Ropeway Manager

Programmazione ad Oggetti

Riccardo Cestaro
Matricola: 1170624

June 22, 2019

Ropeway Manager è una semplice utility che consente di visualizzare, ed eventualmente aggiungere o rimuovere, impianti di risalita a fune.
Ropeway Manager consente inoltre di modificare alcune delle principali caratteristiche dei suddetti impianti.

0.1 Prefazione

Scopo del progetto

Lo scopo del progetto e' realizzare un utility che consenta di memorizzare una lista di impianti di risalita a fune, con le principali caratteristiche.

Ambiente di sviluppo

- Sistema operativo: Kubuntu 19.04 / Kernel 5.0.0-13-Generic 64Bit
- Hardware: Intel Core m3-7y30 CPU @ 1.00GHz / 3,7 GiB di RAM
- Versione Qt: Desktop Qt 5.9.5 GCC 64bit / Qt Creator 4.9.0 64bit
- Versione GCC: 5.3.1 20160406 (Red Hat 5.3.1-6)

Ripartizione Ore

- Analisi del problema: 2 ore;
- Progettazione del modello: 2 ore;
- Progettazione della vista: 30 minuti;
- Programmazione del container e deepptr: 11 ore;
- Programmazione della gerarchia: 10 ore;
- Apprendimento della libreria Qt: 2 ore e 30 minuti;
- Programmazione parte grafica e di controllo: 19 ore;
- Testing: 4 ore;

0.2 Ropeway Manager

Funzionalita' principali

Le principali funzionalita' incluse nell'utility si riassumono in:

- Visualizzazione della lista degli impianti: In cui viene visualizzata una tabella con dei campi principali. Con un doppio click si accede all'area di modifica, con un click si seleziona l'impianto. Viene offerta anche la possibilita' di ordinare gli impianti per tutti i campi della tabella, in ordine ascendente ma anche in ordine discendente. Inoltre si possono filtrare le righe della tabella con una ricerca incrociata tra tipologia di impianto e produttore dell'impianto. Nel basso viene resa la possibilita' di eliminare gli impianti selezionati nella tabella, o di entrare nell'area di modifica tramite un pushbutton.

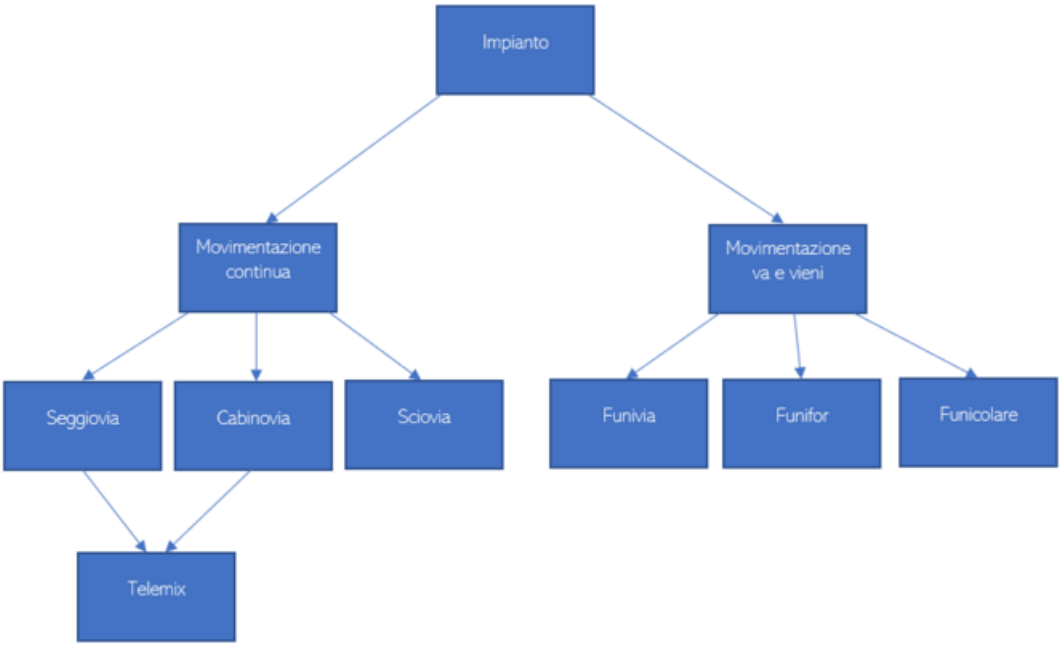
- Aggiunta di un impianto: Questa pagina della *TabBar* offre la possibilita' di aggiugere un impianto di risalita alla lista. Tramite una *ComboBox*, che contiene la tipologia dell'impianto da aggiungere, vengono nascosti o resi visibili i campi da riempire per quella tipologia di impianto scelto. Inoltre viene resa la possibilita' di aggiungere anche una foto dell'impianto.

