull'esito della partita, mentre le terre avranno un valore di forza nullo, in quanto non influenzeranno l'esito della partita di per sé ma dovranno essere incluse nel mazzo necessariamente per poter giocare le magie.

Per determinare il numero di terre da includere nel mazzo, ad ogni magia verranno attribuiti dei requisiti minimi di terre necessarie per giocarla, suddivise tra i 5 tipi esistenti, tramite l'utilizzo di una distribuzione ipergeometrica: dato che la mano iniziale è di 7 carte, si pesca una carta a turno e si può giocare una sola terra per turno, la distribuzione ipergeometrica verrà impostata in modo da calcolare il numero minimo di casi favorevoli (terre che contribuiscono al lancio della magia) da includere all'interno del campione (mazzo) affinché ne vengano estratti almeno in quantità necessaria allo scopo prefissato (lanciare la magia) su 7 + numero di turni trascorsi dall' inizio della partita estrazioni con una probabilità scelta arbitrariamente (i giocatori professionisti indicano una probabilità accettabile tra 80% e 90%).

Il numero di terre necessarie al mazzo per funzionare regolarmente sarà ottenuto dall'unione dei massimi dei 5 requisiti relativi ai rispettivi tipi di terra.

Inoltre, se il massimo requisito di terre in generale sarà maggiore della somma dei massimi dei 5 requisiti di terre specifiche, verranno aggiunte terre suddivise equamente per tipo pari alla differenza.

3 Descrizione del data-set utilizzato

Il dataset utilizzato dall'applicazione è disponibile all'indirizzo https://mtgjson.com/ed è composto da numerose tabelle contenenti tutte le informazioni riguardanti il gioco.

Noi siamo interessati a 2 di queste tabelle, Sets e Cards, la prima contenente tutte le espansioni rilasciate dall'uscita del gioco e la seconda tutte le carte esistenti.

Le espansioni tenute in considerazione ai fini dell'applicazione sono quelle cartacee, distribuite in pacchetti e già rilasciate completamente, non sempre in fase di preview; pertanto viene effettuata una query SQL che seleziona il nome e il codice di tutte le espansioni con isOnlineOnly=0, isPartialPreview=0 e booster is not null. La Fig. 2 riporta un estratto della tabella Sets con le sole colonne prese in considerazione.

code	name
10E	Tenth Edition
2ED	Unlimited Edition
2XM	Double Masters
3ED	Revised Edition
4ED	Fourth Edition
5DN	Fifth Dawn
5ED	Fifth Edition
6ED	Classic Sixth Edition
7ED	Seventh Edition
8ED	Eighth Edition
9ED	Ninth Edition
A25	Masters 25
AER	Aether Revolt
AFR	Adventures in the…
AKH	Amonkhet
ALA	Shards of Alara
ALL	Alliances
APC	Apocalypse
ARB	Alara Reborn
ARN	Arabian Nights
ATQ	Antiquities
AVR	Avacyn Restored
BBD	Battlebond
BFZ	Battle for Zendikar
BNG	Born of the Gods
BOK	Betrayers of Kamig…
CHK	Champions of Kam…

Figura 2 – Tabella Sets.

Gli attributi delle carte ai quali siamo interessati sono il nome, la rarità, il codice dell'espansione di appartenenza, manaCost e manaValue; la differenza tra gli ultimi due consiste nel fatto che il

primo è una stringa dove ogni lettera corrisponde a una terra specifica richiesta (w=pianura, u=isola, b=palude, r=montagna e g=foresta), mentre il secondo è semplicemente il numero totale di terre richieste. La Fig. 3 riporta un estratto di tabella Cards con le sole colonne prese in considerazione.

