Progetto di Programmazione ad Oggetti

A.A. 2018/2019

Vittorio Corrizzato – Matricola 1122288

Progetto: PC Part Picker (Qontainer)

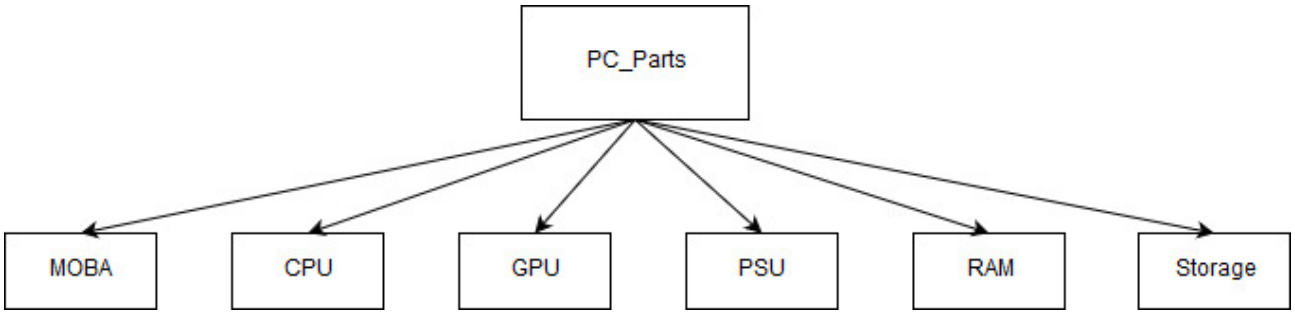
Sommario

1. Descrizione Generale

PCPartPicker è un sito (https://pcpartpicker.com/) che permette di creare configurazioni di computer personalizzate partendo dai singoli componenti di cui fornisce caratteristiche, prezzo e rating (basato sulle recensioni degli utenti).

Lo scopo che si prefigge Qontainer è quindi quello di emulare il funzionamento di questo portale permettendo l'assemblaggio di build con parti scelte dall'utente, indicandone allo stesso tempo specifiche, costo e rating (qui basato sulla qualità del pezzo). È possibile, inoltre, modificare le caratteristiche dei componenti, aggiungerne di nuovi o rimuoverli.

2. Descrizione Gerarchia



La classe base della gerarchia è *PC_Parts* contenente i campi dati che rappresentano attributi comuni a tutti i componenti quali: nome, prezzo, lunghezza, larghezza, produttore e consumo di elettricità.

PC_Parts ha 6 sottoclassi:

- *MOBA*: classe che descrive le caratteristiche di una scheda madre tra cui: socket, form factor, slot di RAM disponibili, quantità di RAM massima supportata e connettori.
- *CPU*: classe che descrive le specifiche di un processore tra cui: frequenza di clock, numero di cores, eventuale supporto di architetture a 64 bit, socket e presenza di un eventuale chip grafico integrato.
- *GPU*: classe che descrive gli attributi di una scheda grafica tra cui: tipo di memoria RAM grafica, quantità di memoria grafica, numero di operazioni in virgola mobile al secondo (FLOPS), frequenza di clock, tipo di interfaccia d'espansione, connettori ed eventuale bisogno di alimentazione supplementare.
- *PSU*: classe che descrive le caratteristiche di un alimentatore tra cui: form factor, wattaggio, tipo di certificazione di efficienza, tipo di modularità ed eventuale possibilità di fornire alimentazione supplementare.
- *RAM*: classe che descrive le specifiche di un banco di RAM tra cui: frequenza di clock, tipo di memoria RAM e quantità di memoria.
- *Storage*: classe che descrive gli attributi di un supporto di archiviazione tra cui: tipo, RPM, quantità di memoria disponibile, interfacce di connessione, form factor e velocità di trasferimento.

3. Descrizione Container

Come container è stata usata una versione modificata di un vector, in quanto non sussistono particolari esigenze di ordinamento, mentre è più importante l'accesso randomico.

