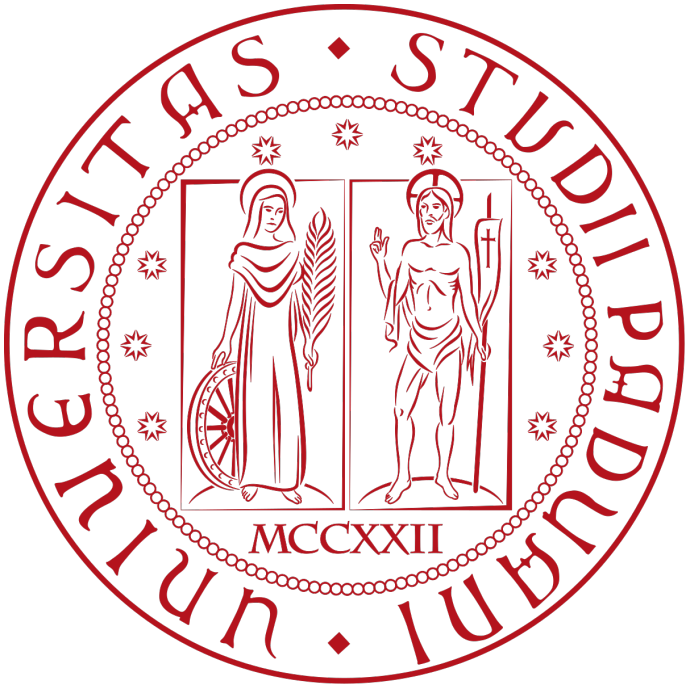
myGreen

Zaccone Rosario, 2043680

Anno Accademico 2022/2023

Relazione progetto Programmazione a Oggetti



Indice

Abstract

MyGreen è un software che permette di creare una propria enciclopedia consultabile per un determinato spazio verde, sia esso un giardino, un parco pubblico, un'oasi naturale o un terreno di campagna. Permette di inserire diversi tipi di organismi, modificarli, visualizzarli ed eventualmente eliminarli. Possono essere inseriti sei tipi di organismi: anfibi, uccelli, mammiferi, rettili, piante erbacee, piante arboree(alberi). Ogni organismo inserito ha una propria scheda personale che illustra le sue principali caratteristiche, come nome, tassonomia, habitat, stato di conservazione IUCN, foto, descrizione generica. Ogni tipologia di organismo ha un layout di scheda diverso, per esempio la scheda degli animali comprende anche una sezione che permette di ascoltare il loro verso, mentre quella delle piante presenta una sezione in cui è possibile visualizzare foglia, fiore e frutto della specie in questione. È disponibile anche una sezione di ricerca, attraverso la quale è possibile ricercare un determinato organismo, compilando campi come nome, tipo, habitat e stato di conservazione. Per la costruzione del layout e la scelta delle informazioni delle schede sono state prese in condiderazione le enciclopedie principali utilizzate nei corsi di studio di Scienze Naturali, per l'aspetto generale del software ci si è ispirati al *Pokédex*[1]. Ho scelto questo progetto poichè ho una grande passione per la natura, e prestando servizio in un'oasi naturalistica ho pensato che mi potrebbe far comodo tenere traccia di tutte le specie viventi presenti in quell'area. Il nome del software è una citazione a *MySpace*[2], social network famoso degli anni 2000.

1 Istruzioni

Il software utilizza la classe *QSoundEffect* della libreria *QT Multimedia*, quindi per avviarlo c'è bisogno di installarla. Per avviare correttamente il programma posizionarsi nella cartella *myGreen* e digitare:

- sudo apt update
- sudo apt install qt6-multimedia-dev
- qmake
- make
- ./myGreen

L'utilizzo della libreria è stato concesso dal professore Marco Zanella.

[1] Il Pokédex è uno strumento elettronico immaginario che appare nei videogiochi, nell'anime e nei manga dei Pokémon.
[2] Myspace (conosciuto in precedenza come MySpace) è un social network lanciato l'1 Agosto 2003, ha avuto una forte influenza sulla musica e sulla cultura pop degli anni '00.

