squadBuilder

September 30, 2025

1 SquadBuilder

Questo notebook ti guiderà nell'individuazione dei tipi di pokemon necessari per batterli tutti!

In questo Notebook andremo a creare una squadra che riesce a battere tutti i tipi pokemon. Per farlo è necessario definire cosa sono i Pokemon e di conseguenza cosa è un Tipo associato ai pokemon

[85]:
```
pokemon_types = {
    "Normale": {
        "debolezze": ["Lotta"],
        "resistenze": [],
        "immunità": ["Spettro"],
        "superefficacie": []
    },
    "Fuoco": {
        "debolezze": ["Acqua", "Terra", "Roccia"],
        "resistenze": ["Fuoco", "Erba", "Ghiaccio", "Coleottero", "Acciaio",␣
 ↪"Folletto"],
        "immunità": [],
        "superefficacie": ["Erba", "Coleottero", "Ghiaccio", "Acciaio"]
    },
    "Acqua": {
        "debolezze": ["Elettro", "Erba"],
        "resistenze": ["Fuoco", "Acqua", "Acciaio", "Ghiaccio"],
        "immunità": [],
        "superefficacie": ["Fuoco", "Terra", "Roccia"]
    },
    "Erba": {
        "debolezze": ["Fuoco", "Ghiaccio", "Veleno", "Volante", "Coleottero"],
        "resistenze": ["Acqua", "Erba", "Terra", "Elettro"],
        "immunità": [],
        "superefficacie": ["Acqua", "Terra", "Roccia"]
    },
    "Elettro": {
        "debolezze": ["Terra"],
        "resistenze": ["Acciaio", "Elettro", "Volante"],
        "immunità": [],
        "superefficacie": ["Acqua", "Volante"]
```

```
    },
    "Ghiaccio": {
        "debolezze": ["Fuoco", "Lotta", "Roccia", "Acciaio"],
        "resistenze": ["Ghiaccio"],
        "immunità": [],
        "superefficacie": ["Erba", "Terra", "Volante", "Drago"]
    },
    "Lotta": {
        "debolezze": ["Volante", "Psico", "Folletto"],
        "resistenze": ["Roccia", "Coleottero", "Buio"],
        "immunità": [],
        "superefficacie": ["Normale", "Roccia", "Acciaio", "Ghiaccio", "Buio"]
    },
    "Veleno": {
        "debolezze": ["Psico", "Terra"],
        "resistenze": ["Erba", "Lotta", "Veleno", "Coleottero", "Folletto"],
        "immunità": [],
        "superefficacie": ["Erba", "Folletto"]
    },
    "Terra": {
        "debolezze": ["Acqua", "Erba", "Ghiaccio"],
        "resistenze": ["Veleno", "Roccia"],
        "immunità": ["Elettro"],
        "superefficacie": ["Fuoco", "Elettro", "Veleno", "Roccia", "Acciaio"]
    },
    "Volante": {
        "debolezze": ["Elettro", "Ghiaccio", "Roccia"],
        "resistenze": ["Erba", "Lotta", "Coleottero"],
        "immunità": ["Terra"],
        "superefficacie": ["Erba", "Lotta", "Coleottero"]
    },
    "Psico": {
        "debolezze": ["Buio", "Spettro", "Coleottero"],
        "resistenze": ["Lotta", "Psico"],
        "immunità": [],
        "superefficacie": ["Lotta", "Veleno"]
    },
    "Coleottero": {
        "debolezze": ["Fuoco", "Volante", "Roccia"],
        "resistenze": ["Erba", "Lotta", "Terra"],
        "immunità": [],
        "superefficacie": ["Erba", "Psico", "Buio"]
    },
    "Roccia": {
        "debolezze": ["Acqua", "Erba", "Lotta", "Acciaio", "Terra"],
        "resistenze": ["Normale", "Fuoco", "Veleno", "Volante"],
        "immunità": [],
```

```
        "superefficacie": ["Fuoco", "Ghiaccio", "Volante", "Coleottero"]
    },
    "Spettro": {
        "debolezze": ["Spettro", "Buio"],
        "resistenze": ["Veleno", "Coleottero"],
        "immunità": ["Normale", "Lotta"],
        "superefficacie": ["Psico", "Spettro"]
    },
    "Drago": {
        "debolezze": ["Ghiaccio", "Drago", "Folletto"],
        "resistenze": ["Fuoco", "Acqua", "Erba", "Elettro"],
        "immunità": [],
        "superefficacie": ["Drago"]
    },
    "Buio": {
        "debolezze": ["Lotta", "Coleottero", "Folletto"],
        "resistenze": ["Spettro", "Buio"],
        "immunità": ["Psico"],
        "superefficacie": ["Psico", "Spettro"]
    },
    "Acciaio": {
        "debolezze": ["Fuoco", "Lotta", "Terra"],
        "resistenze": ["Normale", "Volante", "Roccia", "Coleottero", "Acciaio",␣
 ↪"Erba", "Psico", "Ghiaccio", "Drago", "Folletto"],
        "immunità": ["Veleno"],
        "superefficacie": ["Ghiaccio", "Roccia", "Folletto"]
    },
    "Folletto": {
        "debolezze": ["Acciaio", "Veleno"],
        "resistenze": ["Lotta", "Coleottero", "Buio"],
        "immunità": ["Drago"],
        "superefficacie": ["Lotta", "Buio", "Drago"]
    }
}
```

definiamo delle funzioni per controllare la bontà della nostra squadra in funzione dei tipi che vengono battuti, ed una funzione per controlare cosa rimane da battere