name	rarity	manaCost	manaValue	setCode
Grasp of Darkness	uncommon	{B}{B}	2	OGW
Hobgoblin Bandit Lord	rare	{1}{R}{R}	3	AFR
Pirate's Pillage	common	{3}{R}	4	2X2
Jace Beleren	mythic	{1}{U}{U}	3	SS1
Ghostly Prison	uncommon	{2}{W}	3	C21
Doomed Artisan	rare	{2}{W}	3	C19
Zodiac Pig	uncommon	{3}{B}	4	PTK
Jwar Isle Refuge	uncommon	NULL	0	C18
Demonic Attorney	rare	{1}{B}{B}	3	LEA
Wu Longbowman	common	{2}{U}	3	ME3
Utter End	rare	{2}{W}{B}	4	NCC
Land Grant	common	{1}{G}	2	MMQ
Krosan Grip	uncommon	{2}{G}	3	CMA
Aethertow	common	{3}{W/U}	4	SHM
Seal of Primordium	common	{1}{G}	2	MM3
Simic Initiate	common	{G}	1	MM2
Workshop Warchief	rare	{3}{G}{G}	5	SNC
Emberwilde Caliph	rare	{2}{U}{R}	4	MIR
Sacred Foundry	rare	NULL	0	GTC
Elvish Visionary	common	{1}{G}	2	CMA
Gather the Townsfolk	common	{1}{W}	2	PRM
Morkrut Behemoth	common	{4}{B}	5	DBL
Concordia Pegasus	common	{1}{W}	2	RNA
Raging Regisaur	uncommon	{2}{R}{G}	4	JMP
Liliana's Mastery	rare	{3}{B}{B}	5	MIC

Figura 3 – Tabella Cards.

4 Descrizione strutture dati e algoritmi utilizzati

L'applicazione è stata sviluppata in linguaggio Java con il supporto delle interfacce JavaFX. Sono inoltre implementati il pattern MVC (Model-View-Controller) ed il pattern DAO (Data Access Object).

La struttura dell'applicazione software è divisa in tre package:

Il Controller, denominato mtg_cards, contiene la classe Main per l'avvio dell'applicazione e le classi FXMLController ed EntryPoint che definiscono i metodi che permettono l'interazione con l'utente e la definizione dell'interfaccia grafica.

Il Database, denominato mtg_cards.db, contiene la classe DBConnect utilizzata per la connessione al database e la classe MtgDAO che attraverso query SQL permette il caricamento dei dati presenti nel database all'interno dell'ambiente Java.

Il Model, denominato mtg_cards.model include la classe omonima che contiene tutta la logica applicativa del software. Inoltre, sono presenti anche le classi Java Bean Carta ed Espansione.

Tramite i metodi contenuti nella classe MtgDAO, l'applicazione attinge a tutti i dati necessari all'implementazione dell'algoritmo ricorsivo: il metodo getAllEspansioni crea una lista di tutte le possibili espansioni a cui apparterranno i pacchetti da aprire, mentre getAllCarte, una volta selezionata l'espansione con la query riportata sotto, simula l'apertura di 6 pacchetti ad essa appartenenti.

```
SELECT code,name FROM mtg_cards.sets WHERE isOnlineOnly=0 AND
isPartialPreview=0 AND booster IS NOT NULL
```

Per quanto riguarda la simulazione dell'apertura di un pacchetto, si effettua una query SQL che seleziona dapprima 11 carte con il codice dell'espansione scelta e rarity='common', poi 3 carte con rarity='uncommon' e a seguire 1 carta con rarity='rare' or rarity='mythic', sempre con il codice corrispondente a quello dell'espansione scelta. La corrispondente query SQL è riportata sotto.

```
(SELECT name,rarity,manaCost,manaValue,setCode FROM mtg_cards.cards
WHERE setCode = ? and rarity = 'common'
ORDER BY RAND() LIMIT 11)
UNION ALL
SELECT name,rarity,manaCost,manaValue,setCode FROM mtg_cards.cards
WHERE setCode = ? and rarity = 'uncommon'
ORDER BY RAND() LIMIT 3)
UNION ALL
SELECT name,rarity,manaCost,manaValue,setCode FROM mtg_cards.cards
WHERE setCode = ? AND (rarity='rare' or rarity='mythic')
ORDER BY RAND() LIMIT 1)
```

L'unione delle 3 selezioni corrisponde al contenuto di una bustina, pertanto è sufficiente ripetere la query 6 volte e unire tutte le righe per generare il card pool a cui applicare l'algoritmo ricorsivo per costruire il mazzo ottimale.

Per far sì che le carte vengano estratte in maniera casuale, si utilizza `ORDER BY RAND()` e si selezionano le prime n righe trovate in questo modo con LIMIT.

