Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Inteligencia Artificial:

Constructor de equipos Pokémon

Práctica 4 - Proyecto individual Prolog.

Febe Fariña Aguirre
(alu0101430973@ull.edu.es)

Índice:

1. Introducción.

En esta práctica se plantea realizar un sistema basado en el conocimiento en Prolog para dar solución a un problema. En este caso, hablaremos sobre una herramienta para construir equipos Pokémon. Este programa utiliza una interfaz gráfica de Python para acceder a las reglas y hechos desarrollados en Prolog a través de la librería PySwip. El programa es capaz de añadir y eliminar Pokémon a tu equipo, indicando los tipos del Pokémon, sus debilidades y resistencias. Además, te indicará las debilidades generales de tu equipo y sugerencias sobre qué tipos deberías añadir.

2. Pokémon: Tipos, debilidad y resistencia.

Pokémon es una de las franquicias más conocidas del mundo, la cual empezó como un videojuego en el que capturas monstruos para formar un equipo y luchar contra otros Pokémon. En Pokémon existen 18 tipos, los cuales interactúan de distinta forma con los demás (*Fig 1*). Estas interacciones son: muy efectivo (x2 de daño), poco efectivo (x0.5 de daño) e inefectivo (x0 de daño). Diremos que un tipo A es débil contra un tipo B cuando el tipo B es muy efectivo contra el tipo A. Por otra parte, diremos que el tipo A es resistente contra el B cuando B es poco efectivo o inefectivo contra A.

Efect.	Tipo del Pokémon del oponente																	
Tipo del ataque propio	ACERO	AGUA	BICHO	DRAGÓN	ELÉCT.	FANT.	FUEGO	HADA	HIELO	LUCHA	NORMAL	PLANTA	PSÍQ.	ROCA	SINIE.	TIERRA	VENENO	VOLAD.
ACERO	½	½	-	-	½	-	½	x2	x2	-	-	-	-	x2	-	-	-	-
AGUA	-	½	-	½	-	-	x2	-	-	-	-	½	-	x2	-	x2	-	-
BICHO	½	-	-	-	-	½	½	½	-	½	-	x2	x2	-	x2	-	½	½
DRAGÓN	½	-	-	x2	-	-	-	x0	-	-	-	-	-	-	-	-	-	-
ELÉCT.	-	x2	-	½	½	-	-	-	-	-	-	½	-	-	-	x0	-	x2
FANT.	-	-	-	-	-	x2	-	-	-	-	x0	-	x2	-	½	-	-	-
FUEGO	x2	½	x2	½	-	-	½	-	x2	-	-	x2	-	½	-	-	-	-
HADA	½	-	-	x2	-	-	½	-	-	x2	-	-	-	-	x2	-	½	-
HIELO	½	½	-	x2	-	-	½	-	½	-	-	x2	-	-	-	x2	-	x2
LUCHA	x2	-	½	-	-	x0	-	½	x2	-	x2	-	½	x2	x2	-	½	½
NORMAL	½	-	-	-	-	x0	-	-	-	-	-	-	-	½	-	-	-	-
PLANTA	½	x2	½	½	-	-	½	-	-	-	-	½	-	x2	-	x2	½	½
PSÍQ.	½	-	-	-	-	-	-	-	-	x2	-	-	½	-	x0	-	x2	-
ROCA	½	-	x2	-	-	-	x2	-	x2	½	-	-	-	-	-	½	-	x2
SINIE.	-	-	-	-	-	x2	-	½	-	½	-	-	x2	-	½	-	-	-
TIERRA	x2	-	½	-	x2	-	x2	-	-	-	-	½	-	x2	-	-	x2	x0
VENENO	x0	-	-	-	-	½	-	x2	-	-	-	x2	-	½	-	½	½	-
VOLAD.	½	-	x2	-	½	-	-	-	-	x2	-	x2	-	½	-	-	-	-

Fig 1: Tabla de tipos y sus efectividades contra los demás tipos

Un equipo Pokémon está formado por 6 de estos, teniendo cada uno sus propias debilidades y resistencias. Consideramos que una debilidad está cubierta cuando, para esa debilidad, existe un Pokémon en el equipo que sea resistente a ese tipo. De esta forma, buscamos que todas las debilidades de cada uno de los Pokémon del equipo estén cubiertas.

