Progetto di Linguaggi di Programmazione:Metodologie di Programmazione

Federico Cicchi   Luca Cinti   Lorenzo Masetti

21 maggio 2002

Indice

1 Descrizione del progetto

Il progetto consiste nella realizzazione di un quiz interattivo composto da diversi tipi di gioco.

Il nome del programma è **Tom**, dalle iniziali di "The Object Millionaire" e dal nome del presentatore del gioco, che è il famoso gatto dei cartoni animati.

Particolare attenzione è stata posta alla possibilità di poter aggiungere nuovi giochi in maniera semplice e con poche modifiche. Come esempio abbiamo inserito quattro diversi tipi di gioco:

- Gioco con domande
- Passaparola
- Ruota della fortuna
- Il pranzo è servito

A sua volta il gioco con domande può essere formato da domande di diverso tipo; nella nostra implementazione abbiamo inserito come esempio le domande a risposta chiusa, le domande con immagine e le domande con scelte da ordinare. Il progetto è stato sviluppato interamente in C++. La parte grafica è stata realizzata utilizzando le librerie SDL[1] (Simple DirectMedia Layer). La parte audio è stata realizzata utilizzando la libreria FMOD[2].

2 Analisi del diagramma UML e giustificazione delle scelte di progetto

2.1 Il diagramma UML del progetto

Questo diagramma è stato realizzato utilizzando il programma *Together*.

2.2 Quiz, Gioco e Domanda

La classi su cui si sviluppa tutto il progetto sono la classe `Quiz`, e le classi astratte `Gioco` e `Domanda`. Il `Quiz` è composto da un vettore di puntatori ad oggetti di tipo `Gioco`. È così possibile sfruttare il binding dinamico e mandare in esecuzione a runtime il gioco appropriato. Il metodo `play()` manda in esecuzione i giochi che sono presenti nel vettore (il campo `_playlist`). Il metodo `addGioco()` permette di inserire un nuovo gioco nella `_playlist`.

[1] Scaricabili dal sito http://www.libsdl.org
[2] Scaricabile dal sito http://www.fmod.org

Una specializzazione della classe Gioco, di nome `GiocoADomande`, è un aggregato di domande che crea un collegamento tra la gerarchia dei giochi e quella delle domande. Il metodo `play()` di `GiocoADomande` manda in esecuzione una serie di domande.

2.3 Criteri generali

Il primo importante criterio che abbiamo seguito è stato quello di separare l'implementazione dei vari giochi e delle domande dalla loro interfaccia grafica. A tale scopo abbiamo realizzato per ogni gioco e per ogni domanda una classe base che ne definisce l'implementazione e una classe derivata (il cui nome è lo stesso della classe base preceduto da X) che ridefinisce soltanto i metodi che implementano l'interfaccia. Grazie a questa strategia abbiamo avuto la possibilità di testare il funzionamento dei vari giochi definendo una semplice interfaccia testuale e solo successivamente dedicarci ai dettagli grafici. L'altro vantaggio di questo approccio riguarda l'estendibilità. Infatti per implementare una versione alternativa del nostro progetto con una grafica completamente diversa sarebbe sufficiente ridefinire soltanto le classi che implementano l'interfaccia grafica (quelle con il nome che comincia per X). I metodi che gestiscono l'interfaccia e che devono essere ridefiniti nelle classi derivate sono dichiarati come `protected` dalla classe base perché non devono essere mai richiamati dall'esterno. Fanno eccezione i metodi delle classi derivate da 'Aperta' perché in quel caso i metodi devono essere chiamati direttamente dalla classe 'Passaparola'. Giochi e domande hanno spesso due tipi di costruttori: quelli che prendono come parametro direttamente i vari attributi che devono settare, e quelli che prendono come parametro il nome del file dal quale estrarre in modo casuale una domanda o un gioco.

2.4 La gerarchia dei giochi

La classe *Gioco* funge da interfaccia per i diversi tipi di gioco. Infatti, sebbene ogni gioco differisca dagli altri per diverse particolarità, vi sono due caratterisiche che li accomunano tutti:

- Tutti i giochi hanno un punteggio.
- Tutti i giochi vengono mandati in esecuzione.

Tenendo conto di questi fatti, abbiamo inserito nella classe `Gioco` un attributo `_score` che contiene il punteggio del gioco e un metodo virtuale puro `play()` che manda in esecuzione il gioco e restituisce il punteggio ottenuto dal giocatore. In questo modo è semplice estendere il quiz aggiungendo nuovi giochi: è sufficiente creare una classe derivata dalla classe `Gioco` ed implementare il metodo `play()`.

Le classi derivate di `Gioco` sono in relazione di uso o di contenimento con un'altra gerarchia di classi che derivano da `Domanda`, descritte nel paragrafo 2.5.

La classe `GiocoADomande` implementa un generico gioco a domande che risulta essere una lista di domande. In modo simile a quanto accade per la classe `Quiz`, l'implementazione della lista come un vettore di puntatori ad oggetti di tipo domanda permette di utilizzare il binding dinamico per mandare in esecuzione il tipo giusto di domanda. La classe derivata da `GiocoADomande` che gestisce la parte grafica deve ridefinire solo il metodo `displayScore()` che scrive il punteggio realizzato fino a quel momento.

La classe `Passaparola` realizza il gioco del "Passaparola". Un passaparola è una lista di 21 domande a risposta aperta con la proprietà che le risposte alle domande cominciano ognuna con una delle 21 lettere dell'alfabeto. Non abbiamo scelto, anche se potrebbe sembrare opportuno, di derivare la classe `Passaparola` da `GiocoADomande` perché i vantaggi dell'ereditarietà non sarebbero stati significativi. Abbiamo preferito invece implementare il passaparola in modo più efficiente. Infatti nel gioco a domande la lista delle domande è implementata come un `vector`, mentre nel Passaparola, dove la lunghezza della lista è fissa, abbiamo utilizzato un semplice array.

