UNIVERSITÀ DEGLI STUDI DI CATANIA
DIPARTIMENTO DI INGEGNERIA ELETTRICA ELETTRONICA E INFORMATICA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Mario Roberto Nuñez Pereira

POKÉMON MEMORY

RELAZIONE

Anno Accademico 2021 – 2022

Indice

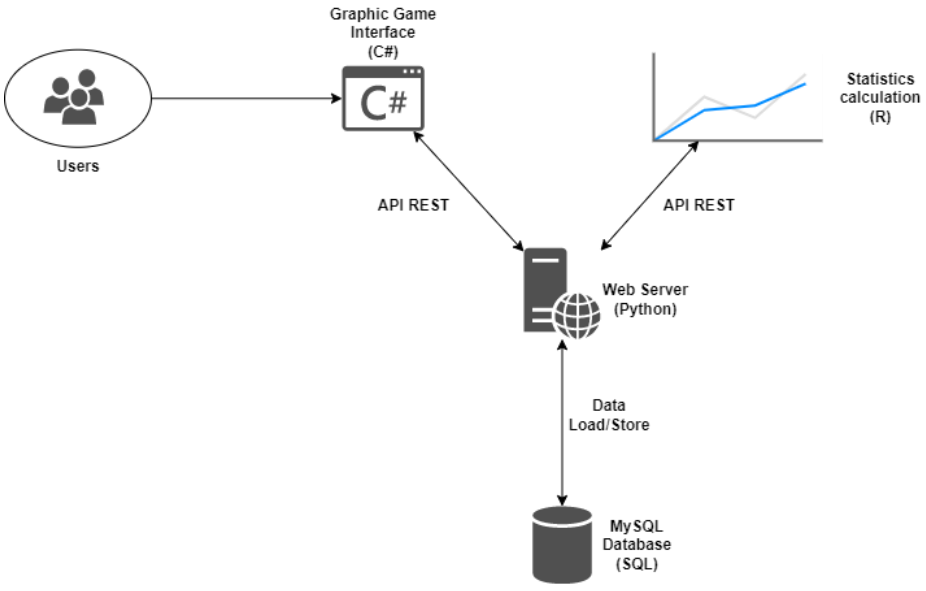
Introduzione

Il progetto **Pokémon Memory** si concentra sullo sviluppo di un'applicazione per il gioco del "memory", il cui obiettivo è cercare ed eliminare nel minor tempo possibile le coppie d'immagini identiche tra quelle nascoste, posizionate in modo casuale all'interno di una griglia di dimensione variabile. Il progetto si suddivide in tre parti, strutturate in una tipica architettura **client-server**, ognuna delle quali offre una diversa funzionalità. La comunicazione tra le varie parti che costituiscono l'applicazione viene gestita mediate l'utilizzo di REST API. All'interno dell'applicazione, ogni utente potrà giocare in tre diverse difficoltà e visualizzare la sua posizione in classifica per ognuna di esse grazie alla persistenza dei dati fornita mediante l'ausilio di un apposito database SQL.

Architettura

Per lo sviluppo di quest'applicazione, è stata scelta un'architettura **client-server** con comunicazione mediante REST API, essendo questa la più indicata e facilmente sviluppabile per l'interazione tra componenti aventi una natura diversa, come in questo caso in cui era richiesto l'uso di diversi linguaggi di programmazione caratterizzati ognuno dalle proprie peculiarità. Le componenti principali dell'applicazione sono:

- Una **Graphic Game Interface** (scritta in **C#**) che permette l'interazione dell'utente;
- Un **Web Server** (scritto in **Python**) che gestisce interamente la logica di load/store dei dati;
- Un processo di **calcolo delle statistiche** (tramite **Script R**) di uno specifico utente ricercato.