[86]:
```
def covered_types(squad):

    if(len(squad)== 0):return set()

    coverage = set(analyzed_type
                   for analyzed_type in pokemon_types
                   for unchecked_type in␣
 ↪pokemon_types[analyzed_type]["debolezze"]
                   for acquired in squad  if unchecked_type == acquired)
```

```
    return coverage

def uncovered_types(squad):
    coverage = covered_types(squad)

    uncovered_types = set(key for key in pokemon_types.keys())

    if len(squad) == 0 or len(coverage)== 0 :
        return uncovered_types
    else:
        return uncovered_types.difference(coverage)

print(covered_types([]))
print(uncovered_types([]))
```

```
set()
{'Acciaio', 'Normale', 'Acqua', 'Spettro', 'Roccia', 'Terra', 'Lotta',
'Elettro', 'Ghiaccio', 'Veleno', 'Drago', 'Folletto', 'Psico', 'Coleottero',
'Erba', 'Fuoco', 'Volante', 'Buio'}
```

Aggiungiamo un "peso" ai tipi, ossia forniamo un parametro per capire quali tipi richiedono più attenzione. Interpreteremo il peso come numero di superefficacie che un tipo ha. Inoltre definiamo delle operazioni da fare sui pesi, come aggiornarli e trovare il "prossimo" ossia il peso maggiore tra quelli presenti

[ ]:
```
def create_weights():return {pkmon_type:
 ↪len(pokemon_types[pkmon_type]["superefficacie"]) for pkmon_type in␣
 ↪pokemon_types}

def calculate_weight(super_effectivness,useless_types):
    weight = len(super_effectivness) - sum(1 for pkmon_type in␣
 ↪super_effectivness for useless_type in useless_types if pkmon_type ==␣
 ↪useless_type)
    if weight == 0:
        return -1
    else:
        return weight

def update_wheights(types_checked):
    return {pkmon_type:
 ↪calculate_weight(pokemon_types[pkmon_type]["superefficacie"],types_checked)␣
 ↪for pkmon_type in pokemon_types}

def next_weight(weighted_types):
    max = 0
    for type in weighted_types:
        if weighted_types[type] > max :
            max = weighted_types[type]
```

```
    return max
```

infine dobbiamo controllare i tipi che hanno solo 1 debolezza, in quanto questi ci forzano a prendere la loro debolezza nella coverage

[88]:
```
def initialize_squad(squad=[]):
    #scrito qui solo per migliorare la leggibilità del codice
    mandatory_types = {pkmon_types[pkmon_type]["debolezze"][0] for pkmon_type␣
 ↪in pokemon_types if len(pokemon_types[pkmon_type]["debolezze"]) == 1}
    return list(dict.fromkeys([*squad,*mandatory_types]))
print (initialize_squad())
```

```
['Lotta', 'Terra']
```

adesso dobbiamo controllare il prossimo tipo da prendere per continuare la copertura, per farlo sfruttiamo i pesi

[89]:
```
def next_types_to_add(coverage):

    if(len(coverage)==0):return set(initialize_squad());

    updated = update_wheights(coverage)

    return set(
            pkmon_type
            for pkmon_type in pokemon_types
            for covered_type in coverage
            if pkmon_type != covered_type and updated[pkmon_type] ==␣
 ↪next_weight(updated)
    )
```

L'obiettivo è quindi minimizzare il numero di pokemon richiesti per battere tutti i tipi, avendo 6 come numero massimo ed 1 come minimo.

la squadra è quindi una lista di Pokemon, che al momento non definiamo in quanto sono una quantità mastodontica, pertanto considereremo solo il massimo numero di tipi che 6 pokemon possono avere ossia 12.