Oltre a costruttore, distruttore e overloading dell'operatore di copia e di assegnazione sono state implementati in *cvector* i seguenti metodi:

- *empty*: controlla se il *cvector* è vuoto e, in caso, ritorna true;
- *clear*: pulisce il *cvector* e ne inizializza uno nuovo;
- *getCapacity*: ritorna la capacità del *cvector*;
- *getSize*: ritorna la dimensione del *cvector*;
- *setSize*: modifica la *size* del *cvector*;
- *resize*: inizializza un nuovo *cvector* con una nuova capacità *c*;
- *operator[]*: overloading dell'operatore di scoping per ritornare l'elemento in una determinata posizione;
- *push_back*: inserisce un nuovo elemento alla fine del *cvector*;
- *pop_back*: elimina l'ultimo elemento di *cvector*;
- *swap*: scambia due elementi del *cvector* tra di loro;
- *erase*: cancella l'elemento in posizione *pos*;
- *search*: controlla se un elemento *v* è presente in *cvector* e, in caso, ritorna la sua posizione nel contenitore, altrimenti torna -1;
- *operator==*: overloading dell'operatore di uguaglianza;
- *operator!=*: overloading dell'operatore di disuguaglianza.

4. Descrizione GUI

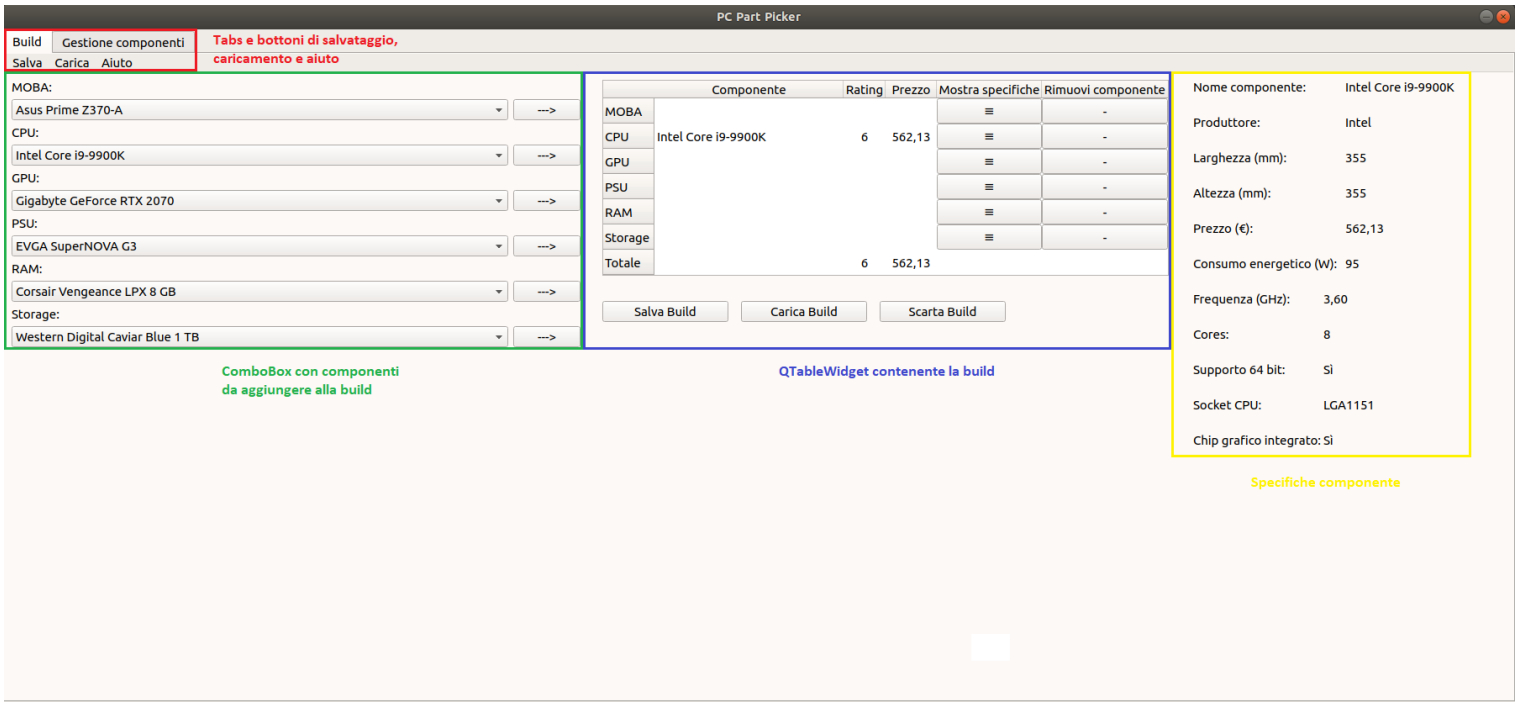


Figura 1: Tabella "Build" di PC Part Picker

La GUI è gestita quasi totalmente dal file *mainwindow.cpp* e dal corrispettivo header *mainwindow.h* che eredita da *QWidget*, il manuale utente è invece inizializzato da *usermanual.cpp* e da *usermanual.h*. L'interfaccia utente segue il pattern Model-View.

La figura 1 sopra mostra la tab *Build* del programma dove è possibile assemblare il proprio computer e vedere le specifiche di ogni componente inserito.

- Nel riquadro rosso abbiamo la possibilità di selezionare una delle due tab del programma (attraverso *QTabWidget*) e, tramite una *QMenuBar*, di salvare o caricare un database di componenti oppure visualizzare la finestra di aiuto (che funge da manuale utente).
- Nel riquadro verde abbiamo un *QFormLayout* che contiene: delle *QComboBox* con i componenti caricati e dei *QPushButton* che permettono l'inserimento dei pezzi nella build.
- Nel riquadro blu abbiamo un *QTableWidget*, rappresentante la build, con relativi *QPushButton* per mostrare le specifiche di un determinato componente. È inoltre possibile salvare, caricare o scartare la build.
- Nel riquadro giallo, infine, abbiamo un *QFormLayout* con delle *QLabel* che indicano le specifiche del pezzo selezionato.