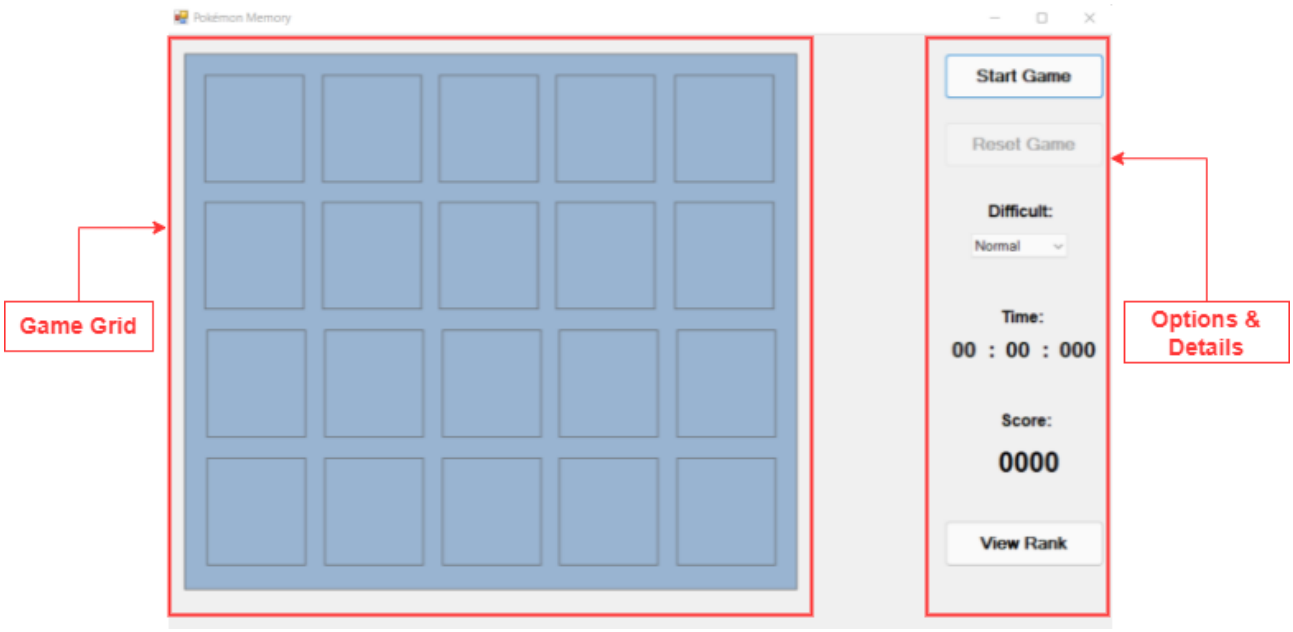


1 – Graphic Game Interface (C#)

1.1 – Panoramica

L'interfaccia grafica, che funge da **Client** principale per l'intera applicazione, permette all'utente di giocare effettivamente al gioco del memory, offrendo una schermata semplice ed intuibile divisa in due macro-sezioni:

- **Griglia di gioco**: una matrice, la cui dimensione varia in base alla difficoltà selezionata, contenente gli slot che una volta avviata la partita saranno inizializzati con le immagini (nascoste) di cui l'utente dovrà trovare i duplicati.
- **Opzioni e dettagli**: tutte le informazioni d'interesse sulla partita e le opzioni selezionabili dall'utente prima, durante e dopo la partita, tra cui:
  - *Start Game*: permette di avviare la partita.
  - *Reset Game*: permette di resettare la griglia di gioco durante la partita nel caso, ad esempio, l'utente voglio ricominciare da capo.
  - *Difficult*: permette di selezionare la difficoltà desiderata dall'utente.
  - *Time*: visualizza il tempo che scorre durante la partita e si ferma alla sua conclusione.
  - *Score*: visualizza il punteggio attuale dell'utente durante la partita
  - *View Rank*: permette di visualizzare le classifiche suddivise tra le tre fasce di difficoltà.



