Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Informatica

Elaborato finale

Sviluppo di un'applicazione per attività didattica in collaborazione con il Muse

Relatore
Alberto Montresor

Laureando
Lorenzo D'Ambrosio

Correlatrice
Carlotta Vielmo

Anno accademico 2023/2024

Ringraziamenti

-Ringrazio Mamma e Papà, per avermi sostenuto in ogni momento della mia vita, sin dai primi giorni di scuola fino a questo giorno così importante. Grazie per il costante supporto trasmesso anche semplicemente tramite una telefonata alla fine del turno al Mc, o per un semplice ma allo stesso tempo significativo messaggio le mattine prima di ogni esame. Se sono quello che sono è anche e soprattutto merito vostro, e non sarò mai abbastanza grato per tutto quello che mi avete donato;

-Ringrazio mia sorella Martina, per aver affrontato insieme a me momenti molto duri, ma dalla quale ne siamo usciti, insieme, più uniti che mai;

-Ringrazio i miei parenti ed in particolare i miei nonni Clara, Carolina, Francesco e Gioacchino per il loro supporto, per la loro spensieratezza e per l'enorme saggezza che mi hanno trasmesso. Sono ogni giorno grato per ogni momento passato insieme a voi, per i pranzi, le cene e le infinite partite a carte;

-Ringrazio Sara, per essere stata una colonna portante di questo mio percorso, senza di te non sono sicuro che sarei riuscito ad arrivare a questo traguardo. Senza il tuo costante supporto, anche nei momenti più bui, e alla tua enorme pazienza, non sarei la persona che sono oggi e sono contento del percorso che stiamo facendo insieme, coi suoi alti e bassi;

-Ringrazio Serena, Arianna, Beatrice, Ludovico e tutti le persone che hanno reso indimenticabili questi giorni passati qui a Trento, grazie per le risate, per le cene infinite al Mc, per i pranzi da Zhang (ogni volta in un tavolo diverso) e per le indimenticabili serate passate insieme, tra semplici aperitivi, serate in Akka o serate giochi da tavolo più tisane;

-Ringrazio Emanuele, che so che si stava già agitando nel non essere nominato, per essere una persona fantastica, dotata di una simpatia (di un ego) e di una bontà per le quali era necessario un ringraziamento personale. Grazie per essere un amico fantastico e sincero, per essermi sempre stato accanto fin da quel lontano 19 ottobre 2022, data del nostro primo e leggendario BeReal. Senza di te questo periodo a Trento non sarebbe stato lo stesso, avrei sicuramente meno ricordi felici, meno sorrisi e molto probabilmente una media più bassa. Avrei anche meno chili e più soldi, ma nessuno è perfetto;

-Ringrazio tutti i colleghi e amici del McDonald di Trento Nord, in particolare ringrazio coloro che mi hanno aiutato a crescere, dal lato lavorativo ma soprattutto da quello umano. Grazie per le risate e grazie per non aver mai mollato, neppure in situazioni di scarso personale che, con sudore e fatica, ci hanno reso un gruppo fantastico, con i loro alti e bassi;

-Ringrazio Mattia, per essere stato un fantastico coinquilino, e un ancor più fantastico amico. Grazie per le giornate trascorse insieme, tra risate, partite al campetto, gelati e ore passate sotto casa tua ad aspettarti, che contraddistinguono la nostra amicizia da ormai più di 17 anni;

-Ringrazio Simone, Joséphine, Luca e tutti gli altri amici di Bolzano e dintorni, che mi hanno permesso di non avere nostalgia di casa. Grazie per avermi accolto in tutti i miei ritorni a Bolzano, grazie per le colazioni, per le partite e per i cori compatti fatti in gradinata;

-Ringrazio Gabriele, Andrea, Marco, Giulio, Enrico e tutti i miei compagni di università, branco compreso. A livello pratico siete sicuramente la principale forza che mi ha permesso di arrivare qui, grazie ad appunti, spiegazioni e risposte a domande spesso banali, senza le quali avrei ancora il 90% degli esami da affrontare. Grazie per i pranzi insieme, per le risate tra una lezione e l'altra e per le ripetute spiegazioni dei vari argomenti che faticavano ad entrare nella mia testolina;

-Ringrazio il professor Montresor, lo staff del Muse ed in particolare Carlotta, per l'opportunità concessa e per la pazienza e la disponibilità mostrata durante questo periodo.

Indice

Sommario

Questa tesi verte sull'attività di tirocinio svolta nell'anno accademico 2024/2025 in collaborazione con il Muse – Museo delle Scienze di Trento [17] e con la dottoranda Carlotta Vielmo, del Dipartimento di Matematica dell'Università di Trento.

L'obiettivo del tirocinio è stato quello di realizzare un'applicazione *user-friendly*, che potesse supportare i partecipanti nell'attività organizzata dal Muse, basata sul progetto di dottorato della dr.ssa Vielmo.

Il progetto, denominato ExploRL, consiste in un'attività educativa progettata per dare ai partecipanti un'idea generale del funzionamento del *reinforcement learning*, tecnica di apprendimento fondamentale nel campo del *machine learning* e dell'intelligenza artificiale.

Il mio ruolo all'interno di questa collaborazione è stato quello di sviluppatore dell'applicazione, lavorando autonomamente e aggiornando periodicamente il Muse e la dottoranda sullo stato del lavoro.

L'applicazione è costituita da una parte introduttiva, nella quale i partecipanti all'attività possono familiarizzare con le principali funzioni, ed una parte di attività vera e propria. Per lo sviluppo è stato utilizzato *Flutter*, un *framework open-source* per lo sviluppo di applicazioni multi-piattaforma (Android, iOS, ...) ed il relativo linguaggio di programmazione *Dart*.

L'elaborato inizia con una contestualizzazione del progetto e dell'idea su cui verte lo sviluppo dell'applicazione, del quale vengono poi mostrati i risultati. Successivamente, sono riportate le criticità rilevate durante l'intero processo ed infine sono presenti delle considerazioni riguardanti l'utilizzo ed il futuro mantenimento dell'applicazione.

In appendice, vi è un file di tutorial, consegnato al Muse e all'azienda che manterrà l'applicazione da qui in avanti, contenente le istruzioni necessarie per il mantenimento della stessa.

1 Introduzione

Prima di mostrare i risultati dello sviluppo, è necessario introdurre alcuni concetti e fornire il contesto su cui si basa l'intero progetto, partendo dal concetto di *Reinforcement Learning* e successivamente focalizzandosi sul progetto della dr.ssa Vielmo.

1.1 *Reinforcement Learning*

Il *reinforcement learning*, o apprendimento per rinforzo, è una tecnica di apprendimento specifica del *machine learning*, che permette di addestrare un sistema insegnandogli a prendere le decisioni corrette. Gli elementi principali presenti in questo contesto sono:

- **Agente**: il protagonista dell'apprendimento, ovvero il soggetto che compie movimenti all'interno dell'ambiente e prende decisioni che influenzeranno il suo apprendimento. Il suo obiettivo è quello di raggiungere un determinato risultato utilizzando il minor quantitativo di risorse possibile;

- **Ambiente**: il luogo all'interno del quale avviene l'apprendimento, dove l'agente compie delle azioni ricevendo, spesso, dei riscontri, definiti *rewards*. Queste ricompense vengono utilizzate dall'agente per poter scegliere più accuratamente la prossima azione da compiere. L'ambiente è composto da vari stati, ovvero varie fasi temporali che lo definiscono;

- **Azioni**: ciò che l'agente può fare all'interno dell'ambiente. Sono focalizzate sul raggiungimento dell'obiettivo compiendo il minor numero di azioni possibile e contribuiscono allo sviluppo di una strategia, definita *policy*. Ad ogni azione può corrispondere una ricompensa (*reward*);

- ***Policy***: un insieme di azioni compiute con lo scopo di arrivare a raggiungere l'obiettivo sprecando il minor numero di risorse possibili. Lo sviluppo della *policy* avviene in contemporanea allo svolgimento delle azioni, basandosi sulle ricompense ottenute durante il percorso;

- ***Reward***: *feedback* forniti dall'ambiente in base all'azione compiuta e / o allo stato raggiunto. Possono essere di vario tipo, generalmente sono definiti come un punteggio guadagnato o perso dall'agente. L'obiettivo dell'agente è quello di massimizzare il quantitativo di ricompense positive, o minimizzare quelle negative, ricevute.

Come descritto da AWS [5], l'apprendimento per rinforzo si basa su un concetto semplice: quando un agente compie un'azione all'interno dell'ambiente, può ricevere una ricompensa oppure no. Nel primo caso, la ricompensa rappresenta una valutazione dell'azione svolta; nel secondo, l'azione non ha comportato una variazione significativa nell'ambiente. È importante notare come una ricompensa negativa non descriva necessariamente una scelta sbagliata; in certi contesti può essere necessario compiere un sacrificio a breve termine per massimizzare la ricompensa finale. In relazione a questo aspetto, distinguiamo due tipologie di approcci: *value-based* e *policy-based*.

1.1.1 Approccio *value-based*

Nel *reinforcement learning* con approccio *value-based*, l'agente si focalizza sul produrre una *value function*, il cui obiettivo consiste nello stimare le ricompense ottenibili raggiungendo un determinato stato. La *value function* può focalizzarsi interamente sullo stato (dove $V(s)$ rappresenta quindi il valore di uno stato s) o sulla combinazione stato-azione (dove $Q(s, a)$ rappresenta il valore di uno stato s, raggiunto eseguendo l'azione a).

È importante notare come il valore di uno stato non si limiti semplicemente alla ricompensa ottenibile raggiungendolo, bensì anche alle ricompense ottenibili partendo da quello stato. Ad esempio, nella variante basata sulla coppia stato-azione, non ci si limita a massimizzare la ricompensa ottenibile

dalla transizione (s, a, s'), bensì anche le ricompense future che si otterranno seguendo la politica π da quello stato in poi. La formula è:

$$Q(s,a) = Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \tag{1.1}$$

Dove r è la ricompensa ottenibile partendo da s, eseguendo a e raggiungendo lo stato s', γ è il fattore di sconto, determinante il peso delle ricompense future e α è il *learning rate*, indica di quanto aggiornare l'attuale valore $Q(s, a)$. Questa tipologia di approccio risulta più semplice e più efficiente (in spazi discreti) rispetto all'approccio *policy-based*.

