**Relazione progetto Programmazione a Oggetti**

Menon Giovanni, mat. 2034301

Endi Hysa, mat. 2046424

**Titolo**: Arcane Oracle

**Introduzione**

In questo momento storico in cui l’intelligenza artificiale si diffonde in ogni ramo di scienza e tecnologia, non solo in relazione all’informatica ma anche in campi molto distribuiti, il nostro progetto vi fa riferimento. Arcane Oracle è difatti un applicativo che permette la creazione di carte le cui immagine vengono generate tramite una AI. Questo permette di rappresentare qualsiasi fantasia dell’utilizzatore finale. In particolare la generazione delle carte avviene grazie all’utilizzo dell’API di DALL‧E 2 , un modello di Intelligenza Artificiale per la generazione di immagini creato da OpenAi. Questo avviene tramite una chiamata Curl contenente la descrizione dell’immagine che l’utente ha scritto. L’applicativo permette inoltre di scegliere tra i vari tipi di carte disponibili, ognuno con attributi e campi differenti, e permette anche la gestione e modifica di carte già realizzate in precedenza. Il tutto è organizzato in mazzi che contengono le varie carte e permetto il salvataggio di esse.

Un punto di forza del progetto è il container, rappresentante un mazzo di carte dell’utente, che è una rivisitazione di una lista doppiamente concatenata. È stato scelto questo tipo di struttura per facilitare la visita all’interno del mazzo e soprattutto per rendere più efficiente lo scorrimento dei vari oggetti in entrambe le direzioni.

Ho scelto questo progetto poiché l’idea di sviluppare una applicazione utilizzando una API rappresentava per me una sfida r5iguardante un argomento nuovo. Inoltre, sia io che il mio collega abbiamo pensato che con la gerarchia scelta si andavano a sfruttare a pieno le nozioni apprese sul polimorfismo in maniera non banale.

**Descrizione del Modello**

Il modello logico del progetto è caratterizzato da svariate componenti fondamentali: la gerarchia degli oggetti, (ovvero le carte ed i loro tipi derivati), il contenitore rappresentato dall’oggetto Mazzo ed infine la classe che implementa la chiamata all’API esterna.

Il diagramma UML, omesso per motivi di dimensione, è in allegato nella consegna e reperibile qui.

**Gerarchia**

Il progetto fa uso di una classe astratta *AbstractCard* che rappresenta una carta generica contenente i metodi e campi comuni a tutte le carte. Questa classe è composta da due classi derivate concrete e da una derivata pura che aggiunge metodi e attributi comuni per le altre tre classi concrete che la derivano. è possibile osservare tale gerarchia e tali attributi nel modello UML allegato.

I campi comuni sono il *nome*, il *costo* (che rappresenta il costo della carta in un ipotetico gioco) e l’*immagine*; sono inoltre presenti altri attributi come *save* che servono per tenere traccia se la carta è salvata o meno.

In particolare va osservata la scelta di utilizzare un campo (*b64url*) per salvare il percorso dell’immagine generata all’interno della directory del mazzo; questa scelta è stata intrapresa per evitare di appesantire l’oggetto *Card* salvando l’intera immagine in codifica B64. Inoltre, per evitare la presenza di *garbage* nel caso di carte eliminate, è stato creato un *garbage collector* che elimina le immagini delle carte non più presenti nel mazzo o non più salvate.

La gerarchia dunque, radicata in *AbstractCard* prevede le seguenti classi derivate concrete:

* *MonsterCard*, rappresentante la carta Mostro utilizzata per rappresentare una creatura.
* *ArtifactCard*, che rapprsenta una carta Artefatto, ovvero un oggetto che va a dare dei potenziamenti.

*EffectCard* è una classe derivata da *AbstractCard* e non implementa il suo metodo puro e di conseguenza anch’essa è considerata una classe Pura. Quest’ultima implementa il campo effect che ne fanno uso le sue tre derivate.

* *TrapCard*, che rappresenta una carta trappola.
* *SpellCard,* rappresentante una magia o un incantesimo.
* *FieldCard,* che rappresenta un terreno o una zona in particolare.