Al termine di ogni partita comparirà una schermata che richiederà all'utente l'inserimento di un nominativo per il salvataggio dei suoi risultati e la messa in classifica (operazioni che verranno di fatto eseguite dal **Web Server Python**)

1.2 – Implementazione

L'implementazione (in C#) della GUI che permette all'utente di giocare è stata possibile grazie alle classi *Forms* del package **System.Windows.Forms**, messe nativamente a disposizione da Visual Studio. Questo genere di classi contiene una serie di metodi e classi per l'interazione tra interfaccia ed utente mediante il richiamo di metodi *Event* al verificarsi di una specifica situazione generata dall'utente stesso o meno (e.g. click su di un elemento/pulsante o chiusura di una finestra). Di seguito verranno brevemente descritte le classi realizzate durante lo sviluppo e le loro caratteristiche.

1.2.1 – GUIGame

Mediante la classe **GUIGame** è stato realizzato il pannello principale dell'interfaccia di gioco, in cui vi sono la griglia e le opzioni/dettagli di gioco. Tale classe contiene tutta la logica ed i metodi per permettere all'utente di giocare e salvare i propri dati, facendo uso in particolare dei packages **System.Net** e **Newtonsoft.Json** per invio di richieste **REST** in formato **JSON** al **Web Server**. I metodi principali che consentono un'esecuzione corretta di questa parte dell'applicazione:

- ***StartGame***: Event Method richiamato al click sull'omonimo pulsante; permette l'avvio della partita richiamando ulteriori metodi che svolgono diversi compiti, come resettare la griglia di gioco ed avviare il timer.
- ***ClickImage***: Event Method richiamato al click su di un immagine nascosta; applica i dovuti controlli sull'immagine appena cliccata confrontandola con quella cliccata in precedenza (se identiche verranno eliminate) e gestisce la politica d'incremento dello score, nel caso non vi siano più immagini da trovare richiamerà il metodo ***EndGame***.
- ***EndGame***: metodo richiamato al reset o termine della partita; in caso di terminazione della partita, mostra una finestra di dialogo in cui si chiede all'utente l'inserimento di un nominativo per il salvataggio del punteggio ottenuto. Tali dati verrano inviati, in formato JSON, mediante POST request al Web Server.
- ***ViewRank***: Event Method richiamato al click sull'omonimo pulsante; recupera i dati JSON delle classifiche degli utenti mediante una GET request al Web Server e li deserializza su di un oggetto Dictionary per poi creare un'istanza della classe Form **GUIRank** che ne permette la visualizzazione.

1.2.2 – GUIRank

Classe Form per la visualizzazione delle classifiche degli utenti nelle varie difficoltà caratterizzata da un unico metodo significativo, ***SetRankTable***, che permette il popolamento delle righe di una data tabella.

1.2.3 – GameCard

Per ottenere una corretta esecuzione della partita è stata creata una classe **GameCard**, rappresentante la singola immagine cliccabile all'interno del gioco, caratterizzata da due proprietà:

- *PicBox* (PictureBox): elemento che verrà utilizzato per mappare le PictureBox della griglia alle immagini che vengono cliccate durante la partita.
- *IsFound* (bool): booleano utilizzato per stabilire se l'immagine è già stata vista o meno ed applicare così le regole d'incremento o decremento dello score durante la partita.

2 – Web Server (Python)

2.1 – Panoramica

Il **Web Server** ha lo scopo per occuparsi di effettuare il salvataggio/recupero dei dati dell'applicazione su/da un database **MySQL** mediante la gestione delle richieste **REST**. Tali richieste vengono effettuate dalle altre parti dell'applicazione (ovvero, interfaccia e script R) grazie all'esposizione da parte del server di endpoint specifici per le diverse funzionalità.

2.2 – Implementazione

La realizzazione pratica del **Web Server** è stata possibile tramite l'uso del framework per applicazioni web **Flask** ed il Python SQL toolkit noto come **SQLAlchemy**.