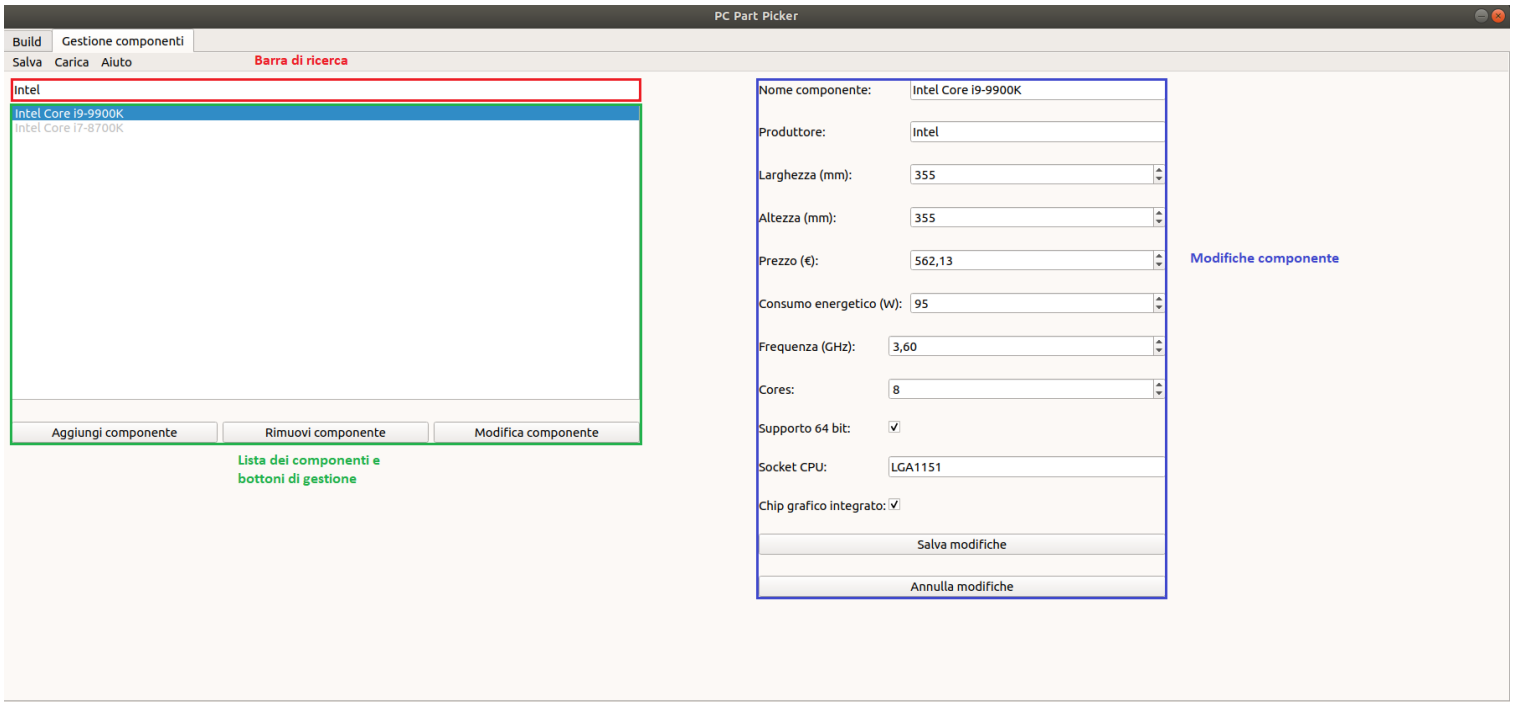


Figura 2: Tabella "Gestione componenti" di PC Part Picker

La figura 2 mostra, invece la tab *Gestione componenti* di PC Part Picker dove è possibile vedere tutti i componenti presenti e procedere alla modifica o alla rimozione di essi. In alternativa, è anche possibile l'inserimento di nuovi elementi.

- Nel riquadro rosso è presente una barra di ricerca implementata con un semplice *QLineEdit*.
- Nel riquadro verde abbiamo un *QListWidget* con tutti i componenti e dei *QPushButton* per inserimento, rimozione o modifica.
- Infine, nel riquadro blu abbiamo un *QFormLayout* che mostra le specifiche dei componenti, in questo caso modificabili con l'utilizzo di *QLineEdit*, *QSpinBox* e *QCheckBox*.

5. Codice Polimorfo

All'interno di Qontainer sono presenti varie chiamate polimorfe, attraverso dei dynamic cast e degli static cast che ritornano un puntatore al tipo di componente verso il quale avviene il cast.

Vengono utilizzati per:

- Inserire il componente corretto nella build;
- Rimuovere il giusto componente dal Qontainer;
- Salvare le modifiche e il database;
- Attuare modifiche ai componenti;
- Mostrare le specifiche corrette dei vari componenti;
- Caricare il database ed inserire i componenti nelle ComboBox adatte.

6. Formato Files

