Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2020-2

27 de Agosto de 2020
Actividad Formativa

Actividad Formativa 01

Programación Orientada a Objetos I

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AF01/`
- **Hora del *push*:** 16:50

Importante: Antes de comenzar, comprueba que Git este funcionando correctamente en tu repositorio privado. Para esto, **sube los archivos base de la actividad de inmediato** (*add*, *commit*, *push*). Se espera que en esta actividad (así como en las demás actividades y tareas) utilices Git a lo largo de **todo tu desarrollo** como una herramienta, no sólo como un método de entrega. Es por esto que recomendamos enfáticamente que vayas subiendo tus cambios constantemente (***push***), ya que **problemas de último minuto** relacionados con la entrega y Git **no serán considerados**.

Introducción

Como ya nos hemos dado cuenta, la pandemia ha sido un periodo muy difícil para todos y todas. Hemos podido ver que las personas enfocan gran parte de sus días exclusivamente a sus labores obligatorias dejando de lado el tiempo de descanso.

Es por esto, que te has propuesto crear los cimientos del juego que cambiará el estado de ánimo de todas las personas: ¡El aclamado ~~Pokémon~~ **DCCriaturas**! Este juego consiste en simular la pelea de las criaturas del usuario contra las criaturas de sus rivales.

Flujo de DCCriaturas

El juego comienza con la creación de un **Entrenador**, que será el personaje controlado por el usuario. Una vez ya creado el entrenador, a este **se le asignan 6 criaturas de manera aleatoria desde la base de datos** (ubicada en `criaturas.csv`) para poder luchar. Inmediatamente después de esto comienza la simulación de batalla.

En la simulación el jugador batalla contra varios rivales y, en cada una, los entrenadores, tienen **Bolsillos** asociados, que básicamente son contenedores para sus criaturas. Al comenzar la simulación de una batalla, el jugador y el rival lanzan **la primera criatura que tengan en su bolsillos**, las que luchan hasta que una de las dos se queda sin puntos de vida. Una vez que pasa esto, el dueño de la criatura que se ha quedado sin puntos de vida, **procede a lanzar su siguiente criatura al combate**. La batalla termina una vez que uno de los entrenadores se queda sin criaturas disponibles para luchar. Si el ganador es el jugador, se cambiará su criatura más débil con la criatura más fuerte del rival, mientras que si el ganador es el rival no ocurrirán intercambios.

Antes de iniciar la siguiente batalla, se sanan todas las criaturas del usuario y se hacen más poderosas (suben de nivel). Posteriormente, comienzan los preparativos para el siguiente combate.

El juego termina una vez que el jugador pierde o derrota a todos los rivales.

Archivos

Archivos de datos

- `criaturas.csv`: En este archivo encontrarás los datos de todas las criaturas. La primera línea del archivo contiene el nombre de las columnas y el resto de las líneas contienen las características de las critauras, separadas por coma, de la forma:
  `Name,Type,HP,Attack,Sp_Atk,Defense`
- `rivales.csv`: En este archivo encontrarás los datos de todos los rivales. La primera línea del archivo contiene el nombre de las columnas y el resto de las líneas contienen, separados por coma, el nombre del rival, y la lista de criaturas que posee. Las criaturas a su vez estarán separadas entre sí por un punto y coma (`";"`), generando líneas de la forma:
  `entrenador,criatura1;criatura2;criatura3;criatura4;criatura5;criatura6`

Archivos de código

- `cargar_datos.py`: En este archivo encontrarás todos los métodos relacionados con el cargado de datos. Deberás completar los métodos `cargar_criaturas`, `cargar_rivales` y `crear_jugador`.
- `entidades.py`: Aquí encontrarás las definiciones de las clases que representan a las entidades del juego. Deberás completar las clases `Criatura` y `Entrenador`.
- `bolsillo.py`: Aquí encontrarás la definición de la estructura de datos que modela el bolsillo de los entrenadores. Deberás completar la clase `BolsilloCriaturas`.
- `batalla.py`: Aquí encontrarás la definición para de la simulación de la batalla. **Este archivo ya está completo y NO debes modificarlo**.
- `main.py`: Este archivo contiene la instanciación de las clases necesarias para ejecutar la simulación, y es el archivo que deberás ejecutar cuando tu implementación esté completa. **Este archivo ya está completo y NO debes modificarlo**.

