Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2019-2

29 de agosto de 2019
Actividad Sumativa

Actividad 02

Estructuras de datos *built-ins*

Introducción

¡Una vez más el DCC se ve en peligro! La gran cantidad de alumnos y alumnas que inscribió el curso IIC2233 este semestre se dio cuenta que, debido a que sobrepasan de manera abismante en número a los ayudantes y han desarrollado impresionantes habilidades en Git, podrían distraer a los ayudantes de manera estratégica para llegar al computador del *Chief* Tamburini y obtener así las soluciones de todas las actividades y tareas que deberán realizar en el semestre. Es por esto que hoy se llevará acabo, por primera vez en la historia de la computación, la gran Batalla DCCampal entre alumnos y ayudantes. Estos últimos se encuentran resguardados en los distintos pisos de la DCCueva corrigiendo tareas y actividades sin parar y tu rol como programador y experto en estructuras básicas es realizar una simulación para ver si será posible ganar la batalla ~~destruyendo~~ distrayendo a los ayudantes.

Figura 1: Logo DCCampal

Batalla

Los contrincantes en la batalla son: ayudantes vs. alumnos. Por un lado, los ayudantes tienen un **nombre**, **rango** y **debilidades** por distintos tipos de comidas. Por otro lado, los alumnos tienen **nombre** y **habilidades** para cocinar distintos tipos de comidas, las que usarán para distraer a los ayudantes. Para poder obtener la victoria, los alumnos deben llegar al **Piso -4** de la DCCueva y lograr distraer al malvado ***Chief* Tamburini** para conseguir todas las soluciones.

Los ayudantes están separados según su rango y distribuidos en los distintos pisos de la DCCueva, de la siguiente forma:

- Nuevos - Piso -1
- Mentores - Piso -2

- Jefes - Piso -3
- *Chief* Tamburini - Piso -4

Programa

En la Batalla DCCampal, deberás desarrollar un programa que cree las entidades de los alumnos y ayudantes y luego simula la pelea. Estas entidades deben almacenarse apropiadamente en distintas estructuras de datos, de tal manera que puedan se accedidas de manera eficiente, con el fin de hacer tu simulación tan rápida como puedas y así saber exactamente si todos los ayudantes de la DCCueva serán distraídos o no.

Los alumnos tendrán la habilidad para cocinar distintos tipos de comidas que serán utilizadas para distraer a los ayudantes. Si un alumno logra distraer a un ayudante con un tipo de comida, este ya no podrá volver a cocinar esa comida, es decir, pierde esa habilidad. Uno a uno, se intentarán distraer ayudantes, piso por piso. Para poder bajar de un piso al siguiente, todos los ayudantes del piso anterior deben ser distraídos por alumnos, hasta llegar al último.

Archivos

Para la actividad tendrás que utilizar los siguientes archivos, indispensables para su desarrollo.

- `main.py`: En este archivo se encuentran todas la acciones propias de la Batalla DCCampal las cuales debes completar para poder simular las interacciones entre alumnos y ayudantes.
- `consultas.py`: En este archivo se encuentran dos consultas que debes completar para obtener información crucial para simular la batalla.
- `cargar_datos.py`: En este archivo deberás completar dos funciones, las que son necesarias para cargar los datos tanto de alumnos y ayudantes.
- `ayudantes.csv`: Esta es la base de datos de los ayudantes, un archivo separado por comas (*Comma Separated Values*). En este caso el separador entre nombre, rango y debilidades será el caracter `";"`, mientras que las debilidades se separan entre ellas con el caracter `","`. El formato de este archivo será:

  `nombre;rango;debilidad_1,debilidad_2,...,debilidad_n`

- `alumnos.csv`: Esta es la base de datos de los alumnos, un archivo separado por comas (*Comma Separated Values*). En este caso el separador entre el nombre y las habilidades será el caracter `";"`, en cambio las habilidades se separan entre ellas con el caracter `","`. El formato de este archivo será:

  `nombre;habilidad_1,habilidad_2,...,habilidad_n`

Se te entregan también los archivos `bonus.py` y `tech-keys.csv` junto a esta actividad, pero **NO** son indispensables para el desarrollo de esta actividad. Sus descripciones se detallan en la **Sección *Bonus***.

