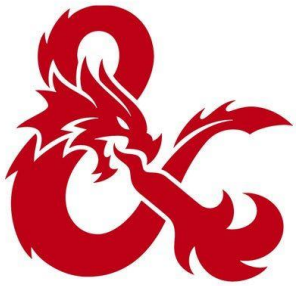


Dungeons & Data

PROGETTO DI PROGRAMMAZIONE AD OGGETTI

Anno Accademico 2021/2022

Realizzato da
Davide Vitagliano 1225414

UNIVERSITÀ DEGLI STUDI DI PADOVA

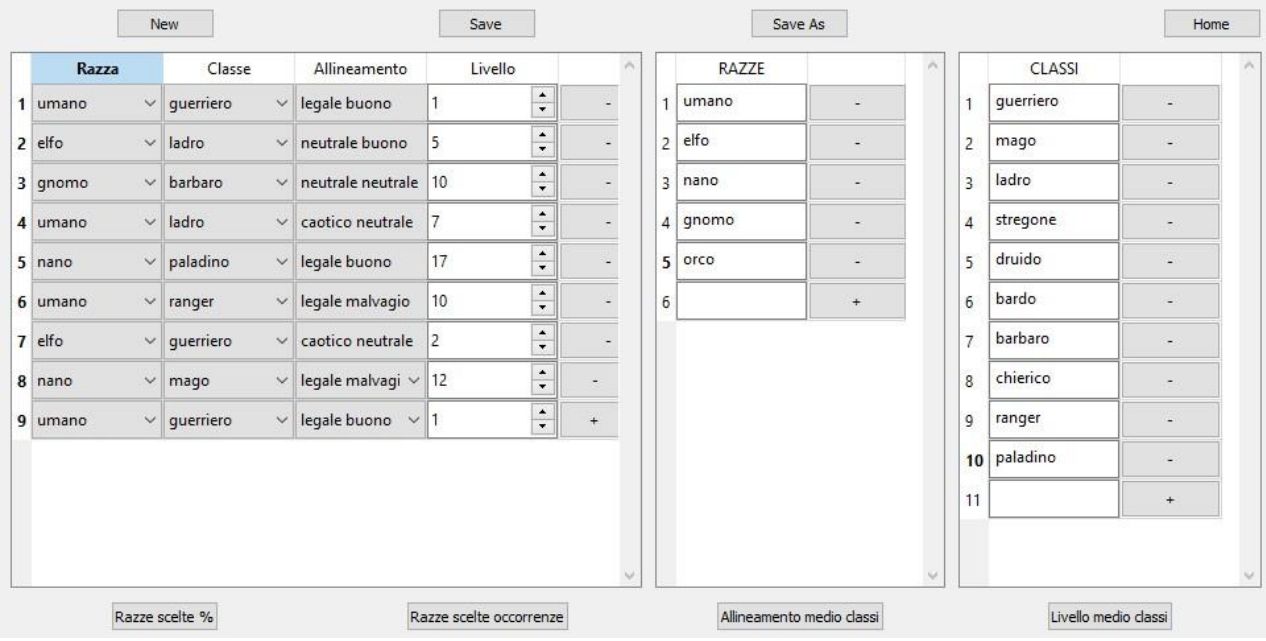
1. **Introduzione**

Dungeons & Dragons è un gioco di ruolo fantasy creato da Gary Gygax e Dave Arneson, pubblicato nella sua prima edizione nel gennaio 1974 e con l'uscita dell'edizione 3.5 nel 2003, guadagna il titolo di gdr più famoso del mondo. Ormai arrivati alla quinta edizione del gioco, milioni di giocatori hanno creato altrettanti personaggi, perciò si vuole formulare alcune statistiche sulle preferenze dei giocatori. In particolare si tiene conto delle razze, delle classi, degli allineamenti e dei livelli dei personaggi creati. Per poi costruire dei grafici per conoscere la preferenza di scelta delle razze, il livello medio a cui arrivano i personaggi, diviso per classe e quale allineamento i giocatori preferiscono, in media, per ogni classe.

2. **Descrizione programma e manuale d'uso GUI**

Dungeons & Data è un programma utile per la statistica, fornisce un applicativo con cui registrare e visualizzare i dati dei personaggi di D&D, inoltre rielabora questi dati e mostra dei grafici utili.
Una volta aperto il programma la prima schermata Home offre la possibilità di iniziare un nuovo progetto senza dati oppure di aprirne uno già esistente prendendo i dati da un precedente progetto salvato su file.
Una volta aperto un progetto viene mostrata la schermata Admin in cui sono presenti tre strutture tabellari su cui inserire i dati.
La prima, a sinistra, rappresenta i Record ovvero i dati dei personaggi creati. La seconda, in centro, rappresenta la lista delle razze disponibili. La terza, a destra invece, rappresenta la lista delle classi disponibili.
Nella tabella dei record si può selezionare la razza e la classe tra quelle disponibili; l'allineamento tra una lista predefinita e, infine, il livello in un range tra 1 e 20.
Una volta immessi i dati, con i pulsanti in alto si può salvare il progetto oppure salvarlo con un altro nome. Sono presenti altri due pulsanti con cui aprire un nuovo progetto o tornare alla schermata Home.