Parte I: Modelación de las entidades

Antes de poder comenzar con la simulación del juego, es importante definir la estructura de las entidades que forman parte de DCCriaturas. Estas son representadas por medio de las clases en `entidades.py`, las que deberás completar en base a los siguientes requerimientos:

- `class Criatura`: Esta es la clase que representa a una criatura, la que combate con otras criaturas durante la simulación. Esta clase incluye los siguientes métodos:
  - `def __init__(self, nombre, tipo, hp, atk, sp_atk, defense)`: Este es el inicializador de la clase, y debe asignar los siguientes atributos:
    - `self.nombre`: Un `str` que representa el nombre de la criatura.
    - `self.tipo`: Un `str` que representa el tipo de la criatura.
    - `self.__hp`: Un atributo privado, de tipo `int`, que representa los puntos de vida variables de la criatura. Siempre debe estar entre 0 y `self.hp_base`
    - `self.hp_base`: Un `int` que representa los puntos de vida bases de la criatura (sus puntos de vida iniciales y a la vez su máximo posible).
    - `self.atk`: Un `int` que representa los puntos de ataque físico de las criaturas.
    - `self.sp_atk`: Un `int` que representa los puntos de ataque especial de la criatura.
    - `self.defense`: Un `int` que representa los puntos de defensa de la criatura.
    - `self.preferencia`: Este es un atributo que llama al método `preferencia_combate` de esta misma clase.
  - `def hp(self)`: *Getter* del atributo `__hp` del objeto `Criatura`.
  - `def hp(self, nuevo_hp)`: *Setter* del atributo `__hp`. Debe verificar que `nuevo_hp` no se salga de los rangos permitidos.
  - `def preferencia_combate(self)`: Este método define la preferencia de combate de la criatura y retorna un *string* con el valor `"Fisico"` o `"Especial"`. Esta elección se hace a partir del tipo de la criatura: si su tipo tiene más de 5 letras preferirá usar ataques de tipo `"Especial"`, mientras que en caso contrario preferirá los ataques de tipo `"Fisico"`.
  - `def recibir_ataque(self, dano)`: Esta función se llamará cada vez que la criatura reciba daño del enemigo. Deberá utilizar correctamente la *property* `hp` para descontar el daño causado.
  - `def __str__(self)`: Este método se llamará cada vez que se imprima en pantalla una instancia del objeto. Puedes elegir el mensaje que retornas, mientras este contenga el `nombre` y el `tipo` de la criatura. Por ejemplo:

```
"Criatura: Arcanine, tipo: Fire"
"Mi nombre es Lucario y soy de tipo Fighting"
```

- `class Entrenador`: Esta es la clase que define a un entrenador, es decir, tanto al jugador como a sus rivales.
  - `def __init__(self, nombre, bolsillo)`: Este es el inicializador de la clase, y debe asignar los siguientes atributos:
    - `self.nombre`: Un `str` que representa el nombre del entrenador

- `self.bolsillo`: Una instancia de la clase `BolsilloCriaturas`. Contiene a las criaturas del entrenador que a su vez son instancias de la clase `Criatura`.

Para probar tus avances en este módulo, puedes correr directamente el archivo `entidades.py`. El código dentro de `if __name__ == "__main__"` instancia la clase `Criatura` y llama a cada uno de los atributos que se espera que tenga, junto con los métodos de los requerimientos. Además, se crea una instancia de `Entrenador`, y se repite el mismo proceso. Si tus clases están definidas correctamente, el *output* debería tener un formato similar a este:

```
"Atributos de Criatura: OK"
"Método preferencia_combate de Criatura: OK"
"Método recibir_ataque de Criatura: OK"
"Atributos de Entrenador: OK"
```

Parte II: Modelación del Bolsillo

El bolsillo es el lugar donde los entrenadores guardan a sus criaturas. Para modelar este bolsillo crearemos nuestra propia estructura de datos utilizando **herencia**, cuya definición se encuentra en `bolsillo.py`.

- `class BolsilloCriaturas(list)`: Esta clase representa el bolsillo y **hereda** del *built-in* `list`. Esta clase incluye los siguientes métodos:
  - `def append(self, criatura)`: Este método recibe una instancia de la clase `Criatura` y se encarga de agregarla al final del bolsillo siempre y cuando este tenga menos de 6 criaturas. Si se trata de agregar una séptima criatura se debe imprimir un mensaje de error, indicando que se llegó al límite, y no agregarla.
  - `def cantidad_criaturas_estrella(self)`: Este método retorna un `int` con la cantidad de criaturas dentro del Bolsillo que son consideradas "criaturas estrella". Una criatura es considerada estrella del equipo si la suma de los atributos `hp_base`, `atk`, `sp_atk` y `defense` es mayor a 400[1].
  - `def __add__(self, bolsillo_enemigo)`: Este método será llamado cuando el entrenador dueño del bolsillo derrote a su enemigo. El método recibirá el bolsillo del perdedor (otra instancia de `BolsilloCriaturas`) y deberá intercambiar la criatura más débil del Bolsillo del entrenador, con la criatura más fuerte del Bolsillo del enemigo. Diremos que una criatura es más débil que otra si es que la suma de los atributos `atk` y `sp_atk` de la criatura es menor que la misma suma para la otra.

