Politecnico di Torino

Dipartimento di Ingegneria Gestionale e della Produzione

Corso di Laurea Triennale in Ingegneria Gestionale – Classe L8

A.A 2025/2026

Analisi di sinergie tra carte e generazione guidata di deck

Relatore

Prof. Averta Giuseppe Bruno

Candidato

Pintus Andrea

Indice

1.Proposta di progetto

1.1 Titolo del progetto

Analisi sinergie tra carte e generazione guidata di deck.

1.2 Descrizione del problema affrontato

Clash Royale è un gioco di strategia digitale in tempo reale in cui il successo dipende dalla costruzione di un mazzo di 8 carte (scelte tra oltre 100 disponibili). Il problema affrontato riguarda la configurazione ottimale di un sistema complesso: non è sufficiente selezionare le carte più forti individualmente, poiché l'efficacia del mazzo dipende dalle interazioni (sinergie) tra le unità. L'utente finale si trova spesso di fronte a un'eccessiva varietà di combinazioni (esplosione combinatoria), che rende difficile identificare mazzi bilanciati senza ricorrere alla copia passiva di soluzioni preesistenti. L'obiettivo dell'applicazione è fornire un supporto algoritmico che, partendo da una singola carta scelta dall'utente (input), generi combinazioni di 8 carte (output) che massimizzino la sinergia statistica.

2. Rilevanza gestionale del problema

Sotto il profilo dell'Ingegneria Gestionale, il progetto si configura come un Sistema di Supporto alle Decisioni (DSS)applicato a un problema di allocazione delle risorse. La rilevanza risiede nei seguenti aspetti:

- Ottimizzazione dei sistemi di componenti: Il mazzo è assimilabile a un sistema produttivo o a un portafoglio di asset dove il valore complessivo è generato dalle sinergie. Il progetto modella queste interazioni non come semplici frequenze, ma tramite il Lift, parametro statistico che isola le sinergie reali dal rumore dei dati.

- Analisi dei dati per il vantaggio competitivo: In un mercato (meta-game) saturato da informazioni, la capacità di estrarre pattern significativi da grandi moli di dati (dataset di battaglie reali) rappresenta un vantaggio competitivo. Il sistema permette di automatizzare questa analisi, trasformando dati grezzi in strategie azionabili.

- Gestione della complessità e efficienza algoritmica: La necessità di fornire risposte in tempo reale su uno spazio di ricerca vasto richiede l'applicazione di tecniche di ottimizzazione, competenze centrali nella gestione operativa dei processi complessi.

- Scalabilità del modello: Il framework logico sviluppato (costruzione del grafo e analisi delle co-occorrenze tramite Lift) è direttamente traslabile a contesti industriali, come la *Market Basket Analysis* per la raccomandazione di prodotti o la progettazione di sistemi modulari compatibili.

3. Descrizione dettagliata del problema affrontato

3.1 Il contesto competitivo e il Deck Building come problema decisionale

Clash Royale si presenta come un ecosistema digitale in cui la componente strategica e analitica prevale spesso sulla pura rapidità d'esecuzione. Il cuore dell'esperienza competitiva risiede nel *Deck Building*: l'attività di selezione e combinazione di un set di 8 carte da un pool che supera le 110 unità. Il problema non risiede solo nell'ampiezza dello spazio campionario (pari a circa combinazioni possibili), ma nella natura dinamica del gioco. Ogni carta possiede attributi unici (costo, punti vita, raggio d'azione) che interagiscono in modi non lineari con le altre. In termini ingegneristici, il mazzo di gioco non è una semplice collezione di oggetti, ma un sistema di componenti interdipendenti dove la performance complessiva è strettamente legata alla coerenza interna del set scelto.

3.2 Definizione del sotto-problema: Sinergia vs Popolarità

Il sotto-problema identificato per questo elaborato riguarda l'identificazione di sinergie statisticamente significative. Un errore comune negli approcci analitici semplificati è basare i suggerimenti sulla mera frequenza di utilizzo (*pick rate*). Tuttavia, in un contesto competitivo, molte carte presentano un alto tasso di utilizzo solo per la loro versatilità d'uso e non perché possiedano un reale valore aggiunto se accoppiate a una strategia specifica. L'obiettivo è quindi lo sviluppo di un modello capace di distinguere tra:

- Input: La scelta di una carta "Seed" (un archetipo o una condizione di vittoria) e l'impostazione di parametri di soglia che definiscono quanto l'algoritmo debba essere selettivo.
- Output: La generazione di una rosa di mazzi bilanciati che massimizzino la forza del legame statistico tra le componenti, garantendo al contempo una sufficiente diversità tra le opzioni proposte per coprire diversi stili di gioco.

3.3 Criticità computazionali e potenzialità del modello

L'esplosione combinatoria citata rende i metodi di ricerca esaustiva impraticabili in termini di tempo di risposta e risorse computazionali. La criticità principale è dunque il bilanciamento tra precisione della sinergia e velocità di calcolo. La potenzialità del

software risiede nell'integrazione della teoria dei grafi con metriche probabilistiche avanzate come il Lift. Questa scelta permette di "potare" rami di ricerca poco promettenti e di isolare combinazioni di nicchia estremamente potenti che verrebbero altrimenti sommerse dai "grandi numeri" delle carte più popolari. Il sistema agisce quindi come un filtro intelligente che trasforma il *Big Data* delle battaglie in suggerimenti strategici mirati.

