Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2019-1

1 de julio de 2019
Actividad Sumativa

Actividad 10 (Recuperativa)

Introducción

El malvado Dr. Herny en conjunto a su orden oscura: Tamburini Maw, Poleus Glave, Danixima Midnight y Cull Wieladian lograron obtener los contenidos del infinito, luego que nuestros héroes, los alumnos de Programación Avanzada, las intentaran proteger de sus manos. Ha pasado casi un semestre del chasquido, y tienes una misión, deshacer la tragedia recolectando los contenidos del infinito:

- Estructuras *built-in* del **Poder**
- Programación Orientada a la **Realidad**
- Estructuras Nodales del **Espacio**
- Excepciones de la **Mente**
- *Threading* del **Tiempo**
- Serialización del **Alma**.

Instrucciones y entrega

- **Lugar de entrega:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AC10`
- **Hora del *push*:** 12:30

A continuación se presentan seis secciones con problemas de programación distintos, cada uno vale **dos puntos**. Cada uno está enfocado en un contenido específico del curso. Tú solo debes desarrollar **tres** de estas secciones, con las cuales se te evaluará esta actividad. Una de las secciones que realices **debe** ser aquella del contenido correspondiente a la actividad sumativa que **reemplazarías** con la nota de esta actividad. Las otras dos secciones, quedan a elección tuya. **Se recomienda comenzar por la sección de un contenido que conozcas bien.**

Solo debes entregar en tu repositorio los archivos correspondientes a las tres secciones que realizaste, y en sus carpetas respectivas. En caso de no realizar la sección de la actividad reemplazada, se considerará como puntaje 0 para un tercio de la evaluación. Y, en caso de entregar más de dos secciones adicionales, se corregirán dos al azar de las entregadas.

1. Estructuras de datos *built-in* (AC01 sumativa)

Para recuperar el contenido del **Poder**, debes encontrar aquel ser que se robó la mayor cantidad de **números del poder** en la última guerra de números.

En esta guerra participaron cientos de extraterrestres, donde todos comenzaron con un conjunto de números distinto. Cuando empezó la guerra, estos extraterrestres comenzaron a robarse entre ellos números, lo que generó que algunos quedaran con pocos números de poder, y otros con muchos. Se encontraron registros escritos de las interacciones entre extraterrestres durante la guerra y es tu deber usarlos para reproducir lo ocurrido, para encontrar al extraterrestre que terminó con más números del poder.

Para lograr esto, debes completar las siguientes funciones:

- `def cargar_extraterrestres(path_extraterrestres)`

  Esta función recibe el *path* a un archivo de extensión `.txt` que contiene la información inicial de los extraterrestres: su nombre, su identificador universal, y su conjunto de números de poder inicial. El conjunto de número que tiene un ser **nunca debe tener repetidos**. Cada línea del archivo viene en el siguiente formato:

  ```
  Iribula,18456,3;5;7;9;10
  ```

  Que por separación de `","` los elementos corresponden respectivamente al nombre, identificador universal y al conjunto de números, que a su vez se separa por `";"`. La función debe leer este archivo y retornar una estructura de datos **adecuada** que contenga instancias de la clase entregada: `Extraterrestre`.

- `def simular_guerra(extraterrestres, path_guerra)`

  Esta función recibe tanto la estructura de datos que contiene los extraterrestres como el *path* al archivo de extensión `.txt` que contiene los registros secuenciales de robos entre extraterrestres durante la guerra. En este archivo se detalla en orden, línea a línea, que extraterrestre le robo a qué otro. El único problema, es que encontraron errores en estos registros, por lo que hay líneas que revierten un robo antes registrado. Hay dos tipos de línea en este archivo:

  - Las líneas que detallan un robo. Por ejemplo: `18456,12650`, que significa que el extraterrestre de identificador `18456` le robó al extraterrestre de identificador `12650`.
  - Las líneas que detallan que hay que revertir el último robo registrado. Estas siempre dicen: `undo`.

  La función debe cargar dicho archivo y aplicar, en orden, los robos sobre los extraterrestres. Y debe hacerlo de tal forma, que considere la aparición de `undo` en los robos. Importante es notar, que pueden aparecen múltiples `undo` sucesivos, que revierten los últimos robos, según la cantidad de `undo`. Eso sí, puedes asumir que nunca habrán más `undo` que robos previos. **No es necesario que esta función retorne un objeto**, ya que alterará los extraterrestres entregados.