3. Desarrollo del programa.

Una vez definido el alcance del programa, necesitamos crear una base de conocimiento que almacene a cierto número de Pokémon, sus tipos y las relaciones entre los mismos. Luego, crearemos las reglas necesarias para buscar las debilidades y resistencias de un Pokémon en particular y, más adelante, de un equipo. Haremos accesibles estas reglas a través de una interfaz gráfica.

3.1. Creación de hechos a través de *PokéAPI*.

PokéAPI es una API por la que puedes solicitar datos de una base de datos con miles de entradas sobre datos de los juegos de Pokémon. A nosotros nos interesa consultar las entradas sobre Pokémon y tipos. En el directorio de trabajo *py-loader* se encuentran los scripts necesarios para descargar los datos de los Pokémon y tipos y añadirlos a la base de conocimiento. Los scripts denominados *X_data_loader.py* son los encargados de llamar a la API y almacenar los datos sobre Pokémon y tipos en archivos json. Por otra parte, los scripts *fact-X-script.py* cogen la información que nos interesa de los json y crea hechos en un archivo Prolog. Los hechos que crea son los siguientes:

- *pokemon(P)*, P es un pokémon
- *type(T)*, T es un tipo
- *have_type(P, T)*, P es del tipo T
- *effective(T1, T2, N)*, el tipo T1 tiene una efectividad contra T2 de N, pudiendo ser el valor de N 0, 0.5, 1 o 2.

3.2. Creación de reglas.

Antes de crear las reglas para buscar debilidades y resistencias de un Pokémon, se han creado reglas para hacer más visual las relaciones entre tipos en lugar de utilizar los números de efectividad. Por ello, se han creado las siguientes reglas:

```
super_effective(T1, T2):-
    effective(T1, T2, 2).

not_very_effective(T1, T2):-
    effective(T1, T2, 0.5).

no_effect(T1, T2):-
    effective(T1, T2, 0).

resistant(T1, T2):-
    not_very_effective(T2, T1).

weak(T1, T2):-
    super_effective(T2, T1).
```

Tal y como describimos anteriormente, T1 será resistente contra T2 si T2 es poco efectivo contra T1. De la misma forma, T1, será débil contra T2 si T2 es muy efectivo contra T1. Una vez creadas estas reglas, definiremos la relación *pokemonResistsAndWeaknesses/3*, la cual recibirá como argumento un Pokémon y devolverá dos listas de tipos: una con las debilidades del Pokémon y otra con sus resistencias. Se ha definido la relación de la siguiente manera:

```
pokemonResistsAndWeaknesses([], [], []).
pokemonResistsAndWeaknesses(T, R, W) :-
    type(T),
    !,
    findall(X, weak(T, X), W),
    findall(X, resistant(T, X), R),
    !.
pokemonResistsAndWeaknesses(P, R, W) :-
  pokemon(P),
  !,
  have_type(P, T),
  pokemonResistsAndWeaknesses(T, R, W).
```

La primera cláusula contempla el caso en el que no se introduzca ningún Pokémon, lo cual será importante cuando definamos la regla que busque las debilidades y resistencias de un equipo. La segunda cláusula contempla como primer argumento un tipo de Pokemon, al cual encontrará todas sus debilidades y resistencias y la añadirá a la lista W y R respectivamente. En la tercera y última

cláusula, el primer argumento es un Pokémon, por lo que tendremos que conseguir los tipos de ese Pokémon y llamar nuevamente a la regla, pero cambiando el primer argumento por el tipo o tipos del Pokémon. Con esto, conseguiremos dos listas con las debilidades y resistencias de un Pokémon dado, pero necesitamos averiguar las debilidades y resistencias de varios Pokémon.
Primero, definiremos lo que es un equipo:

```
team([]).
team([P|TP]):-
    is_list(TP),
    pokemon(P),
    team(TP).
```

