A blue and black logo

Description automatically generated

Dipartimento di Ingegneria Gestionale e della Produzione

Corso di Laurea Triennale in Ingegneria Gestionale

Classe L8 – Ingegneria dell’Informazione

A.A. 2023/2024

**Generatore di build con gli emblemi di Pokémon Unite**

**Relatore** **Candidato**

*Prof. Fulvio Corno* *Andrea Marietti*

*S281374*

Anno Accademico 2023-2024

**INDICE**

[Capitolo 1 – Proposta di Progetto 3](#_Toc160111153)

[1.1 - Studente proponente 3](#_Toc160111154)

[1.2 - Titolo della proposta 3](#_Toc160111155)

[1.3 - Descrizione del problema proposto 3](#_Toc160111156)

[1.4 - Descrizione della rilevanza gestionale del problema 3](#_Toc160111157)

[1.5 - Descrizione dei data-set per la valutazione 4](#_Toc160111158)

[1.6 - Descrizione preliminare degli algoritmi coinvolti 4](#_Toc160111159)

[1.7 - Descrizione preliminare delle funzionalità previste per l’applicazione software 5](#_Toc160111160)

[Capitolo 2 – Descrizione dettagliata del problema affrontato 5](#_Toc160111161)

[2.1 – Contesto aziendale 5](#_Toc160111162)

[2.2 – Analisi del problema affrontato 6](#_Toc160111163)

[Capitolo 3 – Descrizione del dataset 6](#_Toc160111164)

[Capitolo 4 – Descrizione delle strutture dati e algoritmi coinvolti 7](#_Toc160111165)

[Capitolo 5 – Diagramma delle classi delle parti principali 12](#_Toc160111166)

[Capitolo 6 – Videata dell’applicazione e link 13](#_Toc160111167)

[6.1 – Creazione manuale con completamento 13](#_Toc160111168)

[6.2 – Creazione automatica due parametri 13](#_Toc160111169)

[6.3 – Video di presentazione e dimostrazione funzionamento 13](#_Toc160111170)

[Capitolo 7 – Risultati sperimentali ottenuti 14](#_Toc160111171)

[Capitolo 8 – Valutazioni sui risultati ottenuti e considerazioni conclusive 17](#_Toc160111172)

[Licenza 18](#_Toc160111173)

# **Capitolo 1 – Proposta di Progetto**

# **1.1 - Studente proponente**

s281374 Marietti Andrea

# **1.2 - Titolo della proposta**

Generatore di build con gli emblemi di Pokémon Unite

# **1.3 - Descrizione del problema proposto**

Pokémon Unite è un videogioco del genere MOBA in cui due squadre si scontrano per segnare il maggior numero di punti nelle basi avversarie. I giocatori hanno a disposizione dei personaggi standard per tutti, ma di cui possono aumentare (o diminuire) i 7 parametri tramite degli emblemi ottenibili con l'avanzare del livello.  
Ogni emblema ha tre livelli: Bronzo, Argento e Oro. Più alto è il livello (dal metallo di valore inferiore al più prezioso) maggiori saranno le variazioni dei parametri, sia in positivo che in negativo. Tra la community, tuttavia, è caldamente sconsigliato utilizzare i livelli Bronzo e Argento, dati gli scarsi vantaggi che vengono forniti al giocatore.  
La creazione di una combinazione di emblemi è complessa in quanto bisogna trovare la combinazione migliore dati 10 slot da riempire. L'applicazione avrà quindi lo scopo di facilitare la ricerca della combinazione date delle richieste del giocatore tramite un algoritmo ricorsivo.