Per ogni carta estratta in questo modo viene costruito un oggetto di classe Carta con rispettivi nome, rarità, costo, metodi per calcolarne la forza e stabilire il numero di terre di ogni tipo necessarie nel mazzo se si vuole includere tale carta tramite l'utilizzo della distribuzione ipergeometrica. In Fig. 4 sono riportati i metodi della classe Carta che utilizzano la distribuzione ipergeometrica (che a sua volta necessita il calcolo di coefficienti binomiali) per determinare tali prerequisiti.

```
private long binomiale(int n, int k) {
    if(k>n-k) {
        k=n-k;
    }
    long b=1;
    for(int i=1, m=n; i<=k; i++, m--) {
        b=b*m/i;
    }
    return b;
}

private double ipergeometrica(int x, int carteMazzo, int terreMazzo, int cartePescate) {
    return (double) this.binomiale(terreMazzo, x) * (double) this.binomiale(carteMazzo-terreMazzo, cartePescate-x)/ (double) this.binomiale(carteMazzo, cartePescate);
}

public int terreRichieste(Carta c) {
    int terreMazzo = c.getManaValue();
    int i = 0;
    double p = 0;
    while(p<soglia) {
        terreMazzo++;
        i = 0;
        p = 1;
        while(i<c.getManaValue()) {
            p = p-this.ipergeometrica(i, 40, terreMazzo, 7+c.getManaValue());
            i++;
        }
    }
    return terreMazzo;
}

public int coloriRichiesti(Carta c, int col) {
    if(col==0) {
        return 0;
    }
    int coloriMazzo = col;
    int i = 0;
    double p = 0;
    while(p<soglia) {
        coloriMazzo++;
        i = 0;
        p = 1;
        while(i<col) {
            p = p-this.ipergeometrica(i, 40, coloriMazzo, 7+c.getManaValue());
            i++;
        }
    }
    return coloriMazzo;
}
```

Figura 4 – Metodi della classe Carta per il calcolo della distribuzione iperegometrica e la determinazione dei vincoli di colore delle carte.

Tutte le carte estratte durante la simulazione dell'apertura dei pacchetti vengono inserite in una lista, successivamente ordinata per valore di forza decrescente all'inizio della classe model tramite l'utilizzo di un comparator come si può vedere in Fig. 5.

```
private Collection<Carta> ordinaCarte() {
    Collections.sort(carte, new Comparator<Carta>() {
        public int compare(Carta c1, Carta c2) {
            if(c1.getForza()==c2.getForza()) {
                return 0;
            }
            else if (c1.getForza()>c2.getForza()) {
                return -1;
            }
            else return 1;
        }
    });
    return carte;
}
```

Figura 5 – Ordinamento delle carte per valore di forza.

Il metodo "ottimo" all'interno della classe model verifica se l'utente ha selezionato delle carte da includere obbligatoriamente nel mazzo finale, quindi imposta la forza e le terre del mazzo in base

alla forza e al costo delle carte scelte (se l'utente non ha selezionato alcuna carta, tutti i valori della classe model vengono azzerati).

```
public List<Carta> ottimo(Carta c1, Carta c2){
    forza = 0;
    forzaMassima = 0;
    numeroCarte = 0;
    numeroTerre = 0;
    numeroPianure = 0;
    numeroIsole = 0;
    numeroPaludi = 0;
    numeroMontagne = 0;
    numeroForeste = 0;
    numeroTerreOttimo = 0;
    numeroPianureOttimo = 0;
    numeroIsoleOttimo = 0;
    numeroPaludiOttimo = 0;
    numeroMontagneOttimo = 0;
    numeroForesteOttimo = 0;
    iterazioni = 0;
    magie = new ArrayList<>();
    if(c1 != null) {
        int copie = 0;
        List<Carta> temp = new ArrayList<Carta>(carte);
        for(Carta c : temp) {
            if(c.getNome().equals(c1.getNome())) {
                magie.add(c);
                copie++;
                carte.remove(c);
            }
        }
        forza = forza+c1.getForza()*copie;
        numeroPianure = c1.getTerreW();
        numeroIsole = c1.getTerreU();
        numeroPaludi = c1.getTerreB();
        numeroMontagne = c1.getTerreR();
        numeroForeste = c1.getTerreG();
        numeroTerre = c1.getTerre();
        numeroCarte = numeroTerre+copie;
    }
    if(c2 != null) {
        int copie = 0;
        List<Carta> temp = new ArrayList<Carta>(carte);
        for(Carta c : temp) {
            if(c.getNome().equals(c2.getNome())) {
                magie.add(c);
                copie++;
                carte.remove(c);
            }
        }
        forza = forza+c2.getForza()*copie;
        if(numeroPianure < c2.getTerreW()) numeroPianure = c2.getTerreW();
        if(numeroIsole < c2.getTerreU()) numeroIsole = c2.getTerreU();
        if(numeroPaludi < c2.getTerreB()) numeroPaludi = c2.getTerreB();
        if(numeroMontagne < c2.getTerreR()) numeroMontagne = c2.getTerreR();
        if(numeroForeste < c2.getTerreG()) numeroForeste = c2.getTerreG();
        if(numeroTerre < c2.getTerre()) numeroTerre = c2.getTerre();
        if(numeroTerre <= numeroPianure+numeroIsole+numeroPaludi+numeroMontagne+numeroForeste) {
            numeroCarte = numeroPianure+numeroIsole+numeroPaludi+numeroMontagne+numeroForeste+magie.size();
        }
        else {
            numeroCarte = numeroTerre+magie.size();
        }

    }
    cerca(magie, carte, numeroTerre, numeroPianure, numeroIsole, numeroPaludi, numeroMontagne, numeroForeste, forza);
    return magie;
}
```