Sencillamente, la regla comprobará que cada uno de los elementos de la lista sea un Pokémon. Una vez creado, es hora de definir la relación *teamResistsAndWeaknesses/3*. De la misma forma que *pokemonResistsAndWeaknesses*, toma como argumento un equipo de Pokémon y devuelve dos listas con las debilidades y resistencias de cada uno de los Pokémon. La regla es la siguiente:

```
teamResistsAndWeaknesses([], [], []).
teamResistsAndWeaknesses(T, R, W) :-
  team(T),
  !,
  T = [F|B],
  pokemonResistsAndWeaknesses(F, PR, PW),
  teamResistsAndWeaknesses(B, TR, TW),
  append(PR, TR, R),
  append(PW, TW, W).
```

La primera cláusula contempla un equipo vacío, ya que al estar iterando sobre los Pokémon del equipo nos acabaremos encontrando con el final de la lista, es decir, una lista vacía. La segunda cláusula, en primer lugar, comprueba que se ha introducido un equipo de Pokémon. Luego, llama a *pokemonResistsAndWeaknesses* con el primer elemento de la lista y almacena las debilidades y resistencias particulares del Pokémon en PR y PW. La regla se llamará a sí misma con el resto del equipo y guardará sus resistencias y debilidades en TR y TW. Al final, las resistencias y debilidades del Pokémon en particular serán añadidas a las resistencias y debilidades del equipo y almacenadas en R y W. Con esto habremos conseguido las listas de resistencias y debilidades de un equipo dado.

Por último, necesitaremos la relación *teamTypeSuggestions(T, S)*, la cual analiza las debilidades del equipo y encuentra a aquellos tipos que sean resistentes a estos. Además, definiremos una regla adicional *checkTypeSuggestions(T,S)* que comprueba que no se recomiende un tipo de Pokémon que ya tengamos en el equipo. Dichas relaciones son las siguientes:

```
checkTypeSuggestions([], _).
checkTypeSuggestions(T, S) :-
  team(T),
  !,
  T = [F|B],
  not(have_type(F, S)),
  checkTypeSuggestions(B, S).

teamTypeSuggestions(T, S) :-
  teamResistsAndWeaknesses(T, _, W),
  member(X, W),
  findall(S, resistant(S, X), Suggestions),
  member(S, Suggestions),
  checkTypeSuggestions(T, S).
```

Empezando por *checkTypeSuggestions*, se comprueba que T se trate de un equipo Pokémon y, después, que cada uno de los Pokémon que forma el equipo no sea del tipo de la recomendación dada. Luego, en *teamTypeSuggestions*, se hallan las debilidades del equipo y se almacenan en W. Para cada uno de los tipos en W, se buscan todos los tipos que sean resistentes a dichos tipos. Por último, se comprueba que los tipos de las recomendaciones no están ya incluidos en el equipo actual.
Con estas reglas ya somos capaces de desarrollar el programa, lo que resta crear la interfaz gráfica.

3.3. GUI en Python.

Usaremos la librería tkinter para implementar una interfaz gráfica mediante python. El programa consta de un input en el que se introduce el nombre del Pokémon. Si le damos a *Add Pokémon*, se introducirá en la tabla el Pokémon junto a sus tipos, debilidades y resistencias. Esto se logra gracias a pyswip, el cual hace una consulta de las reglas establecidas anteriormente.

```
def get_pkm_weakness(pokemon):
    weak = []
    resist = []
                                        for          matchup          in
prolog.query("pokemonResistsAndWeaknesses("+pokemon+",R,W)"):
        weak.append(matchup['W'])
        resist.append(matchup['R'])
    weak = [item for sublist in weak for item in sublist]
    resist = [item for sublist in resist for item in sublist]
    weak = list(dict.fromkeys(weak))
    resist = list(dict.fromkeys(resist))
    for types in weak:
        if types in resist:
            weak.remove(types)
            resist.remove(types)
    return weak, resist
```

