EasyCharts 

Relazione Progetto
Programmazione ad Oggetti

Anno Accademico 2021/2022

Davide Baggio

2009989

Indice

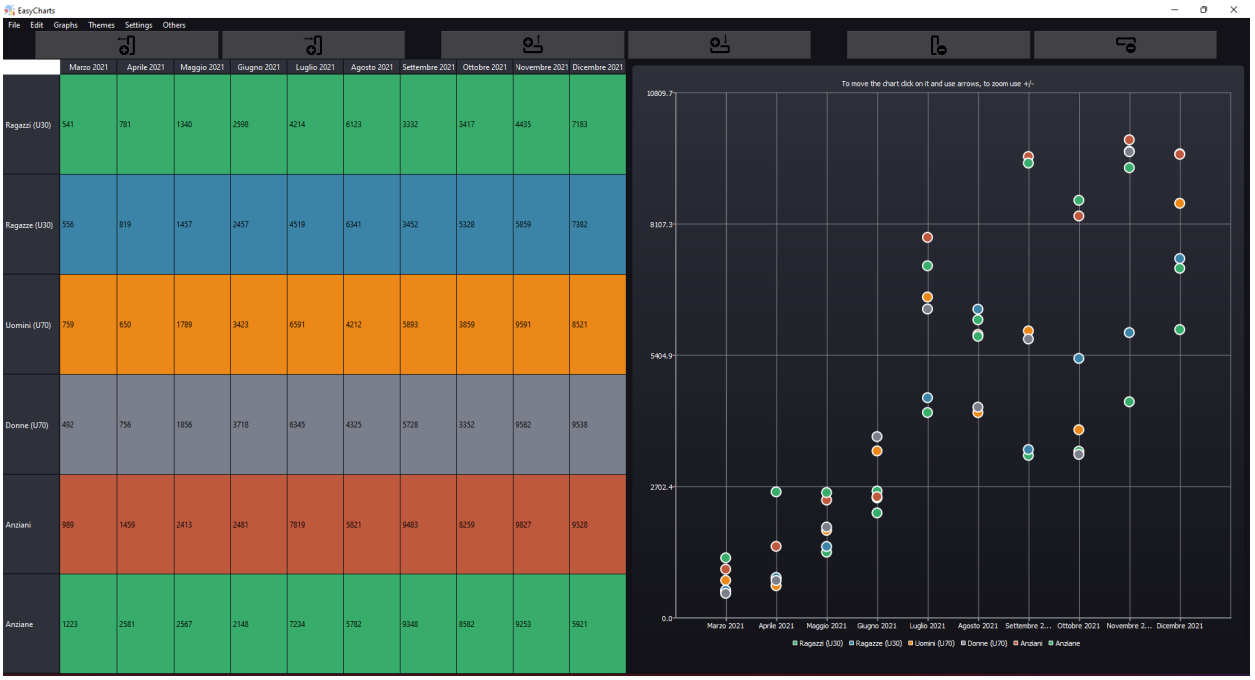
1. Introduzione

EasyCharts è un programma software facile ed intuitivo, in grado di consentire ad ogni utente una semplice gestione di dati, rappresentati mediante una tabella e da essi produrre ben 9 grafici distinti.
Le features di cui il programma dispone sono:

- totale gestione di una tabella bidimensionale di dimensioni variabili a runtime
- personalizzazione dell'interfaccia grafica in ogni contesto:
    - è possibile nascondere sia la tabella che il grafico, con conseguente riadattamento della schermata
    - colorare le celle della tabella con il medesimo colore corrispondente a quello rappresentato nel grafico prodotto (*attivo di default*)
    - selezionare a proprio piacimento uno tra gli 8 temi disponibili
- muovere e zoomare a proprio piacimento i grafici cartesiani
- salvare e importare file in 3 diversi formati: XML, JSON e CSV.

Il programma è stato interamente sviluppato in C++ 11 con l'ausilio del relativo framework Qt.