# **1.4 - Descrizione della rilevanza gestionale del problema**

I parametri che gli emblemi vanno a modificare hanno un grande impatto nel gioco a livello competitivo, dove anche una piccola variazione può segnare l’esito di uno scontro. Un personaggio improntato alla difesa, ad esempio, richiederà sicuramente elevati parametri difensivi, mentre potrà non necessitare di rilevante velocità di movimento.  
Nell’ambito gestionale, la ricerca di una soluzione che massimizza un parametro non è da considerarsi sempre come la soluzione migliore, ma fornisce informazioni utili per avere una stima dei vantaggi e degli svantaggi che conseguono a una serie di scelte combinate tra loro.

# **1.5 - Descrizione dei data-set per la valutazione**

Non esistendo un data-set scaricabile con tutte le informazioni, i dati sono ottenuti effettuando webscraping dal sito: [https://game8.co/games/Pokémon-UNITE/archives/383688](https://game8.co/games/Pokemon-UNITE/archives/383688). Per effettuare questo procedimento è stata usata una estensione per browsers basati su Chromium chiamata “Web Scraper”, che permette di estrarre le informazioni in automatico previa specifica degli elementi HTML in cui presenti e di inserirle in un file csv, facilmente convertibile in SQL.  
Il database è composto da un’unica tabella formata da cinque colonne:  
-Name: è nome del Pokémon che identifica l’emblema;  
-Color: è il colore dell’emblema;  
-Bronze, Silver, Gold: per ogni colonna sono indicati sia il parametro che viene aumentato sia quello viene diminuito.

# **1.6 - Descrizione preliminare degli algoritmi coinvolti**

Per effettuare la ricerca della combinazione richiesta è necessario utilizzare un algoritmo ricorsivo con vari filtri al fine di soddisfare tutte le richieste.  
Verranno proposte all’utente due opzioni per la creazione della build: automatica e manuale.  
Nella prima, l’utente potrà scegliere il parametro da massimizzare e, se desidera, un parametro da non portare a valore negativo. In questo caso l’algoritmo ricorsivo partirà con un insieme parziale vuoto e dovrà, quindi, esplorare tutte le possibili combinazioni. Mentre nel secondo caso sarà possibile scegliere manualmente un numero a piacere di emblemi (fino a 10) e il parametro che si vuole massimizzare. L’insieme parziale partirà con al suo interno gli emblemi scelti e l’algoritmo si occuperà di completare nel modo migliore la combinazione.  
Per permettere entrambe le opzioni senza intaccare l’esperienza dell’utente, si sfrutta il polimorfismo, ovvero verranno create due funzioni omonime che implementano un algoritmo ricorsivo. I due algoritmi presenteranno delle differenze dettate dalla necessità di rendere il processo di ricerca il più efficiente possibile.  
Il problema principale è processare tutte le possibili combinazioni dati 233 emblemi e 10 posizioni. I tempi di esecuzione sarebbero molto grandi, data l’elevata quantità di risorse richieste. Per ovviare a questo problema si potrebbero usare dei parametri di soglia che devono necessariamente essere superati prima del completamente della combinazione, al fine di scartare tutte le combinazioni che non soddisfano gli standard richiesti. Per di creare varietà negli output, non verranno usati tutti gli emblemi, ma un sottoinsieme di 30 elementi estratti casualmente; ciò ridurrà notevolmente anche i tempi di esecuzione.

# **1.7 - Descrizione preliminare delle funzionalità previste per l’applicazione software**

L’applicazione avrà una interfaccia semplice e minimale. Saranno presenti due schede, una per l’opzione manuale e una per quella automatica, così da prevenire ogni possibile errore dell’utente. In entrambe le schede, sotto le opzioni di scelta, sarà presente un’area di testo in cui verrà mostrato l’output composto dall’elenco degli emblemi scelti e dal calcolo dei valori ottenuti.  
Nella schermata per l’opzione automatica saranno presenti due comboBox; una per la scelta del parametro da massimizzare e una, il cui utilizzo è a discrezione dell’utente, per selezionare il parametro da non modificare sotto lo zero. Nell’altra, invece, sarà sempre presente una comboBox per massimizzare una statistica, ma anche altre 10 per permettere all’utente di scegliere manualmente gli emblemi.  
Viene ipotizzata la possibilità di utilizzare questa funzione anche solo per calcolare rapidamente il risultato ottenibile e quindi vengono lasciate 10 opzioni per l’inserimento dei dati, rendendo di fatto inutile la comboBox per il parametro da massimizzare.