La classe `Ruota` realizza il gioco della "ruota della fortuna". Ha come attributo un oggetto di tipo domanda aperta dove sono memorizzate le due stringhe che definiscono questo gioco (la frase da indovinare e la definizione che aiuta il giocatore). Molte funzioni caratteristiche della ruota della fortuna possono essere implementate con una *delegation* alla corrispondente funzione della classe `Aperta` come per esempio nel caso del metodo `getDefinizione()`. La necessità di distinguere la parte di input/output da quella di controllo ha portato ad avere molti metodi nella classe Ruota. I metodi relativi all'input/output sono riscritti nella classe `XRuota` che gestisce l'interfaccia grafica.

La classe `Pranzo` realizza il gioco "Il pranzo è servito". In questo gioco si considera un insieme di portate che il giocatore deve completare per vincere. Al giocatore viene fatta una serie di domande a risposta aperta e, ogni volta che risponde correttamente, viene estratta una portata casuale. Il gioco finisce quando tutte le portate sono state estratte. Il punteggio parte da un certo valore e diminuisce ogni volta che si risponde a una domanda, in modo che chi completa il pranzo con un numero minore di domande abbia un punteggio più alto. Da queste regole risulta evidente che il numero di domande non è fissato al momento della costruzione dell'oggetto ma varia in base allo svolgimento del gioco, per questo la domanda viene costruita quando serve in una variabile locale all'interno del metodo `estraiDomanda()`. Si ha quindi una relazione d'uso tra la classe `Pranzo` e la classe `Aperta`. Questa relazione non è rappresentata nel diagramma UML perché la classe `Pranzo` non ha un attributo di tipo `Aperta`. La classe `XPranzo` ridefinisce i metodi relativi all'input/output per gestire la parte grafica.

2.5 La gerarchia delle domande

La classe `Domanda` fattorizza tutti i metodi e gli attributi comuni ad una generica domanda che è composta da un testo e da un punteggio da assegnare in caso di risposta esatta. Il metodo virtuale purto `vai()`, analogo al `play()` dei giochi, esegue una domanda e restituisce il punteggio totalizzato con quella domanda. La forma della risposta cambia in base al tipo di domanda: una domanda a risposta chiusa contiene diverse risposte fra le quali deve essere scelta quella giusta; una domanda a risposta aperta contiene l'unica risposta giusta. Abbiamo cercato di rendere possibile l'aggiunta di ulteriori diversi tipi di domande, senza dover reimplementare la parte di codice relativa alle operazioni di base (inserimento del testo e del punteggio). Una volta definiti gli elementi base di una `Domanda`, è possibile creare diverse varianti specializzando ed aggiungendo eventuali nuovi attributi e nuovi metodi. Nella nostra implementazione abbiamo inserito come esempio due diverse varianti: le *domande a risposta aperta* e le *domande a risposta multipla*. Le domande a risposta aperta hanno un attributo di tipo stringa che contiene la risposta giusta alla domanda. Una ulteriore specializzazione delle domande a risposta aperta sono le *domande a risposta aperta con immagine*. Gli oggetti di questa classe contengono un attributo (`_fileimg`) che contiene il percorso del file immagine che deve essere visualizzato quando viene mandata in esecuzione la domanda. Ovviamente questa classe deriva dalla versione grafica della domanda aperta; in modalità testuale non è possibile visualizzare immagini. La classe `Multipla` è una classe astratta creata per fattorizzare un elemento comune a due particolari tipi di domande: le *domande a risposta chiusa* e le *domande con scelte da ordinare*. Entrambe sono infatti caratterizzate da un vettore di stringhe contenente le relative risposte possibili, con una piccola differenza:

- Nelle domande a risposta chiusa il vettore delle risposte contiene nella prima posizione la risposta giusta.
- Nelle domande con scelte da ordinare il vettore delle risposte contiene le scelte, ordinate nel modo corretto.

Successivamente, quando le domande devono essere mostrate all'utente, viene applicato l'algoritmo di shuffling al vettore delle risposte in modo da mescolarle.[3] Tutto questo mostra come sia possibile inserire nuovi tipi di domanda (sia per la modalità testuale che per la modalità grafica) facilmente e senza ulteriori sforzi programmativi dovuti alla riscrittura delle parti in comune. L'uso dei puntatori permette poi di sfruttare appieno il polimorfismo ed il binding dinamico; non è necessaria nessuna modifica al codice delle classi `Gioco` e `Quiz` quando vengono aggiunti nuovi tipi di domande.

2.6 Le altre classi

Al di fuori delle due gerarchie finora citate, le classi che completano il nostro progetto sono quelle che gestiscono la grafica e l'audio. La parte grafica è gestita dalla classe `Grafica` che realizza un'interfaccia ad oggetti per le funzioni principali della libreria SDL. La classe `Grafica` è una classe statica perché deve gestire lo schermo che è chiaramente uno solo, e quindi non avrebbe senso creare più di un oggetto di tipo `Grafica`.

Le classi `Rettangolo` e `Bottone` (la seconda è una specializzazione della prima) gestiscono le aree sensibili, utilizzate per sapere in che punto dello schermo l'utente ha cliccato col mouse.

La parte audio è gestita dalle classi `SoundSystem` (statica), `ModAudio`, `CampAudio`, `Sample`, `Stream`.[4] Queste classi costituiscono un'interfaccia alla libreria audio FMOD[5].

La classe `Generalutils` è un'altra classe statica che raccoglie una collezione di metodi di utilità.

La classe `Tom` avvia il programma e gestisce l'interfaccia utente per la scelta dei vari giochi. Nel suo costruttore vengono inizializzati la grafica e l'audio; il metodo `start()` fa partire il gioco.

La classe `Shuffle` implementa un vettore che ha la proprietà di contenere una permutazione dei numeri interi da 0 a $n-1$, se n è la lunghezza del vettore. Questa classe è utile per rimescolare le risposte alle domande.

3 Implementazione delle classi

In questa sezione presentiamo più approfonditamente le classi che costituiscono il nostro progetto. In particolare una domanda (quella a risposta aperta) ed un gioco (il passaparola) vengono presentati più in dettaglio.

[3] Per approfondimenti consultare il paragrafo 3.11

[4] Le classi sono state inserite all'interno del namespace `EricBaginov`, nickname di uno dei tre studenti.

[5] Scaricabile dal sito http://www.fmod.org

3.1 La classe Quiz

La classe `Quiz` implementa una lista di puntatori a oggetti di tipo `Gioco` attraverso un `vector`. L'implementazione di questa classe è banale: il Quiz viene costruito inizialmente vuoto, il metodo `addGioco` permette di aggiungere un nuovo gioco.