  Cuando se simula un robo entre un extraterrestre y otro, siempre lo hacen de la misma forma: el ladrón **solo** quitará a la víctima aquellos números que **no posea el ladrón, pero sí la víctima**. Notar que esto mantiene la propiedad de que un ser nunca tiene números de poder repetidos en su posesión.

  Se espera de esta función que se haga uso de las estructuras de datos *built-ins* **adecuadas** para conseguir el objetivo descrito.

- `def encontrar_maximo_poder(extraterrestres)`

  Esta función recibe la estructura de datos con extraterrestres y retorna aquel ser que tiene la mayor cantidad de números de poder.

Se presenta un ejemplo de una guerra de números a continuación. Hay tres extraterrestres: Iribula, Stariel Lord e Ian Machine. Tienen inicialmente los conjuntos de números: $\{2, 3, 4, 5\}$, $\{3, 4, 5, 6, 7\}$ y $\{2, 3, 8, 9\}$, respectivamente.

```
Iribula,18456,2;3;4;5
Stariel Lord,12650,3;4;5;6;7
Ian Machine,56783,2;3;8;9
```

Los registros de guerra dicen lo siguiente:

```
18456,12650
12650,56783
18456,56783
    undo
    undo
56783,18456
```

Es decir, ocurre que:

- Primero, Iribula le roba a Stariel Lord. Resultando en que tienen $\{2, 3, 4, 5, 6, 7\}$ y $\{3, 4, 5\}$, respectivamente.
- Luego, Stariel Lord le roba a Ian Machine. Resultando en que tienen $\{2, 3, 4, 5, 8, 9\}$ y $\{3\}$, respectivamente.
- Luego, Iribula le roba a Ian Machine. Resultando en que tienen $\{2, 3, 4, 5, 6, 7\}$ y $\{3\}$, respectivamente.
- Apareció un `undo`, por lo que se revierte el último robo. Se mantienen en $\{2, 3, 4, 5, 6, 7\}$ y $\{3\}$.
- Apareció otro `undo`, lo cual revierte el robo de Stariel Lord a Ian Machine. Stariel Lord vuelve a tener $\{3, 4, 5\}$, e Ian Machine a $\{2, 3, 8, 9\}$
- Finalmente Ian Machine le roba a Iribula. Resulta en que tienen $\{2, 3, 4, 5, 6, 7, 8, 9\}$ y $\{2, 3\}$ respectivamente. Ian Machine termina siendo el extraterrestre con más números de poder.

Notas

- Recordar algunas de las operaciones de conjuntos que permiten los `sets`: `-` y `|`.
- Puedes asumir que todos identificador en el archivo de guerra existe previamente y que después de la guerra, solo habrá un extraterrestre con la mayor cantidad de números.

Requerimientos

- (0.50 pts) Completar `cargar_extraterrestres` correctamente y retornar estructura adecuada.
- (1.30 pts) Completar `simular_guerra` con el uso de estructuras adecuadas.
- (0.20 pts) Completar `encontrar_maximo_poder` correctamente.

2. Programación orientada a objetos (AC02 formativa)

Para recuperar el contenido de la **Realidad**, deberás enfrentarte al poderoso **Dr. Herny** con la ayuda de **Antony Stark**, **Capitan Ruz** y **Vichor**. Sin embargo, el malvado doctor tiene en su posesión tres contenidos del infinito, específicamente, el de la **Realidad**, **Poder** y **Tiempo**, por lo cual los Programavengers te piden a ti, ávido programador, hacer una simulación de la batalla haciendo uso de POO para poder maximizar las chances de vencer al malvado doctor.

Se te entregarán tres archivos: `personas.py`, `parametros.py` y `main.py`. En `personas.py` encontrarás las siguientes clases:

- `class Avenger`: Representa a un poderoso **Avenger**. Tiene un `arma`, que es un `bool`, `fuerza`, que es un `int` y `ataque`, que es una `property` que depende de la fuerza y de si tiene arma o no. Especificamente, el ataque es `1.5 * self.fuerza` si se posee un arma, en caso contrario es solo `self.fuerza`.
- `class Programmer`: Representa a un **ávido programador**. Tiene `vida` que es un `int` representada por una `property`, y la capacidad de `decodificar` a su enemigo, que es un **método**.