    * Notar que esta puede ser la parte conceptualmente más compleja, pero la simulación funcionará correctamente. En este caso simplemente no se haría el intercambio de criaturas. **Puedes dejar este método para el final.**

Para probar tus avances en este módulo, puedes correr directamente el archivo `bolsillo.py`. El código dentro de `if __name__ == "__main__"` crea una lista con instancias de `Criatura` (usando una `namedtuple`, por si no has completado la definición de las entidades) e intenta agregarlas a `Bolsillo` utilizando `append`. Luego, intenta agregar una séptima criatura, lo que debería fallar, y finalmente prueba el método `__add__` de `Bolsillo`. Si tus funciones funcionan correctamente, el *output* debería tener un formato similar a este:

[1] Si el valor de uno de estos atributos es mayor a 100, se considera un buen valor, por lo que una suma de 400 significa que la criatura es bastante fuerte.

```
"El bolsillo debería tener 0 criaturas"
"Tiene 0 criaturas"

"El bolsillo debería tener 6 criaturas"
"El bolsillo tiene... 6 criaturas"

<Mensaje indicando que está lleno el bolsillo>
"(Deberías ver un mensaje de error porque no puedes agregar una séptima criatura)"
"El bolsillo tiene... 6 criaturas"

"El bolsillo debería tener 1 criatura estrella"
"El bolsillo tiene... 1 criaturas estrella"

"Método __add__ de Bolsillo: OK"
```

Parte III: Cargado de datos

Ya habiendo completado las distintas clases, deberás instanciar las criaturas y entrenadores a partir de los archivos entregados: `criaturas.csv` y `rivales.csv`, además de crear al entrenador que usará el jugador. Para lograrlo, se deben completar las siguientes funciones:

- `def cargar_criaturas(ruta)`: Esta función recibe un `str` con la ruta al archivo de criaturas y retorna un `dict` donde la key será el nombre (`str`) de la criatura y su value una intancia de la clase `Criatura`, por ejemplo `{"Charmander" : Criatura(...)}`.
- `def cargar_rivales(archivo_rivales)`: Esta función recibe un `str` con la ruta al archivo de los rivales y retorna una lista con instancias de la clase `Entrenador`. Cabe señalar que para instanciar correctamente a los rivales necesitarás instanciar sus bolsillos y agregarles las criaturas correspondientes.
- `def crear_jugador(nombre)`: Esta función recibe un `str` con el nombre del jugador y retorna una instancia de la clase `Entrenador` con 6 criaturas al azar en su bolsillo[2].

Para probar tus avances en este módulo, puedes correr directamente el archivo `cargar_datos.py`. El código dentro de `if __name__ == "__main__"` utiliza las funciones definidas en el módulo para cargar los datos desde los archivos. Luego, verifica que lo retornado por cada función tenga los tipos apropiados: para `cargar_criaturas` y `cargar_rivales` una lista de instancias, mientras que para `crear_jugador` es solo una instancia de clase. Posteriormente se verifican los tipos de datos de las instancias cargadas y que el tamaño del bolsillo del jugador sea el esperado. Si tus funciones funcionan correctamente, el *output* debería tener un formato similar a este:

```
"Lista de Criatura tiene formato correcto"
"Recuerda: cargar_rivales retorna una lista de Entrenador"
"Instancias de Criatura tienen atributos con tipo correcto"
"Jugador tiene la cantidad correcta de Criatura en su Bolsillo"
```

[2]Recuerda siempre revisar las notas, pueden a ayudarte mucho en la actividad

Simulación

Ahora que tienes definidas las entidades, los bolsillos y las funciones para cargar los datos, puedes ejecutar el archivo `main.py`. En este archivo se ejecuta todo el código que desarrollaste, y te permitirá conocer el resultado de la simulación.

Notas

- Puedes usar los métodos `choice(secuencia)` o `sample(secuencia, k)`, de la librería `random`, para obtener 1 (en el caso de `choice`) o k elementos sin repetición (en el caso de `sample`) al azar de una secuencia, como una lista
- Recuerda que esta actividad es de tipo **formativa**, por lo que no necesitas tenerla correcta en un 100 % para tener todo el puntaje.

Objetivos de la actividad

- Modelar clases correctamente, utilizando los atributos, métodos y properties adecuados para simular su comportamiento.
- Crear una estructura de datos personalizada a través de la herencia desde una estructura *built-in*.