Figura 6 – Metodo “ottimo” che verifica le impostazioni dell’utente e chiama il metodo ricorsivo “cerca”.

Dopodiché il metodo procede alla chiamata della funzione ricorsiva "cerca", passandole come parametri la lista di carte eventualmente già incluse nel mazzo, la lista di carte esterne al mazzo, il numero di terre da includere nel mazzo in base alle carte attualmente presenti al suo interno, suddivise per tipo, e il valore di forza attuale del mazzo, corrispondente alla somma dei valori di forza delle singole carte che contiene attualmente (le terre hanno un valore di forza nullo).

All'inizio di ogni chiamata ricorsiva si verifica se il mazzo ha raggiunto le 40 carte; in caso affermativo la forza del mazzo viene confrontata con la forza del mazzo precedentemente scelto come ottimo e, se maggiore, il mazzo appena finito diventa il nuovo mazzo ottimo.

Dopodiché in entrambi i casi i valori utilizzati dalla funzione vengono reimpostati a zero e si procede con la costruzione di un nuovo mazzo.

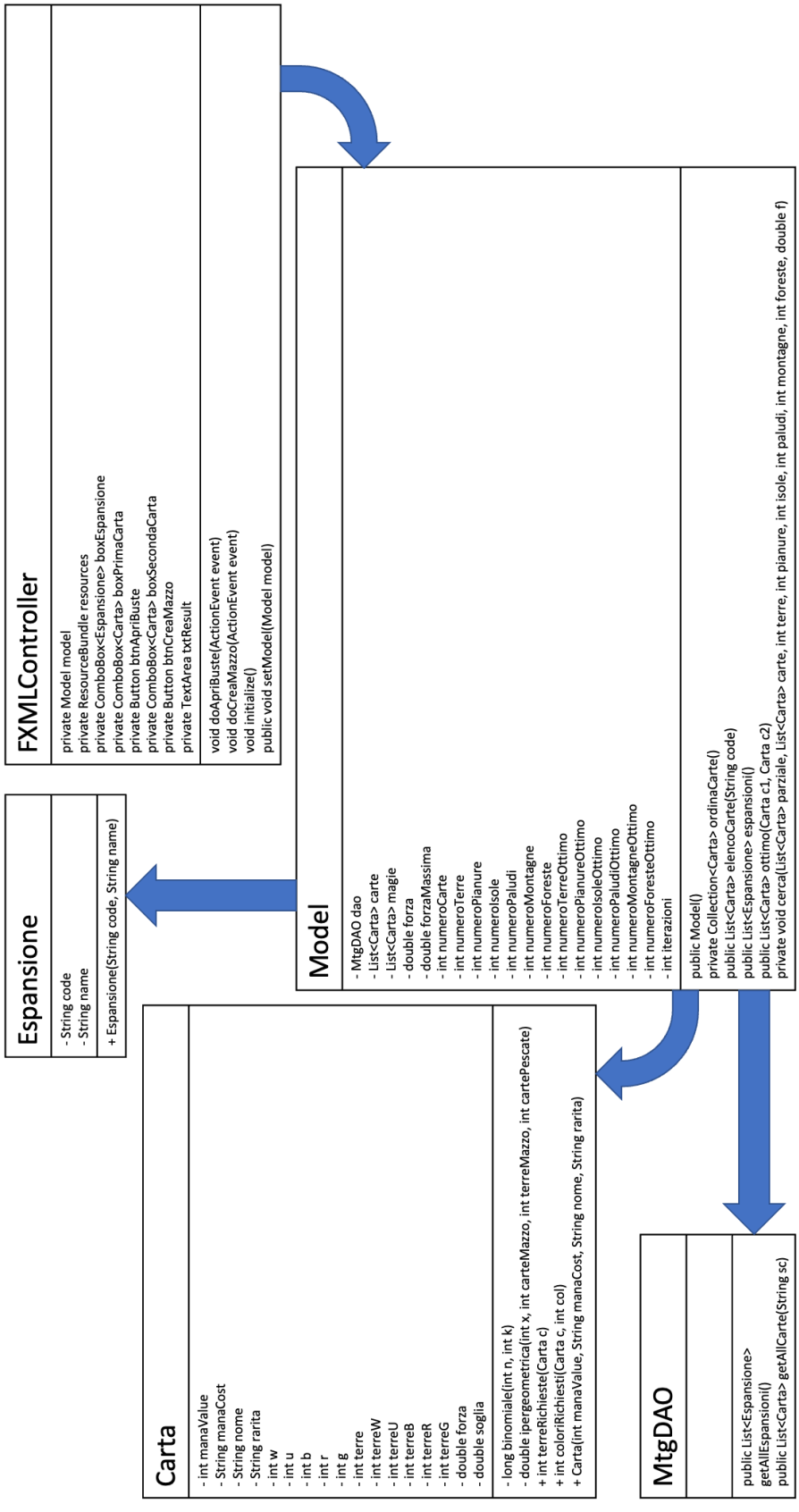
Nel caso generale, all'aggiunta di ogni nuova carta al mazzo si aggiunge la sua forza a quella del mazzo e si confrontano i suoi costi con le terre nel mazzo, uno ad uno per tipo di terra: ad ogni confronto, se il costo della carta è più stringente del numero di terre attualmente nel mazzo, viene aggiunta la differenza.

```
cerca(magie, carte, numeroTerre, numeroPianure, numeroIsole, numeroPaludi, numeroMontagne, numeroForeste, forza);
return magie;


private void cerca(List<Carta> parziale, List<Carta> carte, int terre, int pianure, int isole, int paludi, int montagne, int foreste, double f) {
    if(parziale.size()+Math.max(numeroTerre, numeroPianure+numeroIsole+numeroPaludi+numeroMontagne+numeroForeste) >= 40) {
        if(forza > forzaMassima && parziale.size()+Math.max(numeroTerre, numeroPianure+numeroIsole+numeroPaludi+numeroMontagne+numeroForeste) == 40) {
            magie = new ArrayList<Carta>(parziale);
            forzaMassima = forza;
            numeroTerreOttimo = numeroTerre;
            numeroPianureOttimo = numeroPianure;
            numeroIsoleOttimo = numeroIsole;
            numeroPaludiOttimo = numeroPaludi;