  El método `decodificar` recibe al enemigo y ve si su **vida restante** es divisible por el **ataque del programmer**. En caso de que esto sea así, el progrmmaer recupera un 10 % de su vida.
- `class Jefe`: Representa a ~~la incarnación del mal~~ un ayudante jefe. Tiene `ataque`, que es un `int` y a diferencia de los Avengers, el Jefe tiene total dominio sobre los controles del infinito, por lo cual ataca utilizándolos. Tiene un **método** por cada contenido que tiene en su posesión, cada una de estas recibe al enemigo al cual se le aplicará el efecto.

  Los contenidos son los siguientes:

  - `realidad`: Convierte el arma del oponente en burbujas, destruyéndola.
  - `tiempo`: Retrocede en el tiempo, recuperando un 10 % de la vida máxima de tanto el jefe, como su enemigo. Si la arma del enemigo estaba destruida, esta se recupera.
  - `poder`: Reduce la vida del enemigo en un 20 % de su valor original, es decir, tomaría 5 golpes de este contenido para destruir a cualquier Programavenger.
- `class Programavenger`: Representa a un Programavenger, es tanto un **Avenger** como un **Programmer**. Tiene un `nombre`, que es un `str` y un **método** encargado de `atacar` a **DrHerny**, el cual es recibido como argumento. Al momento de atacar, primero intenta decodificar a su enemigo y luego le resta su ataque a la vida de su enemigo.
- `class DrHerny`: Representa a las fuerzas del mal. Es tanto un **Jefe** como un **Programmer**. Tiene un **método** encargado de `atacar` a un **Programavenger**. Primero intenta decodificar a su enemigo, luego lo ataca con uno de los contenidos del infinito. El contenido es elegido mediante las siguientes reglas (en el orden que se muestran):
  - Si el enemigo tiene un arma y menos del 60 % de su vida máxima, se ocupará el contenido de la **Realidad**.
  - Si la vida de DrHenry esta bajo el 5 %, se ocupará el contenido del **Tiempo**.
  - En caso contrario, se utilizará el contenido **Poder**

En `parametros.py` podrás encontrar todas las estadísticas que se utilizarán para crear las instancias. No es necesario que hagas nada con este archivo, pero puedes jugar con los valores si es que quieres.

En `main.py` podrás encontrar la instanciación de las clases y la simulación del combate. En esta simulación se instanciaran los tres Progravengers definidos en `parametors.py` y a DrHerny, luego tomaran turnos para atacarse los unos a los otros. Tampoco es necesario que hagas nada con este archivo.

Deberás imprimir todo lo que pasa en la simulación de la siguiente forma:

- Cuando DrHerny ocupa un contenido del infinito deberás imprimir lo siguiente si el contenido es el del **Tiempo** o del **Poder**:

  `DrHerny ha ocupado el contenido del {contenido} contra {Progravenger}`

  En caso contrario deberás imprimir:

  `DrHerny ha ocupado el contenido de la Realidad contra {Progravenger}`

- Cuando un Progravenger logre decodificar a DrHerny debes imprimir:

  `{Progravenger} ha decodificado a DrHerny`

  En caso de que DrHerny decodifique a un Progravenger deberás imprimir:

  `DrHerny ha decodificado a {Progravenger}`

- Cuando un Progravenger ataque a DrHerny deberás imprimir:

  `{Progravenger} ha atacado a DrHerny`

  En caso de que este ataque resulte causando que la vida de DrHerny llege a cero, deberás imprimir lo siguiente:

  `Los Progravengers han derrotado a DrHerny`

- Finalmente si DrHerny con uno de sus ataques causa que la vida de un Progravenger llegue a 0 deberás imprimir lo siguiente:

  `Dr Herny a derrotado ha {Progravenger}`

En todos estos *print* {Progravenger} es el nombre del Progravenger involucrado en la interacción y {contenido} es el nombre del contenido.

Notas

- Recuerda que solo debes modificar `personas.py`.
- Recuerda que `main.py` contiene la simulación armada. Podrá ayudarte para entender como interactúan las clases.

Requerimientos

- (1.00 pts) Todas las clases están correctamente implementadas.
  - (0.6 pts) Clases `Jefe`, `Programmer` y `Avenger` correctamente implementadas.
  - (0.4 pts) Clases `Progravenger` y `DrHenry` correctamente implementadas.
- (1.00 pts) El *output* del programa es el correcto.