1.1.2 Approccio *policy-based*

L'approccio *policy-based* si focalizza invece sull'immediato apprendimento della *policy*, senza il calcolo di alcuna *value-function*. L'obiettivo dell'agente è quello di massimizzare il risultato, modificando di volta in volta la *policy* seguita, in modo da ottenere la massima ricompensa possibile. All'interno di una *policy*, indicata come π, la probabilità di scegliere un'azione a trovandosi in uno stato s è data da $\pi(a|s)$. L'obiettivo consiste quindi nel trovare la *policy* ottimale, definita come $\pi^*(a|s)$, che permetta di massimizzare il risultato. Rispetto al *value-based*, l'approccio *policy-based* risulta migliore in spazi d'azione continui o con politiche stocastiche.

1.1.3 Differenze tra tipologie di apprendimento

A prescindere dall'approccio utilizzato, il *reinforcement learning* si distingue dalle altre tipologie di apprendimento per vari aspetti e vantaggi.

Per prima cosa, l'apprendimento per rinforzo risulta ottimale in ambienti complessi, ovvero situazioni con molte regole e variabili che possono influenzare la decisione su quale azione eseguire. In situazioni con un alto numero di fattori, l'essere umano fatica a prendere una decisione che riesca a tenere in considerazione tutte le variabili in gioco; al contrario, un sistema di *reinforcement learning* è in grado di considerare un alto numero di fattori e di conseguenza prendere una decisione più ponderata. Inoltre, questa tipologia di sistemi è in grado di adattarsi rapidamente agli ambienti in continua evoluzione, trovando rapidamente nuove strategie per massimizzare i risultati.

Un secondo vantaggio dell'apprendimento per rinforzo risulta essere l'autonomia rispetto al supporto umano, ovvero la non necessità di un essere umano che etichetti i dati per dare un responso all'algoritmo, a differenza di altri approcci di apprendimento come quello supervisionato o non supervisionato.

1.1.4 *Markov Decision Process*

Il *reinforcement learning* si basa sul *Markov Decision Process* (MDP), un modello matematico sviluppato dal matematico russo Andrey Markov [15], focalizzato sulla modellizzazione del processo decisionale in ambiti con risultati non deterministici. Si parla di MDP in contesti in cui è soddisfatta la proprietà di Markov.

Un processo stocastico $\{S_t\}$ soddisfa la proprietà di Markov se la probabilità di raggiungere uno stato S_{t+1} dipende esclusivamente dallo stato S_t e non dagli stati $S_{t-1}, S_{t-2}, \ldots$
Questa proprietà indica che il sistema non deve avere memoria, bensì tenere conto solo della situazione attuale.
Un MDP è definito da:

- Uno spazio degli stati S.
- Uno spazio delle azioni A.
- Una funzione di probabilità di transizione $P_a(s, s')$, indicante la probabilità di partire dallo stato s e raggiungere lo stato s' eseguendo l'azione a.
- Una funzione di ricompensa $R_a(s, s')$, indicante la ricompensa ottenibile partendo dallo stato s e raggiungendo lo stato s' eseguendo l'azione a.
- Un fattore di sconto γ, compreso tra 0 e 1, rappresentante la differenza tra ricompense attuali e ricompense future.

Il problema principale sul quale si basa il *Markov Decision Process* riguarda l'identificazione della migliore azione eseguibile partendo da un certo stato, quindi lo sviluppo di una *policy* ottimale. Quest'ultimo si divide in cinque fasi:

1. Si parte da uno stato $s \in S$ e da un tempo t
2. Si sceglie, sulla base della *policy* π, la miglior azione $a_t \in A$ e la si esegue
3. Si ottiene un *reward* $r_t \in R$, se disponibile
4. L'ambiente fornisce all'agente un nuovo stato $s' \in S$
5. Si ripetono i punti 2, 3 e 4 finché non si giunge ad una condizione di arresto (come può essere una situazione di vittoria, di sconfitta o di stallo)

1.2 Progetto ExploRL

L'idea di base dietro a questo progetto consiste nel mostrare allo studente come funziona l'apprendimento per rinforzo. L'idea situata alla base di questa tipologia di apprendimento consiste nello sfruttare l'interazione con l'ambiente circostante, quindi i *feedback* ricevuti dopo varie azioni, per guidare l'apprendimento.

Tale approccio presenta molte similitudini con l'approccio base dell'apprendimento umano, sin dalla nascita ogni bambino impara molte cose compiendo azioni, le quali risultano corrette o errate a seconda di *feedback* ricevuti dall'ambiente che lo circonda. Questo apprendimento risulta molto potente poiché permette di mettere in stretta relazione causa ed effetto di un'azione. Continuando con quest'analogia, osserviamo come il bambino scopra l'ambiente che lo circonda, inizialmente sconosciuto, proprio grazie alle azioni che compie e ai risultati che ottiene.

Quest'approccio, naturale nell'essere umano, si è rivelato vincente anche nell'ambito dell'intelligenza artificiale. Sin dagli anni Cinquanta, l'apprendimento per rinforzo ha portato allo sviluppo di algoritmi basati sull'apprendere informazioni dalle azioni compiute e dalle conseguenti ricompense, generalmente di tipo numerico.

Lo scopo dell'attività proposta dalla dr.ssa Vielmo consiste quindi nel mostrare ai partecipanti le principali idee fondanti del *reinforcement learning*, tramite due fasi: la fase agente e la fase progettista.

1.2.1 Fase agente

In questa fase lo studente, interpretando il ruolo di agente in un contesto di *reinforcement learning*, dovrà apprendere la struttura dell'ambiente, sviluppando una strategia che gli permetta di raggiungere l'obiettivo nel minor numero di passi possibili. Questa fase sarà comprensiva di un'introduzione sulla tematica e successivamente di un'esplorazione all'interno del Muse, vestendo a tutti gli effetti i panni di un agente di apprendimento per rinforzo.

L'applicazione sviluppata in fase di tirocinio è il principale strumento di supporto all'interno di questa fase, fornendo all'agente alcune informazioni fondamentali durante l'esplorazione:

- Indicazioni riguardanti l'animale da raggiungere (come, ad esempio, il piano del Muse nel quale si trova)
- *Scanner* di *QR-code*, per permettere all'agente di scoprire le informazioni dell'animale raggiunto
- Lista di animali da visitare (le opzioni per il proseguimento dell'esplorazione)
- Lista di animali visitati (per poter definire una *policy* durante l'esplorazione)
- Dato sull'energia rimanente (per poter calibrare le proprie decisioni)

Al termine di questa analisi, gli studenti rientreranno nell'aula iniziale e analizzeranno i vari tentativi effettuati, focalizzandosi sul rapporto tra percorsi effettuati e risultati ottenuti. In questa fase, l'applicazione sarà di supporto per mostrare agli studenti i percorsi effettuati nei vari tentativi, portando al ragionamento sulla correttezza o meno di una determinata decisione.

1.2.2 Fase progettista

In questa fase lo studente si occuperà, sulla base di ciò che ha appreso nella prima fase, di modellare l'ambiente di esplorazione, focalizzandosi sull'importanza e la delicatezza della scelta di una determinata funzione ricompensa. Questa fase è descritta in questo elaborato per pura completezza, poiché all'interno di essa non è previsto l'utilizzo dell'applicazione.

1.3 Muse

Il Muse è il Museo delle Scienze di Trento ed è composto da vari piani ospitanti esposizioni (permanenti e temporanee) e attività di ricerca. Questa struttura collabora con molte scuole, ospitando attività formative mirate all'apprendimento di temi esterni o innovativi rispetto alle tradizionali attività previste dal programma scolastico.

Il progetto ExploRL permette di affrontare il tema dell'intelligenza artificiale, focalizzandosi in particolare sul *reinforcement learning*, un argomento generalmente non affrontato nell'insegnamento scolastico tradizionale.

1.4 Analogie tra *reinforcement learning* e attività al Muse

Lo scopo del progetto è quindi quello di far interpretare agli studenti il ruolo di agenti e sfruttare il Muse come ambiente per i loro tentativi. Lo spostarsi da un animale ad un altro e la scansione del *QR-code* ad esso associato sono le principali azioni svolte, che porteranno a varie tipologie di *reward*:

- Animale comune o vincente: tramite l'applicazione, lo studente sarà in grado di riconoscere se l'animale raggiunto è quello vincente oppure no
- Informazioni sull'animale: tramite l'applicazione, lo studente riceverà informazioni sull'animale raggiunto, le quali permetteranno all'agente lo sviluppo di una strategia (la *policy*)
- Termine del tentativo: tramite l'applicazione, lo studente sarà in grado di monitorare il consumo dell'energia, al termine del quale dovrà iniziare un nuovo tentativo, ripartendo dal punto iniziale
- Termine dell'apprendimento: tramite l'applicazione, lo studente riceverà una notifica indicante il termine dell'apprendimento e quindi il dover ritornare nell'aula iniziale ospitante l'attività

2 Idea applicazione

L'applicazione nasce dal bisogno di supportare l'esplorazione dello studente durante la sua attività al Muse.

2.1 Necessità

Per offrire la miglior esperienza possibile per gli studenti, si sono stabiliti sin dal principio i punti cardine su cui centrare lo sviluppo dell'applicazione. Dopo un'attenta analisi con la dr.ssa Vielmo, abbiamo individuato alcuni aspetti sui quali focalizzarsi:

Accessibilità Per alcune categorie di utenti, l'applicazione potrebbe risultare difficile da visualizzare. La decisione è stata quindi di realizzare un'applicazione che non penalizzasse in alcun modo studenti affetti da discromatopsia. La discromatopsia, comunemente nota come daltonismo [9], rappresenta un'anomalia visiva, comportante un'alterata percezione dei colori. Questa cecità parziale comporta la scarsa sensibilità ad alcuni colori, principalmente il rosso e il verde. La scelta, quindi, è ricaduta sull'utilizzo di una palette di colori che non ostacolasse l'apprendimento e l'esperienza dello studente affetto da discromatopsia.

Compatibilità Durante l'attività, gli studenti dovranno scaricare l'applicazione sul proprio dispositivo personale. È stato quindi necessario lo sviluppo di un'applicazione compatibile con i principali sistemi operativi presenti su smartphone, in modo da permetterne l'utilizzo alla maggior parte degli studenti. Secondo il sito *Statcounter*[8], per l'intera durata del 2024 i sistemi operativi più utilizzati in Italia sono stati Android (circa 70%) e iOS (circa 30%). Gli smartphone con un sistema operativo differente dai due appena citati riguardano una percentuale di utenti inferiore all'1%.