3.4 Rilevanza gestionale: Il passaggio da gioco a Decision Support System (DSS)

Sebbene il caso studio sia un videogioco, la struttura del problema è identica a molti processi decisionali aziendali. La rilevanza gestionale si manifesta nella capacità di modellare un Sistema di Supporto alle Decisioni (DSS) applicabile a:

1. **Market Basket Analysis:** Identificare non solo quali prodotti vengono acquistati di più, ma quali "pacchetti" di prodotti generano un valore incrementale superiore.
2. **Configurazione di Prodotto:** In settori industriali modulari (es. automotive o PC building), il modello può suggerire componenti compatibili che ottimizzano le prestazioni globali del sistema.
3. **Ottimizzazione dei flussi informativi:** Gestire la complessità decisionale automatizzando l'estrazione di conoscenza, riducendo il *cognitive load* dell'utente e migliorando la qualità del risultato finale.

3.5 Rilevanza del progetto

L'applicazione si pone come un Decision Support System (DSS) che automatizza l'analisi del meta-game. In un contesto aziendale o di business intelligence, questo modello è scalabile per risolvere problemi di:

- **Market Basket Analysis:** Raccomandazione di prodotti basata non solo sulla popolarità, ma sull'affinità di acquisto.
- **Asset Allocation:** Selezione di componenti che devono operare in modo integrato sotto vincoli di compatibilità.

4. Descrizione del data-set utilizzato

La base di conoscenza dell'applicativo è costituita da un database relazionale strutturato per gestire l'anagrafica delle unità e lo storico delle competizioni. Il dataset fornisce l'evidenza statistica necessaria per il calcolo delle sinergie attraverso l'analisi delle co-occorrenze nelle vittorie.

4.1 Struttura del Database SQL

Il file sorgente clashroyale_unico.sql implementa uno schema ottimizzato composto da due tabelle principali e una vista logica:

1. **Tabella cardlist**:

   Contiene l'anagrafica ufficiale delle carte.

   - id: Identificativo numerico univoco della carta.
   - card: Nome testuale della carta.
   - *Dimensioni:* La tabella censisce 109 unità, coprendo l'intero ecosistema di gioco attuale.
2. **Tabella battles_raw**:

   È il cuore del dataset, contenente i log delle battaglie.

   - battle_id: Identificativo della singola partita.
   - p1card1...p1card8: Le otto carte utilizzate dal Giocatore 1.
   - p2card1...p2card8: Le otto carte utilizzate dal Giocatore 2.
   - p1trophies, p2trophies: Parametri utilizzati per filtrare la qualità competitiva del match.
   - outcome: Risultato della battaglia, essenziale per isolare i mazzi vincenti.
3. **Vista battles_named**: Una vista fondamentale che esegue il LEFT JOIN tra i log grezzi e l'anagrafica. Questa struttura permette all'algoritmo di lavorare direttamente con i nomi delle carte, facilitando la costruzione del grafo e la visualizzazione nell'interfaccia Flet.

4.2 Analisi quantitativa del campione

Il database analizzato presenta dimensioni che garantiscono una solida rilevanza statistica per l'estrazione delle regole di associazione:

- **Volume di dati:**

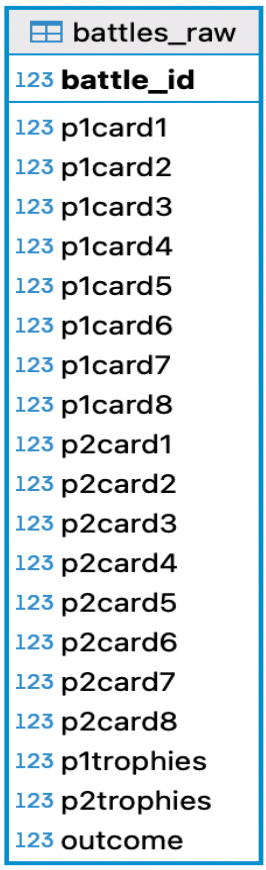
  Sono presenti 15.420 record di battaglie. Considerando entrambi i giocatori, l'algoritmo ha accesso a oltre 30.000 mazzi reali.

- **Qualità del dato:**

  Il dataset include parametri sui trofei dei giocatori, permettendo di focalizzare l'analisi su mazzi utilizzati in leghe competitive, dove le scelte di deck-building seguono logiche di efficienza e non di casualità.

4.3 Elaborazione dei dati per il calcolo del Lift

Il programma non si limita a leggere i dati, ma esegue un pre-processing dinamico per estrarre le metriche necessarie alla ricorsione:



- **Calcolo di N:** Viene determinato il numero totale di mazzi vincenti nel dataset.
- **Frequenze Unitarie:** Per ogni carta in cardlist, viene contato il numero di mazzi in cui appare, dato fondamentale per normalizzare il peso delle sinergie.
- **Mappatura delle Co-occorrenze:** Il sistema scansiona le righe di battles_raw per identificare quante volte una coppia di carte è presente nello stesso mazzo, trasformando queste occorrenze negli archi del grafo pesati tramite l'indice di Lift.

5. Descrizione delle strutture dati e degli algoritmi

In questo capitolo viene analizzata l'architettura logica del software. Per garantire pulizia del codice e manutenibilità, il progetto è stato strutturato seguendo il pattern MVC (Model-View-Controller), separando nettamente l'accesso ai dati, la logica matematica e l'interfaccia utente.