3. Estructuras de datos nodales (parte 2): grafos (AC04 sumativa)

El contenido del **Espacio** ha sido uno de los más preciados y disputados a través del tiempo. Hace muchos años atrás **Red Skullwirth** y su malvada organización utilizó este contenido para tratar de ganar dominar el mundo, todo hubiera resultado en una terrible dominación si no fuera por el sacrificio del **Capitán Ruz**, encerrando su contenido en el **Teseracto**. Años más tarde, el **Teseracto** fue encontrado por el **Dr. Hank Pymto** quién lo mantuvo oculto, hasta que descubrió que el malvado **Dr. Herny** estaba en su búsqueda.

Si eres capaz de dominar el contenido del **Espacio**, podrás crear y modelar el espacio a tu gusto, elemento que es fundamental para el malévolo plan del **Dr. Herny**. Pero para obtener este contenido, primero es necesario que lo saques de su contenedor, el **Teseracto**.

El **Dr. Hank Pymto** decide buscar tú ayuda y la de **Antony Stark** para sacar el contenido del **Teseracto**, y de esta forma utilizarlo en contra del **Dr. Herny** y sus malvados planes.

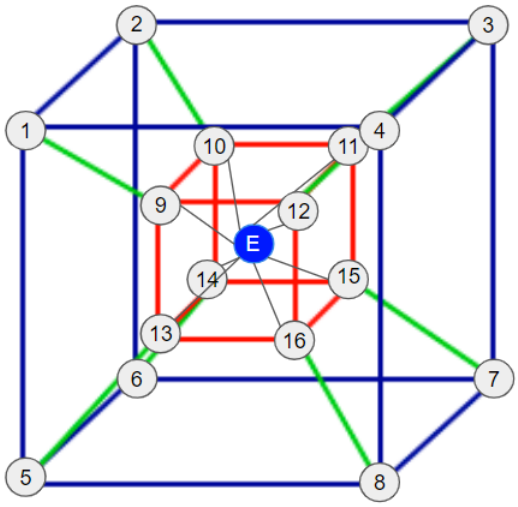

Figura 1: Imagen del **Teseracto** y el contenido del **Espacio**

Tu misión es programar un algoritmo que entregue la ruta óptima desde el nodo 1 del **Teseracto**, hasta el nodo **E**. Para luego ser entregada a **Antony Stark**, y que el logre con su tecnología sacar el contenido del **Espacio** siguiendo tu ruta. El espera que le des una lista de los índices de los nodos, ordenada de forma de que al recorrerla pueda llegar del origen al contenido del Espacio.

Debes saber que solo hay **una** ruta óptima, y entregar una ruta incorrecta a **Antony** podría destruir el espacio como lo conocemos. **Antony Stark** ya hizo un análisis del **Teseracto** y se dio cuenta de que hay **solo una ruta óptima, y de que es fácil de encontrar si tu algoritmo siempre se va por el camino de menor costo entre los posibles, partiendo del origen.**

Antony Stark ya te facilitó entregándote archivos y código base:

- `teseracto.txt`: Contiene la información de las conexiones entre los nodos del **Teseracto**, representado por una matriz de adyacencia. Las conexiones son bidireccionales y con pesos. Si en una casilla de la matriz hay un 0, significa que no hay una conexión entre los nodos que representa, si hay un valor mayor que 0, este valor es el peso de la conexión entre los nodos que representan las casillas. El último nodo, o nodo número 17, es el que representa el nodo **E**.

- `descifrador_teseracto.py`: Contiene el código base para poder cargar el `teseracto.txt`, para luego poder ser recorrido como un grafo.

Notas

- Recuerda los métodos para recorrer grafos como DFS o BFS.

Requerimientos

- (0.50 pts) Completar función `cargar_teseracto()`, debe devolver una matriz de `int`
- (1.50 pts) Completar función `recorrer_teseracto()`, debe retornar la lista de nodos que representen la ruta óptima. Una lista de `int` que representen los índices de los nodos en el orden que deben ser recorridos, para llegar del origen al nodo **E**.

4. Excepciones (AC05 formativa)

Para volver a ver el contenido de la Mente deben llegar hasta el planeta *eFe* en donde está guardada. El problema es que los secuaces del **Dr Herny** sabotearon el simulador de navegación sin el cual el **Capitán Ruz** no podrá fijar destinos y es deber de ustedes el repararlo.

