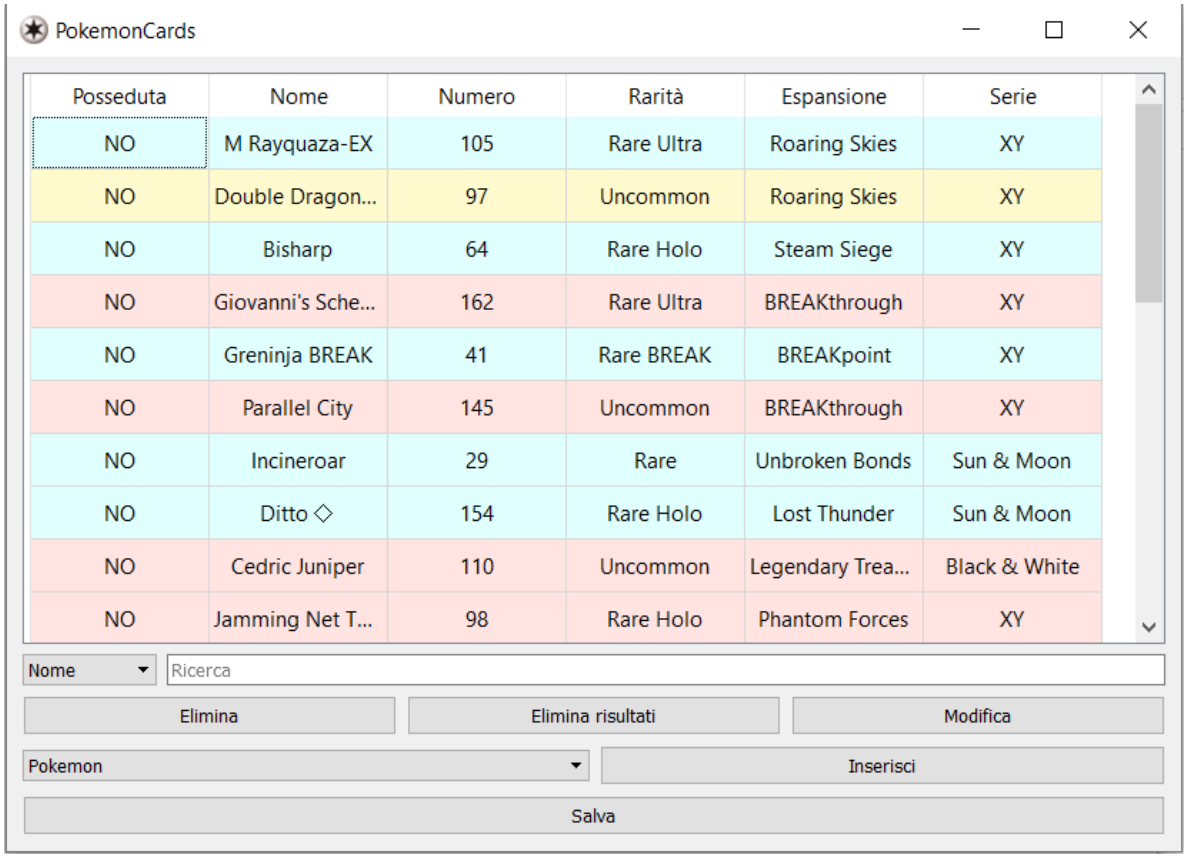
Programmazione ad oggetti

Relazione di Fistarollo Fiorenza Francesca

Matricola 1126101

AA 2018/2019

Relazione del progetto "PokemonCards"