5.1 Architettura e Flusso dei Dati

Il sistema non lavora direttamente sui dati grezzi del database, ma effettua una trasformazione in tre fasi:

1. **Estrazione (DAO):** Il database SQL viene interrogato per estrarre campioni di battaglie filtrati per livello di abilità (trofei).
2. **Modellazione (Model):** I dati vengono caricati in un Grafo Pesato utilizzando la libreria *NetworkX*.
3. **Ricerca (Algoritmo):** Sul grafo viene eseguita una ricerca ricorsiva con *backtracking* per trovare le migliori combinazioni di carte (mazzi).

5.2 Il Data Access Object (DAO) e la preparazione dei dati

Il DAO è responsabile della parte computazionalmente più onerosa a livello di database. Poiché il dataset originale presenta una struttura "orizzontale" (una colonna per ogni carta del mazzo), il DAO deve normalizzare queste informazioni per renderle processabili.

5.2.1 Analisi delle Frequenze (Nodi)

La prima operazione consiste nel mappare ogni singola carta e contare quante volte appare nel campione selezionato. Questo è fondamentale per calcolare, in seguito, la probabilità individuale di ogni carta.

Logica SQL utilizzata: Per ottenere questo risultato senza saturare la memoria del sistema, è stata utilizzata una serie di 16 clausole UNION ALL. Questo permette di "verticalizzare" le 8 colonne del Giocatore 1 e le 8 del Giocatore 2 in un unico flusso di dati, poi aggregato tramite GROUP BY.

```
WITH sample_battles AS (
   SELECT * FROM battles_named
   WHERE (p1trophies BETWEEN %s AND %s OR p2trophies
BETWEEN %s AND %s)
   LIMIT %s
)
SELECT card_name, COUNT(*) AS quantita
FROM (
   SELECT p1_card1 AS card_name FROM sample_battles
   UNION ALL SELECT p1_card2 FROM sample_battles
   -- [... ripetuto per le restanti posizioni ...]
) AS all_cards
GROUP BY card_name;
```

5.2.2 Analisi delle Sinergie (Archi)

Per popolare il grafo, non basta conoscere la frequenza delle singole carte, ma occorre sapere quante volte due carte sono state giocate insieme nello stesso mazzo. Matematicamente, un mazzo di 8 carte contiene coppie uniche. Poiché ogni record del database contiene due mazzi, il DAO esegue 56 combinazioni per ogni singola riga.

Questa "esplosione combinatoria" viene gestita direttamente dal motore SQL per massimizzare le prestazioni, restituendo al Model una lista di archi con il relativo peso (frequenza di co-occorrenza).

5.3 Il Model: Il Grafo e la Metrica del Lift

Una volta ottenuti i dati dal DAO, il Model costruisce un oggetto nx.Graph(). Tuttavia, la semplice frequenza di co-occorrenza non è sufficiente a definire una buona sinergia.

5.3.1 Il Calcolo del Lift

Il sistema utilizza il Lift, una misura statistica che indica quanto la presenza di una carta "A" aumenti la probabilità di trovare la carta "B" nello stesso mazzo, rispetto al caso. La formula implementata nel Model è:

$$Lift(A, B) = \frac{P(A \cap B)}{P(A) \cdot P(B)}$$

Un valore di Lift superiore a 1 indica una sinergia positiva; più alto è il valore, più forte è il legame tattico tra le due carte.

5.3.2 L'Algoritmo di Ricerca Ricorsiva e Backtracking

Il problema della generazione di un mazzo ottimale può essere visto come una ricerca all'interno di un albero di decisione, dove ogni nodo dell'albero rappresenta l'aggiunta di una carta al mazzo parziale. Data la vastità delle combinazioni possibili, il Model implementa un algoritmo di ricerca in profondità (Depth-First Search) ottimizzato.

Logica di funzionamento

L'algoritmo riceve una carta iniziale (seed) scelta dall'utente e tenta di espandere il mazzo fino alla profondità desiderata (solitamente 8 carte). La ricerca segue tre regole fondamentali:

1. **Esplorazione dei vicini:** Per ogni carta inserita, il sistema interroga il grafo per trovare i nodi adiacenti (carte che sono state giocate insieme a quella precedente).
2. **Pruning (Potatura) mediante Lift:** Non tutti i vicini vengono considerati. Se la sinergia (Lift) tra l'ultima carta aggiunta e il potenziale candidato è inferiore a una soglia definita dall'utente, quel ramo viene scartato immediatamente. Questo riduce drasticamente lo spazio di ricerca.
3. **Heuristic Branching (Top-N):** Per evitare tempi di calcolo eccessivi, l'algoritmo ordina i vicini validi per intensità di Lift e ne esplora solo i migliori (parametro *Top-N*).