2.2.1 – Flask

L'utilizzo del framework **Flask** ha permesso la definizione degli endpoint per le richieste REST tramite l'applicazione dell'annotazione *route*, in cui viene specificato il path che dev'essere richiamato dal Client ed il tipo di richiesta/e che ci si aspetta (se GET, POST, PUT, DELETE, ecc.). Tale annotazione viene posta al di sopra della definizione di una funzione python, per indicare al compilatore di

eseguire il codice al suo interno quando si riceverà una richiesta sul path sopra specificato. Inoltre, Flask ha permesso anche di manipolare i dati ricevuti/ritornati da/al Client mediante l'oggetto *request*, contenente tutte le info della richiesta ricevuta (come il body), e *json*, usato per serializzare/deserializzare i vari dati in formato JSON.

```python
@api.route('/user_statistics', methods=['POST'])
def get_raw_user_data():
    if request.content_type == 'application/json':
        rcv_data = request.json

        s_user_id = users.select().with_only_columns(users.c.id_user).where(users.c.name == rcv_data['name'])
        user_id = conn.execute(s_user_id).scalar()

        if user_id is not None:
            results = {
                "EASY": get_user_scores_by_difficult(user_id, "EASY"),
                "NORMAL": get_user_scores_by_difficult(user_id, "NORMAL"),
                "HARD": get_user_scores_by_difficult(user_id, "HARD")
            }
            return json.dumps(results, indent=4, sort_keys=True, default=str), 201
        return json.dumps({"success": True}), 200
    else:
        return json.dumps({"success": False}), 500
```

Tramite Flask, una volta avviato il Web Server, se non diversamente specificato, esso verrà eseguito sull'indirizzo locale *http://127.0.0.1:5000/*, che quindi sarà il root di ogni endpoint a cui verranno effettuate le richieste REST.

2.2.2 – SQLAlchemy

SQLALchemy consente l'esecuzione di query su database, creati tramite i più famosi RDBMS (e.g. *PostgreSQL*, *MySQL*, *SQLite*). Esso permette di considerare il database come un motore di algebra relazionale a cui si accede usando un URL di collegamento al database che presenta i dati fondamentali dell'istanza di quest'ultimo. Inoltre, le query vengono eseguite utilizzando una sintassi molto semplicistica che prevede il richiamo dell'azione e le eventuali condizioni come funzioni python su un oggetto Table (presente nel framework di SQLAlchemy).

```python
users = Table('users', metadata, autoload=True, autoload_with=engine)
```

```python
users.select().where(users.c.name == user_data['name'])
```

3 – Analisi statistiche (R)

Mediante l'esecuzione di un singolo script R scritto appositamente per l'applicazione, denominato "UserStatistics.R", è possibile ottenere le statistiche basilari su di uno specifico utente ricercato tramite nominativo. Eseguendo tale script si otterranno il punteggio medio ed il tempo medio dell'utente e verrà stampato un grafico rappresentante l'andazzo dei punteggi dell'utente nelle varie partite da lui giocate nelle diverse difficoltà. Per recuperare i dati con cui effettuare le suddette statistiche, anche tale script esegue una richiesta di tipo POST al Web Server, inviando il nominativo inserito durante l'esecuzione ed ottenendo in risposta i dati relativi ad esso, che vengono riformattati in modo tale da poterli elaborare.

```
Enter the name of the user whose statistics you want: mario

MARIO's Average Scores:
 - Easy:        506
 - Normal:      572.86
 - Hard:        836.67

MARIO's Average Times:
 - Easy:        00:00:12
 - Normal:      00:00:42
 - Hard:        00:01:09

WARNING!: any NA values simply mean that there is no data

(To see the updated plot go to ../PokémonMemory/Rscripts/Plots/MARIO.jpeg)

Do you want to do another search? [Y/N]: n
```