En este ejemplo, podemos ver cómo se consigue las debilidades y resistencias de un Pokémon en particular. Pyswip realiza la consulta de *pokemonResistsAndWeaknesses* y almacena en las listas weak y resist las correspondientes debilidades y resistencias. Luego se evitan duplicados de tipos para que en casos en los que un tipo tenga efectividad x4 o x1/4. Esto ocurre si un Pokémon tiene 2 tipos y estos son débiles o resistentes al mismo tipo. Luego se anulan las debilidades y resistencias eliminando tipos que están en ambas listas. Esto surge si el Pokémon tiene 2 tipos y existe un tipo tal que el Pokémon es débil y resistente a la vez.
Al añadirse un Pokémon a la tabla, se llama a la función *add_pkm* para añadir la información en la tabla y, además, añadir al Pokémon al equipo. Cada vez que esto ocurra, se ha de actualizar las debilidades generales del equipo y las recomendaciones de tipos por medio de la función *check_weaknesses*.

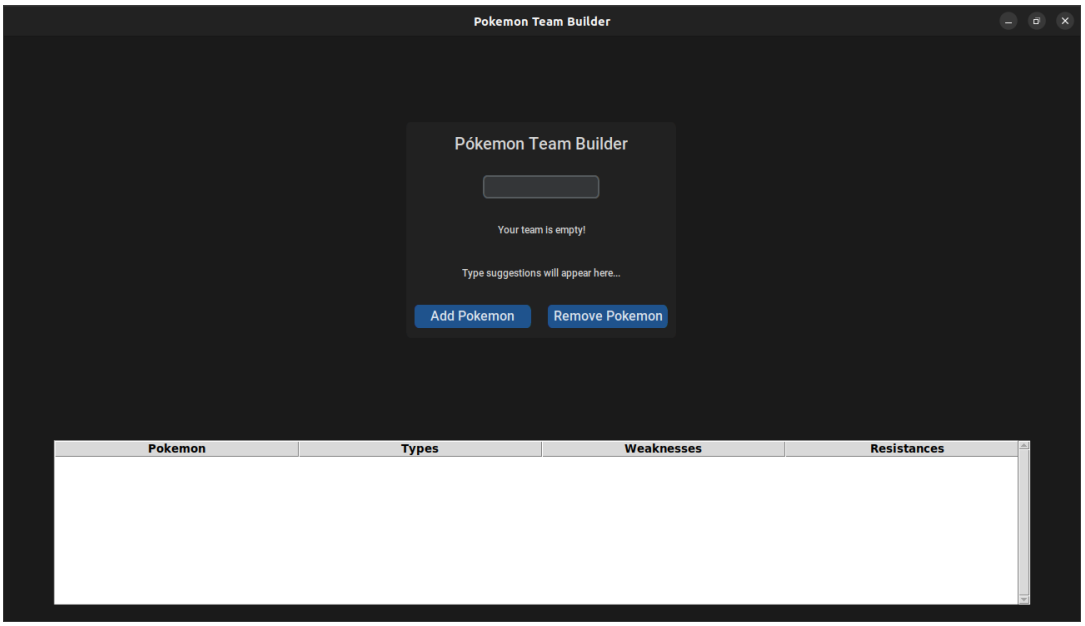
```
def check_weaknesses():
    general_weakness = []
    general_resistance = []
    type_suggestions = []
    [...]
                                        for          match          in
prolog.query("teamResistsAndWeaknesses("+text+",R,W)"):
        general_weakness.append(match['W'])
        general_resistance.append(match['R'])
```

```
    for match in prolog.query("teamTypeSuggestions("+text+",X)"):
        type_suggestions.append(match['X'])
     general_weakness = [item for sublist in general_weakness for item
in sublist]
     general_resistance = [item for sublist in general_resistance for
item in sublist]
    general_weakness_dict = list(dict.fromkeys(general_weakness))
    general_resistance_dict = list(dict.fromkeys(general_resistance))
    type_suggestions_dict = list(dict.fromkeys(type_suggestions))
    for types in general_resistance_dict:
        if types in general_weakness_dict:
            general_weakness_dict.remove(types)
   [...]
```