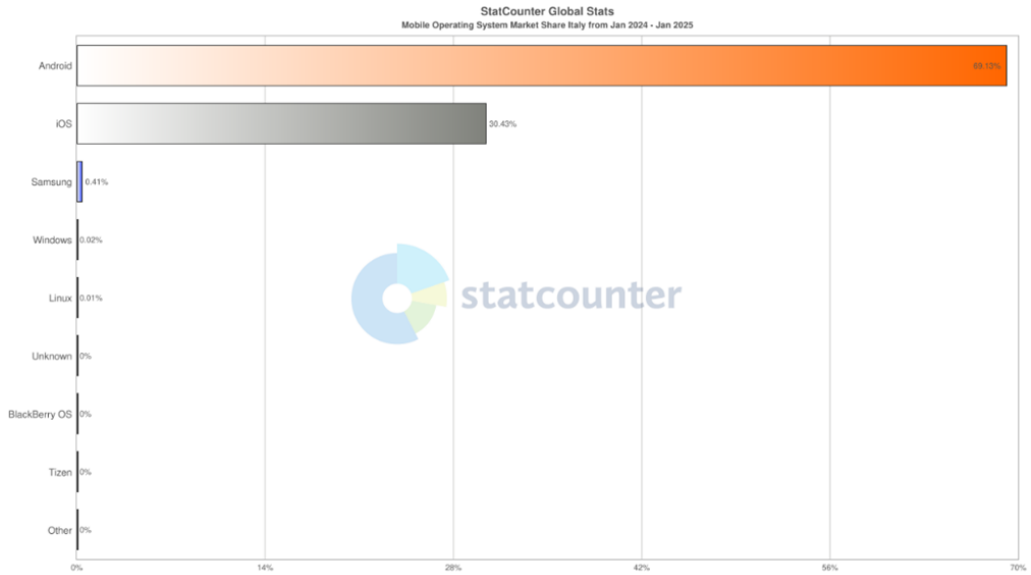

Figura 2.1: Sistemi operativi in Italia nel 2024 secondo *StatCounter*

In seguito a quest'analisi, abbiamo quindi deciso di sviluppare un'applicazione compatibile con i sistemi operativi Android e iOS.

Usabilità Un aspetto fondamentale dell'utilizzo di un'applicazione risiede nella facilità d'utilizzo e quindi nel suo design.

≪Design is not just what it looks like and feels like. The design is how it works≫ [20]

L'applicazione è stata quindi sviluppata in modo da avere un'interfaccia grafica che risultasse facilmente utilizzabile dal principale target d'età dell'attività, ovvero studenti di età compresa tra i quattordici e i diciannove anni circa.

Secondo il sito *Eleken*[19], l'usabilità di un'applicazione è strettamente collegata al suo design e quest'ultimo viene valutato in base a differenti parametri:

- *Success score*: la quantità di *task* eseguite correttamente in relazione al numero totale di *task* eseguite
- *Errors*: la quantità di azioni errate eseguite durante il completamento di una *task*
- *Task time*: la quantità media di tempo impiegata per eseguire una *task*
- Efficienza: definita come la combinazione di *success score* e *task time*

Per valutare questi parametri, si utilizza in genere una scala d'usabilità, compilata da coloro che testano l'applicazione. Abbiamo quindi creato un modulo Google da sottoporre agli studenti durante il test dell'applicazione, in maniera completamente anonima, per monitorarne l'usabilità.
Oltre agli aspetti implementativi, citati in precedenza, sono stati definiti anche alcuni aspetti non strettamente collegati allo sviluppo dell'applicazione, bensì a ciò che ne concerne. Dopo un incontro preliminare online con la dr.ssa Vielmo e gli organizzatori del Muse, sono stati definiti alcuni aspetti fondamentali per l'organizzazione e la pubblicazione dell'applicazione.

Il team del Muse ha fin da subito comunicato alcune *deadline*:

- Inizio novembre: idea di struttura dell'applicazione ben definita
- Inizio dicembre: sviluppo delle principali funzioni dell'applicazione
- Inizio gennaio: applicazione in fase beta, testabile da parte di alcune classi di studenti
- Inizio febbraio: applicazione completata e pubblicata

Durante l'incontro è stato stabilito inoltre che lo sviluppo sarebbe avvenuto in maniera autonoma e da remoto, con periodici incontri online utili per risolvere eventuali dubbi o criticità e monitorare il progredire della creazione dell'applicazione. Durante questa serie di incontri di aggiornamento, sono definiti alcuni aspetti cruciali per lo sviluppo, come il nome dell'applicazione e del progetto stesso, ovvero ExploRL, e alcune linee guida riguardanti gli standard di visualizzazione del Muse, che hanno registrato un forte impatto sul design.

2.2 *Flutter*

Flutter [11] è un *framework open-source*, sviluppato da Google con lo scopo di creare applicazioni multi-piattaforma sfruttando un'unica base di codice. Quest'approccio permette di avere un'alta efficienza nello sviluppo di applicazioni e, contemporaneamente, ridurre costi e tempi di produzione e manutenzione.

≪Write once, run anywhere≫ [16]

Uno dei suoi componenti principali è il *Flutter engine*. Scritto principalmente in C++, esso rappresenta il cuore di *Flutter*, fornendo supporto per il *rendering* a basso livello grazie alla libreria grafica di Google, *Skia Graphics* e permettendo di interfacciarsi in maniera ottimale con piattaforme come Android o iOS. Un punto di forza di *Flutter engine*, così come di *Flutter* in generale, perviene dalla sua scrittura nel linguaggio di programmazione *Dart* [10]. Tramite questo linguaggio, è possibile effettuare un "*hot-reload*" dell'applicazione, permettendo all'utente di verificare in maniera istantanea le modifiche apportate, senza la necessità di un riavvio completo o di un cambio di stato.

È possibile sviluppare applicazioni *Flutter* in vari ambienti di sviluppo, come *Android Studio*, *Visual Studio Code* o *Intellij IDEA*. Nello sviluppo di quest'applicazione, la scelta è ricaduta su *Android Studio*, un ambiente di sviluppo progettato da Google per la realizzazione di applicazioni per Android, succedendo alla versione standard precedente, realizzata da *Eclipse*.

2.2.1 Storia

Nato come progetto interno di Google, *Flutter*, inizialmente noto con il nome "*Sky*", vede la sua prima pubblicazione risalente al 23 ottobre 2014. Tuttavia, la prima versione soddisfacente è datata 4 dicembre 2018 [2], risultando per la prima volta un *framework* stabile. Infine, il 3 marzo 2021, è stato pubblicato *Flutter* 2.0 [3], consentendo per la prima volta lo sviluppo stabile di applicazioni multi-piattaforma (Android, iOS, Windows, MacOS, Linux, Web).

Quest'ultima innovazione, oltre alla capacità di permettere la creazione di interfacce utente moderne, ha portato *Flutter* ad un enorme utilizzo da parte di sviluppatori e aziende, risultando competitivo e in certi aspetti superiore ai principali *competitor* come *React* o *Xamarin*.

2.2.2 *Dart*

Si tratta del linguaggio di programmazione scelto per lo sviluppo di codice. Sviluppato anch'esso da Google e pubblicato nell'ottobre 2011 [1], *Dart* è un linguaggio di programmazione orientato agli oggetti il cui scopo principale fu fin da subito quello di sostituire *JavaScript* nello sviluppo web. Attualmente, *Dart* risulta il principale strumento per sviluppare applicazioni in *Flutter*, offrendo una *Virtual Machine* per la compilazione del codice, oltre a librerie e *repository* per la gestione di pacchetti e funzionalità.

Un aspetto interessante di *Dart* riguarda la sua tipizzazione dinamica, che porta l'utente a scegliere se utilizzare o meno le annotazioni sui tipi. Quest'aspetto permette di evitare problemi di ambiguità in alcuni punti e contemporaneamente agevolare la documentazione, rendendola semplice da leggere. Per quanto riguarda tipi, costrutti, funzioni e altri aspetti della programmazione base, *Dart* si conforma ai principali linguaggi di programmazione preesistenti, come *Python* o *Java*.

Dart si distingue dagli altri principali linguaggi di programmazione grazie alla potenza e versatilità del suo compilatore, che consente lo sviluppo su differenti piattaforme. Questo linguaggio supporta due modalità di compilazione: *Dart Native* e *Dart Web*.

- *Dart Native* risulta essere ottimale per l'esecuzione di programmi su dispositivi mobili, quindi con sistemi operativi come Android o iOS, ma anche su desktop, quindi con sistemi operativi come Windows, MacOS e Linux. Questo approccio permette di ridurre i tempi di esecuzione mantenendo prestazioni elevate, grazie alla compilazione AOT (*Ahead Of Time*), ovvero traducendo l'intero codice in linguaggio macchina prima dell'esecuzione del programma;

- *Dart Web* risulta essere ottimale per l'esecuzione di programmi direttamente nel *browser*. Questa variante consente una programmazione più rapida e flessibile, facilitando *debugging* e *testing* grazie alla compilazione JIT (*Just In Time*), ovvero traducendo il codice in linguaggio macchina solo nel momento in cui risulta essere necessario.

Grazie a questa flessibilità, *Dart* risulta quindi essere una scelta ottimale per lo sviluppo multipiattaforma, consentendo la scrittura di un'unica porzione di codice distribuibile contemporaneamente su web, desktop e *mobile*, il tutto senza riduzioni di prestazioni.

2.3 Progettazione ambiente di gioco

All'interno del *reinforcement learning*, uno dei principali esempi di scenario consiste nell'allenare un agente a trovare l'uscita da un labirinto eseguendo il minor numero di mosse possibili. Generalmente, si assume che non sia consentito il passaggio attraverso i muri e che la struttura del labirinto (compresi punto di partenza e di arrivo) rimanga immutata nel tempo.

Anche all'interno di questo progetto è stata scelta una struttura a labirinto, per due motivi principali: la semplicità con cui vi si collega il tema del *reinforcement learning*, facilitandone la trattazione, e la possibilità del partecipante di immedesimarsi nel ruolo di agente nel tentativo di uscire dal labirinto.

La struttura utilizzata è definita come una griglia composta da quattro righe e cinque colonne, per un totale di venti caselle.

Figura 2.2: Labirinto (vuoto)

Dove la casella gialla rappresenta il punto di partenza, quella azzurra rappresenta il punto di arrivo e le caselle nere dei muri non attraversabili.

È interessante notare la presenza di *loop*, ovvero percorsi ciclici che potrebbero causare un cammino potenzialmente infinito. I cicli sono importanti all'interno del *reinforcement learning* poiché permettono all'agente di riconoscere quando un percorso è ciclico, quindi errato, ed evitarlo in futuro.

L'obiettivo è quello di raggiungere il punto di arrivo attraversando il minor numero di celle possibili, per questo motivo verrà assegnato un *reward* negativo ad ogni passo, incentivando l'agente a percorrere la strada più breve possibile.