Sin embargo, no todo está perdido, **Antony Stark** ha logrado detectar que en el simulador de navegación falla una librería en específico, llamada `libreria_automata.py` y ha encontrado un módulo que lo ocupa, `naveganding.py`, que no está preparado para que esta librería falle; a continuación se te detallan los métodos y funciones que tendrás que revisar:

En el archivo `libreria_automata.py` está la clase `Navegador` que tiene los métodos:

- `def definir_destino(nombre)`

  Existe un diccionario con cada destino conocido de esta galaxia, llamado `destinos` que es un atributo de `Navegador` y al cual este método consulta, busca en `destinos` la posición hacia la que fijar ruta y la retorna. En caso de no existir, debe levantar una excepción.

- `def agregar_destino(nombre, posicion)`

  Cada vez que se quiere agregar un destino, se debe revisar que este destino no exista, sino levantar una excepción personalizada `ErrorRepetirDestino`. Al recibir 3 vez o más veces este error se debe actualizar de igual forma la posición del destino, porque se trata de una emergencia de parte de la base estelar. Esta función **no** retorna nada al ser ocupada, pero puede actualizar el valor del atributo `destinos` del objeto de clase `Navegador`.

- `def destruir_enemigo(enemigo)`

  Recibe como parámetro a un objeto de clase `Enemigo` y setea su atributo `vida` a 0. Levanta un error en caso de que reciba un parámetro de otra clase.

- `def esquivar_obstaculo(posicion_obstaculo, distancia)`

  Este método (de ahora en adelante lo abreviaremos como `e_o`) tiene la funcionalidad de decirle a la nave si se debe mover en alguna dirección según los obstáculos que se le envían. Se debe levantar una excepción cuando el primer argumento no es de largo 2, en este caso el navegador mantiene el rumbo **pero** baja su velocidad en 1 unidad (ve el dato de vital importancia), y también debe levantar error cuando el último argumento no sea `float` o `int`, en este caso el navegador debe mantenerse sin cambios de ningún tipo para seguir con el programa.

  Dato de vital importancia Esta función retorna una tupla de números enteros que representan vectores de movimiento, donde los primeros dos argumentos indican si la nave se debe mover izquierda (-1) a derecha (+1) y de arriba (-1) hacia abajo (+1), mientras que el tercer argumento es la velocidad a aplicar a la nave. Por ejemplo, si sólo quiero subir en posición y también subir en 1 la velocidad, retornaría una tupla de la forma `(0, -1, 1)`. En `naveganding.py`, se debe nombrar esta tupla diciendo: "La nave tomó el vector: {tupla} para evadir el obstáculo".

En el archivo `naveganding.py` tendrán que capturar errores que vayan saliendo al ir leyendo el archivos serializado que es necesario para determinar las fallas de la librería.

Para ayudarte a diagnosticar los problemas, te damos una tabla que describe el comportamiento esperado según los errores encontrados:

Descripción	Solución esperada	Ejemplo	Resultado
El destino definido no existe en el diccionario.	Escoger un destino al azar del diccionario.	'Sala Examen'	'Planeta *eFe*'
Se trata de agregar un destino que ya existe por primera o segunda vez.	No agregar destino.	`a_d('Planeta eFe', (1, 1, 1))`	Imprime 'El destino Planeta *eFe* ya existe, lo han reingresado n veces'
Se trata de agregar un destino que ya existe por una tercera o mayor vez.	Actualizar posición.	`a_d('Planeta eFe', (1, 1, 1))`	Imprime 'El destino Planeta *eFe* ya existe, lo han reingresado n veces. Se actualiza de igual forma.'
Se entrega un objeto que no es clase `Enemigo`.	No quitarle vida.	`Pepino('hola')`	Imprime 'La clase Pepino no es un enemigo'
Se entrega una tupla de largo distinto a 2.	Disminuir velocidad.	`e_o((1, 2, 3), 2)` `e_o((1,),3)`	Se debe disminuir la velocidad en uno: `(0, 0, -1)` y seguir con el programa
Se entrega algo distinto de un `float` o `int` a `e_o`.	Mantener rumbo.	`e_o((1, 2, 3), 'f')`	Se mantiene el rumbo: `(0, 0, 0)` y seguir con el programa