**Al di fuori della gerarchia**

Per quanto riguarda invece la classe relativa alla **generazione dell’immagine** è stato scelto di rendere la chiamata all’API un metodo di un oggetto. Questa scelta è stata fatta in modo da facilitare un possibile futuro cambio di API e rendere il progetto meno dipendente da questa specifica API e dinamico a possibili modifiche.

Si utilizza un file esterno in cui salvare la API KEY e leggerla per facilitare il cambiamento di essa, e si usa la libreria [libCurl](https://curl.se/) per effettuare la vera e propria richiesta.

In particolare la richiesta via Curl avviene in maniera asincrona all’interno del codice così’ da non creare attesa attiva da parte del programma durante il tempo di risposta. Questo avviene grazie all’implementazione e all’utilizzo dei metodi asincroni del framework Qt.

**Classe del Container**

Il contenitore del progetto è l’oggetto Deck. Esso contiene le varie carte in maniera polimorfa in una lista doppiamente concatenata realizzata appositamente. La classe inoltre fornisce una solida gestione del Mazzo e delle carte permettendo oltre che alla permanenza dei dati anche la visione, la ricerca e l’ordinamento delle carte all’interno del contenitore.

In particolare il contenitore implementa la libreria OpenSource JsonCpp per la lettura e scrittura di file in Json. Più precisamente i metodi Save e Load, Scrivono e Leggono il file contenente le carte salvate. Inoltre in questa classe è presente il Metodo *Garbage Collector,* già accennato in precendeza.

Infine grazie all’interfaccia realizzata è facile creare e utilizzare deck multipli completamente separati l’uno dall’altro. Ogni deck è infatti rappresentato da una propria directory contenente il file Json con il salvataggio, le immagini delle carte generate di quel mazzo e le carte complete di quel mazzo. Tutte le cartelle dei vari mazzi sono riunite all’interno della cartella Deck.

**Polimorfismo**

Il polimorfismo in questo progetto è presente non solo nella parte relativa alla gerarchia ma anche nella parte grafica, tramite l’accurata progettazione di classi astratte sia per i *Visitor* che per le *Pagine stesse*.

* ***Gerarchia di AbstractCard***
  + Metodo **Serialize** : Questo metodo si occupa, partendo da un puntatore a AbstractCard, di creare una stringa serializzata che rappresenta i campi dell’oggetto stesso. Ogni oggetto ridefinisce questo metodo aggiungendo i campi propri.
  + Metodo **Accept**: Seguendo dal Visitor Design Pattern permette, accettando un puntatore a visitor generico, la creazione polimorfa di widgets grafici.
* **Nella Parte Grafica**
  + **Gerarchia Grafica *AbstractCard***
    - Per la creazione della carta si è creato un Widget , *CardWidget* , da cui deriva una gerarchia identica a quella di *AbstractCard* in cui ogni Widget eredita e implementa metodi aggiuntivi dalla classe radice *CardWidget*.
    - In particolare si utilizza un CardWidget generico puntatore che ha tipo dinamico e tipo statico diverso. Richiamando in Maniera polimorfa i vari metodi nel tipo dinamico.
    - I Metodi Polimorfi sono :
      * ***Manual* :** Un metodo polimorfo in cui viene definito il manuale per ogni tipo di carta, che successivamente ogni classe derivata implementa.
      * ***GetInput* :** Metodo puro che viene implementato nelle varie classi discendenti. Permette la creazione degli oggetti della gerarchia ***AbstractCard***, ritornando un puntatore polimorfo alla classe *AbstractCard.*
      * ***Check* :** Un metodo polimorfo che controlla che tutti i campi per la creazione della carta sono stati compilati, ogni derivato implementerà i suoi campi esclusivi.
      * ***Generate* :** Metodo polimorfo in cui avviene la chiamata asincrona per la generazione della carta. Ogni classe derivata dopo la generazione dovrà eliminare i propri campi dall’interfaccia Grafica.
  + **Gerarchia Visitor**
    - La gerarchia dei visitor coincide con quella dei CardWidget, in cui è stato ridefinito un costruttore appositamente per l’utilizzo dei Visitor e che appunto serve a sfruttare il Vistor Design Pattern. In particolare i visitor nel progetto servono per la rappresentazione polimorfa della carta durante la fase di modifica di quest’ultima.
* **Graphic User Interface (GUI):**