Il labirinto viene quindi popolato da alcuni animali, uno per ogni cella, con due specifiche caratteristiche: il suo *habitat* naturale (ambiente) e la sua alimentazione prevalente. I possibili ambienti (Africa, America, Eurasia e Oceania) rappresenteranno le righe della tabella, mentre le alimentazioni prevalenti (Carnivoro, Erbivoro, Insettivoro, Onnivoro e Piscivoro) rappresenteranno le colonne.

Avremo quindi la seguente tabella

	Erbivoro	Onnivoro	Carnivoro	Piscivoro	Insettivoro
Americhe	Capibara	Opossum della Virginia		Leone marino sudamericano	Formichiere gigante
Eurasia	Cervo nobile	Tasso	Lince eurasiatica	Tricheco	Riccio comune
Oceania	Wallaby dal collo rosso		Dingo		Ornitorinco
Africa	Springbok	Lemure catta	Leone (africano)	Otaria orsina del Capo	Oritteropo

Figura 2.3: Labirinto (completo)

Le informazioni riguardanti ambiente e alimentazione verranno mostrate dall'applicazione nel momento in cui l'agente raggiunge un determinato animale.

All'interno dell'attività, il *reward* sarà indicato come "punto energia". L'agente inizierà l'attività con un quantitativo di dieci punti energia, i quali diminuiranno di uno per ogni passo effettuato all'interno del labirinto. Una volta esauriti i punti energia, l'agente ripartirà dall'animale di partenza (Opossum della Virginia) effettuando un nuovo tentativo e percorrendo strade alternative, che potrebbero risultare migliori della precedente. Se l'agente dovesse arrivare all'animale vincente (Leone marino sudamericano), esso ripartirà comunque dall'animale di partenza, con lo scopo di raggiungere nuovamente l'obiettivo consumando però un quantitativo di energia inferiore rispetto al tentativo precedente. Il numero limitato di energia è stato studiato per favorire una successione di tentativi (anziché un unico grande tentativo) e la propensione dell'agente al ragionamento, scegliendo accuratamente quale passo eseguire.

3 Descrizione applicazione

Di seguito, vengono mostrate le varie schermate e *feature* attualmente presenti all'interno dell'applicazione, oltre ad alcune sezioni di codice fondamentali all'interno del progetto. Al termine di questo elenco, vi sarà inoltre una sezione dedicata alle criticità emerse in fase di sviluppo e di pubblicazione, analizzate nel dettaglio nel capitolo ad esse dedicate.

Come struttura architetturale, ho scelto di utilizzare una variante della classica MVC (*Model-View-Controller*), adattandola all'ambiente *Flutter*. Abbiamo quindi una MWC (*Model-Widget-Controller*), suddivisa in quattro macro-cartelle: *models* (*model*), *pages* (*widget*), *resources* (*widget*) e *providers* (*controller*)

3.1 *Models*

Questa cartella contiene le classi che rappresentano i dati, all'interno è presente una classe principale, ovvero quella dell'oggetto Animale.

Questa struttura contiene informazioni utili per l'esplorazione dell'utente (nome comune e scientifico, alimentazione, ambiente e numero di piano nel quale si trova l'animale all'interno del Muse) e informazioni utili all'interno dell'applicazione (foto rappresentativa e lista di animali adiacenti nel labirinto).

Per motivi pratici ho scelto di rendere tutti i parametri (esclusa la lista di animali vicini) necessari al momento dell'istanziazione dell'oggetto.

```dart
class Animale {

  String normalName, scientificName, alimentazione, ambiente, resPhoto;
  int nPiano;
  List<Animale> neighbors = [];

  Animale({
    required this.normalName, required this.scientificName, required this.alimentazione, required this.ambiente, required this.nPiano, required this.resPhoto,
  });
}
```

Figura 3.1: Struttura classe animale

3.2 *Pages*

In questa cartella sono presenti i file relativi alle varie schermate dell'applicazione, di seguito sono riportate le principali:

3.2.1 Animali già visitati

In questa pagina vengono mostrati i dati sugli animali visitati nel corso dell'esplorazione, divisi tra round attuale e quelli precedenti. All'interno di questo file viene utilizzato il *package SharedPreferences* per il recupero delle informazioni sui round precedenti.

SharedPreferences [18] rappresenta un metodo di salvataggio dei dati all'interno dell'applicazione, permettendo quindi, nel nostro caso, di mantenere traccia delle informazioni sui round precedenti. In particolare, il pacchetto *SharedPreferences* permette il salvataggio ed il recupero di dati in formato chiave-valore.

```
void loadData(GameProvider gProvR, EnergyProvider enProvR,AnimalProvider exProvR) async {

  final SharedPreferences prefs = await SharedPreferences.getInstance();      //Istanza di sharedPreferences

  gProvR.initListOfMatch();
  gProvR.initListOfEnergy();
  gProvR.setEnergyToDisplay(enProvR.energy);

  late List<String> match = [];

  setState(() {
    late int nPartite = prefs.getInt(MyString.nMatch) ?? 0;                   //Recupero il numero di partite (0 se è la prima)
    for (int i = 1; i <= nPartite; i++) {                                     //La prima partita viene salvata come Game-1
      match = prefs.getStringList(MyString.gameMatch(i)) ?? [];               //Se è null non prendo niente
      int remainingEnergy = EnergyProvider.maxEnergy - match.length;          //enMax - 1 * nExhVisitati
      gProvR.addToListOfMatch(match);
      gProvR.addToListOfEnergy(prefs.getInt(MyString.gameEnergy(i)) ?? remainingEnergy);
    }
    gProvR.setSelectedMatch(gProvR.listOfMatch.length.toString());
    gProvR.setListOfThisRound(context.read<AnimalProvider>().visited);
    gProvR.setListToDisplay(gProvR.listOfThisRound);
  });
}
```

Figura 3.2: Salvataggio dati tramite *SharedPreferences*

L'utente potrà selezionare la lista di animali visitati tramite un menu a tendina e scorrere tale lista tramite semplice trascinamento.

All'interno della lista vengono riportate le informazioni principali per lo sviluppo di una strategia, come: alimentazione e ambiente, oltre al nome e all'ordine di visita.

Figura 3.3: Schermate "Cosa hai già visitato"

A questa schermata vi si potrà accedere in più situazioni:

- Dopo aver raggiunto un animale, lo studente può controllare le scelte fatte in precedenza e, sulla base di ciò sviluppare una strategia che gli permetta di ottenere un risultato diverso. Supponiamo ad esempio che, dopo aver scansionato un animale X, lo studente si trovi davanti a un bivio, che prevede di proseguire verso l'animale A oppure verso l'animale B. Lo studente sceglierà l'animale A e proseguirà in tale direzione, senza riuscire ad arrivare all'obiettivo. Nel round successivo, trovandosi di nuovo allo stesso bivio, lo studente, tramite questa schermata, noterà come la scelta dell'animale A non abbia portato al risultato sperato, sceglierà quindi, sviluppando una strategia, la strada per l'animale B, stabilendo poi se tale scelta sia quella corretta oppure no;

- Al termine del round, lo studente può analizzare il percorso che lo ha portato all'animale vincente o all'esaurimento dell'energia, in questo modo potrà pianificare un nuovo percorso che permetta di migliorare il tentativo del round precedente. Supponiamo ad esempio che, al termine di un round, lo studente sia riuscito ad arrivare all'animale vincente per la seconda volta, ma con due quantitativi di energia rimanenti differenti. Tramite questa schermata, potrà notare come la scelta di una determinata strada sia risultata più corretta di un'altra, seppur entrambe risultanti nella vittoria del round;

- Al termine della partita, una volta tornato in classe e scoperta la struttura del labirinto, lo studente potrà capire gli errori commessi e basarsi su questi per lo sviluppo di una strategia nella fase progettista.

3.2.2 Ricerca animale

Questa schermata si divide in due fasi:

Una fase iniziale, definita fase di ricerca, dove lo studente dovrà localizzare l'animale all'interno del Muse. Per farlo avrà a disposizione alcune informazioni fondamentali come il nome dell'animale e il piano del Muse all'interno del quale si trova. Una volta trovato l'animale, lo studente potrà scansionare il *QR-code* ad esso associato e passare alla fase successiva.

Qualora lo studente avesse difficoltà nel riconoscere l'animale dalla schermata generale, cliccando su di esso potrà ottenere un'immagine più grande, facilitandone il riconoscimento.

Fase di pianificazione Una volta scansionato il *QR-code* associato all'animale, lo studente otterrà informazioni fondamentali per la pianificazione del prossimo spostamento, ovvero la sua alimentazione e la sua località geografica. Sulla base di queste informazioni, e basandosi sui tentativi precedenti, potrà selezionare il prossimo animale da visitare.

Ci sono due eccezioni per quest'ultima fase, qualora lo studente dovesse aver terminato la partita, poiché ha trovato l'animale vincente o ha esaurito l'energia disponibile, oltre a controllare il percorso compiuto, avrà a disposizione un pulsante per iniziare un nuovo round, nel tentativo di migliorare quello precedente.

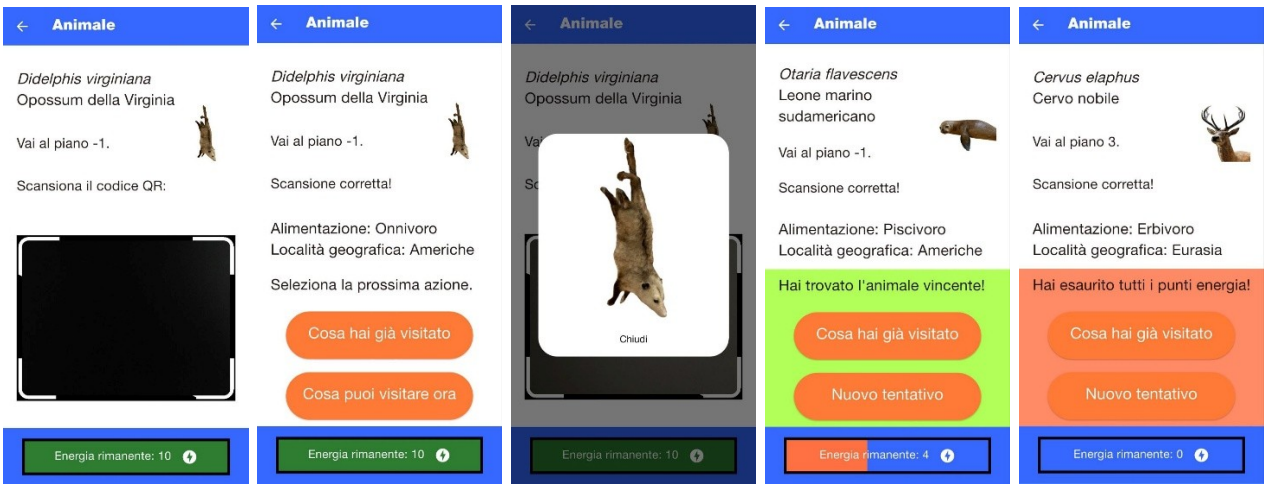

Figura 3.4: Schermate "Ricerca animale"

A questa schermata vi si potrà accedere in più situazioni:

- Avviando un round, una volta avviato un round, si accederà a questa schermata, facendo partire l'utente dall'animale iniziale prestabilito (Opossum della Virginia);

- Selezionando un animale da raggiungere, una volta scelto l'animale da visitare, lo studente verrà reindirizzato a questa schermata, ottenendo le informazioni per raggiungerlo.