2. GUI



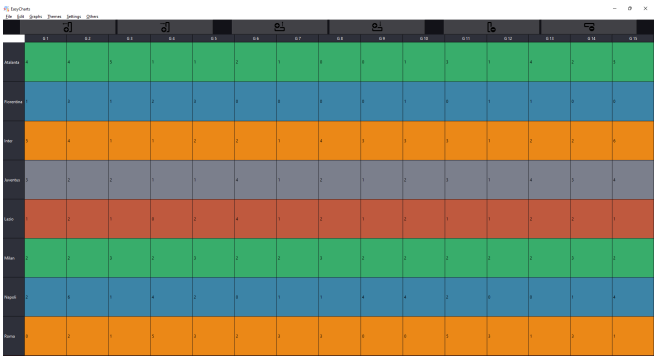
L'interfaccia grafica è stata sviluppata per essere di facile utilizzo, intuitiva e personalizzabile sotto diversi aspetti.
È presente una *QMenuBar* contenente i seguenti menù a tendina:

- *File*: dedicato alla gestione del file
- *Edit*: dedicato alla modellazione della tabella
- *Graphs*: dedicato alla selezione di un tipo di grafico da visualizzare per volta
- *Themes*: dedicato alla selezione del tema visibile
- *Settings*: dedicato alla gestione dell'interfaccia utente a seconda delle esigenze dell'utente
- *Others*: contiene una guida per il programma

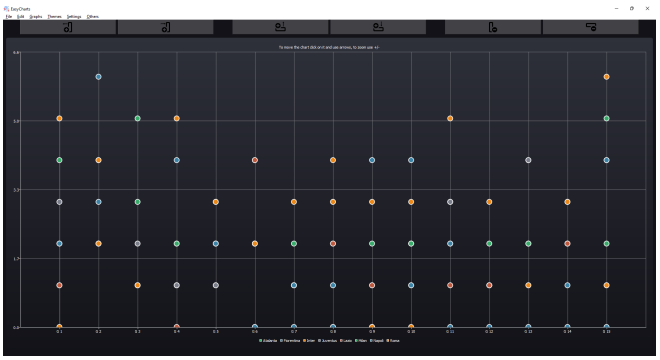
Appena sotto alla menù bar è presente una toolbar con 6 pulsanti che riprendono alcune funzionalità presenti nel menù *Edit* per semplificare l'esperienza dell'utente.
Appena aperto il programma di default è vuoto, l'utente può quindi decidere se:

- aprire un file esistente attraverso *File->Open* (in alternativa *Ctrl+O*)
- creare una nuova tabella mediante *File->New* (in alternativa *Ctrl+New*), così facendo all'utente viene richiesto di inserire quante righe e colonne desideri
- iniziare a creare e popolare una tabella attraverso i pulsanti presenti nella porzione superiore dell'interfaccia

Per consentire all'utente una facile gestione della tabella, si è optato per tenere il grafico nascosto fino a quando non sia l'utente stesso a volerne visualizzare il contenuto.
Inoltre, se nel corso dell'utilizzo del programma l'utente volesse focalizzarsi unicamente sulla tabella o sul grafico, gli basterà semplicemente cliccare sul menù *Settings* per gestire la GUI a proprio piacimento.
Infine, nei grafici con ascissa ed ordinata, l'utente può spostare o zoomare il grafico mediante l'utilizzo delle frecce direzionali e dei tasti +/-.



Esempio con grafico nascosto



Esempio con tabella nascosta

3. Architettura

Per la realizzazione del progetto è stato scelto il pattern architetturale *Model - View - Controller* (in breve MVC) per separare in maniera efficiente e chiara la logica funzionale del programma.

- *Model*: fornisce tutti i metodi necessari per poter accedere e modificare la tabella bidimensionale contenente i dati da gestire e rappresentare
- *View*: riproduce visivamente il contenuto dei dati in forma tabellare e con i svariati grafici, in modo tale da consentire all'utente un'esperienza intuitiva
- *Controller*: si occupa di gestire i comandi e i diversi input forniti dall'utilizzatore del programma a runtime, andando quindi a modificare a sua volta la vista e i dati del modello

