

Progetto di Programmazione ad Oggetti

A.A 2021-2022

**Fly Management**



**Fabio Pantaleo 2009124**

1. **Introduzione**

La recente pandemia mondiale dovuta alla diffusione del virus SARS-Covid-19 ha portato le agenzie viaggi ad adattarsi alla nuova “routine” e ai nuovi ritmi della società.

I costi di trasporto sono diminuiti esponenzialmente, in particolar modo il costo dei viaggi aerei che ha permesso allo sviluppo e la promozione di tantissime nuove offerte che permettono alla gente di viaggiare a prezzi minimi.

Strutture come gli aeroporti, si appoggiano a **società di terze parti** per la prenotazione e la gestione dei voli.

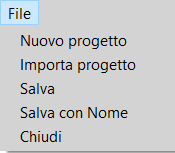
L’idea del progetto è quello di offrire a queste **società** di avere un software apposito per la programmazione, gestione e monitoraggio dei voli per quanto riguarda **Ryanair** e **Air** **Partner**. Il monitoraggio dei dati avverrà tramite dei grafici, i quali si baseranno su dati diversi offrendo varie prospettive.

Avviato il programma l’utente potrà creare il proprio progetto e aggiungere i voli, che saranno di due tipi: Ryanair oppure Air Partner. Inoltre i voli Air Partner offrono un servizio di consegna merci oltre al volo aereo. L’utente che si occupa della loro gestione potrà aggiungere delle tappe che dovrà effettuare il volo Air Partner.

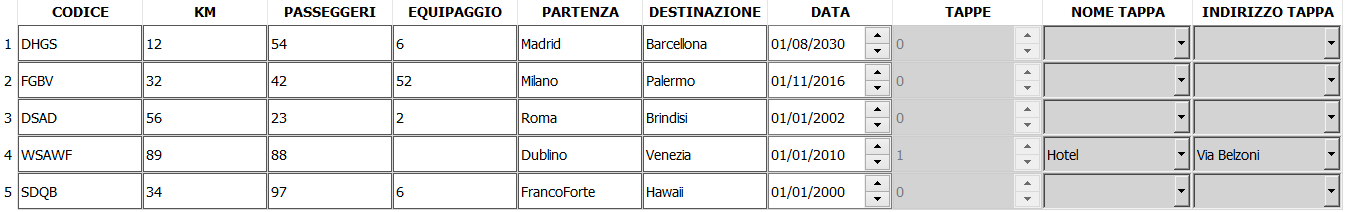
Si offrono anche funzionalità di salvataggio dei progetti, importazioni di progetti già presenti nel proprio dispositivo ed eliminazione dei vari viaggi del file selezionato.

1. **Manuale d’uso GUI**

**Fly Management** è un software che fornisce la visualizzazione dei voli e le loro caratteristiche, andando anche a rappresentare tali attributi su dei grafici.



Avviato il programma si avrà davanti la GUI con la scheda della tabella nella quale verranno rappresentati i viaggi. Ci sarà la barra del menù per i comandi rapidi, come “Nuovo Progetto”, “Salva” ecc… Al di sotto avremo due altre barre nelle quali ci sarà tutte le funzionalità offerte all’utente. Se l’utente vuole aggiungere un nuovo viaggio al proprio documento dovrà cliccare sul bottone “Aggiungi” il quale aprirà una schermata in cui si dovrà scegliere il tipo di volo desiderato, in questo caso **Ryanair** oppure **Air Partner**. Una volta selezionato nella schermata sottostante apparirà una riga dentro la quale si potranno scrivere tutte le informazioni necessarie. Il bottone “Aggiungi Locazione” permetterà di aggiungere una locazione al viaggio **Air Partner** desiderato andando a fornire la riga nel quale si trova e automaticamente la colonna “N Tappe” verrà incrementata.