3.2.3 *Home page*

La pagina iniziale dell'applicazione, all'interno di essa ci sono titolo, testo introduttivo, loghi dei partecipanti al progetto (Università di Trento – Dipartimento di Ingegneria e Scienze dell'Informazione, Università di Trento – Dipartimento di Matematica e Muse – Museo delle Scienze di Trento) e due pulsanti:

- Pulsante di avvio dell'attività, attraverso il quale lo studente potrà accedere al tutorial introduttivo e successivamente avviare l'applicazione;
- Pulsante *credits*, attraverso il quale lo studente avrà a disposizione info sulla progettazione dell'applicazione, sullo sviluppo e sul fotografo.

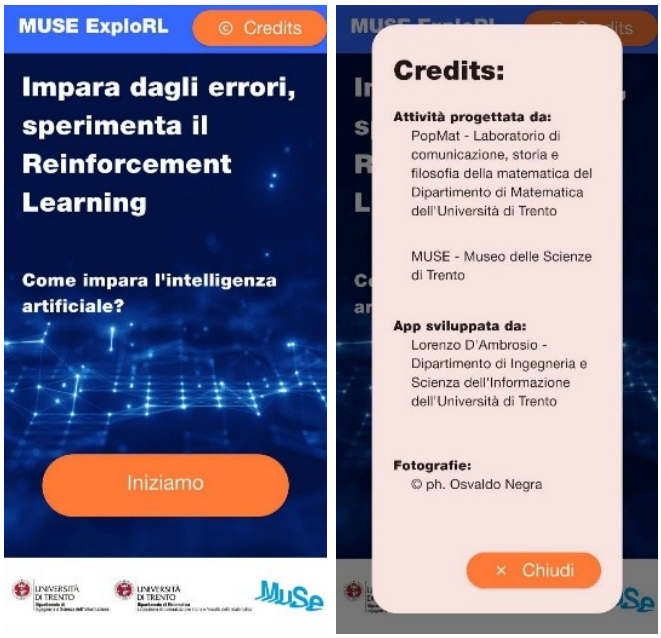

Figura 3.5: Schermate *Home* e *Credits*

A questa schermata vi si potrà accedere in più situazioni:

- Una volta avviata l'applicazione;
- Tornando indietro dalla schermata di tutorial

3.2.4 Animali da visitare

In questa schermata lo studente avrà a disposizione la lista di animali raggiungibili partendo da quello in cui si trova in quel momento. Esso avrà quindi a disposizione due scelte:

- Pianificare la prossima scelta, tramite l'apposito pulsante, lo studente potrà controllare le scelte fatte in precedenza e, sulla base di esse, stabilire quale sia la strada migliore da percorrere;
- Selezionare l'animale da raggiungere, dalla lista di animali a disposizione, l'utente potrà cliccare su uno di essi e successivamente ottenere le informazioni necessarie per raggiungere tale animale.

A questa schermata vi si potrà accedere dopo aver scansionato il *QR-code* relativo all'animale cercato.

Figura 3.6: Schermate "Animali da visitare"

3.2.5 Obiettivo attività

In questa schermata l'utente troverà alcune istruzioni riguardanti l'attività, oltre al pulsante per avviare il round iniziale.

Figura 3.7: Schermata Obiettivo

A questa schermata vi si potrà accedere dopo aver completato il tutorial.

3.2.6 Tutorial

Questa schermata permette allo studente di familiarizzare con l'applicazione, mostrandone le principali funzionalità. Essa è suddivisa in cinque sezioni:

- Sezione di info, dove verranno mostrate alcune informazioni riguardanti applicazione e tutorial stesso.

- Tutorial energia, dove lo studente potrà simulare una scansione e notare il decremento progressivo dell'energia.
- Tutorial scansione, dove lo studente potrà simulare una scansione, con un *QR-code* fornito nella presentazione dell'attività all'interno del Muse.
- Tutorial animali già visitati, dove lo studente familiarizzerà con la tabella contenente le informazioni sulle scelte effettuate durante l'attività
- Tutorial animali da visitare, dove lo studente familiarizzerà con la scelta del prossimo animale da visitare

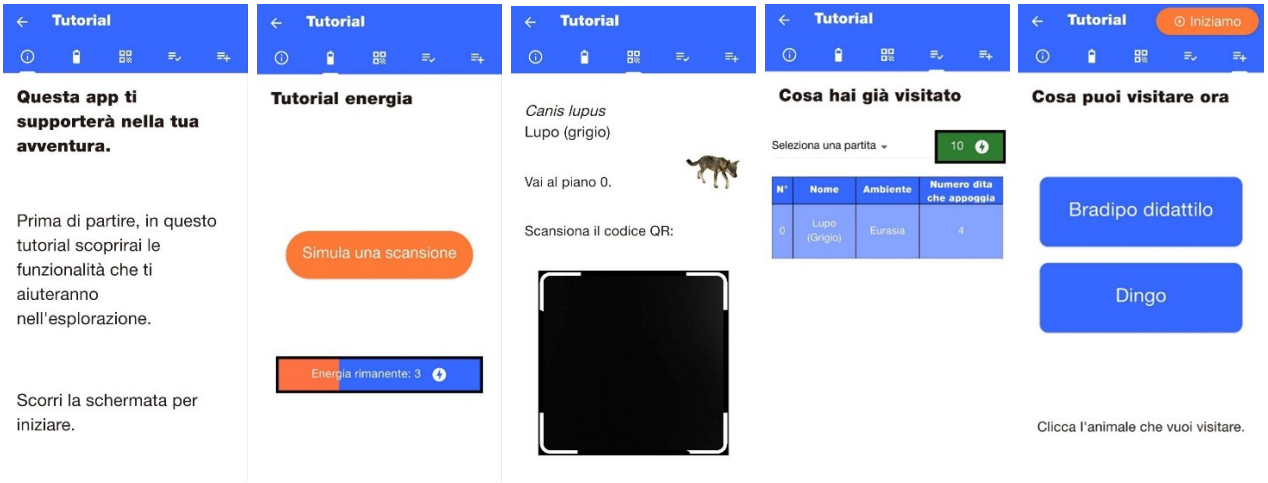

Figura 3.8: Schermate Tutorial

Si noti come soltanto nell'ultima schermata sia presente il pulsante per iniziare l'attività, quest'ultimo infatti comparirà soltanto dopo aver visionato tutte le cinque sezioni del tutorial (non obbligatoriamente nell'ordine qui mostrato).

A questa sezione vi si potrà accedere dalla schermata Home o da quella di obiettivo.

3.3 *Resources*

Questa cartella contiene tutti i *widget* e le altre tipologie di risorse utilizzabili all'interno dell'applicazione, divisi per categoria. L'obiettivo di questa sezione consiste nel mantenere il codice, in particolare le varie pagine, il più pulito e organizzato possibile. Grazie a questa suddivisione, risulta molto più semplice la gestione e la modifica dei componenti all'interno delle varie pagine.

All'interno della cartella, le risorse sono a loro volta divise in due sottosezioni:

- *Widget* veri e propri: come *Column*, *Row*, *AppBar*, ...
- Valori: file contenenti interi, stringhe, colori, ...

Questa struttura permette di migliorare la scalabilità del progetto e facilita l'eventuale espansione futura dell'applicazione, rendendo il codice più organizzato e mantenibile.

I *widget* rappresentano un punto cardine dello sviluppo in *Flutter*, contraddistinguendo quest'ultimo dagli altri *framework*. Un *widget* rappresenta un aspetto dell'interfaccia utente, come elementi grafici, testo o animazioni.

I *widget* si dividono in due macro-tipologie: *stateless widget* e *stateful widget*.

- *Stateless widget*: tipologia di *widget* che impone l'immutabilità del proprio contenuto. Generalmente sono utilizzati per contenere elementi che non necessitano di cambiare nel corso dell'esecuzione dell'applicazione. Gli *stateless widget* più comuni sono i testi o le immagini;

- *Stateful widget*, tipologia di *widget* che consente la mutabilità del proprio contenuto, mantenendo le informazioni ad esso collegate. Questi *widget* sono generalmente utilizzati per contenuti dinamici, che necessitano di variare durante l'esecuzione dell'applicazione. Gli *stateful widget* più comuni sono i pulsanti.

È possibile combinare più *widget* tra loro per ottenerne di più complessi, permettendo lo sviluppo di interfacce grafiche complesse e ottimali per ogni tipologia di sviluppo. *Flutter* contiene inoltre due principali *set* di *widget* conformi a specifici linguaggi di progettazione. Esiste infatti il *set* di *widget Material Design*, che implementano il design di Google (e quindi Android), e il *set* di *widget Cupertino*, che implementano il design iOS.

3.4 *Providers*

Questa cartella contiene vari file, fondamentali nella gestione degli stati all'interno dell'applicazione, ovvero i *provider*. Essi rappresentano un punto centrale, connesso con le varie pagine, tramite il quale i widget riconoscono la variazione di un dato e sono quindi in grado di alterare il proprio stato.

Nel momento in cui un dato all'interno di un *provider* cambia, questa modifica viene segnalata a tutti i *widget* "in ascolto", i quali potranno quindi agire di conseguenza, variando di stato. Questa modifica viene comunicata tramite la funzione *notifyListeners*() della classe *ChangeNotifier*, implementata da ogni classe *provider*.

```
void decreaseEnergy(){ if(_energy > 0) { _energy--; notifyListeners(); } }
```

Figura 3.9: Notifica diminuzione dell'energia

Prendendo l'esempio qui sopra, osserviamo come, all'invocazione della funzione *decreaseEnergy*(), si decrementa (se possibile) il valore dell'energia e successivamente si notifica a tutti i *widget* in ascolto l'avvenuta modifica. Alla ricezione della notifica di cambiamento, i vari *widget* (come, ad esempio, il visualizzatore della batteria) aggiorneranno il proprio stato, mostrando a schermo la nuova quantità di energia.