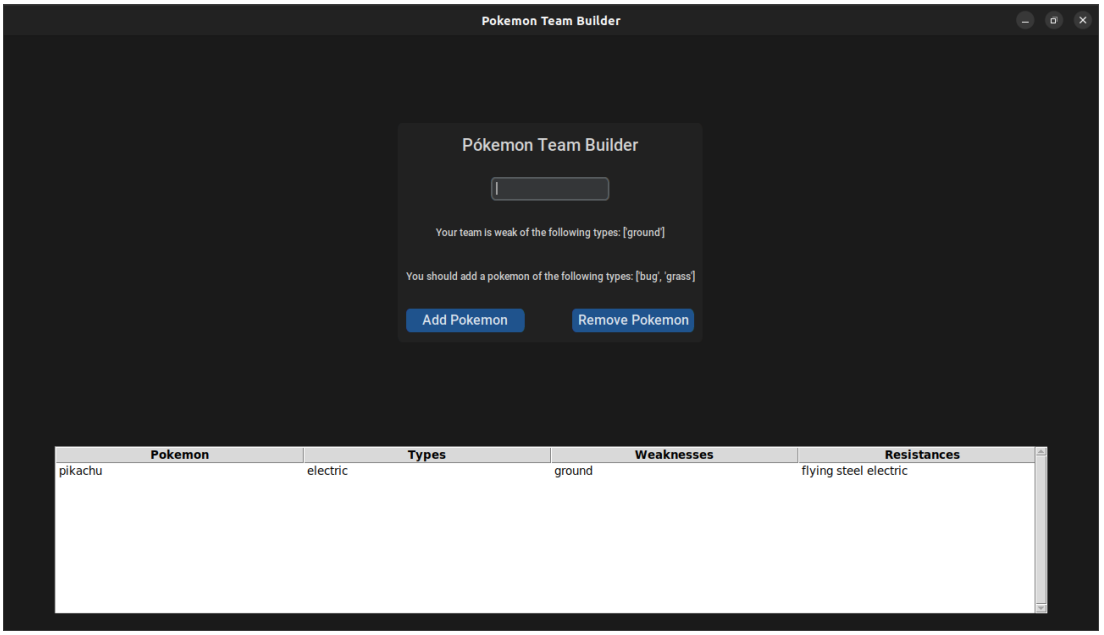
De forma similar a la función *get_pkm_weakness*, se realiza una consulta sobre las debilidades y resistencias del equipo por medio de *teamResistsAndWeaknesses*, además de sugerencias a través de la regla *teamTypeSuggestions*. De la misma manera, se eliminan duplicados y se anulan las debilidades con las resistencias del equipo.