Schermata 1 - Home



Schermata 2 – Admin

Il programma infine elabora questi dati ed offre all'utente degli grafici utili da cui si possono trarne delle considerazioni sui personaggi creati.

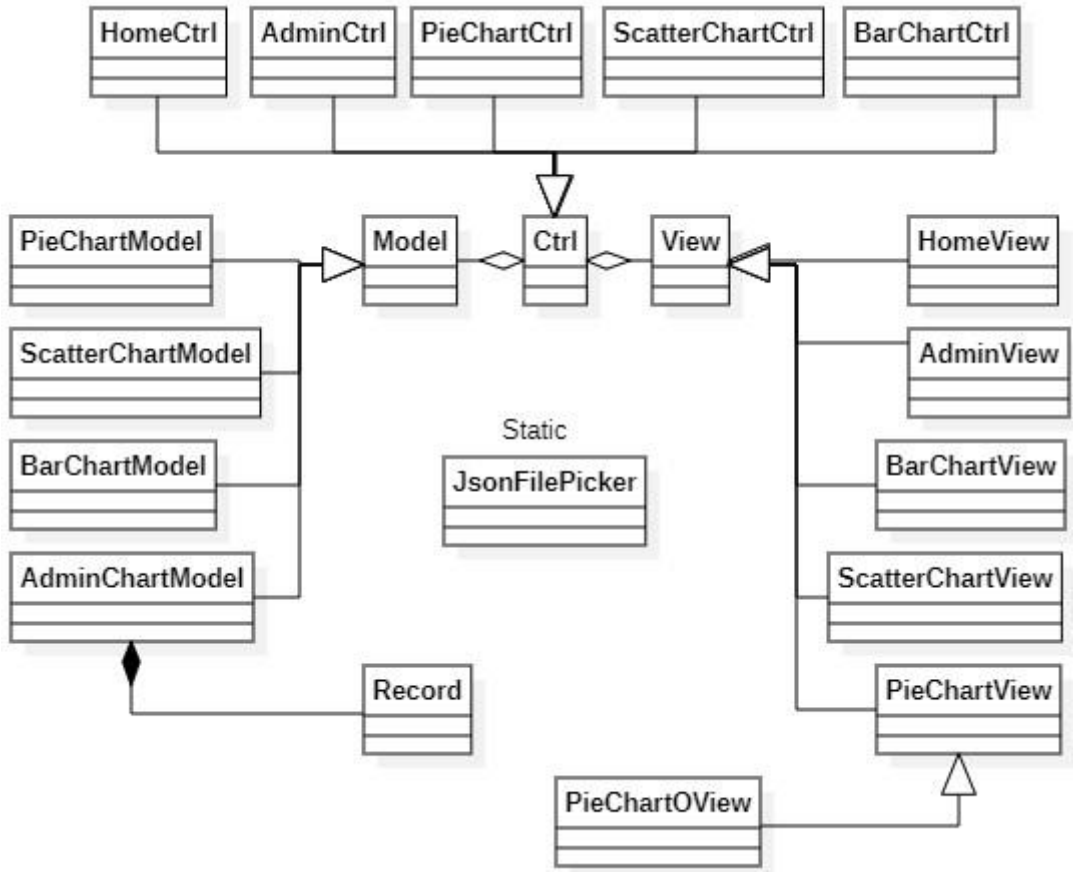
3. Funzionalità

- Inserimento e controllo dei dati immessi mantenendo un'integrità referenziale tra Record-Razze e Record-Classi.
  La tabella dei Record, ovvero dei personaggi, e la tabella delle razze sono in integrità referenziale tra loro e l'applicazione gestisce l'aggiunta, la modifica e l'eliminazione di una nuova razza gestendo anche i rispettivi Record collegati. Lo stesso procedimento avviene con la tabella delle Classi.
  Viene aggiornato in tempo reale il ComboBox (SelectBox) nella lista dei Record con le effettive modifiche effetuate nella lista delle razze e quella delle classi. Vengono, inoltre, gestite la lista delle razze quella delle classi in maniera tale che esse abbiano dei nomi univoci.