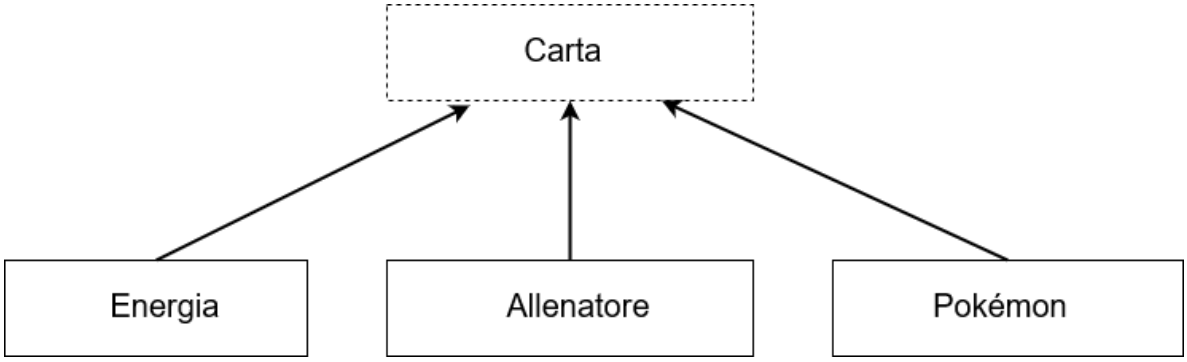
Scopo

Il progetto vuole andare a creare un programma con il quale l'utente può gestire la propria collezione di carte GCC Pokémon man mano che escono nuove espansioni.

Le carte GCC Pokémon si dividono secondo lo standard in tre categorie: Pokémon, che servono per attaccare gli avversari, Energie, il cui scopo è caricare le mosse dei Pokémon e Allenatore, che sono sostanzialmente carte di supporto.

PokémonCards permette all'utente di aprire e modificare carte già esistenti o inserirne di nuove e salvarle su file. Inoltre gli permette di indicare per ogni carta se è in suo possesso oppure no in modo da tenere traccia facilmente della sua collezione.

Gerarchie tipi



La classe base della gerarchia è la classe astratta polimorfa Carta, che rappresenta una generica carta che fa parte della collezione. Contiene nella parte privata solo le informazioni comuni a tutte e tre le tipologie di carte: nome, espansione, numero, rarità, serie(stringhe), e un boolean posseduta che indica se la carta è in possesso dell'utente oppure no.

Le tre sottoclassi concrete polimorfe derivano pubblicamente da Carta e ne implementano il metodo virtuale puro; esse sono:

- **Allenatore**: rappresenta una carta Allenatore, è dotata di due campi di tipo stringa privati che contengono il testo e il tipo dell'allenatore in questione;
- **Energia**: rappresenta una carta Energia, è dotata di un campo di tipo stringa privato che contiene il testo e un campi privato di tipo bool che indica se si tratta di un'Energia Speciale oppure no;
- **Pokémon**: rappresenta una carta Pokémon, la sua parte privata è dotata di 3 campi di tipo unsigned int che contengono il numero del Pokédex, i punti vita e il costo di ritirata del Pokémon in questione, 5 di tipo stringa che contengono il suo sottotipo, il nome del Pokémon da cui è evoluto (in caso il Pokémon corrente sia un'evoluzione) e il nome, il testo e il tipo della sua abilità (se ne ha una), e 2 vettori di stringhe che rappresentano i suoi tipi(ogni Pokémon può averne al massimo 2) e le sue regole di utilizzo. La classe Pokémon

inoltre ha un vettore privato di Mossa (che contiene le mosse) e due vettori privati di DebRes (che contengono debolezze e resistenze) che sono a loro volta due classi annidate all'interno di Pokemon, dichiarate nella sua parte pubblica. Mossa rappresenta una mossa di un Pokémon e contiene 3 stringhe (nome, danno e testo) e un vettore di stringhe (costo) nella sua parte privata. DebRes è invece una classe che rappresenta sia le debolezze che le resistenze in quanto hanno le stesse caratteristiche: un tipo e un valore (entrambe stringhe private).
Sia Mossa che DebRes sono dichiarate friend di Pokémon, in modo che quest'ultima classe possa avere accesso alla loro parte privata, e hanno i costruttori privati in modo che non si possano istanziare oggetto Mossa o DebRes senza un oggetto Pokémon.

Metodi e chiamate polimorfe

La gerarchia fa uso di metodi polimorfi e possiede un distruttore virtuale pubblico in modo da richiamare sempre il distruttore corretto.

virtual Carta* clone()const=0; è un metodo virtuale puro implementato nelle sottoclassi della gerarchia, in quanto concrete. Restituisce un puntatore alla copia dell'oggetto di invocazione.

virtual nlohmann::json exportToJSON() const; è un metodo virtuale che permette l'esportazione in un file json degli oggetti di ognuna delle sottoclassi con quindi campi diversi in base al loro tipo dinamico al momento dell'invocazione. Restituisce il json creato.