- Area di Modifica/Visualizzazione completa dell'impianto: In quest'area, accessibile tramite un doppio click dalla tabella o tramite il *PushButton* visualizza, vengono visualizzati tutti i campi dell'impianto selezionato e viene resa possibile la loro modifica.

0.3 Aspetti Progettuali

Gerarchia

La gerarchia scelta comprende tre classi virtuali astratte e una sottogerarchia a diamante. La gerarchia si riassume con le classi:



- *Impianto*: classe astratta, che contiene le caratteristiche comuni a tutti gli impianti di risalita a fune;
  - *MovimentazioneContinua*: classe astratta, comprende gli impianti con un moto della fune continuo;
    - *Seggiovia*: classe virtuale, comprende gli impianti con dei veicoli del tipo: seggiola;
    - *Cabinovia*: classe virtuale, comprende gli impianti con dei veicoli del tipo: cabina;
    - *Telemix*: classe concreta, derivata da Seggiovia e Cabinovia, comprende gli impianti con dei veicoli del tipo: seggiola e cabina;
    - *Sciovia*: classe concreta, comprende gli impianti con dei veicoli del tipo: gancio;
  - *MovimentazioneVaeVieni*: classe astratta, comprende gli impianti con un moto della fune alternato.
    - *Funivia*: classe concreta, comprende gli impianti con la classica tecnica va e vieni a due funi;
    - *Funifor*: classe concreta, comprende gli impianti con tecnica va e vieni a quattro funi;
    - *Funicolare*: classe concreta, comprende gli impianti su rotaie a terra a fune;

I metodi principali utilizzati nella gerarchia:

- *virtual Impianto* clone() const = 0;* Il metodo *clone()* e' il meotodo di clonazione polimorfa. Permette la clonazione di oggetti mantenendo il tipo dinamico.
- *virtual string getType() const = 0;* Il metodo *getType()* ritorna una stringa con il tipo di oggetto della gerarchia.
- *Metodi getter e setter* Permettono l'ottenimento e l'assegnazione di tutte le specifiche degli oggetti di tipo impianto.
- *Metodi read e write* Permettono la serializzazione e la deserializzazione, verranno spiegati nel paragrafo successivo.

Serializzazione

Per la serializzazione viene implementata una classe con due metodi statici:

- *static void read(string filename)* Il metodo read, che viene richiamato all'avvio dell'eseguibile, nel caso in cui l'utente volesse caricare dei dati preesistenti, oppure nel caso in cui l'utente voglia caricare dei dati preesistenti tramite l'azione *load from XML* dalla toolbar. Il metodo read si preoccupa di leggere tutti i tag del file XML, il cui *path* e' passato come parametro alla funzione. Il metodo adotta un'iterazione *while* che cicla fino a quando i tag di tipo *impianto* sono terminati. La costruzione degli oggetti avviene all'interno della gerarchia, appunto, per offrire una maggiore estendibilita' al codice. Tramite il metodo statico *read(QXmlStreamReader*);* , appunto viene deserializzato e creato un container. Il metodo statico e' presente anche nella classe *Impianto*, il cui metodo viene richiamato dalle classi concrete, ritornando una struct contenente i campi generici di *Impianto*. Le funzioni statiche *read(QXmlStreamReader*);* delle classi concrete, una volta ottenuti tutti i campi dati, creeranno l'oggetto.

- *static void write(string filename)* Il metodo write, viene richiamato nel momento in cui l'utente decide di utilizzare l'azione *save to XML* dalla toolbar. Il metodo si occupa di scrivere il file di salvataggio XML di tutti gli oggetti contenuti nel container. Sfrutta il poliformismo della gerarchia per serializzare il tipo effettivo dell'oggetto da salvare, tramite il metodo virtuale *virtual void write(QXmlStreamWriter*);*.