# **Capitolo 2 – Descrizione dettagliata del problema affrontato**

# **2.1 – Contesto aziendale**

Il problema di trovare la migliore combinazione per massimizzare un parametro nel minore tempo possibile è un problema diffuso. All’aumentare della complessità, si predilige una soluzione buona alla soluzione migliore, poiché quest’ultima potrebbe non esistere o essere troppo onerosa in termini di risorse per essere ottenuta. Ad esempio, vengono utilizzate delle euristiche per gestire le ordinazioni di prodotti per rifornire il magazzino (per esempio l’euristica *Lot4Lot* o *Silver-Meal*); non sempre queste forniscono la soluzione migliore, ma forniscono una soluzione in modo efficiente.

# **2.2 – Analisi del problema affrontato**

Nella ricerca della build che massimizza un parametro, quindi, non necessariamente si otterrà sempre la migliore combinazione, ma quella che, a parità di risorse, fornisce la soluzione migliore.

Se si prendessero tutti gli emblemi e si provassero tutte le combinazioni possibili al fine di trovare la migliore che massimizza il parametro “Attack”, bisognerebbe utilizzare 233 emblemi e combinarli avendo a disposizione 10 slot (senza combinazioni identiche con solo ordine differente) e il numero di combinazioni possibili sarebbe troppo elevato per rendere i calcoli attuabili in tempi ragionevoli. Se a ciò si aggiunge la volontà di non ridurre un parametro secondario, ritenuto rilevante, i tempi si allungherebbero ulteriormente.

Gli input sono variabili e dipendono dalle richieste dell’utente; può venire passato un solo parametro o due, possono venire anche indicati degli emblemi manualmente. L’output sarà, in caso di successo, una lista di dieci emblemi coi cui creare la propria build in gioco. La criticità più rilevante è la quantità di possibili combinazioni da valutare; tuttavia, la creazione del software permetterà agli utenti di risparmiare una notevole quantità di tempo in fase di creazione in gioco.

# **Capitolo 3 – Descrizione del dataset**

Il dataset *emblems* è formato da un’unica tabella omonima, a sua volta divisa in cinque colonne.

|  |  |  |
| --- | --- | --- |
| **campo** | **descrizione** | **variabile** |
| **Name** | Nome del Pokémon rappresentato sull’emblema | VARCHAR |
| **Color** | Colore dell’emblema | VARCHAR |
| **Bronze** | Valori dei parametri per l’emblema livello *bronzo* | VARCHAR |
| **Silver** | Valori dei parametri per l’emblema livello *argento* | VARCHAR |
| **Gold** | Valori dei parametri per l’emblema livello *oro* | VARCHAR |

# **Capitolo 4 – Descrizione delle strutture dati e algoritmi coinvolti**

Il software è stato scritto in linguaggio Java, rispettando i patterns di programmazione MVC e DAO.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

* Il package it.polito.tdp.loadoutBuilder contiene le classi: Main, classe principale del progetto, EntryPoint, che lancia l’interfaccia grafica, e FXMLController, che collega l’interfaccia grafica alla classe Model;
* Il package it.polito.tdp.loadoutBuilder.dao contiene le classi che consentono al software di accedere ai dati del database ed il file .sql relativo ad esso;
* In it.polito.tdp.loadoutBuilder.model è contenuta la classe (logica) che elabora tutti i dati, Model, e la classe (oggetto) Emblem.