Il metodo `play()` di Quiz richiama il metodo `play()` di ognuno dei giochi contenuti nella playlist, grazie al binding dinamico viene selezionato il gioco giusto.

```
int Quiz::play() {
    for (int i=0; i<_playlist.size(); i++)
    {
        displayScore();
        _totalscore+=_playlist[i]->play();
    }
    return _totalscore;
}
```

3.2 La classe Chiusa

Questa classe realizza la domanda a risposta chiusa che è una specializzazione della classe Multipla. L'esecuzione di questo tipo di domande prevede la visualizzazione della domanda e delle risposte in un ordine casuale. Il rimescolamento delle risposte è realizzato con la classe `Shuffle`. Uno shuffle viene passato come parametro al metodo `display` che restituisce la risposta selezionata dall'utente. Si controlla quindi la correttezza della risposta dell'utente. Poiché nella memorizzazione interna la risposta giusta viene mantenuta sempre nella prima posizione dell'array, questo controllo si riduce a verificare che nella posizione scelta dall'utente dello shuffle sia contenuto l'indice 0. Subito dopo si richiama il metodo `displayRisultato` che prende come parametro un booleano che indica se la risposta è giusta o no.

```
int Chiusa::vai()
{
    Shuffle s = Shuffle(getSize());
    int risposta = display(s);
    bool giusto = (s[risposta] == 0);
    displayRisultato(giusto);
    if (giusto)
        return getScore();
    else
        return 0;
}
```

La classe `XChiusa` definisce l'interfaccia grafica per la classe `Chiusa` ridefinendo i metodi `display` e `displayRisultato`.

3.3 La classe Ordinabile

Questa classe, che come la precedente è una specializzazione della classe Multipla, realizza la domanda con scelte da ordinare. L'esecuzione di questo tipo di domande prevede la visualizzazione della domanda e delle scelte da ordinare, in un ordine casuale. Il rimescolamento delle scelte è realizzato con la classe `Shuffle`. L'utente deve selezionare le scelte ordinandole nel modo che ritiene corretto. Poiché nella memorizzazione interna le scelte sono memorizzate nell'ordine corretto, per sapere se l'utente ha inserito la sequenza corretta, basta controllare che

```
_rispGiocatore(s[i]) = i
```

cioè che la risposta di indice `s[i]` del giocatore (dove `s` è il vettore `Shuffle`) sia uguale all'indice `i`. Subito dopo si richiama il metodo `displayRisultato` che prende come parametro un booleano che indica se l'ordine inserito è corretto oppure no.

```
int Ordinabile::vai()
{
    //mescolo le scelte
    Shuffle s = Shuffle(getSize());
    display(s);
    //controllo se la risposta e' giusta
    bool ok = true;
    int i = 0;
    while ((ok) && (i < sizeRispGiocatore()))
    {
        if (getRispGiocatore(s[i]) != i)
            ok = false;
        i++;
    }
    //visualizzo a video la risposta
    displayRisultato(ok);
    if (ok)
        return getScore();
        //se giusta restituisce il punteggio
    else return 0;
}
```

La classe `XOrdinabile` definisce l'interfaccia grafica per la classe `Ordinabile` ridefinendo i metodi `display` e `displayRisultato`. Nella modalità grafica è stata inserita la possibilità per l'utente di ritornare sui propri passi e cambiare l'ordine di selezione. Il giocatore deve semplicemente cliccare col mouse sull'ultima scelta inserita. Per confermare definitivamente l'ordine basta cliccare sul bottone 'Fatto'.

3.4 Una domanda in dettaglio: la classe Aperta

La classe `Aperta` eredita da `Domanda` e descrive domande aperte. Mostriamo qui il file header contenente la descrizione della classe.

aperta.h - *File header della classe Aperta*

CODICE:

```
#ifndef APERTA_H
#define APERTA_H
#include "domanda.h"
#include <iostream>
#include <fstream>
#include <string>

// Classe che definisce le domande aperte
class Aperta : public Domanda {
public:
  // Costruttori
  Aperta();
  Aperta(string testo, int score, string risposta);
  Aperta(string filename);


  // Mutatore della risposta
  void setRisp(string risp);
  // Accessore alla risposta
  string getRisp() const;
  // Restituisce l'iniziale della risposta
  char getInitRisp() const;

  // Visualizza il risultato
  virtual void display() const;
  // restituisce la risposta data
  virtual string readRisposta();
  // Controlla la correttezza della risposta
```

```
  bool checkRisposta(string rispPlayer);
  // Definisce il metodo vai() lasciato virtuale nella
     classe Domanda
  int vai();

private:
  // Risposta corretta
  string _risp;
};
#endif
```

3.4.1 I campi

L'unico campo della classe `Aperta` è `_risp`, che identifica la risposta corretta alla domanda. Il testo della domanda è ereditato dalla classe `Domanda`.

3.4.2 I costruttori

Il costruttore standard richiama il costruttore standard della classe base e inizializza la risposta con una stringa vuota. Oltre ad esso sono forniti altri due costruttori, il primo dei quali prende come parametro il testo della domanda, la risposta e il punteggio da attribuire al giocatore in caso di risposta corretta. L'altro costruttore costruisce una domanda aperta a partire da un file di testo, scegliendone una in modo casuale. Tale costruttore lancia un'eccezione se il file indicato non è presente. I file di testo che contengono le domande aperte sono organizzati in questo modo: ogni riga del file contiene una domanda diversa; il testo della domanda, la risposta e il punteggio sono separati dal carattere '#'.