Contenitore

Come contenitore è stato realizzato un array dinamico. è stato preferito quest'ultimo all'implementazione di una lista per motivi di efficienza: si trattava di dover riempire,al momento dell'inserimento dei dati da file precompilati, classi precostruite di cui veniva messo a disposizione un numero e non un iteratore: con una lista si sarebbe dovuto scorrere, per ogni singola carta, dall'inizio del contenitore fino alla carta stessa, avendone solo la posizione, e già una sola espansione può contenere anche 300 carte; al contrario con un array si ha accesso diretto alla posizione. Inoltre risulta raro dover rimuovere carte dalla collezione, operazione che risulta più gravosa a un array che a una lista, e se si aggiungono nuove carte esse vengono inserite in fondo all'array.

DeepPtr

È stato inoltre implementato un template di classe DeepPtr di puntatori polimorfi; quindi il template di classe Container risulta essere effettivamente un Container di DeepPtr di Carte stesse.

Lettura/Scrittura

È stato scelto di rendere compatibile il progetto con l'API disponibile al sito https://pokemontcg.io/ in modo da poter importare facilmente nuove espansioni che escono, per mantenere così il programma sempre al passo con l'uscita di nuove espansioni (ma non viceversa riesportare con tutti i campi ancora pieni perché alcuni campi non utili come ad esempio il disegnatore vengono tralasciati). Di conseguenza dato che l'API restituisce documenti in formato json, la scelta è ricaduta sulla libreria nlohmann/json (https://github.com/nlohmann/json).

Esempio di Pokémon

```
{
  "id": "xy6-76",
```

```
  "name": "M Rayquaza-EX",
  "nationalPokedexNumber": 384,
  "imageUrl": "https://images.pokemontcg.io/xy6/76.png",
  "imageUrlHiRes": "https://images.pokemontcg.io/xy6/76_hires.png",
  "types": [
    "Colorless"
  ],
  "supertype": "Pokémon",
  "subtype": "MEGA",
  "evolvesFrom": "Rayquaza-EX",
  "ancientTrait": {
    "name": "Δ Evolution",
    "text": "You may play this card from your hand to evolve a Pokémon during
your first turn or the turn you play that Pokémon."
  },
  "hp": "220",
  "retreatCost": [
    "Colorless"
  ],
  "convertedRetreatCost": 1,
  "number": "76",
  "artist": "5ban Graphics",
  "rarity": "Rare Holo EX",
  "series": "XY",
  "set": "Roaring Skies",
  "setCode": "xy6",
  "text": [
    "When a Pokémon-EX has been Knocked Out, your opponent takes 2 Prize
cards.",
    "When 1 of your Pokémon becomes a Mega Evolution Pokémon, your turn ends."
  ],
  "attacks": [
    {
      "cost": [
        "Colorless",
        "Colorless",
        "Colorless"
      ],
      "name": "Emerald Break",
      "text": "This attack does 30 damage times the number of your Benched
Pokémon.",
      "damage": "30×",
      "convertedEnergyCost": 3
    }
  ],
  "resistances": [
    {
      "type": "Fighting",
      "value": "-20"
    }
  ],
  "weaknesses": [
    {
      "type": "Lightning",
      "value": "×2"
    }
  ]
}
```

Esempio di Energia


```
{
  "id": "sm11-213",
  "name": "Weakness Guard Energy",
  "imageUrl": "https://images.pokemontcg.io/sm11/213.png",
  "imageUrlHiRes": "https://images.pokemontcg.io/sm11/213_hires.png",
  "supertype": "Energy",
  "subtype": "Special",
  "number": "213",
  "artist": "5ban Graphics",
  "rarity": "Uncommon",
  "series": "Sun & Moon",
  "set": "Unified Minds",
  "setCode": "sm11",
  "text": [
    "This card provides Colorless Energy. The Pokémon this card is attached to
has no Weakness."
  ]
}
```


Esempio di Allenatore


```
{
  "id": "sm35-68",
  "name": "Ultra Ball",
  "imageUrl": "https://images.pokemontcg.io/sm35/68.png",
  "imageUrlHiRes": "https://images.pokemontcg.io/sm35/68_hires.png",
  "supertype": "Trainer",
  "subtype": "Item",
  "number": "68",
  "artist": "Ryo Ueda",
  "rarity": "Uncommon",
  "series": "Sun & Moon",
  "set": "Shining Legends",
  "setCode": "sm35",
  "text": [
    "Discard 2 cards from your hand. If you do, search your deck for a Pokémon,
reveal it, and put it into your hand. Then, shuffle your deck."
  ]
}
```