Il costruttore della classe Model, ***Model()***, inizializza le variabili dichiarate in precedenza e chiama le funzioni ***creaMappe()*** e ***aggiungiSoglie(),*** che popolano le mappe con dei valori predefiniti. Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

La funzione ***creaBuild()*** è utilizzata due volte, ma con parametri input diversi, sfruttando la proprietà del polimorfismo. È la funzione che si occupa di preparare i dati che saranno poi necessari durante la ricorsione, quali id dei parametri, l’inizializzazione della lista *parziale*, il richiamo della funzione ***randomPull()***, che serve a creare una lista di 30 emblemi casuali su cui lavorare e, nel caso di due parametri in input, controllare se il valore secondario è nullo e, quindi, richiamare la funzione ricorsiva corretta.

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Le funzioni ricorsive chiamate dentro *creaBuild()* si chiamano entrambe ***cerca()***, e proprio come in precedenza viene sfruttata la proprietà del polimorfismo con parametri di input diversi. Sebbene la logica ricorsiva sia la stessa, le due funzioni presentano delle differenze; queste sono una conseguenza della ottimizzazione che è stata fatta per rendere il processo il più rapido possibile.

La funzione ***cerca()*** che si occupa di lavorare con un solo parametro e, se inseriti, degli emblemi di partenza presenti nel *parziale*, presenta come condizione di terminazione della ricerca il raggiungimento di 10 elementi nella lista. Il *parziale* ottenuto viene controllato tramite dei filtri *if()* che si basano sul valore ottenuto del parametro che si vuole massimizzare, mediante la funzione ***calcolaAttuale()***, confrontandolo con la variabile *bestScore*, a cui è assegnato il valore migliore ottenuto in precedenza (0 in partenza). Se il risultato ottenuto è migliore del precedente viene aggiornata la lista *buildFinale* con i valori contenuti nel *parziale*.

La ricerca avviene iterando con un ciclo *for()* gli elementi della lista *emblemi*, ovvero i 30 elementi estratti causalmente i quali, se non contenuti nel *parziale*, vengono aggiunti e rimossi durante il processo ricorsivo. Per ridurre i tempi, è stato inserito un filtro *if()*, con condizione *parziale.size()==8*, che controlla, tramite la funzione *calcolaAttuale()*, il valore del parametro principale e scarta l’ultimo emblema inserito (di fatto chiudendo il ramo che si stava esplorando) se il valore ottenuto è inferiore al valore minimo richiesto.

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

L’altra funzione ***cerca()*** ha come condizione di terminazione una dimensione del *parziale* pari a 5, questa restrizione serve a garantire bassi tempi di esecuzione; verrà richiamata due volte dalla funzione *creaBuild()*, utilizzando due liste con parametri estratti diversi (maggiore varietà di output) e poi uniti insieme. Per consentire questa duplice invocazione, negli *if()* di controllo è presente, nel caso in cui la lista ottenuta contenga un valore del parametro principale migliore (e per quello secondario nullo o maggiore di 0), una varabile booleana *var\_stop* che permette, alla funzione *creaBuild()* di decidere quali delle due liste parziali, che comporranno *buildFinale*, deve essere utilizzata e riempita.

Gli altri filtri presenti in questa parte di codice sono analoghi a quelli dell’altra funzione omonima, ovvero controllano che i valori siano accettabili, aggiornano i valori migliori. A differenza della precedente, anche il parametro secondario è tenuto in considerazione nei controlli (nell’altra funzione non è presente).

La parte di ricerca è formata da un ciclo *for()* e da un *if()* che controlla se l’emblema estratto durante l’iterazione è già contenuto nel parziale.

Immagine che contiene testo, schermata, software, Sistema operativo

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

l parametro id è stato ricavato dalle mappe apposite nella funzione *creaBuild()* e corrisponde all’id del parametro di cui si vuole calcolare il valore complessivo dati gli emblemi del *parziale*.