3.4.3 Metodi principali

La classe `Aperta` fornisce l'accessore e il modificatore alla risposta. Inoltre sono forniti i metodi `display()`, `readRisposta()`, `checkRisposta(string rispPlayer)`, che rispettivamente visualizzano la domanda, leggono la risposta dell'utente e ne controllano la correttezza. I metodi `display()` e `readRisposta()` sono dichiarati virtuali perché saranno poi ridefiniti dalla classe derivata `XAperta`. La definizione del metodo `vai()`, lasciato virtuale dalla classe base, è semplicemente la combinazione dei tre metodi precendenti.

aperta.cc - *Implementazione del metodo vai()*

CODICE:

```
// Pone la domanda e verifica la risposta
int Aperta::vai() {
  display();
  string risposta = readRisposta();
  // Restituisce il punteggio se la risposta e giusta
  if (checkRisposta(risposta))
    return getScore();
  else
    return 0;
}
```

3.4.4 La classe derivata XAperta

La classe `XAperta` eredita da `Aperta` e ne ridefinisce i metodi per la visualizzazione grafica. Gli unici metodi riscritti sono `display()` e `readRisposta()` per la visualizzazione del testo della domanda e la lettura della risposta data dall'utente.

---

xaperta.h - *File header della classe XAperta*

CODICE:

```
#ifndef XAPERTA_H
#define XAPERTA_H
#include "aperta.h"
#include <string>
#include "grafica.h"
// Classe che definisce le domande aperte
class XAperta : public Aperta {
public:
  // Costruttore
  XAperta();
  XAperta(string testo, int score, string risposta);
  XAperta(string filename);

  // Visualizza la risposta corrente
  virtual void display() const;
  // restituisce la risposta data
  virtual string readRisposta();
private:
};
#endif
```

I costruttori di classe non fanno altro che richiamare i costruttori della classe base `Aperta`. Grazie all'ereditarietà quindi abbiamo potuto implementare una domanda aperta in modalità grafica con poche righe di codice.

3.4.5 La classe derivata XApertaimg

Dalla classe `XAperta` abbiamo ulteriormente derivato la classe `XApertaimg` che descrive una domanda aperta il cui testo fa riferimento ad una immagine. Tale classe ha un campo in più rispetto alla classe base che contiene il nome del file in cui è contenuta l'immagine da visualizzare. Inoltre ridefinisce il metodo `display()` e il costruttore da file, che dovrà estrarre anche il nome del file contenente l'immagine.

xapertaimg.h - *Il file header della classe XApertaimg*

CODICE:

```
#ifndef XAPERTAIMG_H
#define XAPERTAIMG_H
#include "xaperta.h"
#include <string>
#include "generalutils.h"

// Classe che definisce le domande aperte
class XApertaimg : public XAperta {
public:
  // Costruttore
  XApertaimg();
  XApertaimg(string testo, int score, string risposta,
     string fileimg);
  XApertaimg(string filename);

  // Visualizza la risposta corrente
  virtual void display() const;
private:
  string _fileimg;
};
#endif
```

Anche in questo caso grazie all'ereditarietà abbiamo potuto definire una domanda con immagine in pochissime righe di codice e riusando il codice già implementato dalle classi che precedono `XApertaimg` nella gerarchia delle classi.

3.5 La classe Ruota

Per realizzare questo gioco bisogna memorizzare due stringhe: la definizione che viene proposta all'utente e la frase che deve essere indovinata.

Molto codice che tipicamente deve essere utilizzato per gestire queste due stringhe è comune alla classe delle *domande a risposta aperta*. Si è utilizzando quindi un attributo `_domanda` di tipo `Aperta`.

La domanda a risposta aperta contiene un testo che corrisponde alla definizione e la risposta che corrisponde al testo da mostrare sul tabellone. Questa relazione d'uso permette di utilizzare alcuni metodi già scritti per la domanda aperta, come ad esempio `checkRisposta()`. L'implementazione della ruota con un attributo di tipo `Aperta` è nascosto dall'esterno, infatti logicamente la classe ruota fornisce due accessori separati alla definizione e al testo della frase. Questi accessori sono facilmente implementati con una delegation.

```
string Ruota::getDefinizione() const {
    return _domanda.getTesto();
}

string Ruota::getTesto() const {
    return _domanda.getRisp();
}
```

Si noti che quello che abbiamo chiamato testo della domanda è la definizione nella ruota, mentre il testo della ruota corrisponde alla risposta nella domanda.

È necessario inoltre tenere traccia di quali lettere il concorrente ha già chiesto per determinare ogni volta che si riscrive il tabellone quale parte della frase deve essere mostrata.

Quindi la classe ha un attributo di tipo array di booleani, di lunghezza pari al numero delle lettere considerate (26).

```
bool _chieste[NUMERO_DI_LETTERE]
```

Il metodo `chiesta` è un accessore a questo attributo:

```
bool chiesta(int i) const;
```

Al momento della costruzione dell'oggetto il vettore `_chieste` viene inizializzato tutto a **false**. Quando il giocatore sceglie una certa lettera l'elemento corrispondente nell'array viene posto a **true**

La classe `XRuota` ridefinisce tutti i metodi che definiscono l'interfaccia e contiene dei campi statici e costanti di tipo intero che contengono le dimensioni delle immagini utilizzate per disegnare il tabellone. Un metodo privato, `formatText`, permette di formattare il testo che deve apparire sul tabellone per fare in modo che le parole vadano a capo nei punti giusti.

3.6 La classe Pranzo

La classe `Pranzo` eredita direttamente dalla classe `Gioco`, ed implementa il gioco de 'Il pranzo è servito'.

Il gioco si svolge con le seguenti regole. Si parte da un punteggio iniziale, che abbiamo stabilito essere di 200 punti. Il giocatore deve rispondere a delle domande aperte. Ad ogni risposta sbagliata vengono tolti 5 punti. Ogni volta che risponde correttamente ad una domanda ha diritto a girare la ruota del pranzo è servito. Sulla ruota può uscire una portata (pasta, pollo, insalata, formaggio, dessert), il jolly, oppure la dieta. Se esce una portata questa viene assegnata al giocatore e vengono tolti dal punteggio i corrispondenti punti relativi alla portata, ovvero le sue calorie. Se esce la dieta al giocatore non vengono assegnati né punti né portate. Se esce il jolly invece si assegna la prima portata mancante e vengono sottratte dal punteggio le relative calorie. Il gioco termina quando il giocatore è riuscito ad avere almeno una porzione di ogni portata (cioè quando il pranzo è servito), oppure quando ha raggiunto un punteggio nullo o negativo. In tal caso gli si assegna comunque un punteggio nullo. Al termine del gioco vengono assegnati i punti rimasti.

