Traffic Lights Optimizer

PROGETTO DI PROGRAMMAZIONE AD OGGETTI

A

Anno accademico 2021/2022

UNIVERSITÀ DEGLI STUDI DI PADOVA

Realizzato da

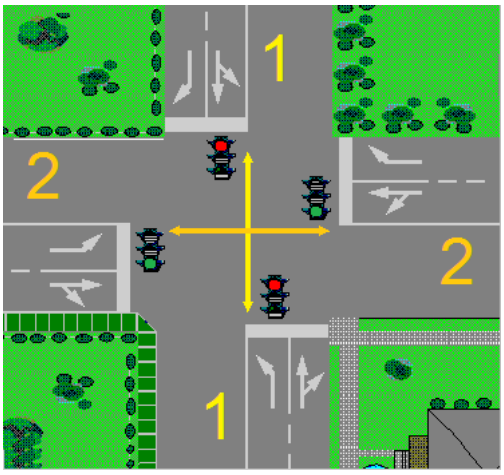
Raul Seganfreddo 1226293

1. Introduzione

Spesso ci ritroviamo ad aspettare semafori rossi interminabili con nessun'auto che passa nella strada dove invece il semaforo è verde. Allora ci chiediamo: "Chi ha progettato questo sistema di semafori? A cosa pensava?"

2. Descrizione del programma

Traffic Lights Optimizer nasce proprio per ovviare a questo problema.
Questa applicazione si propone di acquisire un insieme di dati nel tempo e di proporre l'ottimizzazione dei tempi di un ipotetico impianto semaforico.
Tale impianto gestisce l'intersezione o incrocio di 2 strade (quadrivio);
tali strade d'ora in avanti verranno identificate con Strada 1 e Strada 2.



In base al numero dei passaggi di veicoli lungo la Strada 1 e Strada 2 nel tempo, l'applicazione visualizza, mediante grafici dedicati, sia l'andamento di tali passaggi sia una possibile ottimizzazione dei tempi di attesa (semaforo rosso) e consenso al transito (semaforo verde) per le due direttrici.

La base dati è costituita da:
- suddivisione per Anno/Mese/Giorno/Ora del numero di passaggi
- numero di passaggio su Strada 1 avvenuti in ogni ora della giornata
- numero di passaggio su Strada 2 avvenuti in ogni ora della giornata

Tale base dati potrebbe ipoteticamente essere alimentata da un sistema di sensori posizionati in prossimità dell'incrocio.
Attualmente viene gestita (inserimento/cancellazione/modifica) mediante pagina video dedicata.

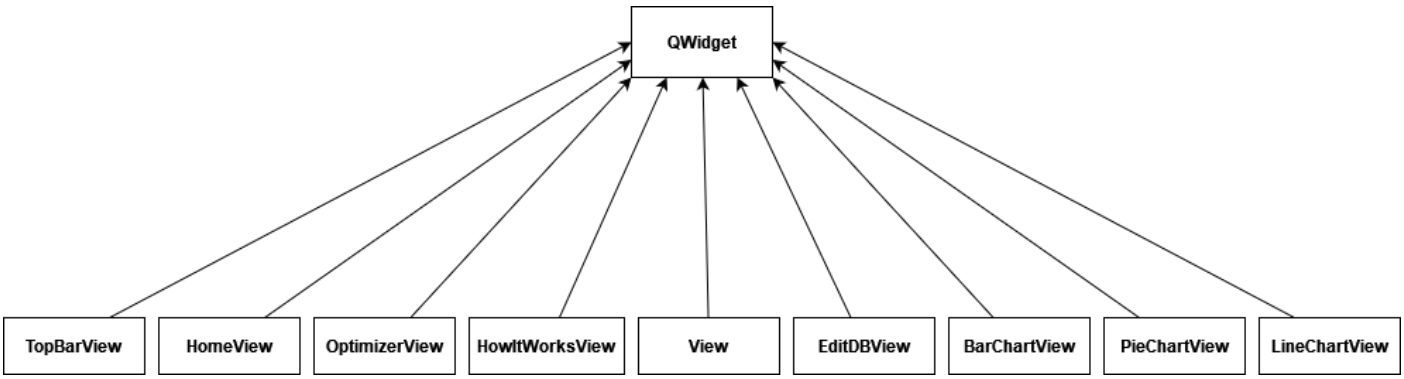
3. Funzionalità

- Elaborazione dei dati e visualizzazione tramite grafici. I grafici offerti sono: un **BarChart** che permette di visualizzare il numero di veicoli che hanno attraversato l'incrocio in un dato giorno, per un range di date deciso dall'utente. Un **PieChart** che permette di visualizzare l'ottimizzazione media in secondi per ciascuna delle due strade in un range di date deciso dall'utente. Infine, un **LineChart** che permette di vedere il variare dell'ottimizzazione giornaliera, ora per ora, per le due strade in una data decisa dall'utente.
- Salvataggio dei dati eventualmente modificati in formato csv nel file database.csv. Questo file viene spesso controllato, nel caso si cerchi di visualizzare dati di giorni non registrati nel database o si cerchi di modificare dati non esistenti. Il file database.csv per essere elaborato viene convertito in una lista di Records, in modo da essere reso leggibile e per agevolare il processo dei dati dal programma.
- I dati possono essere modificati tramite la pagina apposita, per poi essere convertiti e salvati nel file csv e la schermata viene aggiornata in tempo reale.