4. Gerarchia dei tipi

2.1 Chart

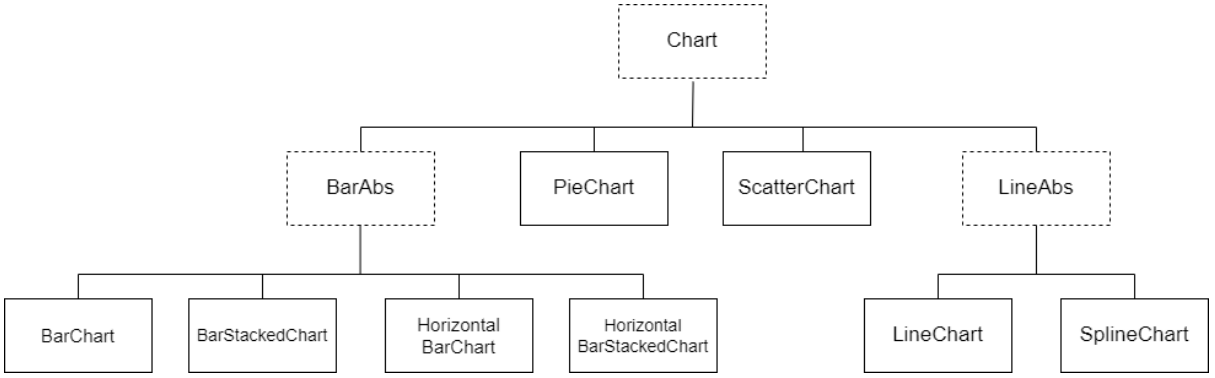


Chart è la classe base astratta per quanto riguarda l'intera gerarchia dei grafici, possiede due campi privati:

- Model: contiene il dataset da cui il grafico estrapola i valori numerici e le label da rappresentare
- QChart: tipo fornito da Qt che consente la creazione di grafici in maniera efficace

Le classi che derivano da *Chart*, per non essere a loro volta astratte, devono implementare i metodi puri *generateChart* e *addColorMapping*.

Le classi che ereditano dalla classe base astratta *Chart* sono:

- *PieChart*: si occupa della creazione del grafico a torta, contiene due campi:
    - Q*PieSeries*
    - double *holeSize*: si è scelto di non creare una classe separata per la creazione di un grafico a ciambella, ma semplicemente di introdurre un tipo double che rappresenta la grandezza del buco interno di un *PieChart*
- *ScatterChart*: si occupa della creazione del grafico di dispersione
- *BarAbs*: è la classe astratta da cui ereditano tutti i grafici a barre. Introduce due metodi con template per agevolare l'effettiva creazione dei seguenti grafici:
    - *BarChart*: si occupa della creazione del grafico a barre verticali
    - *BarStackedChart*: si occupa della creazione del grafico a barre segmentate
    - *HorizontalBarChart*: si occupa della creazione del grafico a barre orizzontali
    - *HorizontalStackedBarChart*: si occupa della creazione del grafico a barre segmentate orizzontali
- *LineAbs*: è la classe astratta da cui ereditano tutti i grafici a linee. Introduce due metodi con template per agevolare l'effettiva creazione dei seguenti grafici:
    - LineChart: si occupa della creazione del grafico a linee
    - SplineChart: si occupa della creazione del grafico a linee con interpolazione

2.2 Model

Model è la classe dedicata a rappresentare il modello di dati usato dal programma; eredita pubblicamente la classe *QAbstractTableModel*.
Include metodi che vanno a richiamare direttamente quelli presenti in *TableData*, classe totalmente indipendente dal framework fornito da Qt.

2.3 MainWindow

MainWindow è la classe che gestisce il comparto grafico del programma; eredita pubblicamente la classe *QWidget*. In particolare i campi privati di tipo *TableView* e *ChartView* (che eredita pubblicamente da *QChartView*), controllano rispettivamente il comportamento della tabella e del grafico a seconda delle preferenze dell'utente.

5. Chiamate polimorfe

Le chiamate polimorfe per i grafici si basano sui metodi puri appartenenti alla classe *Chart*:

- *virtual QChart* generateChart() = 0*: crea il grafico a partire dal modello fornitogli, estraendone i dati numerici e le stringhe per rappresentare correttamente i valori a seconda del grafico selezionato.
- *virtual void addColorMapping() = 0*: ricava il colore dei dati rappresentati sul grafico per fornirlo al modello e colorare le celle ad essi corrispondenti nella tabella.
- *virtual ~Chart() = default*: il distruttore è stato marcato virtuale in modo tale che anche le classi derivate vengano distrutte in qualsiasi contesto venga invocato.

6. Formati Input e Output

Il programma lascia all'utente massima libertà di scelta su come salvare ed importare i propri file, che possono essere gestiti in 3 formati diversi: XML, JSON e CSV.
In particolare quest'ultimo è stato reso compatibile con il diffuso programma di gestione dati *Microsoft Excel*, per cui i file con estensione *.csv* generati da *EasyCharts* possono essere aperti e modificati con estrema facilità anche su *Excel*.
Inoltre è possibile salvare lo stato del grafico attualmente visibile in formato PNG.

Nella cartella contenente il codice sorgente del programma è presente una cartella denominata "Saved Files", al cui interno sono presenti 3 file d'esempio, uno per ciascun tipo di estensione.
Questa stessa cartella è quella utilizzata dal programma di default per salvare i file generati nel caso l'utente non decidesse un percorso alternativo a proprio piacimento.