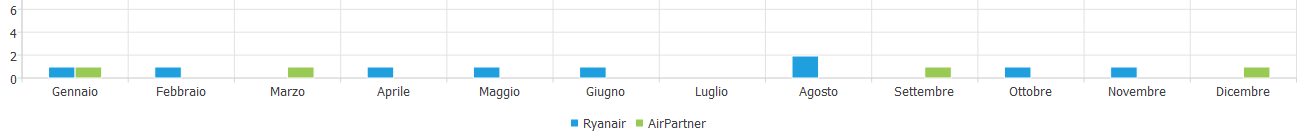
**Attenzione**: per quanto riguarda i viaggi Air Partner non sarà possibile aggiungere nessun dato nella colonna “Equipaggio”; invece nei viaggi “Ryanair” non è possibile aggiungere una locazione; la data minima per i singoli viaggi è l’anno 2000 mentre la massima è l’anno 2030.

Per eliminare un viaggio basta cliccare su “Elimina Viaggio” e dare in input la riga che si vuole eliminare.

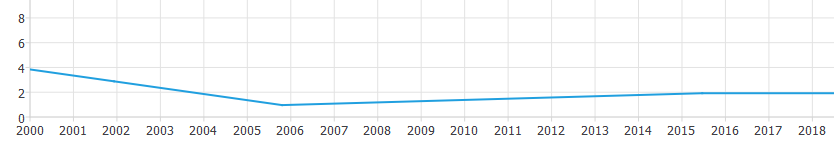


Nella seconda barra sono presenti 3 icone che rappresentano ognuna il tipo di grafico usato. Cliccandoci si andrà nella seconda scheda dove verrà visualizzato il grafico scelto.

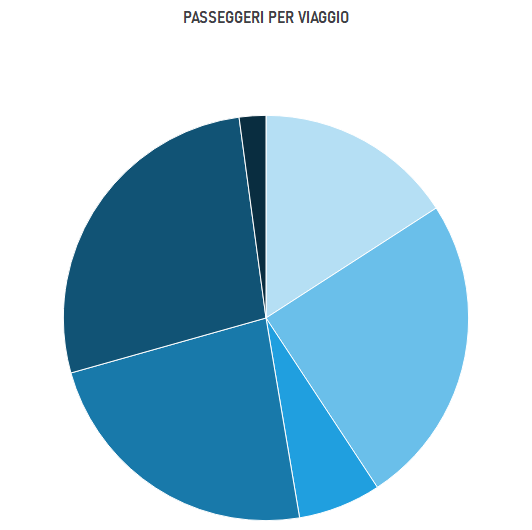
**Il grafico a barre offre una visuale in base alla quantità di tipo di volo per ogni mese.**



**Il grafico a linea mostra quanti voli ci sono per ogni anno**



**Il grafico a torta ci mostra la quantità dei passeggeri per ogni viaggio**



1. **Funzionalità fornite**

**Inserimento** 🡪 si è deciso di creare una tabella per la visualizzazione dei voli inseriti. La tabella prenderà i dati da un vector di classe “**rowViaggio**” finalizzata per rappresentare i dati del viaggio in una riga; quindi all’inserimento di un nuovo viaggio la tabella creerà una nuova istanza di rowViaggio e farà il push\_back all’interno del vettore della tabella. Verrà creata la riga e all’interno delle celle della riga ci sarà il riferimento alle **QLineEdit** dell’istanza rowViaggio creata. In questo modo i dati verranno memorizzati nel vettore che conterrà tutte le “righe” con tutte le caratteristiche dei voli. La classe rowViaggio fornisce inoltre due **QComboBox** utile per la registrazione delle locazioni per i vari voli **Air Partner**. Quindi qualora si volesse inserire una locazione, basterà specificare la riga del viaggio e le informazioni relativa alla locazione e questa verrà inserita nella **QComboBox**.

**Eliminazione** 🡪 l’eliminazione di un viaggio avverrà facendo il pop\_back dal vettore secondo la riga fornita dall’utente. Quindi in questo modo non solo verrà “eliminata” la parte grafica, ma verrà eliminato l’istanza che rappresenta quel viaggio all’interno del vettore della tabella.