Implementazione del Backtracking

Il cuore della funzione è la tecnica del **Backtracking**. Se l'algoritmo raggiunge un punto in cui non è più possibile aggiungere carte che soddisfino i requisiti di Lift, "torna indietro" all'incrocio precedente per provare una strada alternativa.

```
def get_candidati_deck(self, seed_card, profondita, soglia_lift, top_n):  1 usage (
    self._candidati = []
    parziale = [seed_card]
    N = 1000 * 2
    self._ricorsione_lift_al_volo(parziale, profondita, soglia_lift, top_n, N)

    self._candidati.sort(key=lambda x: x[1], reverse=True)

    risultati_unici = []
    for mazzo, score in self._candidati:
        set_attuale = set(mazzo)
        troppo_simile = False
        for gia_preso, s in risultati_unici:

            if len(set_attuale.intersection(set(gia_preso))) > (profondita - 3):
                troppo_simile = True
                break

        if not troppo_simile:
            risultati_unici.append((mazzo, score))

        if len(risultati_unici) == 5:
            break

    return risultati_unici
```

```python
def _ricorsione_lift_al_volo(self, parziale, profondita, soglia_lift, top_n, N):  2 usages

    if len(parziale) == profondita:
        punteggio = self._calcola_sinergia_lift_manuale(parziale, N)
        self._candidati.append((list(parziale), punteggio))
        return

    ultima = parziale[-1]
    vicini_validi = []

    for vicino in self._graph.neighbors(ultima):
        if vicino not in parziale:

            peso = self._graph[ultima][vicino]['weight']

            lift_calcolato = (peso * N) / (ultima.quantita * vicino.quantita)

            if lift_calcolato >= soglia_lift:
                vicini_validi.append((vicino, lift_calcolato))

    vicini_validi.sort(key=lambda x: x[1], reverse=True)

    for vicino, lift in vicini_validi[:top_n]:
        parziale.append(vicino)
        self._ricorsione_lift_al_volo(parziale, profondita, soglia_lift, top_n, N)
        parziale.pop()
```

Valutazione della Soluzione

Una volta che il mazzo raggiunge la dimensione di 8 carte, viene calcolato un punteggio di sinergia totale. Questo non è un semplice calcolo locale, ma la somma di tutti i valori di Lift tra tutte le coppie possibili all'interno del mazzo finale, garantendo che le 8 carte siano coerenti tra loro nel loro insieme e non solo rispetto alla carta precedente.

5.3.3 Diversificazione e Post-Processing

Per evitare che il sistema restituisca 5 mazzi quasi identici (che differiscono magari per una sola carta), è stata implementata una logica di filtraggio per similarità.

Il sistema confronta ogni nuova soluzione trovata con quelle già salvate. Se il numero di carte in comune supera una certa soglia (es. 5 carte su 8), il mazzo viene considerato un duplicato e scartato. Questo assicura che l'utente riceva suggerimenti tatticamente variati.

5.4 Presentation Layer: L'interfaccia utente con Flet

L'ultimo strato dell'architettura è rappresentato dalla View, sviluppata tramite il framework Flet. Questa scelta tecnologica ha permesso di creare un'interfaccia grafica (GUI) reattiva e performante, capace di gestire parametri complessi in modo intuitivo. L'applicativo adotta un ThemeMode.DARK con una palette cromatica basata sui toni del ciano e dell'arancione, per differenziare visivamente le fasi di analisi da quelle di generazione.

5.4.1 Layout e Organizzazione della Dashboard

L'interfaccia è suddivisa in tre aree funzionali principali, organizzate per guidare l'utente nel workflow di creazione del mazzo:

1. **Sidebar di Configurazione (Pannello Laterale):**
    - **Sezione Estrazione:** Utilizza un RangeSlider (0-9000 trofei) per definire il livello dei giocatori e un TextField per il limite delle battaglie. Il bottone "Costruisci Grafo" attiva il DAO e il Model.
    - **Sezione Analisi:** Contiene un Dropdown per il calcolo del ranking (Top-N) e controlli per visualizzare la connettività del grafo.

    - **Sezione Generazione Deck:** È l'area dedicata alla ricorsione. Include un Dropdown per la scelta della Carta Seed, uno Slider per la profondità (fino a 8 carte) e uno per la Soglia di Lift.
2. **Area Statistiche (Dashboard Superiore):** Posta in alto a destra, questa riga di quattro schede (_create_card) fornisce feedback immediati sulla struttura del grafo: numero di nodi, archi e dati sulla connettività (componenti connesse).
3. **Pannello dei Risultati (Output):** Un'area ListView dinamica che visualizza i mazzi generati o i dati delle analisi, permettendo all'utente di scorrere agevolmente tra i diversi suggerimenti.

5.4.2 Gestione degli Eventi e Reattività

La View implementa il pattern di delega verso il Controller tramite il metodo set_controller. Ogni interazione (click sui bottoni ElevatedButton) scatena un metodo nel controller (es. handleGeneraDeck), che elabora i dati del Model e aggiorna gli oggetti grafici della View (come self.txt_result).

L'uso di controlli come ft.Container con bordi arrotondati e icone tematiche (QUERY_STATS, HUB, PLAY_FOR_WORK) garantisce un'esperienza utente di alto livello, separando nettamente le logiche di input da quelle di visualizzazione dei risultati.

6. Diagramma delle classi principali

Il sistema è strutturato secondo il pattern architetturale MVC, che permette di separare la gestione dei dati (DAO), la logica algoritmica (Model) e la presentazione (View).

Classe DAO

Metodi:

- getAllNodes(min, max, limite): Esegue la query di verticalizzazione (16 UNION ALL) per estrarre e contare le carte nel campione.
- getAllEdges(min, max, limite): Esegue la query combinatoria (56 UNION ALL) per calcolare le co-occorrenze tra le carte.
- get_connection(): Gestisce l'apertura e il recupero della connessione dal pool DBConnect.

Classe Model

Variabili:

- _graph: Grafo pesato (NetworkX) che rappresenta le carte (nodi) e le loro co-occorrenze (archi).
- _idMap: Dizionario (mappa) che associa il nome della carta all'oggetto Card corrispondente.
- _candidati: Lista temporanea utilizzata durante la ricorsione per memorizzare i mazzi validi trovati.