I campi della classe `Pranzo` sono

```
string _nomiPortate[N_PORZIONI+1];
int _calorie[N_PORZIONI];
int _piatti[N_PORZIONI];
```

Sono tutti array di dimensione $N\_PORZIONI = 6$, visto che abbiamo 5 portate (pasta, pollo, insalata, formaggio, dessert) e la dieta, con l'eccezione di `_nomiPortate` che ha una posizione aggiuntiva per la dicitura 'Errori'. Il campo `_calorie` indica per ogni portata il numero di calorie corrispondenti, mentre `_piatti` indica il numero di piatti per portata che il giocatore è fino a quel momento riuscito ad ottenere.

La classe `Pranzo` mette a disposizione il solo costruttore standard, il quale pone a 0 tutti gli elementi di `_piatti` ed ai valori opportuni gli altri due campi.

Per tutti gli attributi sono definiti metodi accessori. La classe `XPranzo` ridefinisce i metodi virtuali che implementano l'interfaccia grafica.

3.7 Un gioco in dettaglio: la classe Passaparola

3.7.1 Regole del gioco

Il gioco consiste nel rispondere a 21 domande aperte, ognuna delle quali inizia con una lettera dell'alfabeto diversa. Per ogni lettera si può rispondere immediatamente oppure passare per rispondere poi al giro successivo. Se il giocatore risponde correttamente gli viene assegnato il punteggio relativo alla domanda, altrimenti passa alla domanda successiva senza ottenere nessun punto. Quando e se il giocatore raggiunge le 18, 20 o 21 risposte corrette ha diritto a dei bonus (che abbiamo stabilito essere di 20, 40 e 60 punti rispettivamente). Il gioco termina quando le domande sono esaurite oppure quando termina il tempo a disposizione (stabilito in 120 secondi).

passaparola.h - *File header della classe* ***Passaparola***

CODICE:

```
#ifndef PASSAPAROLA_H
#define PASSAPAROLA_H
#include "gioco.h"
#include "aperta.h"
#define NUMERO_LETTERE 21

class Passaparola : public Gioco {
public:
    Passaparola();
    Passaparola(int score);
    Passaparola(int score, int a18, int a20, int a21);
    // Carica un passaparola da file
    Passaparola(int score, int a18, int a20, int a21,string
        datadir);
    virtual ~Passaparola();
    // Accessori a _a18, _a20, _a21
    int getA18() const;
    int getA20() const;
    int getA21() const;
    // Restituisce la i-esima domanda
    Aperta* getDomande(int i) const;
    // Restituisce l'iniziale della iesima risposta
    char getInitRisp(int i) const;
    // Aggiunge una domanda al passaparola
    void addAperta(Aperta* domanda);
```

```
    // Fa partire il passaparola
    virtual int play();

protected:
    // Visualizza la prima lettera della risposta corretta
    virtual void initDisplay(int l) const;
    // Visualizza OK se la risposta e corretta
    virtual void displayRis(int sc, int l) const;
    // Schermata iniziale del gioco
    virtual void initDisplayGame() const;
    // Restituisce il numero di punti realizzati con la
       risposta grazie agli abbuoni
    virtual int showPremi(int indovinate) const;
    // Visualizzazioni a gioco terminato
    virtual int finalDisplay() const;

private:
    // Inizializza le variabili prima di giocare
    void initGame();
    // Controlla se la i-esima lettera deve essere posta
    bool getLetter(int i) const;
    // Pone la iesima lettere da non rifare
    void setLetter(int i);
    // Stabilisce quale e la prossima lettera da richiedere
    int nextToDo();

    // Array di domande del passaparola
    Aperta* _domande[NUMERO_LETTERE];
    // Risposte gia date
    bool _doneLetters[NUMERO_LETTERE];
    // Ultima domanda fatta
    int _ultima;
    // Punti da aggiungere a 18, 20, e 21 risposte esatte
    int _a18;
    int _a20;
    int _a21;
    // Risposte indovinate fino a quel momento
    int _indovinate;
};
#endif
```

3.7.2 I campi

Il campo principale della classe è `_domande` che individua l'array contenente le domande da proporre al giocatore. `_doneLetters` è invece un array di valori booleani che indicano se una risposta è già stata data o se deve essere ancora proposta. I campi `_a18`, `_a20` e `_a21` indicano quanti punti vanno assegnati al giocatore se raggiunge 18, 20 o 21 risposte corrette, mentre in `_indovinate` è posto il numero di risposte corrette fornite. Il campo `_ultima` è riservato per uso interno alla classe e indica quante domande sono state caricate nel passaparola, in maniera da impedire di giocare fino a quando al passaparola non è stata fornita una domanda per ogni lettera dell'alfabeto.

3.7.3 I costruttori

La classe `Passaparola` fornisce tre costruttori. I primi tre costrutturi, compreso quello standard, inizializzato tutti i campi della classe, ma lasciano vuoto il vettore delle domande. Sarà poi l'utente della classe a dover fornire le domande utilizzando il metodo `addAperta(Aperta* domanda)`, che prende un puntatore a una domanda aperta e lo pone nella giusta posizione dell'array. Non sarà comunque possibile iniziare a giocare fin tanto che non vengono fornite tutte le domande. Il quarto costruttore invece inizializza tutti i campi della classe ai valori forniti come parametro e carica le domande da file cercandoli nella directory passata come parametro. Se per qualche motivo la lettura da file di qualche domanda fallisce il costruttore rilascerà una eccezione, non raccogliendo quella proveniente dal costruttore delle domande aperte da file. Il costruttore descritto va a prendere per ogni lettera dell'alfabeto la domanda aperta nel file `Naperta.dat`, all'interno della directory passata come parametro, dove N è il numero corrispondente alla lettera presa in considerazione. Questo file contiene solo domande la cui risposta comincia per la lettera indicata da N. Riportiamo di seguito il codice dei due costruttori principali.

passaparola.cc - *Costruttori della classe* `Passaparola`

CODICE:

```
// Costruttore
Passaparola::Passaparola(int score, int a18, int a20, int
    a21) : Gioco(score), _a18(a18), _a20(a20), _a21(a21),
    _indovinate(0), _ultima(0)
{ }
```

```
// Costruttore da File
Passaparola::Passaparola(int score, int a18, int a20, int
   a21, string datadir) : Gioco(score), _a18(a18), _a20(a20
   ), _a21(a21), _indovinate(0), _ultima(0)
{
  // Si crea una domanda aperta per ogni lettera
  for (int i=0; i < NUMERO_LETTERE; i++) {
    string filename = datadir;
    filename = filename.append(Generalutils::toString(i
       +1));
    filename = filename.append("aperta.dat");
    // Creazione e aggiunta della domanda presa da file
    addAperta(new Aperta(filename));
  }
}
```