4. Architettura del progetto

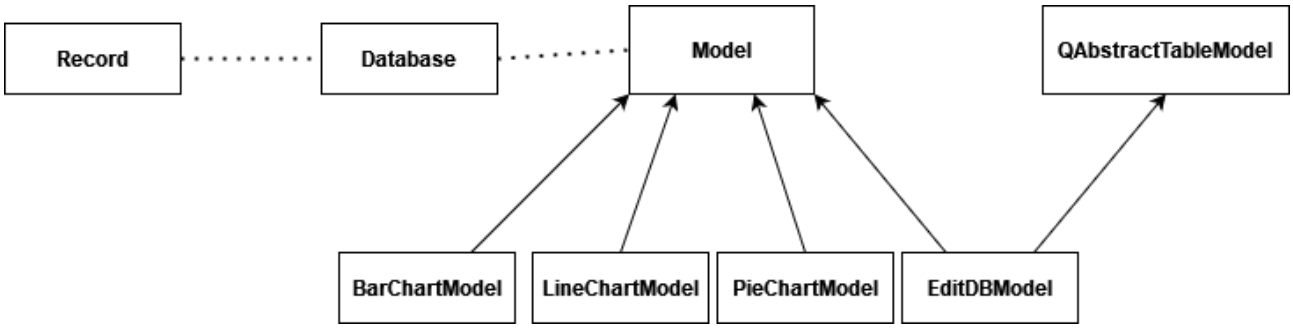
4.1 Gerarchia Vista

Lo sviluppo del progetto adotta il pattern ***Model-View*** di Qt visto il supporto di Qt stesso per questo tipo di pattern.



Le classi che riguardano la ***view*** derivano dalla classe *QWidget* in quanto permette l'utilizzo dei widget per costruire le varie finestre. La classe **View** gestisce la ricezione dei vari segnali per l'apertura delle finestre e contiene le varie funzioni che permettono l'apertura di esse.
HomeView rappresenta la schermata **Home**, la principale che troviamo quando apriamo l'applicazione. Dalla **Home** possiamo andare nella pagina **HowItWorks**, dove si può trovare una breve descrizione dell'applicazione. Sempre dalla **Home** possiamo andare alla pagina **EditDB**, dove è possibile visualizzare e modificare il database. Come ultima pagina dalla home possiamo andare alla **OptimizerView**, una pagina di transizione, dalla quale possiamo aprire le *tre pagine* per visualizzare i *tre tipi di grafico*. Infine **TopBarView** è una classe utilizzata per creare la *TopBar* delle varie pagine.

4.2 Gerarchia Modello



Il **Modello** è rappresentato dalla classe **Model**, padre dei *modelli dei vari charts* e di **EditDBModel**.
Il database è rappresentato dalla classe **Database**, la quale dato principale è una lista di *Records*, che rappresenterebbe l'intero database. La lista è formata da **Records**, un tipo che contiene quanto necessario per gestire una "linea" del database: data, ora, auto passate nella strada 1 e auto passate nella strada 2.
La classe *Model* fa anche da tramite per passare i dati dal database ai modelli dei vari charts e viceversa. La classe **EditDBModel** gestisce la pagina di visualizzazione e modifica del database. Visto che, graficamente parlando, la visualizzazione dei dati è in forma di tabella, il model, per poter gestire al meglio i dati rappresentati in questa forma, ha come padre oltre a model, anche **QAbstractTableModel**, che fornisce delle funzioni ottime per la gestione e rappresentazione dei dati in forma di tabella.

4.3 Polimorfismo

Il polimorfismo è stato utilizzato soprattutto nella classe **EditDBModel**. Visto la natura astratta di **QAbstractTableModel**, bisognava fare *l'override* di vari metodi, tra cui **HeaderData**, **RowCount**, **ColumnCount**, **Data**, **SetData** e **Flags**.

4.4 Note

- Dove possibile si è preferito utilizzare tipi predefiniti da Qt invece che la loro controparte della libreria STL per evitare di dover fare troppe conversioni.

- In alcuni punti è stata intenzionalmente violata la regola di separazione tra model e view, in quanto nel model sono stati tipi di dati forniti dal framework Qt e pure una classe nel model (EditDBModel) è derivata da una classe astratta di Qt (QAbstractTableModel). Questo è stato fatto per comodità d'implementazione e di interazione con la parte grafica.

5. Conclusioni

5.1 Ore di lavoro

Attività	Ore impiegate
Analisi delle specifiche	3
Progettazione della GUI	5
Studio Framework Qt	7
Progettazione e implementazione della struttura dati	7
Progettazione e implementazione del modello e dei grafici	16
Implementazione del sistema di I/O tramite Record e CSV	5
Progettazione e implementazione della parte grafica (view)	8
Test e debugging	4
Stesura della documentazione	4
Ore Totali	59

5.2 Ambiente di sviluppo

Per lo sviluppo è stato utilizzato Visual Studio Code, text editor di proprietà Microsoft. Per la compilazione è stato utilizzato QMake (fornito da Qt) e Make. Lo sviluppo è avvenuto utilizzando Qt5 (5.12.8) e gcc 11.2.0 in esecuzione su Ubuntu 22.04.1 LTS, eseguito attraverso Windows Subsystem for Linux 2.

5.3 Compilazione ed esecuzione

Per la corretta compilazione, prima bisogna installare alcuni componenti aggiuntivi di Qt:

- qt5-default
- libqt5charts5-dev

La compilazione avviene con QMake mediante il file .pro.
Utilizzando i comandi “qmake” -> “make” arriviamo alla creazione dell’eseguibile ./TLO3 con il quale si può avviare l’applicazione.

5.4 Note

Quest’attività è stata molto interessante e costruttiva. Mi ha messo per la prima volta veramente alla prova con lo sviluppo di un’applicazione che fosse qualcosa di più delle solite poche centinaia di righe di codice e soprattutto con una parte grafica. Ho riscontrato delle difficoltà nello sviluppo soprattutto a causa della non perfetta documentazione di Qt.