4. Ejemplos de ejecución.

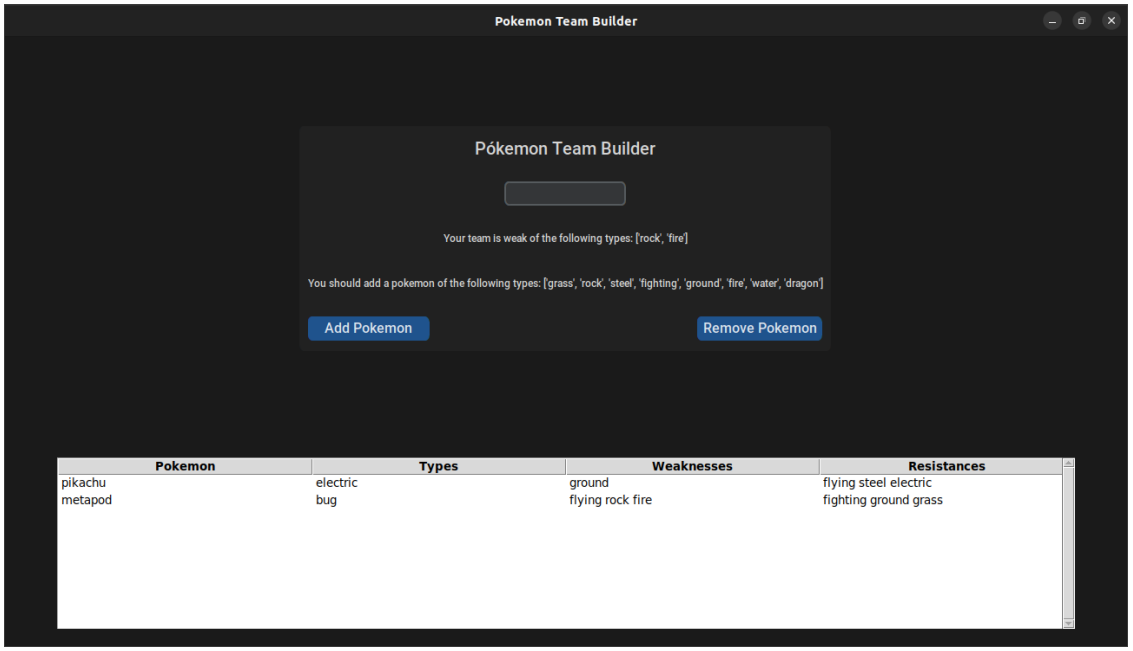
A continuación se presenta un ejemplo de ejecución del programa, en el que añadiremos Pokémon hasta conseguir un equipo equilibrado. Al ejecutar el programa, nos encontraremos con la siguiente interfaz:



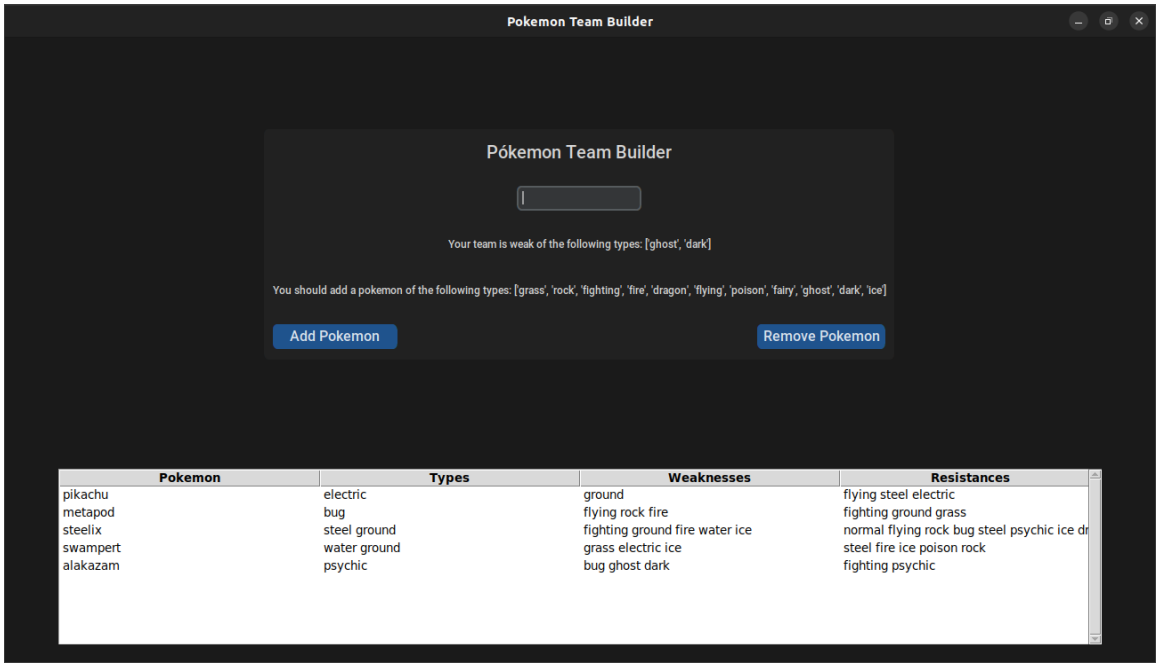
Como primer Pokémon vamos a introducir a Pikachu, Pokémon de tipo eléctrico. Para ello, escribiremos su nombre en el cuadro de input y le daremos a “Add Pokémon”. La tabla de abajo se actualizará con la información del Pokémon.