            numeroMontagneOttimo = numeroMontagne;
            numeroForesteOttimo = numeroForeste;
            iterazioni = 0;
        }
        numeroTerre = terre;
        numeroPianure = pianure;
        numeroIsole = isole;
        numeroPaludi = paludi;
        numeroMontagne = montagne;
        numeroForeste = foreste;
        forza = f;
        return;
    }
    for(Carta c : carte) {
            parziale.add(c);
            double forzaUltima = forza;
            int numeroTerreUltima = numeroTerre;
            int numeroPianureUltima = numeroPianure;
            int numeroIsoleUltima = numeroIsole;
            int numeroPaludiUltima = numeroPaludi;
            int numeroMontagneUltima = numeroMontagne;
            int numeroForesteUltima = numeroForeste;
            forza = forza+c.getForza();
            if(numeroPianure < c.getTerreW()) numeroPianure = c.getTerreW();
            if(numeroIsole < c.getTerreU()) numeroIsole = c.getTerreU();
            if(numeroPaludi < c.getTerreB()) numeroPaludi = c.getTerreB();
            if(numeroMontagne < c.getTerreR()) numeroMontagne = c.getTerreR();
            if(numeroForeste < c.getTerreG()) numeroForeste = c.getTerreG();
            if(numeroTerre < c.getTerre()) numeroTerre = c.getTerre();
            if(numeroTerre <= numeroPianure+numeroIsole+numeroPaludi+numeroMontagne+numeroForeste) {
                numeroCarte = numeroPianure+numeroIsole+numeroPaludi+numeroMontagne+numeroForeste+magie.size();
            }
            else {
                numeroCarte = numeroTerre+magie.size();
            }

            List<Carta> cartemp = new ArrayList<Carta>(carte);
            cartemp.remove(c);
            iterazioni++;
            if(iterazioni>10000000) return;
            cerca(parziale, cartemp, terre, pianure, isole, paludi, montagne, foreste, f);
            parziale.remove(parziale.size()-1);
            forza = forzaUltima;
            numeroTerre = numeroTerreUltima;
            numeroPianure = numeroPianureUltima;
            numeroIsole = numeroIsoleUltima;
            numeroPaludi = numeroPaludiUltima;
            numeroMontagne = numeroMontagneUltima;
            numeroForeste = numeroForesteUltima;
    }
}
```