Notas

- No modifiques nada de la librería que no sea para levantar errores.
- Para poder confirmar que un objeto es instancia de una clase, existe `isinstance(objeto, clase)`.
- Recuerda que existen los atributos de clase y de instancia, incluso en las excepciones personalizadas.
- Llevar conteo es más fácil con una estructura que está en `collections` ;)

Requerimientos

- (0.8 pts) Identificación de errores
  - (0.2 pts) Reconocimiento correcto del error al no existir un destino.
  - (0.2 pts) Reconocimiento correcto del error al entregar un objeto que no es clase `Enemigo`.
  - (0.4 pts) Reconocimiento correcto de errores al esquivar obstáculos.
- (0.6 pts) Creación y manejo de `ErrorRepetirDestino`
  - (0.2 pts) Correcta implementación del error personalizado.
  - (0.4 pts) Uso adecuado en el programa principal.
- (0.6 pts) Manejo de errores en el programa principal
  - (0.2 pts) Manejo adecuado del error en `definir_destino`.
  - (0.1 pts) Manejo adecuado del error en `destruir_enemigo`.
  - (0.3 pts) Manejo adecuado de los errores en `esquivar_obstaculo`.

5. *Threading* (AC06 formativa)

Muchos eones atrás el contenido del **Tiempo** fue encontrado por el hechicero **Juanmotto**, quien lo escondió en un dispositivo llamado **El Ojo de Juanmotto**. Posteriormente **Doctor Hundstrange** la entregaría deliberadamente a **Dr. Herny** a cambio de la vida de **Antony Stark** tras ver los 14 millones de futuros posibles, donde ésta era la única forma de vencer al malvado titán.

Para lograr esta ~~difícil~~ misión deberás utilizar tu conocimiento en *threading* para realizar un ataque coordinado con la ayuda de los Progravengers y recuperar el contenido del infinito de las manos del titán.

Para esto debes completar las siguientes clases:

- `class Threatan`:

  Representa al malvado **Titán**, posee las siguientes características:

  - `nombre`: es un atributo, representado por un `str` que corresponde al nombre del Titán.
  - `robado`: es un atributo, representado por un `bool` que corresponde a si le han robado el contenido del **Tiempo** al Titán. Este siempre comienza en valor `False`.
  - `escudo`: es una `property` que corresponde al escudo de defensa. A medida que disminuye, aumenta el porcentaje de **daño de movilidad** recibido.
  - `movilidad`: es una `property` que corresponde al estado de la movilidad del titán. A medida que disminuye, aumentan las posibilidades de recibir un **robo efectivo**.

- `class Progravenger`

  Representa a un **Progravenger** de tu equipo, posee las siguientes características:

  - `nombre`: atributo representado por un `str` que corresponde al nombre del Progravenger.
  - `objetivo`: atributo representado por un `str` que corresponde la característica a atacar.
  - `poder`: atributo representado por un `float` que corresponde a la fuerza con que atacará.
  - `tiempo_de_ataque`: atributo representado por un `float` que corresponde a que tan seguido ataca en segundos.

  Tu programa instanciará tres de ellos, donde cada uno tendrá un rol distinto según su `objetivo`:

  - `"escudo"`: En cada ataque al titán su escudo recibe un daño igual al poder del Progravenger.
  - `"movilidad"`: En cada ataque al titán su movilidad disminuye en cada ataque según la formula:

    $$\texttt{poder} - (\texttt{poder} \times (\texttt{escudo}/100)$$

    Donde `poder` corresponde al poder del Progravenger y `escudo` al escudo del titán.
  - `"robar"`: En cada ataque al titán la posibilidad de lograr robarle el contenido corresponde a:

    $$\texttt{poder} - (\texttt{poder} \times (\texttt{movilidad}/100))$$

    Donde `poder` corresponde al poder del Progravenger y `movilidad` a la movilidad del titán.

- `class Batalla`:

  Es la encargada de instanciar a todos los participantes de la batalla, para luego realizarla. Sus métodos son:

  - `progravengers_assemble`: debe instanciar al Threatan y a tres Progravengers para combatirlo. Los datos son entregados como en el diccionario `DATOS` en `main.py`.
  - `attack`: deben iniciar cada participante de la batalla como un *thread*.

  Cada segundo de la batalla se debe imprimir el detalle del estado del Titán:

  - Nivel de escudo del titán
  - Nivel de movilidad del titán.