Poblar el sistema

En esta sección deberás cargar el contenido de los distintos archivos a tu programa. Para ello, tendrás que completar de manera adecuada las funciones que se presentan a continuación, las que se encuentran

en `cargar_datos.py`. El ***Chief* Tamburini** espera de ti una batalla digna, por lo que es importante que guardes la información de cada entidad de manera eficiente y acorde a lo pedido, por lo que tendrás que elegir sabiamente qué **estructuras de datos (*built-ins*)** de **Python** utilizar. **Está prohibido el uso de clases para la modelación del problema.**

- `def cargar_alumnos(ruta_archivo_alumnos):` Esta función lee el archivo `alumnos.csv` usando el argumento `ruta_archivo_alumnos` y carga su contenido con el fin de crear todos los alumnos como entidades. Los alumnos están ordenados según su ~~banner~~ número de alumno, de tal forma que los últimos en agregarse a la estructura de datos, **serán los primeros en atacar durante la simulación**. Esta función retorna la estructura de datos con todos los alumnos en su interior, y que luego será utilizada por la simulación para obtener los alumnos atacantes.
- `def cargar_ayudantes(ruta_archivo_ayudantes):` Esta función lee el archivo `ayudantes.csv` usando el argumento `ruta_archivo_ayudantes` y carga su contenido creando a los ayudantes como entidades. Cada grupo de ayudantes debe estar organizado según el piso en el que se encuentran. Los ayudantes de cada piso están ordenados de tal forma que **los primeros en agregarse a la estructura de datos, serán también los primeros en intentar defender la DCCueva**. Además de cargar la información del archivo, cada ayudante **debe tener un campo `comiendo` que inicialmente es una lista vacía**. Esta función retorna una estructura de datos donde se encuentran los ayudantes organizados por piso[1] y será usada luego por la simulación para obtener los ayudantes defensores de cada piso.

Es importante destacar que en el archivo `alumnos.csv` y `ayudantes.csv`, las habilidades y debilidades pueden estar repetidas debido a que nadie se ha preocupado de ir actualizando las bases de datos del DCC en el tiempo. Por lo tanto, para mayor eficiencia del programa **debes cargar esta información sin que se repita**. También, es importante considerar que las estructuras retornadas por las funciones anteriores serán utilizadas por la simulación **y se espera las entidades puedan ser removidas de ellas**, por lo que el número de entidades almacenadas variará durante la simulación.

Consultas

A modo de tener información relevante sobre la batalla durante la simulación, debes completar las siguientes funciones que se encuentran en el archivo `consultas.py`.

- `def resumen_actual(ayudantes, alumnos):` Esta función recibe como parametros las estructuras que contienen los ayudantes y los alumnos. Esta función es utilizada en la simulación para informar los números a ambos bandos de la batalla. Debe mostrar el número total de alumnos y ayudantes almacenados en las estructuras, y para el caso de los ayudantes, mostrar el detalle de número de ayudantes por piso. A continuación, se muestra un ejemplo del resultado impreso de una llamada a la función[2].

```
------------------------------------------------
Alumnos restantes: 100
Ayudantes restantes: 42
Ayudantes Piso -1: 25
Ayudantes Piso -2: 14
Ayudantes Piso -3: 2
Ayudantes Piso -4: 1
------------------------------------------------
```

[1] Recuerda que el piso en que se encuentre depende de su rango

[2] Este es solo un ejemplo, no debe ser exactamente el mismo resultado mientras se muestren los datos solicitados.

- `def stock_comida(alumnos):` Esta función recibe como parámetro la estructura de datos en la cual se encuentran los alumnos y debe retornar una lista de tuplas con los nombres de todas las comidas junto con el número de alumnos que tienen esa comida como habilidad. La lista retornada debe ser de la forma:

```
[("Pizza", 7),("Gohan", 19),...]
```

Acciones en DCCampal

Una vez creadas las entidades de ayudantes y alumnos, deberás liderar la gran **Batalla DCCampal** completando las siguientes funciones que se encuentran en `main.py`:

- `def distraer(alumno, ayudante):` Esta función recibe como parámetros un alumno y un ayudante (como entidades), y si es posible, realiza la acción de distraer a dicho ayudante. El ayudante será distraído si el alumno tiene como habilidad alguna de las debilidades del ayudante. El alumno al distraer a un ayudante pierde la habilidad con la que lo distrajo y se debe incluir la comida que se usó para distraer al campo `comiendo` del ayudante que se distrajo (esto quiere decir que el ayudante esta distraído comiendo). En el caso de que un alumno pueda distraer a un ayudante con más de una comida, debes elegir aleatoriamente[3] una comida para distraerlo (y que perderá el alumno). Finalmente, tu función debe retornar `True` en el caso de que logre distraer al ayudante y `False` en el caso contrario.
- `def simular_batalla(alumnos, ayudantes):` Para predecir si los alumnos saldrán victoriosos de DCCampal, debes completar esta función donde simularás la batalla que se producirá. **Se te entrega un esqueleto del flujo de la simulación, por lo que tu debes completarla utilizando las estructuras que escogiste para almacenar las distintas entidades**. La batalla se desarrolla de la siguiente manera:
  - La simulación avanza piso por piso, comenzando por el -1. Antes de avanzar al siguiente piso, se intenta distraer a todos los ayudantes del tal piso, y en caso de hacerlo se avanza de piso.
  - Los alumnos y ayudantes atacan y defienden uno a la vez según cierto orden, especificado en la **Sección Poblar el sistema**.
  - Un ayudante defiende hasta que es distraído por un alumno. De ser distraído, este se remueve y pasa el siguiente ayudante para defender.
  - Un alumno intentará distraer todos los ayudantes que pueda. En caso de encontrarse con un ayudante que no puede distraer, entonces debe irse a su casa (se remueve), y el siguiente alumno lo seguirá intentando.
  - En caso de que no queden alumnos para distraer o no se logre avanzar de piso, la simulación termina y se declaran ganadores los ayudantes.
  - Si se logra alcanzar el último piso y derrotar al *Chief* Tamburini, entonces **¡felicitaciones lograron obtener las soluciones de las actividades y tareas!**

[3] *Hint:* Te recomendamos usar el método `sample` de la librería `random`. Dada una secuencia de elementos `X`, podemos pedir que nos retorne una lista de `n` elementos. Para pedir un solo elemento, basta algo así: `elemento = random.sample(X, 1)[0]`

Notas (*hints*)

- **¡Cuidado!** Si utilizas solamente listas para resolver toda la actividad no obtendrás el puntaje completo en la sección correspondiente.

Bonus (5 décimas): *Teck-Keys*

IMPORTANTE: Se recomienda leer y comenzar esta sección SOLO en caso de terminar el resto de la actividad. Si no lo has hecho, por ahora sáltate esta sección y revisa la distribución de puntajes en Requerimientos.

El *Chief* Tamburini y el resto de los ayudantes se enteraron del plan de los alumnos de distraerlos para acceder a las evaluaciones, por lo que instalaron un sistema de seguridad en la DCCueva, inventado por el Dr. Robertriki. Este coloca seguros en las puertas entre cada piso, que para abrirlas, son necesarias las *Tech-Keys* correctas. Cada *Tech-Key* se construye a partir de una combinación secreta de gemas

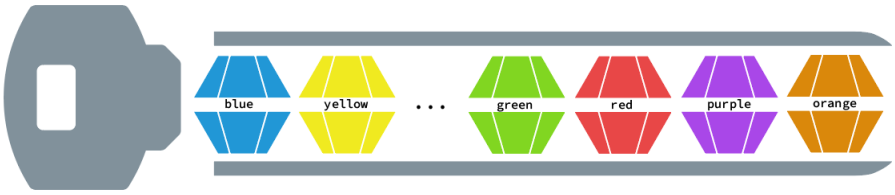

Figura 2: Ejemplo de una *Tech Key*

Esto agrega complejidad a la misión de los alumnos. Una vez que se distraen todos los ayudantes de un piso y se llega al final de este, además hay que utilizar las *Tech-Keys* para intentar desbloquear la puerta al próximo piso. De lograrlo, se pasa al siguiente piso. Afortunadamente para los alumnos, por el descuido del nuevo ayudante Coyuduki luego de la ayudantía de Git, uno de los alumnos logró conseguir las combinaciones correctas de gemas para formar las *Tech-Keys* de cada piso, y las compartió con todos sus compañeros.