4 *Testing*

Una volta sviluppata una versione beta dell'applicazione, è stato effettuato un test con la classe al primo anno del Liceo Linguistico Sophie Magdalena Scholl. Le risposte sono state in forma completamente anonima. Al termine di quest'ultimo, è stato proposto agli studenti un questionario anonimo da compilare, per valutare l'usabilità di quanto prodotto.

Il questionario si è basato sulla *System Usability Scale* (SUS), ovvero una delle metriche più utilizzate per capire, tramite valutazioni soggettive, l'usabilità di un'applicazione.

Questa scala è stata sviluppata nel 1986 dall'inglese John Brooke [7] il quale stabilì come l'usabilità di un sistema fosse strettamente dipendente da tre principali parametri: il contesto di utilizzo, il motivo di utilizzo e l'ambiente di utilizzo. La SUS è composta da dieci domande dove, per ognuna di esse, il compilante del questionario dovrà fornire un valore compreso tra uno e cinque, dove uno equivale a "Fortemente in disaccordo" e cinque equivale a "Assolutamente d'accordo". Le domande sono:

1. Utilizzerei nuovamente l'applicazione in futuro
2. Trovo l'applicazione inutilmente complessa
3. Penso che l'applicazione sia facile da usare
4. Penso che avrei bisogno di supporto per l'utilizzo di quest'applicazione
5. Trovo che le varie funzionalità dell'applicazione siano ben integrate
6. Penso che ci sia troppa incoerenza all'interno dell'applicazione
7. Penso che la maggior parte delle persone possa imparare velocemente a utilizzare l'applicazione
8. Trovo l'applicazione difficile da usare
9. Mi sono sentito a mio agio nell'utilizzare l'applicazione
10. Ho dovuto imparare molte cose prima di poter utilizzare l'applicazione

Vista la natura fortemente circostanziale dell'applicazione, è stata esclusa la prima domanda dal sondaggio, poiché non ne è previsto un utilizzo al di fuori dell'attività per la quale è stata progettata e sviluppata.

Una volta ottenuti i risultati, sono stati utilizzati per calcolare il "valore SUS", dato dall'equazione [13]:

$$S_{TOT} = 2.5 \left(20 + \sum \{S_1, S_3, S_5, S_7, S_9\} - \sum \{S_2, S_4, S_6, S_8, S_{10}\}\right) \tag{4.1}$$

Dove:

- Il valore 20 iniziale ha lo scopo di normalizzare il punteggio di base, evitando di avere valori negativi in caso di valutazione pessima, ottenibile assegnando un valore minimo alle domande "positive" (quindi $\{S_1, S_3, S_5, S_7, S_9\}$ e massimo alle domande "negative" (quindi $S_2, S_4, S_6, S_8, S_{10}$)
- Dal risultato di ogni domanda, si sottrae uno
- Il fattore moltiplicativo 2.5 serve per rendere il risultato in un *range* tra 0 e 100, anziché in un *range* tra 0 e 40

Trovo che l'app sia inutilmente complessa.	Penso che l'app sia facile da usare.	Penso che avrei bisogno di supporto per usare quest'app.	Trovo che le varie funzionalità dell'app siano bene integrate.	Penso ci sia troppa incoerenza nell'app.	Penso che la maggior parte delle persone possano imparare a usare l'app facilmente.	Trovo che questa app sia difficile da usare.	Mi sono sentito a mio agio nell'usare l'app.	Ho avuto bisogno di imparare molte cose prima di riuscire a usare al meglio l'app.	Valore SUS
1	5	1	5	1	5	1	5	5	85
1	1	1	4	1	5	1	4	2	77,5
2	5	4	5	1	5	2	4	4	72,5
2	5	2	5	2	5	1	5	1	87,5
3	3	3	3	3	3	3	3	3	50
2	5	2	4	1	5	1	4	2	82,5
1	4	1	4	1	4	1	5	2	85
1	5	1	5	1	5	1	5	1	95
4	4	1	4	3	1	2	5	1	65
1	5	1	4	1	4	1	5	1	90
4	4	1	3	1	3	2	2	1	65
2	5	1	5	1	5	1	4	2	87,5
2	3	5	4	1	3	5	1	4	42,5
1	4	1	4	1	4	1	5	1	87,5
1	5	1	4	1	5	1	5	1	92,5
3	3	4	4	3	4	3	4	5	50
1	5	1	5	1	5	5	5	1	85
1	4	1	5	1	2	1	5	1	85
2	5	1	2	2	5	1	5	2	80
4	5	1	4	3	5	2	4	3	70
1	5	1	5	1	5	1	5	1	95
2	4	2	4	1	4	2	4	3	72,5

Figura 4.1: Risultati sondaggio

Non avendo posto il quesito numero uno, è stato scelto di assegnare un valore pari a tre, in modo da non influenzare il calcolo e rimanendo quindi neutrale.
Il sondaggio è stato sottoposto a 18 studenti e 4 operatori interni del Muse (le cui risposte sono evidenziate in azzurro), ottenendo i seguenti risultati:

Facendo una media dei valori SUS per ogni singola risposta, otteniamo un valore di 77.39% (75.97% escludendo gli operatori del Muse) un valore che, all'interno della scala di valutazione finale stabilita da Aaron Bangor, Philip Kortum e James Miller [6], corrisponde al grado C "*good*".

Prendendo però il valore di moda di ogni singola risposta (escludendo quindi possibili *outlier*), notiamo come la valutazione cresca notevolmente:

Trovo che l'app sia inutilmente complessa.	Penso che l'app sia facile da usare.	Penso che avrei bisogno di supporto per usare quest'app.	Trovo che le varie funzionalità dell'app siano bene integrate.	Penso ci sia troppa incoerenza nell'app.	Penso che la maggior parte delle persone possano imparare a usare l'app facilmente.	Trovo che questa app sia difficile da usare.	Mi sono sentito a mio agio nell'usare l'app.	Ho avuto bisogno di imparare molte cose prima di riuscire a usare al meglio l'app.	Valore SUS
1	5	1	4	1	5	1	5	1	92,5

Figura 4.2: Risultati sondaggio

Ottenendo una valutazione corrispondente al grado A, "*best imaginable*". Questa discrepanza è dovuta al bacino d'utenti del sondaggio, contenente un gruppo di studenti con interessi differenti e una differente volontà di partecipare attivamente all'attività. A prescindere dalla valutazione considerata, è evidente come l'applicazione sia stata percepita dagli utenti come intuitiva, facile da usare e ben progettata, senza evidenziare particolari ostacoli nell'interazione. Il completamento dei vari compiti viene quindi valutato facile e l'utilizzo dell'applicazione risulta piacevole.

5 Problemi affrontati

Durante le varie fasi dello sviluppo dell'applicazione, sono emersi vari problemi, in merito all'apprendimento, alla burocrazia, allo sviluppo e alla tempistica:

Accessibilità e Compatibilità Come già accennato nella sezione riguardante le necessità, è stato concordato l'obiettivo di sviluppare un'applicazione che:

- Fosse disponibile su Android e iOS
- Fosse visualizzabile anche da soggetti affetti da discromatopsia

Per risolvere il primo punto è stato quindi scelto di lavorare utilizzando *Flutter* e *Dart*, i quali permettono lo sviluppo di applicazioni multipiattaforma. Per quanto riguarda la discromatopsia, è stato scelto l'utilizzo di una palette di colori tendente all'azzurro (ricordante i colori caratteristici del Muse), in modo da risultare visualizzabile a tutti.

Conoscenze tecniche *Flutter* e *Dart* sono due strumenti con i quali non avevo alcuna familiarità e dei quali avevo una conoscenza minima. È stata quindi necessaria una fase iniziale concentrata esclusivamente sull'apprendimento e sulla familiarizzazione dei due. Entrambi sono risultati strumenti molto conosciuti a livello globale, quindi il quantitativo di informazioni e tutorial a disposizione si sono rilevati più che sufficienti per l'apprendimento.

Linee guida del Muse Come la maggior parte degli enti che associano il proprio marchio ad un prodotto, anche il Muse richiede che tale prodotto rispetti alcune linee guida, come ad esempio:

- Modalità di scrittura del marchio Muse
- *Font* e dimensione testo prestabiliti
- Modalità d'inserimento del logo Muse
- Comunicazione testuale e termini da utilizzare

Queste indicazioni sono risultate un ostacolo poiché fornite in seguito allo sviluppo della prima versione dell'applicazione; tuttavia, sono state risolte e concordate con i responsabili del Muse.

Pubblicazione La pubblicazione è stata più complicata del previsto, dovuta anche a causa di vincoli economici e tecnici:

- Pubblicazione su Android: secondo le linee guida fornite da Google per la pubblicazione di applicazioni sul *PlayStore*, essa può avvenire solo da parte di un account sviluppatore [12], che ha quindi versato una quota forfettaria pari a 25$. Account sviluppatore e quota d'iscrizione sono stati pagati dal Muse;
- Pubblicazione su iOS: secondo le linee guida fornite da Apple per la pubblicazione di applicazioni sull'*AppStore*, essa può avvenire solo tramite un dispositivo Mac e solo da parte di un account sviluppatore [4], che ha quindi sottoscritto un abbonamento da 99$ all'anno. Account sviluppatore e quota d'iscrizione sono stati pagati dal Muse, mentre per la pubblicazione è stato utilizzato un Mac presente all'interno del Muse.

Necessità di un timer In seguito ai primi test effettuati con le classi, riportate nella sezione *Testing* di questo elaborato, il principale problema emerso ha riguardato l'impossibilità di comunicare ai partecipanti il termine dell'esplorazione, con conseguenti perdite di tempo dovute alla comunicazione manuale e allo spostamento verso l'aula principale. Per risolvere tale problema, è stato pensato di inserire un timer di 30 minuti, al termine del quale l'applicazione, tramite notifica *pop-up*, avrebbe indicato agli studenti il termine dell'attività, invitandoli a tornare in aula.