2 Architettura

2.1 Gerarchia

La gerarchia base del progetto è rappresentata nella Fig.1. Rappresenta i vari tipi di organismi viventi che è possibile inserire dentro myGreen. È presente la classe astratta *Organism* che rappresenta in modo astratto una specie vivente, successivamente troviamo le classi astratte *Plant* e *Animal*, figlie di *Organism*, che rappresentano rispettivamente una pianta generica e un animale generico. Fra gli attributi di *Organism* troviamo:

- **name**, che rappresenta il nome comune dell'organismo.
- **taxonomy**, che è un vettore di 4 stringhe, ciascuna rappresentante rispettivamente specie(nome scientifico), genere, famiglia e ordine dell'organismo.
- **habitat**, che rappresenta l'habitat principale dell'organismo all'interno dell'area verde esaminata,
- **cs**, che rappresenta lo stato di conservazione IUCN[3] dell'organismo. Ad esempio, la *Rana latastei* ha stato di conservazione VU(Vulnerabile).
- **length**, che rappresenta la lunghezza dell'organismo.
- **imgPath**, che rappresenta il path dell'immagine che raffigura l'organismo.
- **description**, che rappresenta una descrizione generica, utilizzata per inserire informazioni più discorsive e non incluse negli altri attributi.
- **rarity**, che indica la rarità dell'organismo all'interno dell'area verde esaminata.

La classe *Animal* ha come attributi aggiuntivi **noisePath**, che rappresenta il path del file audio WAV che contiene il verso dell'animale, e **weight**, che rappresenta il peso. La classe *Plant* ha come attributo caratteristico **lffImgPath**, che è un array che contiene i path delle immagini relative alla foglia, fiore e frutto della pianta. Infine abbiamo le classi *Herb* e *Tree*, figlie di *Plant*, e le classi *Amphibian*, *Bird*, *Mammal* e *Reptile*, figlie di *Animal*. Quest'ultime sono classi concrete che rappresentano rispettivamente piante erbacee, alberi, anfibi, uccelli, mammiferi, rettili. Di seguito sono elencati gli attributi esclusivi di ciasciuna di quest'ultime classi:

- la classe *Amphibian* ha l'attributo **larvalStageImgPath**, che rappresenta il path della foto dello stato larvale dell'anfibio.
- la classe *Bird* ha l'attributo **wingspan**, che è l'apertura alare dell'uccello.
- la classe *Mammal* ha l'attributo **footprintImgPath**, che rappresenta il path della foto dell'impronta del mammifero.
- la classe *Reptile* ha l'attributo **scaleType**, che rappresenta il tipo di squama del rettile.
- la classe *Herb* ha l'attributo **uses**, che rappresenta un vettore di flag indicanti gli usi della pianta erbacea. Ad esempio, un array uses settato a 1,0,1 indica che la pianta erbacea ha uso culinario, medicinale ma non cosmetico.
- la classe *Tree* ha l'attributo **diameter**, che rappresenta il diametro del tronco dell'albero.

La gerarchia è stata strutturata basandosi sulla classificazione tassonomica degli organismi viventi, una prima suddivisione in *Plant* e *Animal* è stata fatta basandosi sulla classificazione per regno[4]. La specializzazione della classe astratta *Animal* in *Amphibian*, *Bird*, *Mammal*, *Reptile* è stata fatta basandosi sulla suddisione degli animali vertebrati per classi tassonomiche. Un osservatore attento può notare la mancanza di una classe per rappresentare i pesci, che non è stata sviluppata vista l'assenza o la poca presenza di pesci nei giardini, terreni e oasi naturalistiche. La specializzazione della classe astratta *Plant* in *Herb* e *Tree* è stata sviluppata basandosi sulla suddisione delle piante in base al loro fusto(erbaceo nel caso delle piante erbacee e legnoso nel caso degli alberi). Poteva essere scelta una classificazione più specifica, ad esempio basandosi sul sistema di riproduzione delle piante[5], ma si è preferito evitarla per avere una classificazione più user-friendly e conosciuta da tutti.