I file sono strutturati nei seguenti modi:

- JSON

```
{
   "Columns": [
       "Prima colonna",
       ...
       "Ultima colonna"
   ],
   "Rows": {
       "Prima riga": [
           valore 1,
           ...
           valore N
       ],
       "...": [
          ...
       ],
       "Ultima Riga": [
       ...
       ]
   }
}
```

- XML

```
<Graph>
<Columns number="N colonne">
 <Col value="titolo prima colonna"/>
       ...
 <Col value="titolo della colonna N"/>
</Columns>
<Rows number="N righe">
 <Row value="titolo prima riga"/>
       ...
 <Row value="titolo della riga N"/>
</Rows>
<Data>
 <Row>
  <Num value="valore della prima riga-prima colonna"/>
  <Num value="valore della prima riga-N colonna"/>
 </Row>
...
<Row>
  <Num value="valore della N riga-prima colonna"/>
  <Num value="valore della N riga-N colonna"/>
 </Row>
</Data>
</Graph>
```

- CSV

```
;colonna_1; ... ; colonna_N;
riga_1; valore_colonna1_riga1; ... ; valore_colonnaN_riga1;
...
riga_N; valore_colonna1_rigaN; ... ; valore_colonnaN_rigaN;
```

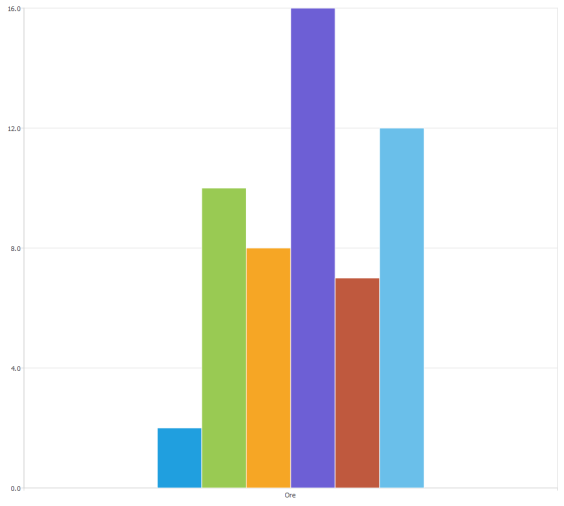
7. Istruzioni per la compilazione ed esecuzione

Per far sì che il programma venga correttamente compilato ed eseguito, digitare i seguenti comandi da terminale nella cartella *EasyCharts* :

```
$ qmake EasyCharts.pro
$ make
$ ./EasyCharts
```

8. Ore impiegate

ATTIVITÀ	ORE
Analisi dei requisiti	2
Documentazione Qt	9
Codifica TableData	8
Codifica Charts	16
Codifica Controller	7
Debug + Test	12
Totale	**52**



Le 2 ore richieste in più, rispetto alle 50 consentite dal docente, sono state impiegate per finalizzare la fase di testing specificatamente sulla macchina virtuale e verificare quindi il corretto comportamento del programma in diversi contesti d'utilizzo.

9. Suddivisione dei compiti

Davide Baggio matricola 2009989:
- analisi delle specifiche del progetto
- progettazione e codifica "*TableData*"
- progettazione e codifica grafici
- progettazione e codifica controller
- test e debug

Samuel Scarabottolo matricola 2012435:
- analisi delle specifiche del progetto
- progettazione e codifica modello
- progettazione e codifica comparto GUI
- test e debug

10. Ambiente di sviluppo e testing

Software	**Sviluppo**	**Testing**
Sistema Operativo	Windows 11 Home build 22000.856	Ubuntu 18.04.3 LTS
Compilatore	GCC 8.1.0 MinGW-W64	GCC 7.4.0
Libreria Qt	5.9.5	5.9.5
IDE	Qt Creator 5.0.3	Qt Creator 5.0.3

Inoltre, per poter svolgere in maniera produttiva il progetto di coppia, è stato utilizzato il sistema di versionamento *Git*, che si è rivelato essere molto efficace nel tenere traccia dei cambiamenti apportati al progetto da entrambi i due componenti del gruppo grazie ad una repository remota presente su *GitHub*. Questo ci ha permesso di avere chiaro fin da subito su come procedere nello sviluppo del software grazie alla project board automatica rappresentante le feature da implementare e segnalare a vicenda l'eventuale presenza di errori e comportamenti anomali del programma.