Nella pagina relativa alla scelta del tipo della carta viene utilizzato uno slot in maniera polimorfa per il passaggio di un elemento, derivato della gerarchia, alla pagina della creazione della carta. Questo passaggio avviene tramite l’utilizzo di un puntatore alla classe radice CardWidget che tramite un segnale polimorfo (*cardsignal()*) viene passato ad un’altra pagina. In quest’ultima in base al tipo dinamico verrà rappresentato il Widget derivato relativo al tipo dinamico del puntatore, così da permettere la creazione della carta del tipo selezionato.

**Persistenza Dei Dati**

Per la persistenza dei dati viene utilizzato il formato Json, e in particolare questo avviene tramite la librearia OpenSource JsonCpp. Ogni Deck ha un file json in cui si salva il nome di quest’ultimo e un Json-Array di carte in cui ogni elemento dell’array è una carta del mazzo con i propri vari campi.

Si è optato per questo tipo di formato per garantire più libertà possibile nei caratteri utilizzabili, poiché JsonCpp gestisce in maniera automatica l’escaping dei caratteri, e per la facilità di formattazzione dei dati. Difatti, scrittura e lettura avvengono in maniera molto efficiente poiché si accede alle varie carte come se esse fossero un vettore e ai suoi dati come se fossero un dizionario, facilitandone di molto l’uso.

Un esempio di formattazione è reperibile all’interno della cartella “*Asset\Deck\TestDeck*”.

**Funzionalità Implementate**

Per semplicità si dividono le varie funzionalità in: Estetiche, Funzionali, Documentative.

Funzionali:

* Gestione di 5 tipologie di dati differenti;
* Conversione e salvataggio in formato Json;
* Funzionalità di ricerca tramite Nome e Descrizione (sia per Stringa che per Sottostringa);
* Funzionalità di Ordinamento in base al costo;
* Funzionalità di Ordinamento in base al nome;
* Possibilità di avere più Mazzi diversi;
* Possibilità di esportare singole carte come immagini;
* Possibilità di effetuare LogOut dal mazzo scelto;
* Possibilità di eliminare in maniera definitiva un elemento del Mazzo;
* Possibilità di eliminare in maniera definitiva un intero Mazzo;
* Possibilità di scorrere il mazzo come fosse una lista di carte;
* Possibilità di modificare i campi delle singole carte;
* Salvataggio Automatico: Per migliorare l’esperienza utente, ad ogni aggiunta di una carta in un mazzo esso viene salvato per evitare eventuali ed accidentali perdite di carte. Questo avviene anche dopo ogni modifica.

Grafiche :

* Pulsanti per ordinare, con icone diverse in base al tipo di ordinamento;
* Campo di ricerca, in cui ad ogni pressione aggiorna la vista delle carte in base alla sottostringa;
* Animazione di caricamento durante la generazione;
* Pulsanti per l’esportazione di uno screen della carta;
* Richiesta conferma all’utente in caso di rimozione del mazzo;
* Richiesta conferma all’utente in caso di Log Out dal mazzo;
* Creazione di un logo;
* Utilizzo di icone nei pulsanti;
* Interfaccia diversa in base al tipo scelto;
* Numerosi effetti grafici al passaggio del mouse;
* Icone Cliccabili rappresentative per i vari tipi ed attributi delle carte;
* Utilizzo di colori e stili grafici non banali;
* Vista nella Home dell’ultima carta del mazzo.

Documentative :

* Presenza di un manuale nella HomePage che in maniera esaustiva spiega l’utilizzo dell’applicazione;
* Warning tramite message box in caso di modifiche errate;
* Warning tramite label in caso di nomi di carte o mazzi non ammissibili (già utilizzati o contenenti caratteri non ammessi);
* Warning in caso di errori da parte dell’API con il relativo errore;
* Presenza di una guida esaustiva per ogni tipo di interfaccia delle carte;
* Richiesta di conferma tramite message box in caso di tentativo di eliminare un mazzo;
* Richiesta di conferma tramite message box in caso di tentativo di log out da un mazzo;
* Richiesta di conferma tramite message box in caso di tentativo di salvataggio del mazzo.