Fig. 7 – Metodo ricorsivo cerca.

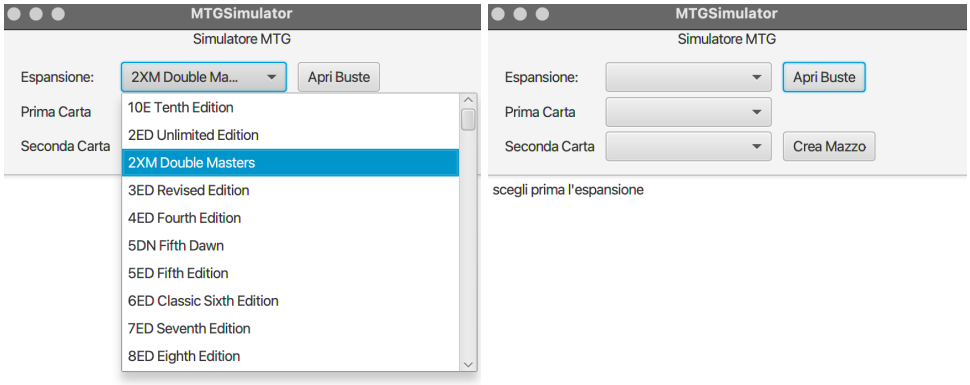
5 Diagramma delle classi principali

Questa sezione riporta il diagramma UML delle classi principali, dove i metodi get, set e to_string non sono riportati per favorire la leggibilità.