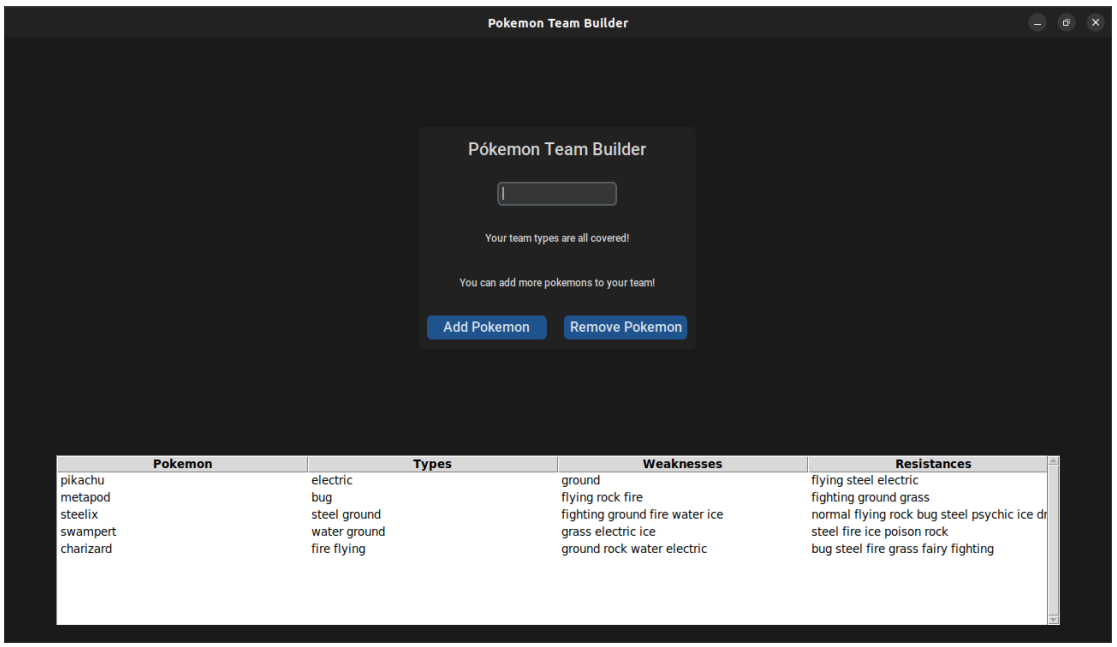
Siguiendo las recomendaciones del programa, deberíamos añadir un Pokémon de tipo *bug* o *grass*, al ser estos dos resistentes al tipo *ground*. Por esta razón, introduciremos al Pokémon de tipo *bug* Metapod a nuestro equipo.