3.7.4 Metodi principali

La classe `Passaparola` fornisce vari accessori ai campi, quali `get18()`, `get20()`, `get21()` e `getDomande(int i)`. Gli accessori dei campi , `_ultima` e `_indovinate`, non sono stati forniti perché tali campi sono di uso interno. Di particolare importanza è il metodo `addAperta(Aperta* domanda)` che prende un puntatore ad una domanda aperta e la aggiunge al passaparola. Riportiamo qui il codice relativo all'implementazione del metodo.

---

passaparola.cc - *Il metodo addAperta(Aperta* domanda)*

CODICE:

```
// Aggiunge una domanda al passaparola ...
void Passaparola::addAperta(Aperta* domanda) {
     // ... se non sono gia tutte piene ...
     if (_ultima < NUMERO_LETTERE) {
     // ... la aggiunge in posizione _ultima
          _domande[_ultima] = domanda;
          _ultima++;
     }
}
```

Il metodo aggiunge una domanda al vettore delle domande se il passaparola non è completo. In questo contesto il campo `_ultima` indica quante domande sono state fornite al passaparola; si potrà eseguire il metodo `play()` solo quando il vettore è stato correttamente riempito.

La classe contiene inoltre metodi virtuali per la visualizzazione dei messaggi all'utente; messaggi che verranno poi ridefiniti dalla classe derivata `XPassarola` per la gestione della parte grafica.

Infine viene definito il metodo `play()` (lasciato virtuale dalla classe `Gioco`) di cui di seguito riportiamo il codice implementativo.

passaparola.cc - *Il metodo play()*

CODICE:

```
// Fa partire il passaparola
int Passaparola::play() {
  // Il passaparola inizia solo se sono presenti tutte le
     domande
  if (_ultima >= NUMERO_LETTERE) {
    initGame();
    initDisplayGame();
    do {
      initDisplay(_ultima);
      _domande[_ultima]->display();
      string risp = _domande[_ultima]->readRisposta();
      // Se il giocatore passa non devo fare niente perche
         il suo elemento in doneLetters e gia a false
      int thiscore = -1;
      if ((risp != "PASSA") && (risp != "")) {
        setLetter(_ultima);
        thiscore = _domande[_ultima]->checkRisposta(risp);
        // Se la risposta e corretta si incrementa il
           numero di domande esatte
        if (thiscore != 0) {
          _indovinate++;
          thiscore += showPremi(_indovinate);
        }
        // Se la risposta e corretta aggiunge il suo
           punteggio a quello del passaparola
        setScore(getScore() + thiscore);
      }
      displayRis(thiscore, _ultima);
      // Controlla la successiva da richiedere
      _ultima = nextToDo();
    } while ((_ultima != -1) && (Generalutils::getTimeleft
      () > 0));
    _ultima = NUMERO_LETTERE;
```

```
      return finalDisplay();
    } else {
      // se non sono presenti tutte le domande non si puo
         iniziare
      stringstream tmp;
      tmp << (NUMERO_LETTERE - _ultima);
      runtime_error e("Devi ancora fornire " + tmp.str() + "
         risposte per giocare");
      throw e;
    }
}
```

Il metodo `play()` implementa le regole di gioco del passaparola. Il gioco può iniziare solo se il vettore delle domande è stato completamente riempito; in caso contrario viene lanciata una eccezione il cui messaggio contiene il numero di domande mancanti. Dopo le inizializzazioni opportune e l'avvio del contatore del tempo, si procede entrando in un ciclo `do-while` e proponendo la prima domanda. In questo contesto il campo `_ultima` assume il significato di indicare il numero della domanda che si sta sottoponendo all'utente. Il metodo `InitGame()` inizializza il campo `_ultima` a 0 e tutti gli elementi del vettore `_doneLetters` al valore **false**. Dopo la lettura della risposta da parte dell'utente, se ne controlla il valore. Se il giocatore ha digitato 'PASSA' o ha premuto solo 'Invio' si passa alla domanda successiva senza assegnare nessun punteggio e si identifica la domanda come da riproporre all'utente; altrimenti si verifica la correttezza della risposta utilizzando il metodo `checkRisposta()` della classe `Aperta`. Se il numero di risposte corrette è 18, 20 o 21 si assegna il premio corrispondente. Si stabilisce poi l'indice della prossima domanda da sottoporre al giocatore attraverso l'uso del metodo `nextToDo()`, che scandisce in modo circolare l'array `_doneLetters` restituendo l'indice del successivo elemento uguale a **false**. Il gioco termina quando non ci sono più domande da proporre oppure quando il tempo concesso all'utente è teminato.

3.7.5 La classe derivata XPassaparola

La classe `XPassaparola` ridefinisce i metodi che implementano l'interfaccia grafica.

xpassaparola.h - *Il file di header di* `XPassaparola`

CODICE:

```
#ifndef XPASSAPAROLA_H
#define XPASSAPAROLA_H
#include "passaparola.h"
#include "xaperta.h"
#include "grafica.h"
#ifndef NOAUDIO
#include "EB_CampAudio.h"
#include "EB_Sample.h"
#include "EB_Stream.h"
#include "EB_ModAudio.h"
#include "EB_SoundSystem.h"
using namespace EricBaginov;
#endif

class XPassaparola : public Passaparola {
public:
    XPassaparola();
    XPassaparola(int score, int a18, int a20, int a21, int
       cx, int cy, int r);
    XPassaparola(int score, int a18, int a20, int a21);
    // carica da file
    XPassaparola(int score, int a18, int a20, int a21,
       string datadir);


protected:
    // Visualizza la prima lettera della risposta corretta
    virtual void initDisplay(int l) const;
    // Colora la lettera a seconda della correttezza della
       risposta
    virtual void displayRis(int sc, int l) const;
    // Disegna inizialmente tutte le lettere
    virtual void initDisplayGame() const;
    // Colora la i-esima lettera
    void coloraLettera(int i, string colore) const;
    // Visualizzazioni finali
    virtual int finalDisplay() const;
    // Visualizza i premi
    virtual int showPremi(int indovinate) const;

private:
    // setta le coordinate del centro e il raggio della
       ruota
```

```
    void setGeo();
    // Restituisce le coordinate della figura della i-esima
        domanda
    int getCoordX(int i) const;
    int getCoordY(int i) const;
    // Coordinate del centro su cui disporre la ruota
    int _cx;
    int _cy;
    // Raggio della ruota
    int _raggio;
    // Thread su cui visualizzare il tempo che passa
    //mutable SDL_Thread* th;
    #ifndef NOAUDIO
    mutable Stream _sigla;
    #endif
};
#endif
```