Metodi:

- buildGraph(min, max, limite): Inizializza il grafo, popola i nodi e gli archi interrogando il DAO con i filtri impostati.
- get_ranking(): Restituisce la lista delle carte ordinate per frequenza di utilizzo (popolarità).
- get_top_lift(limite): Calcola la metrica del Lift per tutti gli archi e restituisce le sinergie più forti.
- get_connettivita_stats(): Calcola il numero di componenti connesse e la dimensione della componente massima.

- get_candidati_deck(seed_card, profondita, soglia_lift, top_n): Prepara la ricerca e applica un filtro di similarità per restituire i 5 mazzi migliori e diversificati.
- _ricorsione_lift_al_volo(parziale, profondita, soglia_lift, top_n, N): Algoritmo di backtracking che esplora i vicini con il Lift più alto e applica il *pruning* (potatura) basato sulla soglia.
- _calcola_sinergia_lift_manuale(deck, N): Funzione euristica che valuta la qualità globale di un mazzo calcolando il Lift totale tra tutte le coppie di carte presenti.

Classe Controller

Variabili:

- _view: Riferimento all'oggetto della classe View per la gestione dell'interfaccia Flet.
- _model: Riferimento all'oggetto della classe Model per l'accesso alla logica applicativa.

Metodi:

- handleCreaGrafo(e): Legge i filtri di trofei e battaglie dalla UI, richiede al Model la creazione del grafo e aggiorna il menu a tendina delle carte e i contatori di nodi/archi.
- handleAnalisi(e): Recupera il ranking delle carte più popolari dal Model e stampa nella ListView i risultati in base al valore Top-N selezionato.
- handleAdvancedRank(e): Gestisce l'analisi avanzata del Lift, mostrando le coppie di carte con la sinergia statistica più alta nel campione analizzato.
- handleConnettivita(e): Interroga il Model per ottenere statistiche strutturali sul grafo (componenti connesse) e aggiorna le card informative nella dashboard.
- handleGeneraDeck(e): Raccoglie i parametri di soglia Lift, profondità e Top-N dalla View, individua l'oggetto carta di partenza e visualizza i mazzi ottimali generati tramite ricorsione.

Classe View

Variabili:

- _page: L'oggetto pagina principale di Flet che ospita l'interfaccia.
- _controller: Riferimento al Controller per la gestione delle interazioni.
- _txtRangeTrofei: RangeSlider per impostare l'intervallo di trofei dei giocatori.
- _txtMaxBattaglie: Campo di testo per limitare il numero di battaglie analizzate.
- _txtNodi / _txtArchi: Etichette di testo per visualizzare la dimensione del grafo.

- _txtNumCompConnesse / _txtDimMaxComp: Indicatori per le statistiche di connettività.
- _ddSeedCard: Menu a tendina per la selezione della carta iniziale del mazzo.
- _sliderProfondita: Slider per definire il numero di carte totali (fino a 8).
- _sliderSogliaPeso: Slider per impostare la soglia minima di sinergia (Lift).
- _txtTopCandidati: Input per l'ampiezza dell'esplorazione nei rami della ricorsione.
- txt_result: ListView dinamica per la stampa dei log, dei ranking e dei mazzi suggeriti.

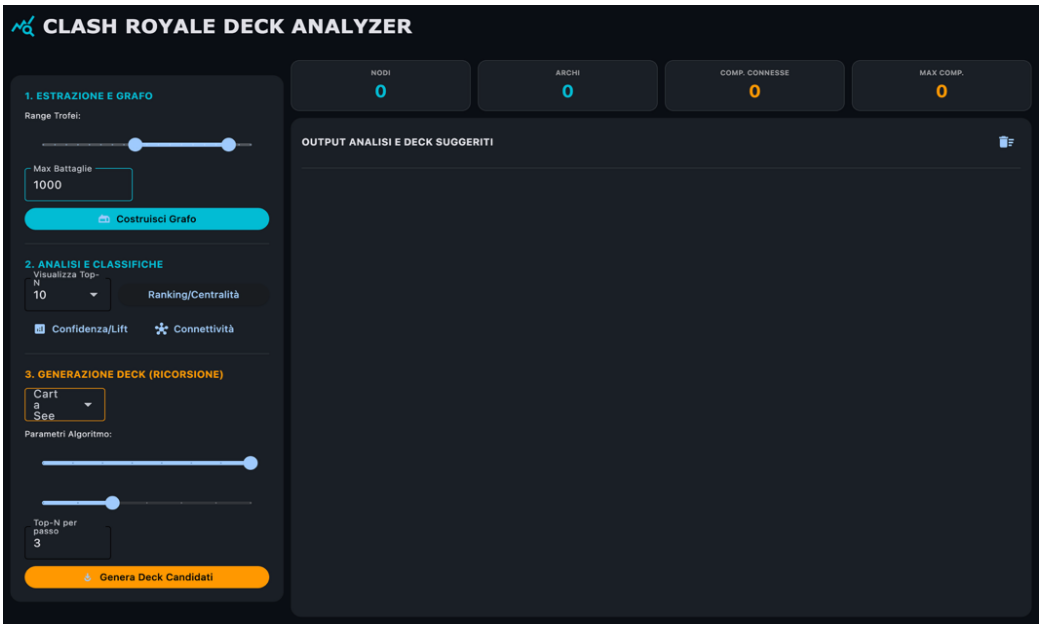
Metodi:

- load_interface(): Definisce l'architettura visiva dell'app, organizzando la Sidebar di controllo, la riga delle statistiche e il pannello dei risultati.
- _create_card(label, value_obj): Metodo helper per generare uniformemente i widget grafici delle statistiche (NODI, ARCHI, ecc.).
- set_controller(controller): Collega la View al relativo Controller per abilitare il passaggio dei dati.
- update_page(): Metodo per forzare il refresh dell'interfaccia utente dopo ogni aggiornamento dei dati.