Infine per avere il risultato minore possibile partiremo da una squadra vuota, ma è sempre possibile aggiungere i tipi per vedere come completare la squadra. Per raggiungere il nostro obiettivo dobbiamo rendere iterativa questa procedura, per farlo necessitiamo di una funzione che unisca le varie funzioni definite prima

[90]:
```
def find_best_types(squad,want_print = True ):
    #cerco i tipi richiesti
    requested_types = uncovered_types(squad);
    #se non ne ho allora non devo fare niente
    if len(requested_types)== 0 :return print('La tua squadra batte già tutti i␣
 ↪tipi, complimenti!')
    #creo la coverage dei tipi in base alla squadra
```

```
    coverage = covered_types(squad)
    #cerco i prossimi tipi da far aggiungere all'utente
    suggested_types = next_types_to_add(coverage)
    if want_print: print(f'La squadra ha bisogno di questi tipi:␣
 ↪{suggested_types}')
    return suggested_types
```

Infine dobbiamo aggiungere l'interazione con l'utente per permettere di inserire il tipo, con dei controlli per avere la prima lettera in uppercase e tutto il resto della stringa in lower

[91]:
```
def insertType(next_type):
    next_type = next_type.lower()
    next_type = next_type[0].upper()+next_type[1:]
    assert next_type in pokemon_types, f"{next_type} non è un tipo Pokémon␣
 ↪valido"
    return next_type
```

Adesso bisogna solo iterare il procedimento finchè la copertura non è completa

[92]:
```
def interactive_calculate_coverage(squad):
    first = True
    while len(covered_types(squad)) <18:
        print(f'Squadra attuale: {squad}')
        if first:
            squad = initialize_squad(squad)
            first = False
            print(f'Alla tua squadra sono stati aggiunti i tipi:␣
 ↪{initialize_squad()}, in quanto sono necessari per battere tipi con una sola␣
 ↪debolezza')
            continue
        find_best_types(squad)
        next_type = insertType(input('Inserisci il prossimo tipo'))
        print(next_type)
        squad.append(next_type)
    print(f'La tua squadra {squad} batte tutti i tipi')
    return squad

# interactive_calculate_coverage([]);
```

Mentre se preferiamo avere un'algoritmo autonomo possiamo modificare leggermente il codice, consentendo sempre all'utente di scegliere la composizione iniziale

[93]:
```
def calculate_coverage(squad,want_print = True):
    first = True
    while len(covered_types(squad)) <18:
        if want_print: print(f'Squadra attuale: {squad}')
        if first:
            squad = initialize_squad(squad)
```

```
            first = False
            if want_print:print(f'Alla tua squadra sono stati aggiunti i tipi:␣
 ↪{initialize_squad()}, in quanto sono necessari per battere tipi con una sola␣
 ↪debolezza')
            continue
        suggested_types = find_best_types(squad,want_print)
        if want_print:print(suggested_types)
        if suggested_types: squad.append(suggested_types.pop())
    print(f'La tua squadra {squad} batte tutti i tipi')
    return squad

calculate_coverage([],False);
```

```
La tua squadra ['Lotta', 'Terra', 'Ghiaccio', 'Spettro', 'Volante', 'Elettro',
'Veleno'] batte tutti i tipi
```

Adesso che la copertura è completa possiamo passare al calcolo delle combinazioni di tipi pokemon disponibili, per farlo definiamo una funzione che data una squadra ci restituisce un set di combinazioni

[94]:
```
import itertools

def calculate_combinations(squad):
    return list(itertools.combinations(squad, 2))

def pretty_print_combs(combs_list):
    i = 1
    for comb in combs_list:
        print(comb,end="\t")
        if i%7== 0 :
            print('\n')
        i+=1
    return

squad = ["Terra","Lotta","Elettro","Acqua","Volante"]
combs = calculate_combinations(calculate_coverage([],False))
print(f'Hai a disposizione {len(combs)} combinazioni di tipi, che sono elencate␣
 ↪di seguito')
pretty_print_combs(combs)
```

```
La tua squadra ['Lotta', 'Terra', 'Ghiaccio', 'Spettro', 'Volante', 'Elettro',
'Veleno'] batte tutti i tipi
Hai a disposizione 21 combinazioni di tipi, che sono elencate di seguito
('Lotta', 'Terra')      ('Lotta', 'Ghiaccio')   ('Lotta', 'Spettro')
('Lotta', 'Volante')    ('Lotta', 'Elettro')    ('Lotta', 'Veleno')
('Terra', 'Ghiaccio')

('Terra', 'Spettro')    ('Terra', 'Volante')    ('Terra', 'Elettro')
('Terra', 'Veleno')     ('Ghiaccio', 'Spettro') ('Ghiaccio', 'Volante')
```

```
('Ghiaccio', 'Elettro')

('Ghiaccio', 'Veleno')  ('Spettro', 'Volante')  ('Spettro', 'Elettro')
('Spettro', 'Veleno')   ('Volante', 'Elettro')  ('Volante', 'Veleno')
('Elettro', 'Veleno')
```

L'obiettivo è stato raggiunto anche se in maniera parziale, in quanto questi sono solo tipi e non vengono considerate le combinazioni di tipi non attualmente esistenti e soprattutto questi non sono pokemon. Per fare questo dobbiamo appoggiarci al web scraping(devo ancora capire come farlo) recuperando in primo luogo tutte le combinazioni di tipo esistenti, e solo successivamente tutti i pokemon che rientrano nelle combinazioni di tipi esistenti.

[95]:
```
def refine_types(invalid_types):

    return
```