Per quanto riguarda la grafica, le figure delle lettere dell'alfabeto sono disposte su una circonferenza con centro nella finestra di visualizzazione e raggio opportuno e definito in base alle dimensioni della finestra stessa. A tale scopo si fa uso della funzione privata `setGeo()`, richiamata da ogni costruttore della classe.

xpassaparola.cc - *Il metodo setGeo()*

CODICE:

```
void XPassaparola::setGeo() {
  _cx = (Grafica::width() - 160)/2 + 160;
  _cy = (Grafica::height() - 160)/2;
  _raggio = (Grafica::height() - 160)/2 - 40;
}
```

La posizione della singola lettera è poi calcolata con l'uso delle funzioni seno e coseno ed in base ai valori del raggio e del centro della circonferenza su cui si devono porre le lettere. A questo scopo si utilizzano i metodi privati `getCoordX(int i)` e `getCoordY(int i)`, che restituiscono la posizione sullo schermo della i-esima domanda.

xpassaparola.cc - *I metodi getCoordX(int i) e getCoordY(int i)*

CODICE:

```
// Restituisce la coordinata x della i-esima lettera
int XPassaparola::getCoordX(int i) const {
  double k = (2*PI*i) / NUMERO_LETTERE;
  int x = (int) ((sin(k) * _raggio) + _cx);
  return x;
}

// Restituisce la coordinata y della i-esima lettera
int XPassaparola::getCoordY(int i) const {
  double k = (2*PI*i) / NUMERO_LETTERE;
  int y = (int) ((cos(k) * _raggio) + _cy);
  return y;
}
```

3.8 La classe Generalutils

Nella classe `Generalutils` sono raccolti alcuni metodi di utilità che devono essere utilizzati da più di una classe.

upperCase Prende come parametro una stringa e la restituisce tutta maiuscola.

toString Converte un intero in una stringa.

bool isVocale(char) Restituisce **true** se il carattere è una vocale.

int casuale(int n) Restituisce un numero casuale da 0 a n.

Inoltre nella classe `Generalutils` è memorizzato il nome della directory dove sono memorizzati i dati.

La classe `Generalutils` contiene anche il codice per la gestione del conto alla rovescia attraverso i thread forniti dalla libreria SDL. Nella classe Generalutils viene fornita una semplice interfaccia per gestire un timer, attraverso i metodi `startClock(int time)`, che fa partire il conto alla rovescia dal numero di secondi specificato, e `stopClock()`. Il tempo rimasto è memorizzato nell'attributo `_timeleft`, per accedere al quale vengono forniti un metodo accessore, `getTimeLeft()`, e un metodo modificatore, `setTimeLeft`.

3.9 La classe Grafica

La parte grafica utilizza la libreria SDL e due sue estensioni: *SDL_Image* (che gestisce le immagini di tipo `gif` e `jpg` mentre la SDL standard gestisce solo immagini bitmap) e *DFont* (che gestisce le font per scrivere facilmente del testo in modalità grafica). Di queste tre librerie due sono scritte in C e solo una (DFont) in C++. La classe `Grafica` è un'interfaccia al sottoinsieme delle funzionalità che abbiamo sfruttato di queste tre librerie. I metodi della classe `Grafica` permettono di visualizzare immagini, di scrivere del testo e di leggere l'input in modalità grafica. Inoltre questa classe gestisce una lista di aree sensibili al click del mouse. Le aree sensibili possono essere oggetti di tipo `Rettangolo` o `Bottone`. Un bottone è un rettangolo con in più un'immagine associata. Le aree sensibili vengono aggiunte attraverso i metodi `addArea` e `addBottone`. Il metodo `waitForClick()` attende che l'utente clicchi su un'area sensibile e restituisce l'area sensibile selezionata. La lista di aree sensibili è implementata con un `vector` di puntatori a oggetti di tipo `Rettangolo`, che per polimorfismo possono essere anche di tipo `Bottone`. Le aree sensibili possono essere eliminate dalla lista con il metodo `removeArea`.

Questa classe comprende anche la parte grafica del codice relativo al thread utilizzato nel passaparola. Poiché due thread concorrenti devono utilizzare la stessa risorsa (devono entrambi scrivere sullo schermo) la classe `Grafica` implementa anche la gestione della concorrenza con un semaforo. Anche i semafori sono strumenti messi a disposizione dalla libreria SDL.

3.10 Le classi per la gestione del sonoro

La classe `SoundSystem` è una classe statica che gestisce il sistema sonoro del computer per l'esecuzione di file audio. Il suo scopo è quello di attivare e disattivare il sistema audio e di gestire l'insieme dei campioni audio che vengono suonati in un certo istante.

La classe `ModAudio` gestisce la riproduzione di *moduli* audio, cioè file con estensione `.MOD .S3M .XM .IT .MID` e `.RMI`.

La classe `CampAudio` gestisce la riproduzione di *campioni* audio, cioè file con estensione `.RAW .WAV .MP2 .MP3` e `.WMA`. Questa classe è astratta perché l'esecuzione effettiva dei campioni audio viene gestita dalle due classi derivate `Sample` e `Stream`. In genere un campione audio viene caricato interamente in memoria prima di essere riprodotto. Poiché vi possono essere campioni audio di grosse dimensioni, il cui caricamento in memoria richiede alcuni secondi, si è preferito scomporre in due parti la loro gestione.