**Nuovo Progetto** 🡪 permetterà di creare un nuovo progetto da zero. Quindi tutto quello presente nel Modello e nella parte View verrà azzerato e se già presente un file, **questo verrà automaticamente salvato.**

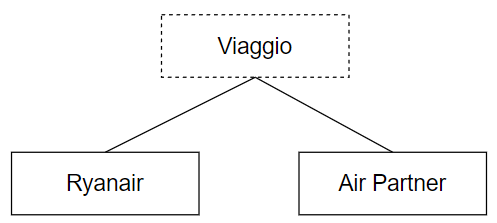
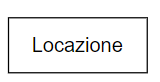
**Salvataggio** 🡪 Il software offre la funzionalità di salvataggio all’utente. Una volta cliccato su “**Salva**”**,** i dati della tabella verranno caricati nel model che eseguirà dei controlli come il controllo sugli **omonimi dei codici dei viaggi**. **Se i controlli vanno a buon fine i dati verranno memorizzati** in un vettore di Viaggi che sarà contenuto in una classe **Vettore**, volta a rappresentare il raggruppamento di tutti i viaggi nel nostro progetto e a offrire le funzionalità utile per gestirli. Avvenuto il caricamento nel model, il salvataggio avverrà proprio attraverso il **vettore** fornitoci dal modello. I file verranno salvati con formato **.xml**. Si fornisce anche una funzionalità “**Salva con Nome**”.

**Costruzione Grafici** 🡪 la GUI fornisce un tasto “**Aggiorna Dati**” che una volta cliccato andrà a prendere il vettore di classe rowViaggio presenta nella tabella, lo analizzerà e aggiornerà i dati dei grafici. Qualora l’utente apporti modifiche alla tabella, per visualizzare tali modifiche dovrà **necessariamente** cliccare su “Aggiorna Dati”. Come anticipato prima ogni grafico prenderà in considerazione una porzione di dati diversa.

**ATTENZIONE** 🡪 nel Model si potranno notare alcuni metodi che non sono utilizzati dalla view. Si è scelto di lasciarli in quanto potrebbero essere utili nel caso in cui si voglia usare il progetto senza la parte grafica o utili per estensioni future del codice.

1. **Gerarchia Tipi**

La gerarchia nel progetto vuole rappresentare la tipologia di volo, in questo caso Ryanair e Air Partner e gli attributi richiesti da queste, per offrire nel modo più completo possibile le informazioni sul viaggio e visualizzare queste caratteristiche in una tabella e sui grafici.

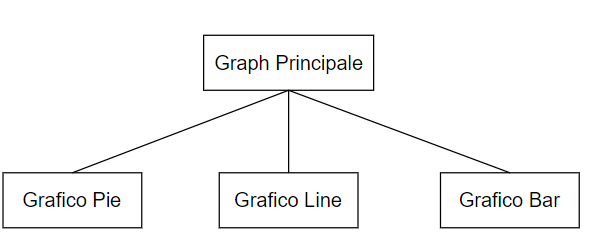


* Viaggio 🡪 è la classe generale che racchiude gli attributi comuni tra le classi che la derivano. Comprende i seguenti attributi: COD, Km, Passeggeri, Partenza, Destinazione e Data. Sono presenti due costruttori: uno riceve tutte le informazioni del Viaggio, mentre l’altro riceve un puntatore a un oggetto Viaggio. Viaggio rappresenta la classe padre e verrà derivata da Ryanair e AirPartner.
* Ryanair 🡪 deriva Viaggio con la clausola public, ne riprende tutti gli attributi e ne aggiunge altri: Equipaggio. Implementa quei metodi che sono stati dichiarati **virtual** nella classe padre.
* Air Partner 🡪 deriva Viaggio con la clausola public, ne riprende tutti gli attributi e ne aggiunge altri: il numero tappe e un vettore di tipo **Locazione** che è classe che rappresenta utile per costruire gli oggetti Locazione. Anche Ryanair implementa i metodi **virtual** di Viaggio, però in modo diverso da Ryanair.

Gli oggetti del nostro Model verranno costruiti tramite una classe separata chiamata “**CreateViaggio**”.