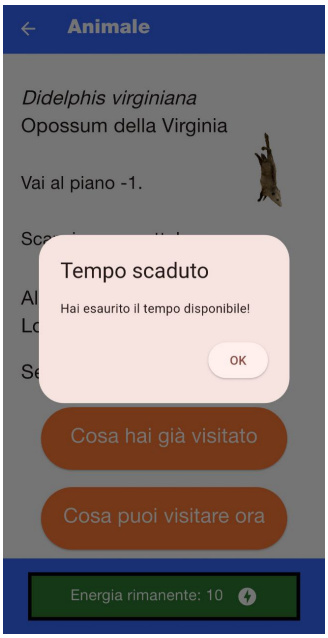

Figura 5.1: Notifica tempo scaduto

Mantenimento futuro Una volta terminata l'esperienza di tirocinio, l'applicazione verrà mantenuta da un'azienda esterna collaborante con il Muse. Per poter fare ciò, l'azienda deve conoscere come e quali parametri modificare per aggiornare l'applicazione in futuro. A tal proposito, ho realizzato una guida per il mantenimento dell'applicazione, riportata integralmente in appendice. Essa si divide nelle seguenti macro-categorie:

- Modifiche all'ambiente dell'attività
- Modifiche alla teoria dell'attività
- Aggiornamenti sugli *store*

Queste categorie vengono spiegate maggiormente nel tutorial in appendice.

6 Conclusioni

Questa tesi presenta lo sviluppo ed il risultato finale del tirocinio svolto in collaborazione con il Muse. Il tirocinio è iniziato con qualche incertezza e difficoltà dovute principalmente all'inesperienza in questo ambito, scomparse con fasi di studio e di apprendimento riguardanti il mondo di *Flutter* e *Dart*. Successivamente si sono presentate alcune divergenze con il Muse, nel tentativo di trovare un punto in comune tra le loro esigenze e l'idea della dr.ssa Vielmo, oltre alla praticità dello sviluppo, si è giunti ad un accordo grazie a degli incontri online e delle spiegazioni più dettagliate. Infine, vi sono state criticità in fase di *testing* e di sviluppo, risolte grazie alla collaborazione ed al supporto generale.

6.1 Sviluppi futuri

Dando uno sguardo generale all'attività appena conclusa, sottolineo alcuni aspetti di sviluppo, quindi di possibili aggiornamenti futuri, all'interno dell'applicazione.

A livello di codice, uno sviluppatore esperto e con un buon bagaglio d'esperienza relativo allo sviluppo con *Flutter*, potrà sicuramente ottimizzare alcune porzioni di codice, modificarne la struttura ed alterarne il funzionamento. Probabilmente io stesso, con l'esperienza guadagnata da questo tirocinio, mi approccerei allo sviluppo dell'applicazione in maniera differente.

A livello di mantenimento, come accennato in precedenza, l'applicazione verrà mantenuta da un'azienda affiliata con il Muse. Dopo un incontro avvenuto con tale azienda e tramite il supporto fornito da questo elaborato, in particolare il tutorial presente in appendice, l'azienda potrà effettuare le modifiche necessarie al mantenimento dell'applicazione durante tutto il suo ciclo vitale all'interno delle attività del Muse. L'obiettivo sarà sempre quello di supportare al massimo gli studenti ed in generale i partecipanti all'attività [14].

Per quanto riguarda possibili miglioramenti futuri, oltre alle possibili modifiche all'ambiente o alla teoria dell'applicazione, si può parlare di miglioramenti del design o aggiunta di nuove funzionalità. Un esempio di nuova funzionalità potrebbe essere l'implementazione di una mappa che tiene traccia, in maniera più visiva rispetto alla tabella presente nella sezione "Animali già visitati", dei percorsi intrapresi durante l'esplorazione.

6.2 Considerazioni personali

Quest'esperienza è stata fondamentale per la mia crescita personale ed avrà sicuramente un forte impatto nel mio percorso. Grazie ad essa ho potuto avere un assaggio di cosa comporta un lavoro del genere: dallo sviluppo autonomo, al rapportarsi con i clienti (in questo caso il Muse), all'avere delle scadenze e al fornire periodici aggiornamenti sul lavoro prodotto. Tramite questo tirocinio ho inoltre avuto l'opportunità di familiarizzare con uno dei principali *framework* per lo sviluppo di applicazioni mobile, ovvero *Flutter*.

Sono grato al Muse che mi ha supportato ed aiutato in questo percorso, fornendomi gli strumenti (come il Mac per il caricamento dell'applicazione sull'*AppStore*), gli accessi (agli account sviluppatore di Google ed Apple) e la disponibilità al confronto in qualsiasi momento.

Sono contento della quantità di concetti affrontati in vari corsi universitari (come Ingegneria del *software*, Programmazione di sistemi mobili e tablet, Programmazione 1 e 2, Algoritmi e strutture dati, *Introduction to Machine Learning* e altri) ed applicati in maniera attiva e reale all'interno di questo tirocinio ed in generale sono soddisfatto del lavoro svolto, anche se per natura tendo sempre a focalizzarmi sui possibili nuovi aspetti da migliorare.

Bibliografia

[1] Annuncio dart. https://gotocon.com/aarhus-2011/presentation/Opening%20Keynote:%20Dart,%20a%20new%20programming%20language%20for%20structured%20web%20programming.

[2] Annuncio flutter 1. https://web.archive.org/web/20180925220106/https://www.apptunix.com/whats-new-in-googles-flutter-for-mobile-app-developers/.

[3] Annuncio flutter 2. https://developers.googleblog.com/en/announcing-flutter-2/.

[4] Linee guida pubblicazione sull'appstore. https://developer.apple.com/it/support/terms/.

[5] Cos'è l'apprendimento per rinforzo. https://aws.amazon.com/it/what-is/reinforcement-learning/.

[6] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: adding an adjective rating scale. *J. Usability Studies*, 4(3):114–123, May 2009.

[7] John Brooke. *SUS – a quick and dirty usability scale*, pages 189–194. Taylor & Francis, 01 1996.

[8] Classifica so in italia. https://gs.statcounter.com/os-market-share/mobile/italy.

[9] Daltonismo. https://www.humanitas.it/malattie/daltonismo/.

[10] Dart. https://pub.dev/.

[11] Flutter. https://flutter.dev/.

[12] Linee guida pubblicazione sul playstore. https://support.google.com/googleplay/android-developer/answer/9859751.

[13] James R. Lewis. The system usability scale: Past, present, and future. *International Journal of Human–Computer Interaction*, 34(7):577–590, 2018.

[14] Link attività sul sito del muse. https://ilmuseperlascuola.muse.it/attivita-didattica/reinforcemente-learnig/.

[15] Andrej andreevič markov (wikipedia). https://it.wikipedia.org/wiki/Andrej_Andreevi%C4%8D_Markov_(1856-1922).

[16] Sun Microsystems. Write once, run anywhere. https://en.wikipedia.org/wiki/Write_once,_run_anywhere, 1995.

[17] Muse - museo delle scienze di trento. https://www.muse.it/.

[18] Package shared preferences. https://pub.dev/packages/shared_preferences.

[19] Usability metrics. https://www.eleken.co/blog-posts/usability-metrics.

[20] Rob Walker. The guts of a new machine. *The New York Times Magazine*, 2003.

Allegato A Mantenimento

All'interno di questo file, vengono spiegate e mostrate alcune modifiche apportabili all'applicazione ExploRL, un'applicazione di supporto per l'attività "Impara dagli errori, sperimenta il Reinforcement Learning" presente al Muse – Museo delle Scienze di Trento. Il link dell'attività è: `https://ilmuseperlascuola.muse.it/attivita-didattica/reinforcemente-learnig/`

Questo file si divide in più sezioni, ognuna riguardante un tutorial per una specifica modifica possibile. In caso di problemi o necessità di ulteriori informazioni, scrivere una mail a lorenzo.dambro@gmail.com, mettendo come oggetto "Problemi mantenimento app ExploRL". La repository ufficiale dell'applicazione è reperibile dal link: `https://github.com/lorenzo2330/MUSE_ExploRL` I tutorial sono aggiornati a febbraio 2025, sono stati realizzati con Android Studio Koala 2024.1.2, e fanno riferimento alla versione 1.0.1 dell'applicazione.

I tutorial presenti all'interno di questo file sono:

- Generare QR-code
- Importare delle foto
- Modificare parametri energia e timer
- Modificare struttura animale
- Modificare la lista di animali
- Realizzare e caricare una nuova versione

Infine, viene fornita una sezione contenente tutti i QR-code, utile per il debug.

A.1 Generare QR-code

Per generare un QR-code, è necessario andare su un qualunque sito in grado di svolgere tale compito. L'unico vincolo è che, una volta scansionato il QR-code, esso fornisca un semplice testo, equivalente a quanto riportato nella sezione "normalName" della struttura dell'animale (mostrata nella relativa sezione "Modificare struttura animale"), compresi spazi e lettere maiuscole/minuscole.

Tutti i QR-code attualmente presenti all'interno dell'applicazione sono stati realizzati tramite il sito `https://qrplanet.com/designer-qr-code-with-embeded-logo#text`, il quale permette la realizzazione permanente (anche sotto forma di semplice testo) di QR-code in formato .png, .svg o .pdf (ai fini dell'utilizzo all'interno dell'applicazione, il formato del QR-code è equivalente).

A.2 Importare delle foto

Per importare una foto all'interno della struttura dell'applicazione, vi sono vari modi, il più semplice consiste nel salvarsi la foto in locale, copiarla (anche con un banale CTRL + C), selezionare la cartella MUSE_ExploRL > images e incollare all'interno la nuova foto.

Nota: se si sta modificando la foto di un animale, assegnare un nome coerente con quanto riportato nella struttura dell'animale (vedi paragrafi "Modificare struttura animale" e "Modificare lista animali").

A.3 Modificare parametri energia e timer

Per l'energia:

- Aprire il file MUSE_ExploRL > lib > providers > energy_provider.dart
- Modificare, nella prima riga della classe, il parametro maxEnergy

Per il timer:

- Aprire il file MUSE_ExploRL > lib > providers > game_provider.dart
- Modificare le righe relative a minuti e secondi

Figura A.1: Modifica parametri

A.4 Modificare struttura animale

Andare nel file MUSE_ExploRL > lib > models > exhibit.dart
Inserire un nuovo dato all'interno della classe ed eventualmente anche all'interno del costruttore (quello con i vari required).
NOTA: una volta effettuata la modifica, sarà necessario applicarla a mano a tutti gli exhibit già esistenti (vedi paragrafo "Modificare lista degli animali")

exhibit.dart

```dart

class Animale {

  String normalName, scientificName, alimentazione, ambiente, resPhoto;
  int nPiano;
  List<Animale> neighbors = [];
  //TipoAttributo nomeAttributo = valoreAttributo;    //Aggiungere un parametro

  Animale({
    required this.normalName, required this.scientificName,
    required this.alimentazione, required this.ambiente,
    required this.nPiano, required this.resPhoto,
    //required this.nomeAttributo                    //Aggiungere un parametro
  });
}
```

Figura A.2: Modifica struttura

A.5 Modificare la lista di animali

Andare nel file MUSE_ExploRL > lib > models > exhibit_list.dart
Se si vuole modificare un valore di un animale già presente:

- Cercare la struttura relativa all'animale del quale si vuole modificare un valore
- Modificare il valore, facendo attenzione a ciò che potrebbe comportare tale modifica (ad esempio, se si sta modificando l'immagine o il nome, aggiornare anche il nome dell'immagine o il QR-code generato)

Nota: se si modifica il normalName, è necessario modificarlo anche nelle funzioni getExhibitByName e scannedExhibit (sempre all'interno del file MUSE_ExploRL > lib > models > exhibit_list.dart).