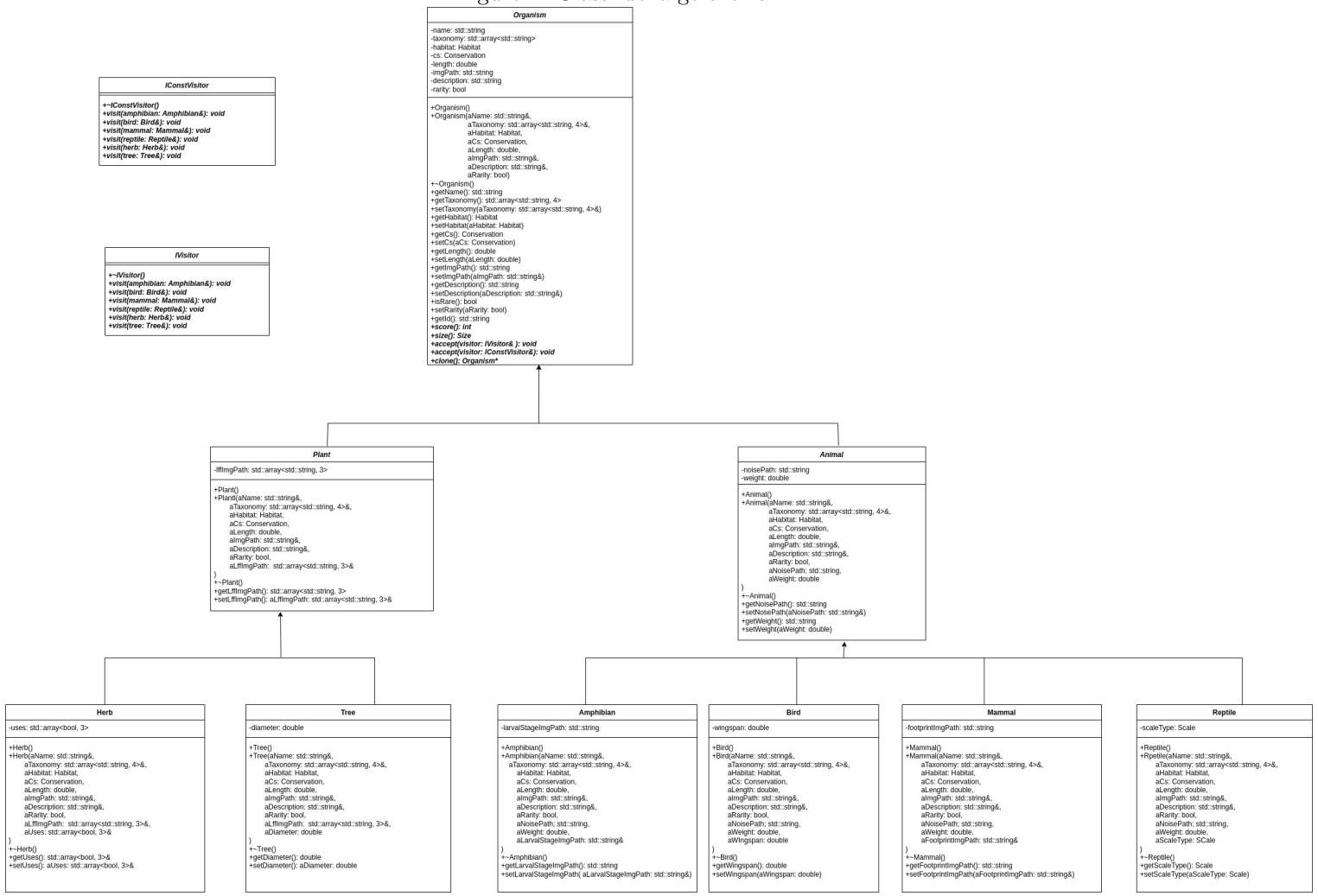
Le classi della gerarchia, oltre ad avere attributi che rappresentano le loro caratteristiche e relativi getter/setter per rispettare i principi di encapsulation, hanno anche due metodi particolari: **score()** e **size()**. **score()** calcola un punteggio basandosi su alcune caratteristiche dell'organismo che lo invoca, mentre **size()** ne calcola la taglia rispetto alla tipologia di appartenenza.

[3] L'Unione internazionale per la conservazione della natura (abbreviato in IUCN dall'inglese International Union for the Conservation of Nature), è una organizzazione non governativa (ONG) internazionale. Lo IUCN è responsabile della pubblicazione della Lista Rossa IUCN, una raccolta dello stato di conservazione delle specie viventi sul nostro pianeta.

[4] Gli organismi viventi si classificano per regno in: Monere, Protisti, Animali, Piante e Funghi. Sono stati lasciati da parte i funghi per mancanza di tempo, monere e protisti per scelta logica, dato che nelle enciclopedie di una determinata area non sono mai presenti.

[5] In questo caso avremmo avuto alghe, muschi, felci, angiosperme e gimnosperme.

Figure 1: Classi della gerarchia



2.1.1 Enum classes

Sono state definite sei *enum classes*: *Taxonomy* per rappresentare le categorie tassonomiche principali(ordine, famiglia, genere, specie), *Habitat* per rappresentare i vari habitat(prateria, montagna, foresta, palude...), *Conservation* per gli stati di conservazione dettati dallo IUCN(LC, NT, VU, EX...), *Size* per le taglie(small, medium, large), *Scale* per i tipi di squama dei rettili(scudata, imbricata, guscio, tuberculiforme) e *Use* per gli usi delle piante erbacee(culinario, cosmetico, medicinale).

