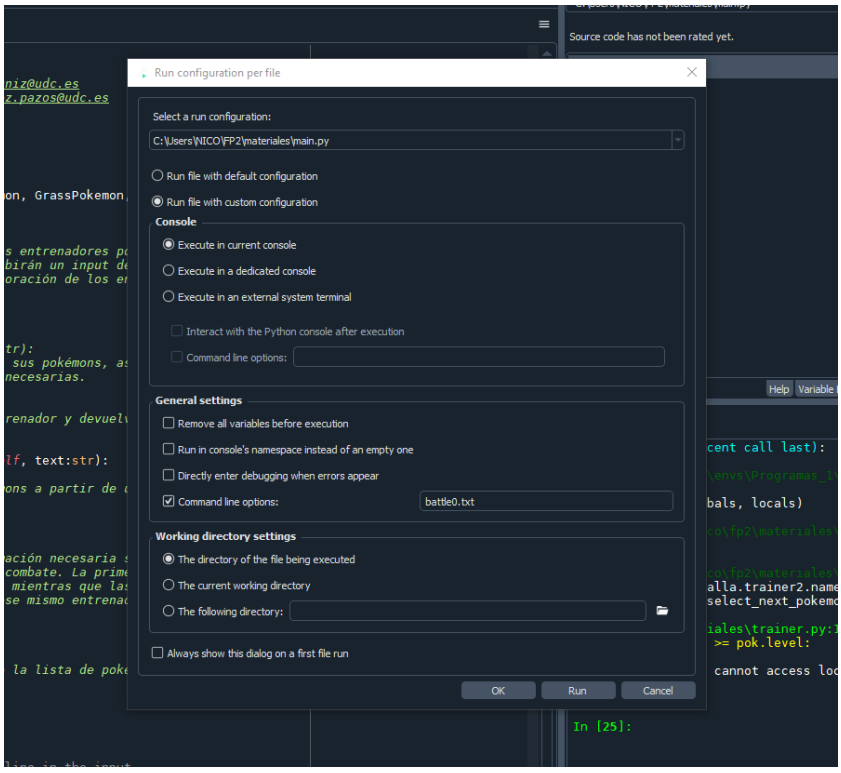
MANUAL DE USUARIO

Nicolás Muñiz Rodríguez : nicolas.muniz@udc.es
Pablo José Pérez Pazos : pablo.perez.pazos@udc.es

· Mecanismo de ejecución:

Para conseguir ejecutar el programa lo primero que debemos hacer es extraer los archivos de en código python del archivo en formato zip adjunto (“pokemon.py”, “trainer.py” y “main.py”) por ejemplo a una carpeta del escritorio.

A continuación, debemos abrir el archivo “main.py” mediante Spyder, habiendo abierto este último desde el navegador de Anaconda. Desde esta ventana, haremos *click* en la pestaña “Run” y seleccionaremos la opción “Configuration per file…”, que nos abrirá la siguiente pestaña:



Aquí tendremos que tener marcadas las opciones “Run file with custom configuration”, “Execute in current console”, “The directory of the file being executed” y “Command line options” donde, a la derecha de esta última, escribiremos el nombre del archivo de texto que contiene la información de los entrenadores y sus pokémons, en mi caso “battle0.txt”.

Por último, haremos *click* en el botón de “Run” de esta última pestaña y observaremos los resultados en la terminal de spyder.

· Fases de desarrollo:

Para comenzar a moldear el código, decidimos construir primero el bloque básico de los combates: los pokémons. Empezamos por construir la clase abstracta “Pokemon”, que hace referencia al concepto general de pokémon. A esta clase le asociamos las variables relevantes para los combates (vida, nivel, daño, etc.) como variables privadas (denotadas por “_” delante del nombre) y creamos algunos de los métodos necesarios para la correcta funcionalidad de los pokémons como por ejemplo “basic_attack”, “is_debilitated” o el método abstracto “effectiveness”, que añadiremos en cada subclase. Además, para en un futuro poder llamar atributos que inicialmente son privados, creamos los *setters* y *getters* de estos, excepto los setters de “name” y “total_hp”.

Habiendo ya casi completado la clase “Pokemon”, definimos sus clases hijas, es decir, las clases de los diferentes tipos de pokémon. Estas subclases heredan todos los atributos de la clase padre, además de cada una definir su atributo “pokemon_type” como un string (“Water”, “Grass” o “Fire”) y el método “effectiveness”, que devolverá un valor diferente dependiendo del oponente. A la par de esto, empezamos a desarrollar la funcionalidad del entrenador, creando la clase “Trainer” con sus atributos, *getters*, *setters* y métodos. De esta sección, lo que más nos costó implementar fue el método “select_next_pokemon” con diferencia, ya que a la hora de programar la organización de los combates y las batallas nos daría algún que otro problema.

Las etapas siguientes del desarrollo se centraron sobre todo en implementar correctamente todos los métodos de las clases “WaterPokemon”, “FirePokemon” y “GrassPokemon”, con sus respectivos atributos únicos, *getters*, *setters* y especificaciones de “effectiveness”, como mencionamos anteriormente. No surgieron demasiadas complicaciones en el desarrollo su funcionalidad a diferencia de en la última etapa, pero sí es cierto que tuvimos que parchear alguna función para que al ejecutar el archivo de prueba “unit_tests_pokemon” nos devolviese un valor de *OK* y poder continuar hacia el archivo “main”.

Al llegar a la edición y organización de la batalla y los combates entre los pokémons de los entrenadores, lo primero que hicimos fue leer y entender el código previamente escrito, para tener un mayor grado de control sobre el archivo y su funcionalidad, lo cual no fue complicado teniendo el conocimiento sobre clases de python y la programación orientada a objetos que poseíamos de haber completado el resto de archivos. Sustituimos los *TODO* por el código necesario y pasamos a crear la clase “Batalla”, que gestionará la selección de pokémons, la batalla y determinará un ganador.

Esta fue sin duda la etapa más ardua del desarrollo. Obtener finalmente un resultado fue bastante complicado debido a que tuvimos que re-editar tanto partes anteriores del código como en la propia clase “Batalla” y eso nos costó tiempo y un par de reelaboraciones del modelo del código del que partimos. Aún así, tras esos pequeños pasos hacia atrás, conseguimos que ambos entrenadores luchasen y que se determinase un entrenador ganador en la batalla.