Se si vuole aggiungere un animale:

- Utilizzare il template già presente
- Rimuovere i simboli di commento /∗ e ∗/
- Assegnare i parametri

Se si vogliono aggiungere o rimuovere i "vicini" degli animali

- Scorrere in basso fino alla funzione setNeighbors()
- Modificare i vicini dell'animale desiderato

Se si vogliono modificare gli animali di partenza, di fine o di tutorial:

- Scorrere in basso fino alle tre variabili: startingExhibit, winnerExhibit e tutorialExhibit
- Sostituire gli animali che si vogliono modificare

```
/*  Esempio di creazione di un nuovo animale
static Animale nome = Animale(
    normalName: "nome animale", //Dev'essere uguale al QR-code
    scientificName: "nome scientifico",
    alimentazione: "...",
    ambiente: "...",
    nPiano: ...,
    resPhoto: "..."     //Dev'essere uguale all'immagine nella cartella images
    );
 */
```

Figura A.3: Aggiungere un animale

```
static void setNeighbors() {
  //Game
  //Nome.neighbors = [neighbor1, neighbor2, ...];
  springbok.neighbors = [lemureCatta, wallabyDalColloRosso];
```

Figura A.4: Modifica vicini

```
static Animale startingExhibit = opossumDellaVirginia;

static Animale winnerExhibit = leoneMarinoSudamericano;

static Animale tutorialExhibit = lupoGrigio;
```

Figura A.5: Modifica animale vincente, perdente o tutorial

A.6 Realizzare e caricare una nuova versione

Dopo aver apportato modifiche significative, rendendo quindi necessaria la pubblicazione di una nuova versione dell'applicazione, è necessario andare nel file MUSE_ExploRL > pubspec.yaml, cercare la riga "version", sostituire il numero di versione prima del + e incrementare il numero di versione dopo il +. È importante che quest'ultimo numero sia sempre maggiore del precedente.

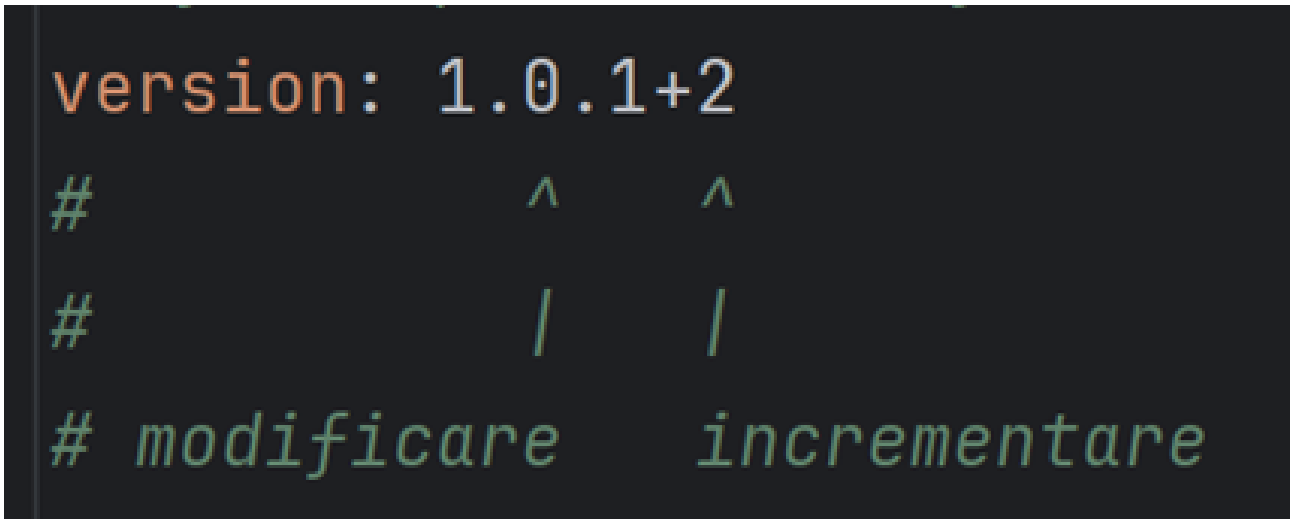

Figura A.6: Modificare la versione

A questo punto è necessario aggiornare le dipendenze digitando nel terminale: flutter pub get.

Il proseguimento si divide lato Android e lato iOS

A.6.1 Android

Sempre da terminale, digitare flutter build apk −−release, dopo qualche secondo, verrà generato un file nella cartella MUSE_ExploRL/build/app/outputs/bundle/release chiamato app-release.aab
A questo punto, è necessario andare sul sito Google Play Console ed accedervi con l'account sviluppatore.

Dalla dashboard basterà selezionare l'applicazione MUSE_ExploRL > Testa e rilascia > Produzione e selezionare il pulsante in alto a destra con scritto "Crea nuova release". Nella schermata che si apre, caricare il file appBundle generato precedentemente e compilare tutti i campi presenti, una volta fatto il tutto, dopo un po' di tempo, necessario per far si che lo staff di Google approvi l'aggiornamento, la nuova versione sarà caricata sul PlayStore.

A.6.2 iOS

Andare sul sito AppStoreConnect ed entrare con le proprie credenziali da sviluppatore. Tra le varie applicazioni, selezionare MUSE ExploRL e, nel menu che si aprirà, cliccare sul + azzurro sotto la scritta MUSE ExploRL. Inserire il numero di versione e procedere.

NOTA: tutte le seguenti operazioni dovranno essere eseguite solo su un dispositivo Mac.

Aprire xCode e aprire il file MUSE_ExploRL > ios > Runner.xcworkspace
Aggiungere il proprio account sviluppatore a xCode (xCode > Settings > Accounts), inserendolo o aggiornandolo (Download manual profiles) qualora non fosse presente. Assicurarsi inoltre di avere i permessi di development per applicazioni iOS (vengono gestiti dall'amministratore dell'account).

Tornando su xCode, selezionare la voce Runner sotto la sezione TARGETS, a questo punto cliccare sulla sezione Signin & Capabilities e selezionare il proprio account sviluppatore alla voce Team. A questo punto caricare una nuova build dell'applicazione cliccando su Product (nella barra superiore del

Mac) > Archive, dopo qualche secondo si genererà una build. Successivamente cliccare su Distribute App > App Store Connect e cliccate Distribute.

Al termine del caricamento, tornare sul sito AppStoreConnect e sotto la sezione TestFlight, dopo qualche minuto, comparirà la nuova build. A questo punto compilare eventuali campi richiesti e salvare la versione, rendendola pronta per il rilasci, dopo una supervisione da parte di Apple.

A.7 Immagini di QR-code da scansionare

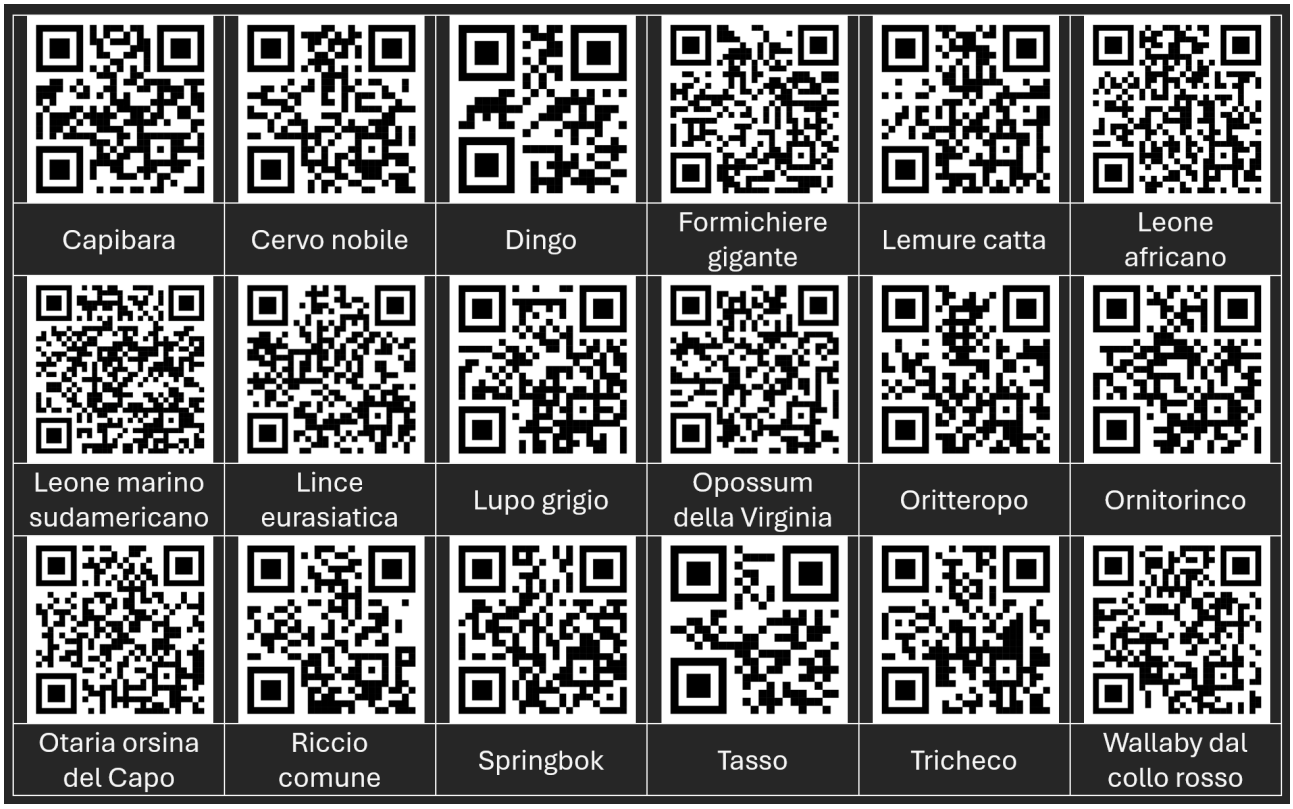

Figura A.7: QR per debug