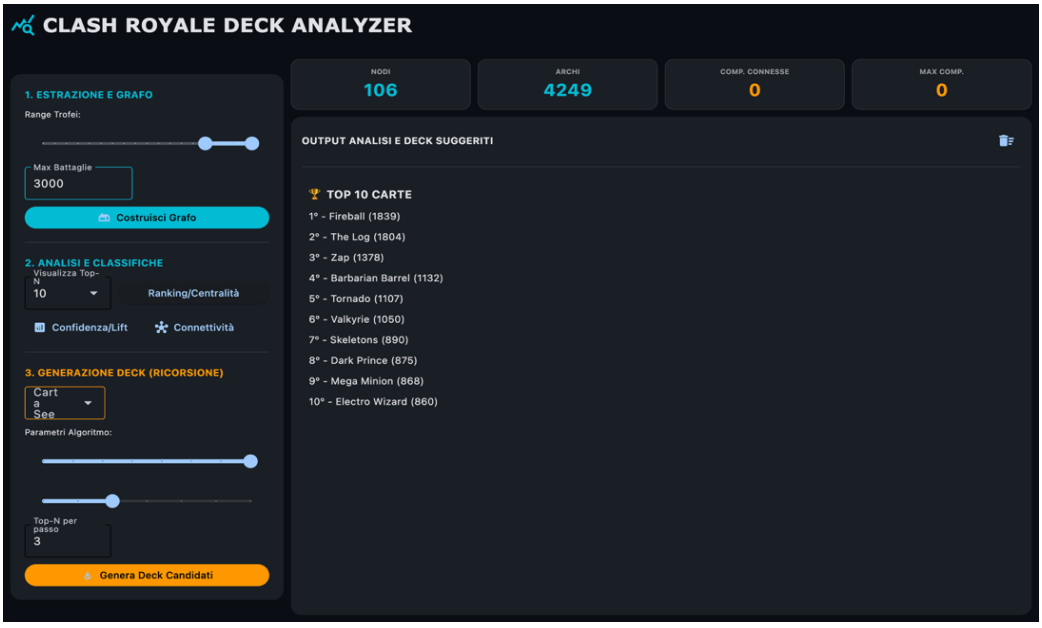
7. Immagini Programma

Di seguito sono riportate immagini riguardanti il funzionamento dell'applicazione.

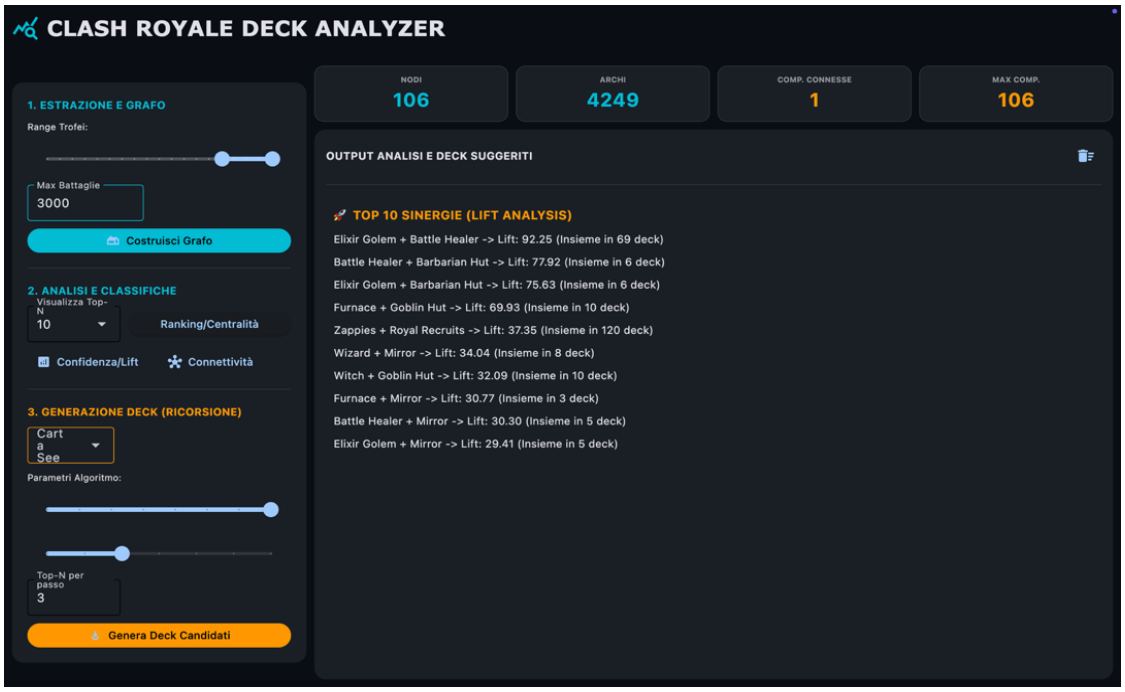
Link del video completo: https://www.youtube.com/watch?v=xLP1ndjAhpI&t