I metodi usati per la lettura e scrittura su file json sono presenti nella parte pubblica della classe jsonio.

static Container<DeepPtr> importFromJson (std::string); è il metodo per leggere da json; crea il contenitore mazzo e lo riempie facendo push_back con le carte costruite da tre funzioni apposta della classe CardBuilder, il cui compito è appunto creare le carte di diverso tipo in base al supertipo presente nel file json. Il supertipo infatti è quello che divide le carte nelle tre classi Pokémon,Energia e Allenatore.

static void exportToJSON(std::string,Container<DeepPtr>); è il metodo che scrive su un file json: scorre il mazzo e richiama la funzione virtuale che converte una carta di un tipo nel suo corrispondente in json e restituisce il nuovo file risultante.

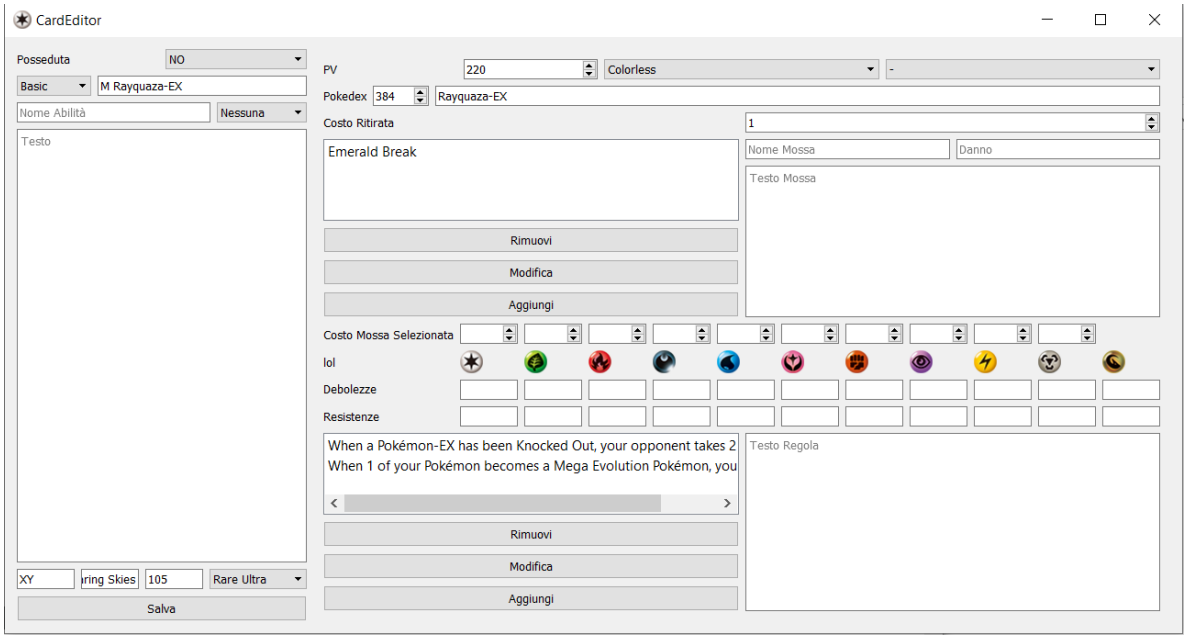
Manuale utente della GUI

All'apertura dell'applicazione viene chiesto di aprire il file json desiderato. Nell'ambito di questo progetto viene fornito un file json carte.json che contiene già 23 carte.