Tu objetivo en este *bonus* es extender el programa considerando esta nueva restricción: Debes cargar las combinaciones de gemas para formar las *Tech-Key* de cada uno de los pisos, y utilizarlas para desbloquear cada piso luego de distraer todos los ayudantes. Debes completar dos funciones en el archivo `bonus.py`.

El archivo `tech_keys.csv` contiene las combinaciones de gemas (colores) necesarias para formar las *Tech-Key* que abrirán cada uno de los pisos. El número del piso con la llave está separada por el caracter `";"`, y las diferentes combinaciones de gemas están separadas por el caracter `","`. El formato de este archivo sería: `piso-X;color_1,color_2,...,color_3,color_4` y se carga mediante la función `cargar_llaves`.

`def cargar_llaves(ruta_archivo_llaves):` Está función lee el archivo `tech_keys.csv` y carga en una estructura de datos las combinaciones de gemas para formar las *Tech-Keys* a modo de acceder a cada piso. Es importante señalar que el orden en que se encuentran las combinaciones de gemas en el archivo `tech_keys.csv` no debe perderse al momento de agregarlas a la estructura de datos respectiva, debido a que si el orden no es el mismo que el que solicita la compuerta no podrán acceder el piso. Esta función debe retornar una estructura con todas las llaves, que será utilizado luego por la simulación.

`def desbloquear_pisos(llaves, piso):` Para poder desbloquear los pisos y seguir en tu lucha por conseguir las soluciones, debes completar esta función la que recibe como parámetros la estructura de datos que contiene las *Tech-Keys* que cargaste en la función `cargar_llaves` y el piso que quieres desbloquear.

Debes realizar la comparación de las *Tech-Keys* que cargaste en tu estructura de datos y las que te entregamos al principio de `bonus.py`. En el caso de que las combinaciones de gemas sean las mismas debes retornar `True` e imprimir un mensaje señalando que lograste desbloquear la puerta.

Finalmente, debes alterar `main.py` para incluir tu implementación de estas funciones y agregar el comportamiento de las llaves en la simulación.

Requerimientos

- (3.00 pts) Poblar el sistema
  - (1.5 pts) `cargar_alumnos`.
    - (0.5 pts) Se cargan correctamente a todos los alumnos del archivo `alumnos.csv` como entidades.
    - (0.5 pts) Las habilidades de cada alumno se guardan en la estructura adecuada.
    - (0.5 pts) Retorna la estructura de datos adecuada.
  - (1.5 pts) `cargar_ayudantes`.
    - (0.5 pts) Se cargan correctamente a todos los ayudantes del archivo `ayudantes.csv` como entidades.
    - (0.5 pts) Las debilidades de cada ayudante se guardan en la estructura adecuada.
    - (0.5 pts) Retorna la estructura de datos adecuada.
- (1.50 pts) Consultas
  - (0.75 pt) `resumen_actual` imprime correctamente la información pedida.
  - (0.75 pt) `stock_comida` retorna correctamente una lista de tuplas.
- (1.50 pts) Acciones en DCCampal
  - (1.0 pt) `distraer`.
    - (0.2 pts) Obtiene correctamente las habilidades del alumno.
    - (0.2 pts) Obtiene correctamente las debilidades del ayudante.
    - (0.6 pts) Revisa correctamente si el ayudante puede ser distraído por el alumno.
  - (0.5 pts) `simular_batalla` logra completar esqueleto de simulación.
- (0.50 pts) *Bonus*: *Teck-Keys*
  - (0.2 pts) `cargar_llaves`.
    - (0.1 pts) Las *Tech-Keys* se guardan en la estructura de datos adecuada por piso.
    - (0.1 pts) Retorna la estructura de datos adecuada.
  - (0.1 pts) `desbloquear_pisos` compara correctamente las *Tech-Keys*.
  - (0.2 pts) Alterar simulación en `main.py`

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AC02/`
- **Hora del *push*:** 16:30