  El combate se realiza segundo a segundo, donde los tres Progravengers pelearán al mismo tiempo, pero organizándose de alguna forma ~~ver notas~~ para no atacar exactamente al mismo tiempo. En otras palabras, dos Progravengers pueden atacar en el mismo segundo pero no pueden alterar al titan exactamente en el mismo momento.

  Recuerda que el Titán en sí no ataca a los Progravengers, y este es derrotado cuando efectivamente le logran robar el contenido del **Tiempo**.

Notas

- Puede ser útil utilizar `Lock` cuando un Progravenger interactué con Threatan.
- Recuerda que puedes usar `is_alive()` para ver el estado de un *thread*.

Requerimientos

- (1.00 pts) Implementación correcta de método `run` en `Progravenger`
  - (0.70 pts) Diferenciación según objetivo
  - (0.30 pts) Implementación de `Lock`
- (0.50 pts) Implementación correcta de método `run` en `Threatan`.
  - (0.40 pts) Imprime estadísticas cada segundo
  - (0.10 pts) *Thread* termina al ser robado por un `Threavenger`
- (0.50 pts) Implementación correcta de métodos en `Batalla`
  - (0.25 pts) Instancia a `Threatan` y `Threavengers` en método `progravengers_assemble`
  - (0.25 pts) Inicia *threads* de `Threatan` y `Threavengers` en método `attack`

6. *Paths*, I/O y serialización (AC08 sumativa)

Para recuperar el contenido del **Alma**, debes encontrar un archivo corrupto, arreglarlo y deserializarlo:

6.1. `def buscar_contenido()`

Para eso, debes completar la función `buscar_contenido()` que te retornará el *path* (como *string*) donde se encuentra el archivo `contenido.alma`, la cual se encuentra en alguna sub-carpeta del directorio `Multiverso`.

6.2. `def obtener_contenido(path_alma)`

Se te acerca un personaje de extraña reputación, **Red Skullwirth**, y te entrega el siguiente mensaje: "El contenido del alma tiene cierta sabiduría, para conseguirlo, debes completar la siguiente función"

La función `obtener_contenido(path_alma)` debe recibir el *path* obtenido de la función anterior, es decir, el *path* del archivo `contenido.alma`, y desencriptarlo utilizando el siguiente algoritmo:

1. Leerlo el archivo completo en forma de *bytes*.
2. Deberá obtener pedazos de a 5 *bytes* del archivo y eliminar el máximo de ese *chunk* de *bytes*, ignorando el *chunk* que reste al final.
3. Debes sacrificar el último *byte* obtenido del total (es decir, eliminarlo del `bytearray`)

Terminado esto, debes escribir los *bytes* resultantes en un archivo nuevo llamado `desencriptado.alma`.

6.3. `def deserializar_contenido(path_serializado)`

Para obtener el contenido del infinito, deberás cargar la información del archivo generado en la función anterior mediante algún método de deserialización. Este archivo esta serializado en uno de los métodos de serialización visto en las clases de **Fernando Banner**, el cual debes identificar abriendo el archivo con tu editor de texto.

Primero deberás completar el método de deserialización que corresponda para el tipo de archivo generado (`__setstate__` u `object_hook`). Este método deberá filtrar los valores de los parámetros del objeto, y solo mantener aquellos parametros cuyos nombres están dentro de la tupla `params` (que es una constante entregada). Además, el método debe agregar el atributo `sacrificio` cuyo valor será un *string* con lo que usted sacrificó para esta misión (es decir, lo que pudo haber estado haciendo hoy en vez de la actividad)

Finalmente, completar la función `deserializar_contenido(path_serializado)` que retornará una instancia de la clase `ContenidoDelInfinito`, mediante alguno de los métodos de serialización descritos.

Notas

- Recuerda que un *byte* tiene un valor hexadecimal asociado.
- `bytearray` tiene un método `remove`.
- Para identificar cual de los dos métodos de serialización es, recuerde cual es legible y cual no.

- La librería `shutil` esta prohibida.

Requerimientos

- (0.4 pts) Completa la función `buscar_contenido()` correctamente.
- (0.8 pts) Completa la función `obtener_contenido(path_alma)` correctamente.
- (0.8 pts) Serialización
  - (0.4 pts) Completa `__setstate__` o `object_hook` según corresponda
  - (0.4 pts) Completa la función `deserializar_contenido(path_serializado)` correctamente.