# **Capitolo 5 – Diagramma delle classi delle parti principali**

**Immagine che contiene testo, schermata, Parallelo, documento

Descrizione generata automaticamente**

# **Capitolo 6 – Videata dell’applicazione e link**

# **6.1 – Creazione manuale con completamento**

**Immagine che contiene testo, schermata, software, schermo

Descrizione generata automaticamente**

# **6.2 – Creazione automatica due parametri**

**Immagine che contiene testo, schermata, software, Pagina Web

Descrizione generata automaticamente**

# **6.3 – Video di presentazione e dimostrazione funzionamento**

<https://youtu.be/hGSeeiFzU6E>

# **Capitolo 7 – Risultati sperimentali ottenuti**

I test per i 3 possibili utilizzi sono stati effettuati, ognuno, su 30 esecuzioni usando come parametro principale *Attack*.

|  |  |  |
| --- | --- | --- |
| Opzione | Tempi medi | Note |
| Manuale  (5 emblemi in input manuale *coerenti*) | ≈1s | Nel 6% dei casi non è stato in grado di trovare una build |
| Automatico - 1 parametro | ≈1s | Nel 3% dei casi non è stato in grado di trovare una build |
| Automatico - 2 parametri (HP secondario) | ≈3s | Nel 17% dei casi non è stato in grado di trovare una build |

**Immagine che contiene testo, schermata, software, Pagina Web

Descrizione generata automaticamente**

**Parametri usati per il primo test**

**Immagine che contiene testo, schermata, software, Pagina Web

Descrizione generata automaticamente**

**Parametri usati per il secondo test**

**Immagine che contiene testo, schermata, software, Pagina Web

Descrizione generata automaticamente**

**Parametri usati per il terzo test**

# **Capitolo 8 – Valutazioni sui risultati ottenuti e considerazioni conclusive**

Le build create dal programma hanno lo scopo di velocizzare ed aiutare l’utente in questo processo. Da questo punto di vista, le risposte fornite sono soddisfacenti. Nonostante l’estrazione casuale degli emblemi, le build che vengono create si adattano in modo coerente ai personaggi del gioco.

Il programma rimane limitato dai 30 emblemi estratti; se si potessero esplorare tutte le combinazioni, utilizzando quindi tutti gli emblemi, sicuramente si potrebbero ottenere risultati migliori, ma ad un costo di risorse computazionali molto più alto. Riguardo all’utilizzo delle risorse necessarie, tuttavia, i risultati sono stati molto soddisfacenti: nelle prime versioni, i tempi di esecuzione superavano i due minuti per la modalità manuale e non era possibile ottenere un output in tempi ragionevoli per quella automatica.

L’aggiornamento del dataset non influirebbe in alcun modo sul programma; inoltre, esso risulta facilmente aggiornabile, in caso di implementazione di nuove meccaniche di gioco.

In conclusione, l’applicazione rispetta le premesse con cui era stata ideata, discostandosi poco dal progetto originale.

# **Licenza**

Questa relazione tecnica è rilasciata con licenza Creative Commons BY-NC-SA.

Tu sei libero di:

* Condividere - riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare questo materiale con qualsiasi mezzo e formato
* Modificare - remixare, trasformare il materiale e basarti su di esso per le tue opere

Alle seguenti condizioni:

* Attribuzione - Devi riconoscere una menzione di paternità adeguata, fornire un link alla licenza e indicare se sono state effettuate delle modifiche. Puoi fare ciò in qualsiasi maniera ragionevole possibile, ma non con modalità tali da suggerire che il licenziante avalli te o il tuo utilizzo del materiale.
* Non Commerciale - Non puoi utilizzare il materiale per scopi commerciali.
* Stessa Licenza - Se remixi, trasformi il materiale o ti basi su di esso, devi distribuire i tuoi contributi con la stessa licenza del materiale originario.

Generatore di build con gli emblemi di Pokémon Unite © 2023 by Andrea Marietti is licensed under CC BY-NC-SA 4.0. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/4.0/