La classe `Sample` è stata idealmente concepita per la riproduzione di piccoli campioni audio (in genere di dimensioni inferiori ad 1 Mb); questa classe carica completamente il campione in memoria prima di riprodurlo.

La classe `Stream` non aspetta che il campione audio venga caricato completamente in memoria ma lo riproduce contemporaneamente al suo caricamento.

3.11 La classe Shuffle

Abbiamo considerato lo Shuffle come un tipo di dato astratto definito come una lista di lunghezza n che contiene una permutazione casuale dei numeri interi da 0 a $n-1$. Il costruttore di uno Shuffle prende come parametro la dimensione dello Shuffle. Al momento della costruzione lo shuffle viene automaticamente riempito con il classico algoritmo dello shuffling. Le altre operazioni definite sullo shuffle sono il metodo `redo()` che rimescola gli elementi dello shuffle, il metodo `size()` che ne restituisce la dimensione e l'accesso all'i-esimo elemento. Per quest'ultima operazione abbiamo ridefinito l'operatore [ ] in modo che restituisca l'opportuno elemento del vettore, ma con accesso solo in lettura perché una scrittura nello shuffle violerebbe la sua proprietà fondamentale.

```
int Shuffle::operator[] (int x) const {
    return _v[x];
}
```

Per implementare lo Shuffle abbiamo utilizzato un `vector` di interi, memorizzato nel campo `_v`.

4 Manuale utente

4.1 Installazione

Requisiti

Il gioco è stato realizzato su un sistema Linux con il compilatore C++ della GNU, gcc. Per funzionare ha bisogno delle librerie SDL, SDL_Image, DFont e, se interessa anche il sonoro, FMOD. Tutte queste librerie sono software libero e possono essere scaricate gratuitamente a partire dai siti `http://www.libsdl.org` e `http://www.fmod.org`.

Compilazione

Una volta installate queste librerie si possono estrarre i sorgenti e compilare con il comando `make`. Se non si dispone di una scheda audio o comunque non si vuole sentire la colonna sonora del gioco si può compilare con il comando `make noaudio`. Una volta compilato il gioco lo si fa partire con il comando `./tom`.

Il progetto è stato compilato con successo su varie distribuzioni Linux: Debian Woody, Red Hat 7.1 e Mandrake 7.2.

File di configurazione

Il file di configurazione si chiama `tom.cfg`. Questo è il file di configurazione di default:

```
800
600
24
FULLSCREEN
data/
```

Il significato delle righe è il seguente: largezza dello schermo, altezza dello schermo, profondità (numero di colori), titolo della finestra, directory dei file che contengono le domande. Se nella riga del titolo della finestra si specifica FULLSCREEN il gioco non apparirà in una finestra ma a tutto schermo. Le modifiche nel file di configurazione hanno effetto non appena si fa ripartire il gioco.

4.2 Come giocare

Alla schermata iniziale cliccare su ”Gioca” per far partire il gioco. Nella schermata successiva si può scegliere quali giochi fare. Si può scegliere tra: domande a risposta chiusa, ordinare gli elementi di una lista secondo i criteri indicati, domande a risposta aperta con immagine, passaparola, ruota della fortuna, il pranzo è servito. Una volta scelto il numero di giochi di ogni tipo cliccando sulle freccette sotto al simbolo di ogni gioco, cliccare su ”Fatto” per iniziare a giocare.

Le interfacce dei vari giochi sono molto semplici. I punteggi di tutti i giochi vanno a sommarsi per dare un punteggio totale.

Il primo gioco consiste in una semplice domanda a risposta chiusa: viene mostrata una domanda e le possibili risposte. Bisogna cliccare sulla risposta giusta.

Nel gioco che prevede di ordinare una lista, se ci si accorge di avere sbagliato, prima di cliccare su ”Fatto”, si può annullare la scelta fatta cliccandoci sopra un'altra volta.

Il terzo gioco prevede di rispondere a una domanda a risposta aperta che si riferisce all'immagine che viene mostrata.

Nel Passaparola si deve rispondere a 21 domande a risposta aperta, le cui risposte cominciano ognuna con una diversa lettera dell'alfabeto. Per rispondere a tutte le domande si hanno a disposizione due minuti. Se non si conosce la risposta a una certa domanda si può passare alla domanda successiva premendo invio, la domanda verrà eventualmente ripetuta al giro successivo. Se si risponde correttamente a 18, 20 o 21 domande si ottiene un bonus rispettivamente di 20, 40 e 60 punti.

Le regole della Ruota della Fortuna sono queste: ad ogni passo il giocatore può girare la ruota o dare la soluzione. Dalla ruota può uscire un punteggio che va da 10 a 50 punti, nel qual caso si può giocare la prossima mossa, oppure la casella nera. Se esce la casella nera si perdono tutti i punti realizzati fino a quel momento e la ruota viene girata nuovamente. Se il punteggio totale è di almeno 100 punti si può comprare una vocale. Quando si compra una vocale si perdono 100 punti e non si aumenta il punteggio, qualsiasi sia il numero di lettere trovate. Se invece si sceglie una consonante si totalizza un punteggio pari ai punti usciti dalla ruota moltiplicato per il numero di lettere trovate. Quando si dà la soluzione si vince un bonus di 20 punti. La strategia vincente è quindi un compromesso tra il rischio di azzerare il punteggio che si ha ogni volta che si gira la ruota e l'obiettivo di realizzare un punteggio alto che suggerisce di girare la ruota il più possibile anche quando già si conosce la soluzione.

Nel gioco "Il Pranzo è servito" si deve rispondere a delle domande a risposta aperta. Ogni volta che si risponde correttamente viene estratta casualmente o una delle portate, o un jolly che vale per la prima portata mancante, o la dieta. L'obiettivo del gioco è quello di completare tutte le portate. Il punteggio parte da 200 punti e diminuisce di un punteggio pari alle calorie della portata che è uscita ogni volta che si risponde a una domanda, in modo che chi completa tutte le portate con un minor numero di domande ottiene un punteggio più alto. Se si arriva a 0 senza aver completato tutte le portate il gioco finisce, ovviamente con punteggio nullo.

Una volta completati tutti i giochi viene mostrato il punteggio totale e si ritorna alla schermata iniziale dove si può scegliere se giocare ancora o uscire dal gioco.