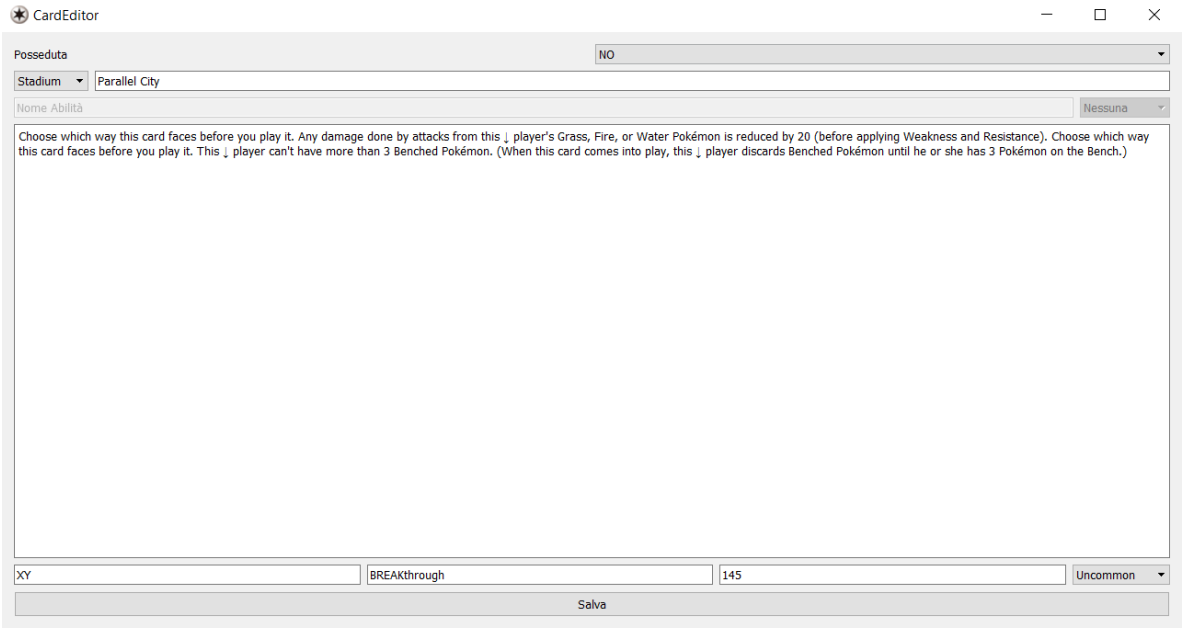
Per quanto riguarda l'interfaccia sono state utilizzate classi derivate dalle librerie Qt; in particolare è stato fatto uso del pattern pattern Model/View, con la classe TableView che deriva da QTableView,utile per dimensionare correttamente la view, e due classi ausiliarie: QFilterProxyModel(derivata da QSortFilterProxyModel), classe proxy che funzionerà da intermediario fra QListModelAdapter e TableView per permettere alla vista di visualizzare solo certi elementi sulla base dell'input di ricerca inserito pur non dovendo effettuare alcuna operazione sul reale modello dei dati sottostante e senza duplicare dati, e QListModelAdapter(derivata da QAbstractTableModel) che serve a implementare tutti i metodi che vanno a influire sui dati (sia sulla visualizzazione che per modifica) per garantire così modularità (in particolare, massima separazione tra parte logica e grafica del codice).

L'interfaccia si presenta come una tabella di carte, in cui vengono mostrati solo i campi comuni a tutte quante. Le carte sono divise per colori in base al loro tipo (rispettivamente azzurro, rosso e giallo per Pokémon, Allenatori, Energie). La MainWindow mette a disposizione una ricerca per filtri, la rimozione di un elemento selezionato o di tutti gli elementi filtrati, la modifica o l'inserimento di una carta e il salvataggio su file.
Per la vista, la modifica e la rimozione si apre una nuova finestra, i cui campi cambiano in base al tipo di carta selezionato. Le classi che si occupano di costruirla sono CardBuilder e PokemonBuilder. Quest'ultima si occupa in particolare della parte specifica dei Pokemon, che è nascosta negli altri casi. Esse si occupano di mostrare, modificare e salvare i campi delle carte, fare controlli (ad esempio che non ci siano più mosse con lo stesso nome) ed eventualmente far apparire messaggi di errore.

Vista di un Pokémon



Vista di un Allenatore



Compilazione

Il progetto necessita di un project file (progetto.pro) per qmake diverso da quello ottenibile tramite l'invocazione di qmake -project, per compilare occorre eseguire i seguenti comandi
qmake
make
nella root del progetto.

Tempistica

Circa 50 ore impiegate, suddivise in :
Progettazione modello: 3 ore
Progettazione GUI: 4 ore
Scrittura modello: 17 ore
Scrittura GUI: 24 ore
Testing e Debug: 5 ore

Ambiente di sviluppo

Sistema Operativo: Microsoft Windows 10 Pro 64
Versione Qt: 5.9.5
Qt Creator: 4.8.1