Per dividere la parte Logica dalla parte Grafica si è scelto il modello MVC. Si è inserita una classe controller che funge da “intermediario” tra la View e il Model, in modo da non far interagire direttamente la View con il Model, ma sarà la View che richiamerà dei metodi del Controller oppure manderà dei segnali e sua a volta il Controller richiamerà i metodi dal Model. Quindi il Controller avrà un riferimento alla View e uno Model in modo tale da far comunicare queste due. Inoltre nel Controller abbiamo un riferimento alla classe di IO che verrà usato ogni qual volta si dovrà andare a scrivere o leggere un file. Principalmente il Controller viene usato in fase di salvataggio in quanto l’interazione tra Model e View avverrà nel momento in cui si vorrà salvare il file; infatti l’utente immetterà tutti i dati nella View, ma in un primo momento questi dati non verranno subito inseriti nel Model; soltanto quando l’utente vorrà salvare il file avverrà la chiamata da parte della View dello SLOT “Salva” e qui il Controller andrà ad analizzare i dati ricevuti e chiamerà i metodi nel Model per l’aggiunta dei nuovi elementi. Stesso procedimento per i metodi di lettura da file: la View riceverà i dati letti da file dal Controller e richiamerà i metodi per riempire la tabella.

Nella parte della View, per la creazione dei grafici si è deciso di creare una classe generale **GraphPrincipale** che sarà poi derivata da **GraficoPie**, **GraficoLine** e **GraficoBar**.



* GraphPrincipale 🡪 è la classe generale, il costruttore richiede un riferimento alla classe TableRya utile per la costruzione dei grafici e ha un metodo virtuale puro che verrà implementato in modo diverso dalle figlie.
* Grafico Pie, Grafico Line, Grafico Bar 🡪 implementa il metodo virtuale puro “Aggiorna” e costruisce il grafico. Nel costruttore avrà un riferimento alla tabella che passerà al costruttore di GraphPrincipale. (Le implementazioni del metodo sono diverse in tutte e tre le classi)

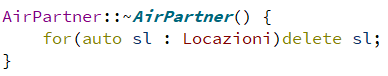
1. **Polimorfismo**

* **Distruttori Polimorfi**

Sono stati implementati distruttori virtuali polimorfi. Per esempio in Viaggio abbiamo il distruttore che sarà re implementato da **Ryanair** e **AirPartner** i due diversi modi. In **Ryanair** abbiamo il distruttore sarà quello standard:



Mentre in **AirPartner** dobbiamo andare a modificare il distruttore affinché vada a distruggere anche il contenuto del vettore delle locazioni:

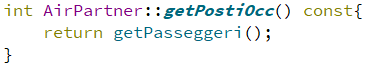


In questo caso si vanno ad eliminare gli oggetti puntanti dai puntatori presenti nel vettore Locazioni

* **get posti occupati**

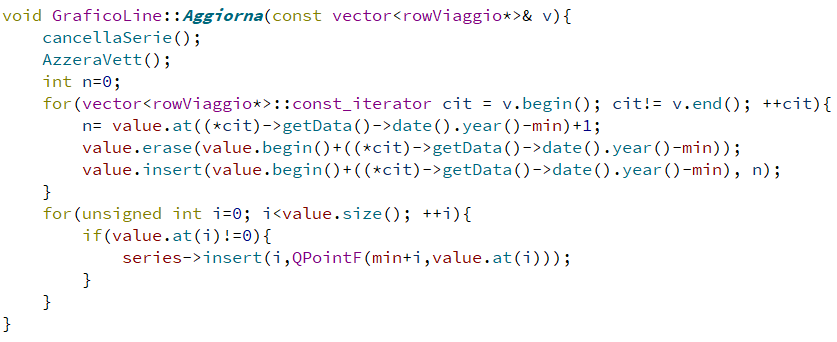
In Viaggio, getPostiOcc() è un metodo virtuale puro. Quando si andrà a richiamare questo metodo, si vedrà il tipo dell’oggetto e in base a questo si seleziona in che modo vengono ritornati i posti occupati. Infatti in Ryanair e AirPartner sarà implementato diversamente:

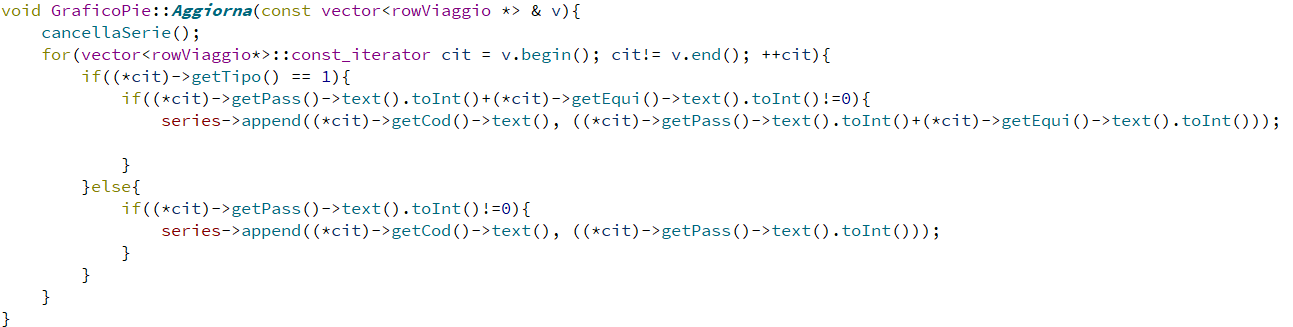
**Ryanair**  **Air Partner**



* **Aggiorna Grafici**

Nella classe GraphPrincipale è presente il metodo virtuale puro **Aggiorna**. Infatti dato che si hanno diversi grafici con diversi dati in input il metodo verrà implementato diversamente per ognuno dei grafici. Avremo void “Aggiorna (const vector<rowViaggio\*>&) **override**” :





1. **Descrizione formato Input e Output**

Il programma offre un meccanismo di salvataggio e caricamento dei propri file.

Il programma salverà e leggerà file con estensione **.XML**. Si è creata una classe apposita per l/O del programma dato che il file dovrà avere una struttura ben specifica. Il file sarà così costituito:

🡪 è la root principale

Nella root faremo un **appendChild()** che ci permetterà di inserire all’interno della root i nostri due tipi di viaggi, Ryanair o AirPartner.



Ovviamente ogni elemento avrà i propri attributi come si vede dall’immagine. Inoltre ogni elemento **AirPartner** avrà degli elementi figli che rappresenteranno le locazioni per quel viaggio.

Se si scegli di caricare un file che non ha il formato predefinito dal programma si potrebbero avere degli errori. Inoltre il programma non caricherà il file in quanto ha delle procedure per riconoscere il formato giusto. Per esempio caricare un file che sostituisce Ryanair e AirPartner porterà al caricamento del file vuoto.

1. **Compilazione ed esecuzione del Progetto**

Il progetto avrà i file .h, file .cpp e il **file .pro necessario** per la compilazione.

Istruzione per compilare da terminale:

1. qmake FlyManagement.pro
2. make
3. ./FlyManagement per eseguire il programma
4. **Sviluppo**

Il progetto è stato sviluppato su Windows 10 con **versione Qt 5.9.5** e **compilatore** **MinGW 64 bit**

Successivamente è stato testato sulla macchina virtuale con sistema operativo **Ubuntu**, versione **Qt 5.9.5** e **compilatore** **GNU g++ 7.3**

**ATTENZIONE** 🡪 testando il progetto sulla macchina virtuale si potrebbero riscontrare alcuni bug a causa della versione di Qt che però non inficiano sul rendimento del programma. Inoltre il formato XML su Ubuntu viene salvato come un file di testo, quindi nel programma se si vuole caricare un file, questo non apparirà nella finestra di selezione per il caricamento. **Si consiglia di testare questo aspetto su Windows.**

1. **Ore di Lavoro**

|  |  |
| --- | --- |
| Analisi preliminare del problema | 1.5 ore |
| Progettazione modello e GUI | 5 ore |
| Apprendimento libreria Qt e video tutorato | 10 ore |
| Codifica modello e Controller | 10 ore |
| Codifica GUI | 15 ore |
| Apprendimento XML e Codifica classe I/O | 5 ore |
| Debugging | 10 ore |
| Testing | 2 ore |
| TOTALE | **58,5** |

Le 50 ore sono state superate in quanto:

* Alcuni bug dovuti alla versione di Qt che sono stati aggirati
* La creazione della GUI ha avuto un tempo di progettazione lungo dovuto allo studio preliminare di alcune librerie offerte da Qt
* Visione dei video dei tutorati che hanno però aiutato alla gestione del MVC
* Apprendimento del funzionamento della libreria XML
* Cambiamenti continui al modello e alla View per rendere il più ottimizzato possibile