2.2 Container (MapTree<K, V>)

Per il container degli oggetti della gerarchia si è scelto di realizzare una map basata su un albero binario di ricerca AVL. Si è scelta una map rispetto ad una comune lista per evitare elementi duplicati. È stato preferito un albero AVL al comune BST poichè gli alberi bilanciati hanno tempi computazionali migliori. La map poteva poggiarsi su una tabella hash per garantire prestazioni migliori, ma si è preferita la struttura ad albero binario per mantere gli elementi ordinati, visto che nelle enciclopedie gli organismi sono listati in ordine alfabetico. Nella map la chiave è una codifica del nome comune dell'organismo, che lo converte in snake case e successivamente lo priva dell'underscore, per garantire un giusto ordinamento alfabetico. Il valore consiste invece in un puntatore ad un oggetto di tipo *Organism*. Di seguito sono specificate le informazioni relative alla complessità computazionale del container.

Time complexity			
Operazione	Caso migliore	Caso medio	Caso peggiore
Inserimento	O(log n)	O(log n)	O(log n)
Rimozione	O(log n)	O(log n)	O(log n)
Ricerca	O(1)	O(log n)	O(log n)
Attraversamento	O(log n)	O(log n)	O(log n)

Sono state realizzate due classi di eccezioni *MissingKeyException* e *DuplicateKeyException* per gestire rispettivamente i casi in cui si cerca un elemento con una chiave non presente nella mappa e in cui si cerca di aggiungere un duplicato ad essa.

2.3 Persistenza dei dati

Per memorizzare i dati si è scelto di usare un unico file Json che contiene tutti gli attributi dei singoli oggetti, con in aggiunta un attributo **type**, utilizzato per riconoscere la loro classe d'appartenenza durante la lettura da file. Si è preferito il formato Json vista la sua semplicità, la presenza di librerie per gestire l'I/O fornite da Qt e la quantità non elevata di informazioni da memorizzare. In caso di un volume di informazioni nettamente maggiore(facciamo caso il software disponesse di connettività ad internet e prevedesse l'utilizzo contemporaneo di più persone) si sarebbe optato per un database non relazionale come MongoDB. Un esempio di file è presente nella cartella Samples.
Per gestire l'I/O sono state sviluppate due classi:

- La classe *JSONReader* che si occupa della lettura da file Json. Un oggetto di questa classe viene inizializzato con un percorso che indica il nome del file da leggere. La funzione **readJ()** apre il file che ha come percorso l'attributo **path** dell'oggetto *JSONReader* e dopo aver creato una MapTree la popola con gli oggetti letti dal file. Come chiave viene utilizzata una codifica dell'attributo **name** dell'oggetto, mentre come valore associato viene posto un puntatore all'oggetto stesso allocato dinamicamente.
- La classe *JSONWriter* che implementa l'interfaccia *JSONWriteVisitor* e si occupa della scrittura degli oggetti della gerarchia su file Json. Un oggetto di questa classe viene inizializzato con una MapTree e con il percorso del file su cui scrivere. La scrittura viene effettuata tramite il metodo **writeJ()**, che scrive tutti i valori della MapTree sul file indicato. In base all'oggetto da scrivere viene invocato un metodo diverso grazie al pattern *Visitor*, si veda la sezione dedicata al polimorfismo per maggiori dettagli. Il nome comune dell'organismo e le relative categorie tassonomiche vengono codificate in snake case.

2.4 Modello

IL sofware si basa sull'architettura Model/View di Qt. La classe Model che rappresenta il modello è sottoclasse di *QAbstractListModel*[6]. La classe Model ha come attributi *memory* che rappresenta la MapTree contenente gli organismi, e *path* che rappresenta il file aperto dal modello. Model mette a disposizione diversi metodi per l'inserimento, la modifica e la rimozione degli organismi. Il costruttore di questa classe inizializza un oggetto della classe *JSONReader*, che apre il file e popola la MapTree.
È presente una classe *Proxy*, sottoclasse di *QSortFilterProxyModel*[7], che si occupa di filtrare gli oggetti in fase di ricerca. La classe *Proxy* usa come modello di riferimento *Model*.