- Salvataggio del progetto su file, i dati vengono salvati in formato Json.
  Il nome del progetto è il nome del file ed esso ha una estensione Jsonj. Viene gestito il caso in cui si da un file non valido al programma e viene effettuato un controllo anche sul JSON che viene fornito.
  In particolare viene controllato che il JSON abbia come struttura almeno
  {"**classi**": [],"**razze**": [],"**records**": []}
  Il file può essere usato per riaprire il progetto in un secondo momento ed effettuare ulteriori modifiche oppure per essere usato con altri programmi.

- Elaborazione dei dati e visualizzazione di essi tramite dei grafici.
  Viene offerto un **PieChart** in 2 versioni con cui si può vedere la preferenza di scelta delle razze utilizzate sia in forma numerica che in forma percentuale.
  Successivamente è disponibile uno **ScatterChart** che rappresenta l'allineamento medio di ogni classe: sulle ascisse la componente di legalità e sulle ordinate la componente di bontà.
  L'ultimo grafico che viene fornito è un **BarChart** grazie al quale è possibile vedere per ogni classe il livello medio dei personaggi.

4. Gerarchia di tipi



Si è deciso di utilizzare il modello MVC per dividere parte Logica dalla parte Grafica. La classe base astratta "**Ctrl : public QObject**" ha il compito di essere Controller di una view (schermata). Un Controller ha una View e un Model e riceve segnali dalla View, li elabora, e ne ritorna il responso alla View oppure fa delle modifiche al Model di dati. Un Controller ha una lista di Slot collegati a segnali della View. Un Controller ha anche la possibilità di creare un altro Controller e ciò corrisponde ad aprire una nuova schermata.
La classe base astratta "**View: public QWidget"** ha il compito di essere la parte visiva di una specifica schermata, contenendo dei Widget e fornendo segnali e metodi al Controller, il quale in base alle sue operazioni ne modifica lo stato e ne aggiorna il Model.
Un "**Model"** è la rappresentazione logica di dati per una specifica schermata o funzionalità.
Tutte le schermate della applicazione hanno specifici **Ctrl** che sono classi derivate ed introducono specifiche funzioni per ogni schermata in questione.
Per esempio "**void AdminCtrl::onRazzeTableAdded(const QString &r);"** offre uno SLOT nella schermata Admin, per la gestione di una razza aggiunta alla lista di razze. Invece "**void HomeCtrl::onOpenDungeon() const;"** offre uno SLOT nella schermata Home, per l'apertura di una collezione di dati esistente.
Lo stesso ragionamento viene effettuato anche per le **View** le quali hanno una specifica implementazione per ogni schermata con ciascuna i propri Widget, segnali e i propri metodi.
Infine, ogni schermata può avere un suo **Model**. La gerarchia di classi si può capire con molta facilità guardando i costruttori dei Controller in quanto essi descrivono con precisione di cosa ha bisogno ogni schermata.

```
HomeCtrl(HomeView*, Ctrl*);
AdminCtrl(AdminView*, AdminModel*, Ctrl*);
PieChartCtrl(PieChartView*, PieChartModel*, Ctrl*);
LineChartCtrl(LineChartView*, AdminModel*, Ctrl*);
BarChartCtrl(BarChartView*, BarChartModel*, Ctrl*);
```

Si ricorda infine che oltre al meccanismo di distruzione di Qt attraverso i parent, si sono utilizzati anche i distruttori virtuali profondi ridefiniti dove servisse e aggiungendo al loro normale comportamento anche la rimozione del parent prima della distruzione in maniera da non creare conflitti con il distruttore di Qt (doppio delete su un ptr).
Si è infine deciso di utilizzare la classe statica **JsonFilePicker** che offre solo metodi statici e quindi viene usata all'occorrenza per la lettura e la scrittura di file. Si è optato per questa soluzione perché non serviva istanziare questa classe ad ogni occorrenza perché offre solamente una sovrastruttura al meccanismo di lettura di file di Qt.

5. **Uso di Polimorfismo**
   - **Distruttori Polimorfi**

Sono stati implementati i distruttori virtuali polimorfi, un esempio è quando si chiude una schermata, in particolare quando si cancella il Controller:

```
Ctrl::~Ctrl() {
   setParent(nullptr);
   for(auto child : children())
      delete child;
   delete view;
   delete model;
}
```

In questo caso viene invocato il distruttore sulla **view** e sul **model** e non importa quale è il vero tipo dinamico di questi due puntatori, vengono comunque eliminati correttamente dallo Heap.

   - **closeEvent**

Quando si preme sul pulsante **X** di una finestra **View** essa essendo un **QWidget** invoca automaticamente il metodo

```
virtual void QWidget::closeEvent(QCloseEvent* event);
```

Esso è in overriding in:

```
void AdminView::closeEvent(QCloseEvent* event) override;
void HomeView::closeEvent(QCloseEvent* event) override;
void View::closeEvent(QCloseEvent *event) override;
```

In base al tipo dinamico di una view viene invocato la giusta versione del metodo che effettua diverse operazioni come chiudere la schermata chiedendo una confermalj con un PopUp a schermo, oppure effettuare delle operazioni di pulizia prima.