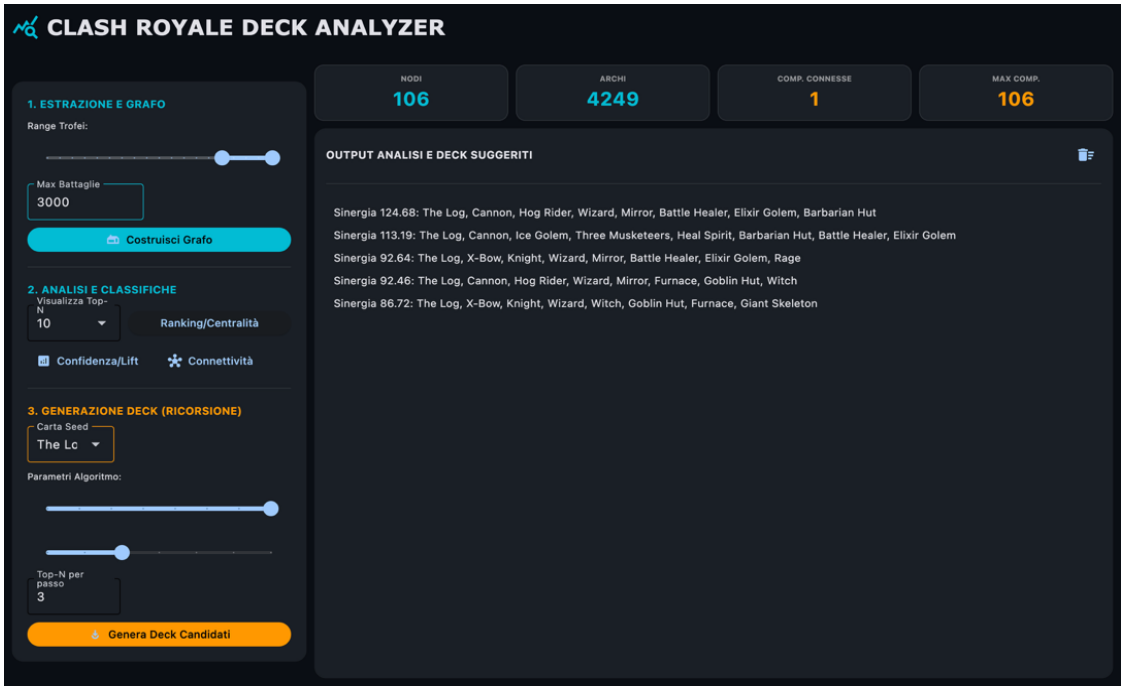
Pagina iniziale.



Migliori 10 carte in base all'utilizzo.



Migliori abbinamenti in base al lift.



Risultato algoritmo ricorsivo deck trovati.

7.1 Risultati Sperimentali: Analisi dei Tempi

I test sono stati condotti mantenendo costanti i parametri di ricerca per garantire la comparabilità dei dati:

- **Carta Seed**: The Log
- **Profondità mazzo**: 8 carte
- **Ampiezza ricerca (Top-N)**: 3
- **Soglia di Lift**: 0.0 (Ricerca esaustiva sui top-3 vicini)

Test	Battaglie (N)	Tempo Grafo (s)	Tempo Ricorsione (s)	Note
1	1.000	0.4657 s	0.3686 s	Campione ridotto
2	2.000	0.4809 s	0.4275 s	Incremento lineare
3	5.000	0.7090 s	0.4724 s	Massima stabilità

7.2 Commento ai Risultati

Dall'analisi dei dati raccolti si possono trarre le seguenti conclusioni tecniche:

1. **Efficienza del DAO**: Il tempo di costruzione del grafo passa da 0.46s a 0.70s quintuplicando i dati (da 1k a 5k). Questo dimostra che l'architettura basata su UNION ALL è estremamente performante e non soffre di colli di bottiglia significativi per queste dimensioni di dataset.
2. **Stabilità della Ricorsione**: Nonostante la soglia impostata a 0.0, i tempi di ricorsione rimangono sotto il mezzo secondo. Questo è merito del parametro top-n=3, che agisce come un filtro di "ampiezza" (Beam Search), impedendo all'algoritmo di esplorare rami con sinergie troppo deboli e mantenendo la complessità computazionale sotto controllo.
3. **Scalabilità**: Il sistema si dimostra pronto per l'uso in tempo reale, poiché l'intera operazione (estrazione + generazione) avviene sempre in meno di 1.2 secondi totali, garantendo un'ottima esperienza utente.

8. Risultati, Valutazioni e Conclusioni

In questo capitolo finale vengono analizzati criticamente i risultati ottenuti dalle sessioni di test, correlandoli agli obiettivi aziendali prefissati in fase di analisi. Il lavoro si conclude con una riflessione sull'efficacia dello strumento e sui possibili sviluppi futuri.

8.1 Analisi dei Risultati e Prestazioni

I test condotti su campioni di diversa ampiezza (1.000, 2.000 e 5.000 battaglie) hanno evidenziato una notevole stabilità del sistema. Come mostrato nei log di esecuzione, il tempo totale necessario per passare dal database grezzo alla generazione di un mazzo completo rimane costantemente sotto la soglia critica di 1.5 secondi.