[6] Dala documentazione ufficiale di Qt: *QAbstractListModel provides a standard interface for models that represent their data as a simple non-hierarchical sequence of items. It is not used directly, but must be subclassed.*

[7] Dalla documentazione ufficiale di Qt: *QSortFilterProxyModel can be used for sorting items, filtering out items, or both. The model transforms the structure of a source model by mapping the model indexes it supplies to new indexes, corresponding to different locations, for views to use.*

2.5 GUI

La classe principale è *MainWindow*, sottoclasse di *QMainWindow*. La finestra principale presenta un menu con tre sezioni: una dedicata all'I/O, una dedidicata all'inserimento e la terza dedicata alla modifica e alla rimozione degli elementi.
La finestra presenta, oltre al menu, tre componenti principali:

- *SearchWidget*, il widget di ricerca
- *QListView*, il widget dedicato alla lista degli organismi
- *CardBox*, il widget dedicato alle schede gli organismi

Nella Fig.2 è mostrato uno schema sintetico e delle classi più importanti della GUI. Sono presenti anche delle classi minori come *ImgPopUp*, sottoclasse di *QDialog*, usata per mostrare la legenda dei simboli e *LabelImage*, sottoclasse di *QWidget*, utilizzata per creare label con didascalia e una serie di immagini che possono essere aggiunte dinamicamente.

2.5.1 SearchWidget

SearchWidget è dedicato alla ricerca. È possibile inserire un nome, un tipo, un habitat e uno stato di conservazione. Una volta premuto il bottone di ricerca il proxy verrà aggiornato per filtrare i risultati desiderati. Rigurado il nome, un organismo verrà mostrato se il nome inserito è sottostringa del suo nome comune.

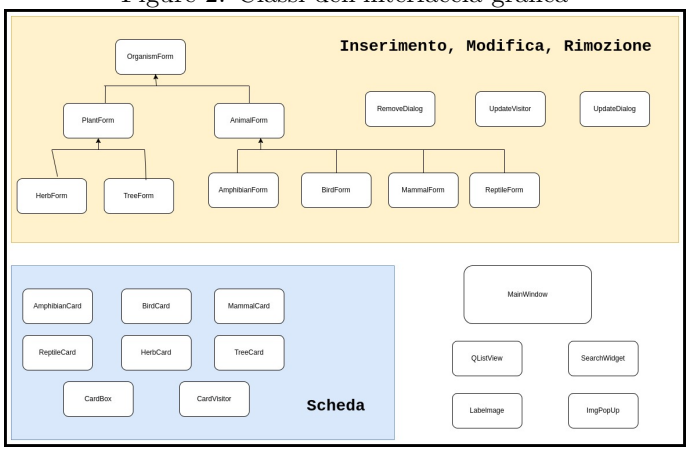
2.5.2 CardWidget

CardWidget è costituito da un bottone, che se premuto mostra le legenda delle varie icone delle schede, e dal widget dedicato alle schede. Quest'ultimo è un generico *QWidget*, a cui viene assegnata una *AmphibianCard, BirdCard, MammalCard, ReptileCard, HerbCard o TreeCard* in base all'organismo selezionato nella *QListView*.

2.5.3 Widget di inserimento, modifica e rimozione dati

Per l'inserimento sono stati progettati diversi form, uno per ogni tipo di oggetto della gerarchia. Per facilitare la costruzione dei form dedicati alle classi concrete, sono stati realizzati anche form di inserimento per le classi astratte *Organism, Animal e Plant*. Ad esempio *BirdForm*, il form dedicato all'inserimento degli uccelli, è sottoclasse di *AnimalForm*, che a sua volta è sottoclasse di *OrganismForm*. *OrganismForm* nel suo costruttore costruisce la sezione del form dedicata agli attributi che tutti gli organismi hanno in comune, *AnimalForm* nel suo costuttore costruisce *OrganismForm* e aggiunge al form i campi comuni a tutti gli animali. Infine *BirdForm* nel suo costruttore costruisce *AnimalForm* e aggiunge la sezione dedicata alle informazioni esclusive degli uccelli. *ReptileForm*, altro esempio di form, nel suo costruttore costruisce *AnimalForm* e aggiunge la sezione di inserimento dedicata alle informazioni esclusive dei rettili. Si è scelto di utilizzare questo approccio per evitare duplicazione di codice e garantire una migliore modularità. I form di inserimento sono sottoclassi di *QDialog* e hanno la funzionalità di bloccare in input dati non validi, come ad esempio nomi troppo lunghi o altezze negative.
I form di aggiornamento sono gli stessi di quelli di inserimento, ma vengono costruiti con un costruttore diverso che precompila il form con i dati dell'organismo da aggiornare. Il nome dell'organismo da modificare viene inserito tramite un *UpdateDialog*, sottoclasse di *QDialog*.
Per la rimozione, il nome dell'organismo da modificare viene inserito tramite un *RemoveDialog*, sottoclasse di *QDialog*.