Contenitore

Il contenitore viene realizzato replicando buona parte dei metodi disponibili nella libreria standard stl del vector. Inoltre vengono implementati attraverso classi annidate le classi iterator e const iterator. I metodi principali implementati:

- Classe Container
  - Costruttori, Distruttori e ridefinizione dell'operatore di assegnazione
  - Metodi begin(), end(), cbegin() e cend()
  - Metodi per l'inserimento e l'assegnazione
  - Metodi per l'eliminazione
  - Metodi per l'ottenimento degli oggetti
  - Ridefinizione degli operatori di confronto
- Classe Iterator
  - Costruttori
  - Ridefinizione degli operatori di confronto
  - Ridefinizione degli operatori
- Classe Const Iterator
  - Costruttori
  - Ridefinizione degli operatori di confronto
  - Ridefinizione degli operatori

Puntatore Profondo

Il puntatore profondo viene implementato grazie alla classe templetizzata *DeepPtr* che permette di costruire un puntatore profondo per ogni oggetto con il quale vogliamo gestire piu' intelligentemente la memoria. Piu' in dettaglio i metodi implementati:

- *DeepPtr(T* q = nullptr);*

  Il costruttore consente di creare un oggetto DeepPtr che implementi al suo interno un puntatore al tipo T.

- *DeepPtr(const DeepPtr& q);*

  Il costruttore di copia consente di creare un oggetto DeepPtr tramite un oggetto DeepPtr eseguendo una copia profonda, tramite una copia polimorfa grazie al metodo *clone()* implementato nelle classi della gerarchia.

- *~DeepPtr();*

  Il distruttore viene ridefinito per eseguire una eliminazione profonda, garantendo l'eliminazione della memoria puntata dal puntatore all'interno dello stesso oggetto DeepPtr.

- *T* operator→() const;*

  Ridefinizione dell'operatore freccia. Ritorna il puntatore dell'oggetto con cui e' stato creato il DeepPtr.

- *T& operator*() const;*

  Ridefinizione dell'operatore di dereferenziazione. Ritorna il riferimento all'oggetto a cui punta il puntatore privato del DeepPtr.

- *DeepPtr& operator=(const DeepPtr& dp);*

  Ridefinzione dell'operatore di assegnazione. Permette di assegnare un oggetto DeepPtr ad un altro eseguendo una copia profonda, tramite una copia polimorfa grazie al metodo *clone()* implementato nelle classi della gerarchia. Ovviamente la memoria viene gestita profondamente.

Grafica Qt

La View consiste principalmente di tre viste principali, di cui due sono visualizzabili tramite la *QTabBar*, TabBar fornitaci dalla libreria di Qt. Dalla TabBar si visualizza in primis la vista relativa alla visualizzazione degli oggetti di tipo impianto, oppure la vista relativa all'aggiunta degli impianti a fune.

- Nella vista relativa alla visualizzazione degli oggetti di impianto, viene implementata una *QTableWidget* che consente di visualizzare gli oggetti con alcune loro caratteristiche, e che facilita, tramite apposite librerie di Qt, la ricerca e l'ordinamento degli oggetti di tipo impianto. Infatti tramite delle classi di tipo *QTableProxyModel* viene possibile creare un modello da utilizzare nella tabella. Tramite questa classe viene implementata la ricerca incrociata su due campi fondamentali, effettuando l'overloading del metodo *bool filterAcceptsRow(...) const*. Inoltre l'ordinamento viene reso possibile su ogni campo tramite un click nel header della tabella, su il relativo campo, tramite il metodo *orderByColumn(int)* offerto dalla classe Qt *QTableProxyModel*. La classe *QHeaderView* viene estesa per opportune modifiche grafiche, quali gli effetti di *hover* del mouse, oppure la dimensione della colonne rese *fixed*.

Con un doppio click dalla tabella, oppure con una selezione e il click sul button visualizza si accede alla terza vista, che permette la visualizzazione di tutte le caratteristiche dell'impianto, oltre alla sua modifica.

- La vista che permette di visualizzare interamente o modificare tutti i campi di un oggetto impianto, chiamata *ShowAllSpec*, viene modellata tramite la tipologia di impianto da visualizzare/modificare. Una possibile autocritica al progetto e' l'uso di *dynamic_cast*, necessari, vista la gerarchia a diamante. Infatti utilizzando questo tipo di cast, nella classi *seggiovia , cabinovia e telemix* si ritarda il cast a runtime. Naturalmente nella altre classi concrete della gerarchia viene utilizzato lo *static_cast*.

Con un click nella sezione *Aggiungi* della TabBar si raggiunge la vista relativa all'aggiunta degli impianti.

- La vista che permette l'aggiunta degli impianti, chiamata *InsertRopeway*, contiene una *ComboBox*, che contiene la tipologia dell'impianto da aggiungere. Essa rende possibile visualizzare i campi di aggiunta relativi a quella tipologia di impianti. Viene utilizzato un *QFormLayout* che consente la creazione di un layout predisposto per i form.