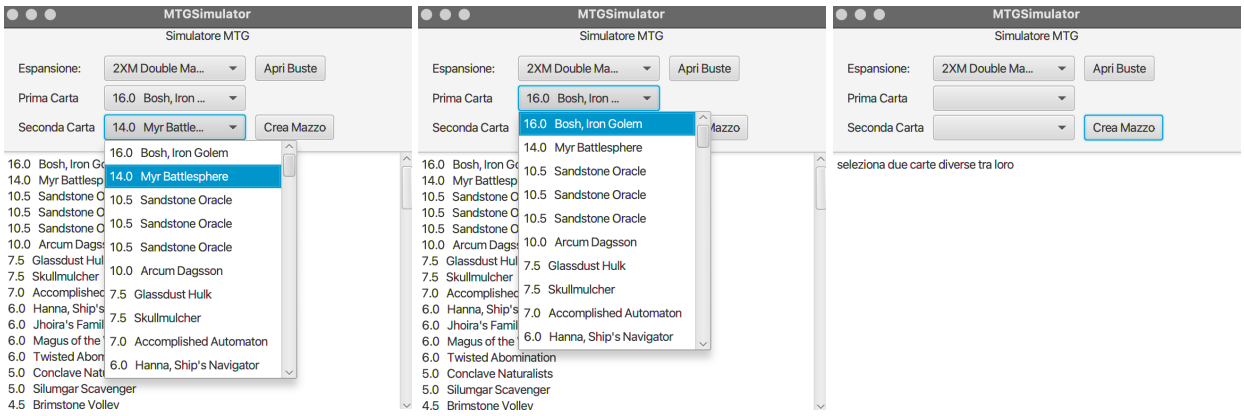


6 Videate dell’applicazione e collegamento al video

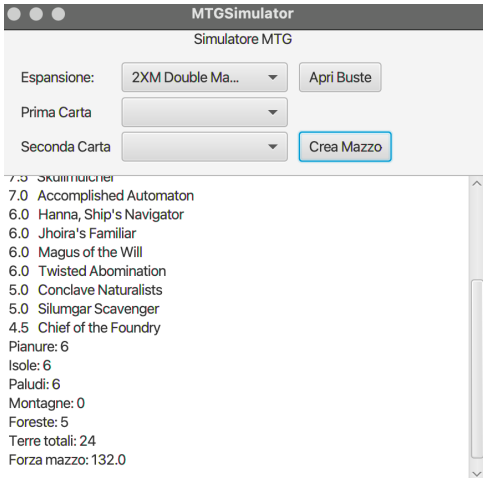
Il link al video dimostrativo del funzionamento dell’applicazione è:
https://youtu.be/8ZBVpQn2PCs



La prima tendina dell’interfaccia utente permette di selezionare l’espansione; nel caso in cui si prema il bottone “Apri Buste” prima di aver selezionato un’espansione apparirà un messaggio di errore.



Una volta aperte le buste, l’utente ha la possibilità di scegliere una o due carte tra quelle trovate da inserire forzatamente nel mazzo; nel caso in cui la stessa carta venga selezionata due volte, apparirà un messaggio di errore.



Cliccando il bottone “Crea Mazzo” si avvia la funzione ricorsiva che costruisce il mazzo migliore secondo i parametri dell’applicazione, e ne stampa la lista indicando le terre da aggiungere alle carte trovate nelle buste e il valore di forza complessivo.

7 Conclusioni e valutazione dei risultati ottenuti

Lo scopo principale dell'applicazione è quello di simulare un draft di magic e guidare i giocatori nella scelta delle carte da includere nel mazzo per aumentare le proprie possibilità di vittoria.

L'utilizzo di una distribuzione statistica nel fare ciò è il suo maggior punto di forza, in quanto l'unico valore stabilito arbitrariamente è il valore di forza da assegnare a ciascuna carta e tutto il resto del procedimento si affida a principi matematici.

Il valore di forza di una carta è dato dal prodotto di costo e rarità (ad ognuna delle 4 rarità viene attribuito un valore numerico equidistante dal precedente e dal successivo) e può comunque essere corretto moltiplicandolo per un coefficiente se ci si accorge che i risultati sono migliorabili.

Un'altra funzione interessante è quella di far scegliere all'utente una o più carte attorno a cui strutturare il mazzo, pensata per chi predilige un certo tipo di strategia o chi semplicemente ha trovato all'interno dei pacchetti la sua carta preferita e vuole giocarla a tutti i costi.

Un altro modo di utilizzare l'applicazione potrebbe essere a scopo di allenamento in vista di un torneo importante, simulando ripetutamente l'apertura dei pacchetti e provando a creare il mazzo per conto proprio prima di vedere quello scelto dall'algoritmo ricorsivo e fare il confronto.

Naturalmente tutto ciò ha anche delle limitazioni: ottimizzare il proprio mazzo non necessariamente porta alla vittoria, in quanto sono in gioco anche fortuna, abilità nel saper giocare le proprie carte al momento giusto e poter prevedere le mosse dell'avversario; nonostante ciò, in un gioco con così tante sfaccettature avere un aiuto esterno può fare la differenza, soprattutto per i principianti.

Inoltre, è stato necessario imporre un limite di 10,000,000 di chiamate ricorsive effettuate durante la costruzione dello stesso mazzo per avere un tempo di risposta ragionevole. Poiché la lista delle carte da provare ad inserire nel mazzo viene precedentemente ordinata per valore di forza decrescente, il risultato ottenuto resta comunque un’approssimazione valida del mazzo ottimo.

Link al video dimostrativo: https://youtu.be/8ZBVpQn2PCs

Link al progetto GitHub: https://github.com/TdP-prove-finali/PriamiLeonardo