Figure 2: Classi dell'interfaccia grafica



3 Polimorfismo

3.1 Visitor

L'utilizzo principale del polimorfismo riguarda il design pattern *Visitor*. Le due interfacce messe a disposizione sono IVisitor e IConstVisitor, entrambi localizzate nella cartella Model/Core. Le classi che implementano queste interfacce sono quattro:

- JSONWriteVisitor
- TypeVisitor
- CardVisitor
- UpdateVisitor

JSONWriteVisitor ha il compito di scrivere gli oggetti su un file Json, ogni tipologia di oggetto ha degli attributi che gli altri non hanno e quindi necessita di un trattamento diverso.
TypeVisitor viene utilizzato principalmente nella fase di filtraggio per verificare se un dato oggetto appartiene ad una determinata classe. Viene utilizzato quindi come alternativa al dynamic casting.
CardVisitor è sicuramente il visitor più interessante, dato che in base alla tipologia di organismo costruisce una scheda descrittiva con layout, informazioni e colori diversi.
UpdateVisitor viene utilizzato per invocare form diversi in base alla tiplogia di organismo da modificare. Ad esempio, se si vuole modificare un rettile, grazie all'UpdateVisitor verrà chiamato il rispettivo *ReptileForm*.

3.2 Metodi virtuali score() e size()

Sono presenti inoltre altre forme meno interessanti di polimorfismo, come ad esempio i metodi astratti **score()** e **size()** della classe *Organism*, implementati diversamente da ogni sottoclasse concreta.
Il metodo **score()** calcola un punteggio relativo ad ogni specie, che indica la sua particolarità. Nel calcolo viene coinvolto l'attributo **rarity** e un insieme di attributi che varia in base alla classe dell'oggetto. La formula è diversa per ogni classe, di seguito sono elencate due esempi di formule:

```
unsigned int Bird::score() const
{
    return isRare() ? std::lround(1.5 + wingspan/2) : std::lround(wingspan/2);
}

unsigned int Herb::score() const
{
    double bonus{0.0};
    for (int i{0}; i < 3; i++) {
        if (uses[i])  bonus++;
    }
    return isRare() ? std::lround(1 + bonus/2.0) : std::lround(bonus/2.0);
}
```

Il metodo **size()** calcola la taglia della specie in base alla tipologia a cui appartiene. Ad esempio, un albero di 1m viene considerato di taglia piccola, ma un anfibio di 1m viene considerato di taglia grande.

4 Funzionalità

Le funzionalità implementatre sono:

- gestione di sei tipologie di organismi viventi
- conversione e salvataggio in formato Json
- funzionalità di ricerca con filtro
- ordinamento in ordine alfabeto automatico
- barra di menu
- sezioni dei menu disabilitate in specifiche condizioni
- utilizzo di icone nel menu
- scorciatoie da tastiera(mostrate anche nelle voci del menu)
- controllo della presenza di modifiche prima di chiudere l'applicazione
- presenza di una legenda delle icone relative alla scheda descrittiva
- gestione del ridimensionato
- utilizzo di stylesheet css
- utilizzo di colori
- animazioni grafiche alla pressione dei pulsanti con immagini
- form di inserimento diversi per ogni tipologia di organismo
- scheda descrittiva diversa con layout diverso per ogni tipologia di organismo
- presenza di effetti sonori(versi degli animali)
- blocco dell'inserimento di elementi duplicati

5 Rendicontazione ore

Ore impiegate	
Attività	Ore impiegate
Studio e progettazione	12
Sviluppo del codice del modello (gerarchia, I/O, classe Model e Proxy)	10
Studio e sviluppo del codice del container	10
Sviluppo del codice della gui	16
Studio del framework Qt	15
Test e debug	4
Stesura della relazione	7
Totale	74

Sono state impiegate in totale **74 ore** rispetto alle 50 previste. È stato impiegato più tempo del previsto visto lo studio approfondito degli alberi AVL(non affrontati nel corso di Algoritmi e Strutture Dati), la presenza di sei tipologie diverse di oggetti e lo studio di Latex per quanto riguarda la relazione.

5.1 Note finali

Le icone sono state scaricate dal sito flaticon.com.