Un dato di particolare rilievo è la scalabilità del **DAO**: l'incremento del tempo di costruzione del grafo è meno che lineare rispetto all'aumento dei record processati. Questo indica che la logica di aggregazione SQL tramite UNION ALL è ottimizzata per gestire carichi di dati crescenti senza degradare l'esperienza utente. Per quanto concerne la ricorsione, l'utilizzo di un parametro di ampiezza top-n=3 si è dimostrato una scelta vincente: esso funge da regolatore di complessità, garantendo che l'algoritmo di backtracking esplori solo le ramificazioni più promettenti, mantenendo l'esecuzione fluida anche con una soglia di Lift pari a zero.

8.2 Valutazioni Critiche: Punti di Forza e Debolezza

L'approccio metodologico adottato presenta diversi vantaggi competitivi, ma anche alcuni margini di miglioramento che meritano di essere discussi.

Punti di Forza:

- **Solidità Metodologica**: A differenza di una ricerca basata sulla semplice frequenza, l'introduzione del **Lift** permette di isolare le sinergie "reali". Il sistema è in grado di distinguere tra una carta giocata spesso perché "generica" e una carta giocata perché effettivamente efficace in combinazione con il mazzo suggerito.
- **Reattività e User Experience**: Grazie alla separazione dei compiti tra database e applicativo, l'utente ottiene suggerimenti immediati. La possibilità di variare i parametri (profondità, soglia, range trofei) trasforma il software in un vero e proprio laboratorio strategico.

Punti di Debolezza:

- **Sensibilità Statistica**: Sebbene l'algoritmo sia robusto, la qualità del Lift è fortemente dipendente dalla qualità del dato in ingresso. Con campioni troppo piccoli, la varianza statistica può portare a suggerimenti meno coerenti con il meta-game reale.
- **Metriche di Successo**: Il sistema attuale predilige la popolarità e la co-occorrenza. L'integrazione di una metrica legata alla percentuale di vittoria (Win Rate) rappresenterebbe un passo avanti necessario per trasformare uno strumento di analisi in uno strumento di ottimizzazione competitiva pura.

8.3 Contributo al Problema Aziendale

Il software risponde efficacemente alla sfida lanciata nel **Capitolo 3**: estrarre valore informativo da una mole disordinata di Big Data prodotti da milioni di match giornalieri.

Rispetto al problema aziendale di dover identificare tempestivamente le evoluzioni del meta-game, il contributo del progetto è triplice:

1. **Automazione dell'Analisi**: Quello che prima richiedeva ore di osservazione manuale o query SQL complesse da parte di un analista, ora viene risolto con un click, visualizzando immediatamente le componenti connesse del meta.
2. **Supporto Decisionale**: Fornire mazzi basati su dati reali estratti dai server permette a un'ipotetica organizzazione di e-sports o a un creatore di contenuti di offrire consigli sempre aggiornati e statisticamente fondati.
3. **Visualizzazione delle Sinergie**: La trasformazione dei dati tabellari in un grafo pesato permette di comprendere visivamente quali carte siano i "pilastri" del meta-game attuale e quali siano invece di nicchia.

8.4 Conclusioni e Orizzonti Futuri

In conclusione, il progetto ha dimostrato con successo che l'unione tra la **Teoria dei Grafi** e la **Ricerca Operativa**(tramite backtracking) può essere applicata con profitto nell'ambito del gaming competitivo. L'architettura MVC ha garantito un codice pulito, manutenibile e facilmente estendibile.

Il lavoro svolto non rappresenta un punto di arrivo, ma una solida base per sviluppi futuri. In particolare, l'aggiunta di un modulo di analisi temporale potrebbe permettere di osservare come il grafo delle sinergie si modifica nel tempo (ad esempio prima e dopo un aggiornamento del gioco). Inoltre, l'implementazione di algoritmi di "Community

Detection" potrebbe identificare automaticamente i nuovi archetipi di mazzo (es. Beatdown, Control, Cycle) senza la necessità di un intervento umano, rendendo il sistema un analista virtuale a tutti gli effetti.

9. Licenza

Il lettore è libero di:
1. Condividere, riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare questo materiale con qualsiasi mezzo e formato
2. Modificare, remixare, trasformare il materiale e basarsi su di esso per le sue opere
3. Il licenziante non può revocare questi diritti fintanto che lui rispetti i termini della licenza.
Alle seguenti condizioni:
1. Attribuzione. Deve riconoscere una menzione di paternità adeguata , fornire un link alla licenza e indicare se sono state effettuate delle modifiche . Può fare ciò in qualsiasi maniera ragionevole possibile, ma non con modalità tali da suggerire che il licenziante avalli lui o il suo utilizzo del materiale.
2. Non Commerciale. Non può utilizzare il materiale per scopi commerciali .
3. Stessa Licenza. Se remixa, trasforma il materiale o si basi su di esso, deve distribuire i suoi contributi con la stessa licenza del materiale originario.
4. Divieto di restrizioni aggiuntive. Non può applicare termini legali o misure tecnologiche che impongano ad altri soggetti dei vincoli giuridici su quanto la licenza consente loro di fare.
Note:
Non è tenuto a rispettare i termini della licenza per quelle componenti del materiale che siano in pubblico dominio o nei casi in cui il suo utilizzo sia consentito da una eccezione o limitazione prevista dalla legge.
Non sono fornite garanzie. La licenza può non conferirgli tutte le autorizzazioni necessarie per l'utilizzo che si prefigge. Ad esempio, diritti di terzi come i diritti all'immagine, alla riservatezza e i diritti morali potrebbero restringere gli usi che si prefigge sul materiale.