Per il progetto sono stati usati files in formato JSON per salvataggio e caricamento dei documenti. È stato scelto questo tipo di formato perché, rispetto ad altri linguaggi come XML, risulta più leggibile e di facile utilizzo sacrificando, però, la quantità di tipi di dati interpretabili. Ai fini del progetto, comunque, risulta sufficiente la manipolazione degli elementi offerta da JSON che garantisce allo stesso tempo una buona velocità computazionale.

All'interno della cartella di consegna è presente:

- un file *database.json* che contiene un database iniziale di componenti;
- un file *build.json* che contiene una build di esempio;
- un file *databasevuoto.json* che contiene un .json vuoto che fallisce il caricamento.

7. Analisi tempistiche

- Progettazione preliminare (2 ore)
- Sviluppo della gerarchia e del container (10 ore)
- Apprendimento Qt (15 ore)
- Sviluppo della GUI (20 ore)
- Ottimizzazione e testing (3 ore)

8. Piattaforme di sviluppo

Il programma è stato sviluppato inizialmente su una macchina Windows 10 con Qt 5.12.2, dopodiché si è passati ai computer del labp036 e del labTA su un sistema operativo Ubuntu 16.04.3 LTS con Qt 5.5.1 ed infine il progetto è stato ultimato sulla macchina virtuale fornita con Ubuntu 18.04.1 LTS e Qt 5.9.5.

9. Comandi per compilazione ed esecuzione

Nella cartella dove è contenuto il progetto eseguire i seguenti comandi in ordine:

1. qmake
2. make
3. ./Qontainer