**Rendicontazione ore**

|  |  |  |
| --- | --- | --- |
| **Attività** | **Ore Previste** | **Ore Effettive** |
| Studio e progettazione | 8 | 10 |
| Sviluppo del codice del modello | 13 | 15 |
| Studio del framework Qt | 6 | 6 |
| Sviluppo del codice della GUI | 12 | 17 |
| Test e debug | 6 | 7 |
| Stesura della relazione | 5 | 5 |
| **totale** | 50 | **60** |

Il monte ore è stato leggermente superato in quanto sviluppo del codice di modello e GUI ha richiesto più tempo di quanto previsto. In particolare i motivi principali sono stati l’inizializzazione della richiesta e il recupero dell’immagine ricevuta che ha portato ad un aumento delle ore, poiché è stato richiesto uno studio di un procedimento mai effettuato in precedenza. Inoltre, l’utilizzo di metodi asincroni all’interno del progetto ha richiesto un ulteriore impiego di tempo poiché anche in questo caso ha portato la necessità di documentazione su tale ramo della programmazione.

**Attività svolte**

**Gerarchia :**

* Intera classe Deck;
* Creazione del template Dlist e la lista doppiamente concatenata;
* Metodo Clone in Abstract Card;
* Implementazione in particolare del metodo Save, Load e Garbage di Deck;
* Intera classe Dall\_eapi (in collaborazione al collega).

**Classi Aggiuntive :**

* Classe ClickableLabel svolta col collega.

**Testing:**

* Testing del garbage in maniera approfondita;
* Testing del container in maniera approfondita.

**Parte Grafica :**

In solitario:

* Creazione della Gerarchia di CardWidget:
  + Corrispetiva interfaccia e implementazione dei metodi per ogni widget:
    - MonsterWidget;
    - SpellWidget;
    - SrapWidget;
    - FieldWidget;
    - ArtifactWidget.
* Creazione della TypePage;
* Creazione della CardPage;
* Metodo Manual e le corrispettive guide.

Con il collega :

* Creazione della DeckHomePage;
* Metodi Generate di CardWidget.

**Compilazione e Installazione**

Il progetto utilizza due librerie esterne ovvero :

* libCurl;
* JsonCpp.

Per l’esecuzione corretta del programma queste due librerie devono essere presenti. Si è quindi creato uno script in bash per l’installazione e la compilazione del programma.

Per eseguirlo basterà digitare nel terminale all’interno cartella “./install.sh” ed in questo modo le dipendenze verranno scaricate ed automaticamente verrà effettuata la compilazione del codice con i comandi qmake e make.

**Nota Importante**

Per la corretta generazione dell’immagine è necessaria una connessione Internet. Inoltre è necessaria una APIKEY con cui fare richiesta delle immagini.

Il credito di questa API KEY (che sarà fornita alla consegna) può esaurirsi poiché ogni generazione di immagine ha un costo di $0.0018.

Il progetto compila ed esegue correttamente anche senza una API KEY funzionante, ma essa è necessaria per la generazione delle immagini tramite AI.

In tutti i casi in cui la chiamata produce un errore comparirà al posto dell’immagine generata un’immagine di default che rappresenta l’errore.

**Note finali**

Lo sviluppo di questo progetto è da me considerato una delle attività più utili e formative svolte durante il periodo universitario, ed e’ stato oggetto di interesse personale e lavorativo. In particolare, cio che questo progetto mi ha lasciato sono competenze concrete relative alla gestione dei compiti di gruppo ed abilita informatiche nell’approccio a nuove sfide.

**Documentazioni Usate :**

* [*libCurl*](https://curl.se/libcurl/);
* [*JsonCpp*](https://open-source-parsers.github.io/jsoncpp-docs/doxygen/index.html);
* [DALL·E API](https://openai.com/blog/dall-e-api-now-available-in-public-beta);
* [Qt Asynchronus](https://www.qt.io/blog/asynchronous-apis-in-qt-6);