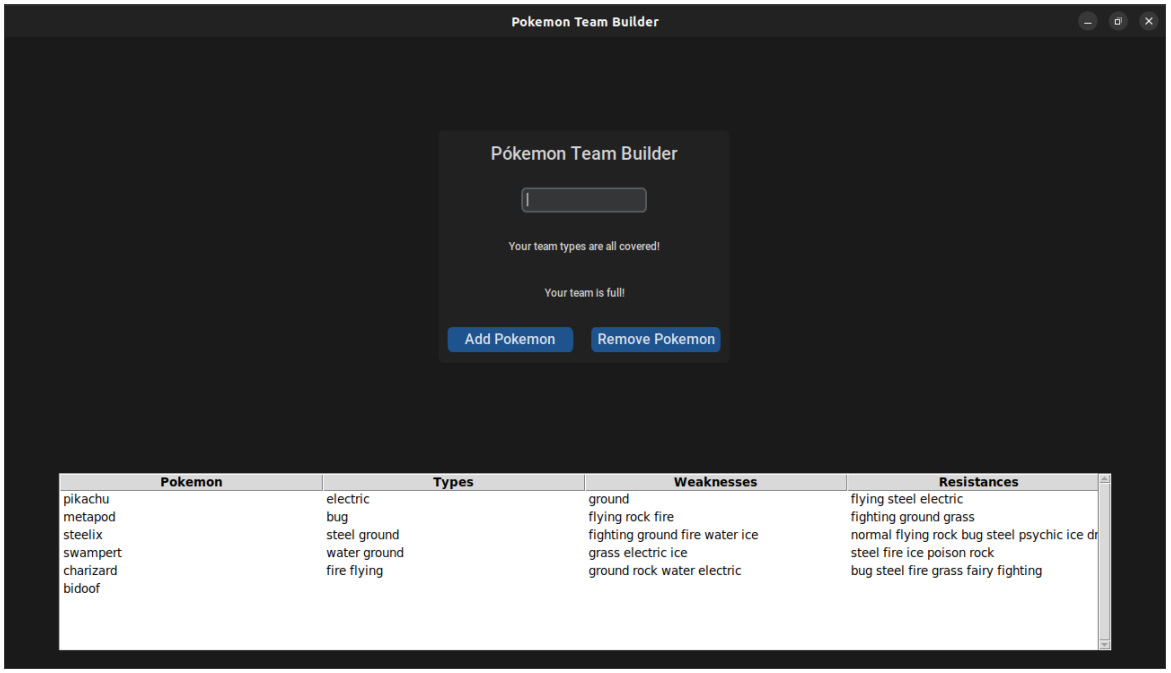
Ahora mismo, nuestro equipo es débil a los tipos *fire* y *rock*. Siguiendo las sugerencias del programa, añadiremos dos Pokémon: Steelix, de tipo *steel y ground*; y Swampert, de tipo *water* y *ground*. Además, añadiremos provisionalmente un pokémon que no nos recomiendan, Alakazam de tipo *psychic*.



No conformes con nuestra decisión, haremos click al botón de “Remove Pokémon”. Con esto, se eliminará la última entrada añadida a la tabla, en nuestro caso el Pokémon Alakazam. Siguiendo las recomendaciones de nuestro programa, añadiremos a Charizard, de tipo *fire* y *flying*.



Con esto hemos conseguido un equipo con todas sus debilidades cubiertas, aunque el programa nos avisa que el equipo aún no está finalizado. Añadiremos finalmente a Bidoof de tipo *normal*.



Finalmente, hemos conseguido un equipo completo de 6 Pokémon sin ninguna debilidad en conjunto.

5. Conclusión.

Prolog es un lenguaje muy completo para el desarrollo de programas que utilizan bases de conocimiento. Integrarlo con un lenguaje de propósito general como Python puede abrir muchas posibilidades a la hora de desarrollar programas. De esta forma, se ha desarrollado una herramienta simple con amplia posibilidad de expansión como, por ejemplo, importar y exportar tus equipos, mejorar la GUI con una lista de los Pokémon disponibles, etc.

El trabajo realizado en Prolog ha resultado complicado debido a la naturaleza de los lenguajes de programación declarativos. No obstante, el uso de Python para la integración de Prolog en una interfaz gráfica ha resultado en un trabajo diferente al acostumbrado y a un alivio de la carga de trabajo en Prolog.