- **setViewTitle**

È stato definito:

```
virtual void View::setViewTitle(const QString& title);
```

Questo metodo serve a impostare alla schermata un Titolo da vedere sulla finistra del Desktop.
La sua prima implementazione all'interno di **View** si limita solamente a trascrivere la stringa title sulla finestra.
Esso è in overriding in :

```
void AdminView::setViewTitle (const QString &title) override;
```

Questo metodo non si limita a trascrivere la stringa title sulla finestra ma aggiunge anche l'etichetta "Dungeon :" davanti al titolo. Esso è infati ridefinito per la **AdminView** la quale ha necessità di specificare che si tratta di un Dungeon (collezione di personaggi).
L'invocazione polimorfa avviene all'interno del **AdminCtrl** :

```
view->setViewTitle(last);
```

Se in futuro ci fosse una evoluzione della **AdminView** questo metodo in overriding potrebbe implementare altre funzionalità quali modificare anche l'icona in base al title , etc.

- **onViewClosed**

Quando una **View** viene chiusa essa emette il segnale: "**void viewClosed() const;**"
Il Ctrl connette a questo segnale uno SLOT virtuale puro.

```
connect(view,SIGNAL(viewClosed()),this,SLOT(onViewClosed()));        // this == const Ctrl*
```

Lo SLOT è:

```
virtual void Ctrl::onViewClosed() const = 0;
```

Questo SLOT è in overriding in tutte le Classi derivate di Ctrl, ecco alcuni esempi:

```
void HomeCtrl::onViewClosed() const override;
void AdminCtrl::onViewClosed() const override;
void HomeCtrl::onViewClosed() const override;
```

Una volta chiusa una View questo SLOT decide cosa fare dal **Ctrl** ed in base alla sua implementazione e al tipo dinamico del **Ctrl** possono essere eseguite diverse operazioni in base alla specifica funzionalità della schermata.

- **PieChartView**

Per la View **PieChartView** è stato ipotizzato un particolare caso di estensibilità in cui la classe veniva reimplementata con una classe derivata :

```
class PieChartOView : public PieChartView
```

È stato implementato il metodo virtuale :

```
virtual void PieChartView::applyGraphics();
```

Esso è in overriding in:

```
void PieChartOView::applyGraphics() override;
```

6. Descrizione formati di input/output

Come anticipato precedentemente il programma offre come funzionalità la possibilità di salvare i dati del progetto sotto forma di file.
I dati immessi da utente tramite tastiera vengono interpretati come Modelli, essi sucessivamente, se richiesto, vengono salvati su file sotto forma di Json. Il file Json ha una struttura ben specifica e l'unico test semantico che viene effettuato è che esso assomigli a {"**classi**": [], "**razze**": [], "**records**": []}.
Viene rigettato ogni altro tentativo di imettere dei File con altre estensioni, che non siano Json oppure che non abbiano una struttura simile a quella su mostrata.
Attenzione al fatto che non sono stati implementati ulteriori meccanismi di protezione e si potrebbe andare incontro a undefined behaviour se si passano file Json alterati esternamente con una struttura,specialmente per i records, diversa.
Il file salvato può essere successivamente riutilizzato per riaprire il progetto in altre esecuzioni del programma.

7. **Istruzione compilazione ed esecuzione**

Il Progetto è fornito insieme al file **Progetto.pro**
Deve essere uttilizzato questo file .pro e non se ne devono generare altri.
Al progetto servono i sudetti pachetti Qt:

qt5-default
libqt5charts5-dev

Le istruzioni per la compilazione sono le seguenti :

qmake Progetto.pro → make

8. Ore di lavoro richieste

Analisi preliminare del problema	1 ore
Progettazione GUI e funzionalità	2 ore
Apprendimento libreria Qt	5 ore
Creazione View GUI - View Home/Admin - Creazione Grafico PieChart - Creazione Grafico ScatterChart - Creazione Grafico BarChart	 7 ore 2 ore 4 ore 3 ore
Creazione Controller - Modello	15 ore
Scrittura su file e salvataggio progetto	6 ore
Debuging testing	5 ore
TOTALE	**50 ore**

9. Ambiente di Sviluppo

L'IDE di sviluppo usato è stato Qt Creator.
Lo sviluppo principale è stato effettuato sulla piattaforma :

Sistema Operativo	Windows 10
Compilatore	MinGW 7.3.0 64-bit
Versione Qt	Qt 5.12.0 (compatibile con Qt 5.9.5)

Inoltre è stato utilizzato lo strumento GIT per usuvruire della feature di version control.