



Actividad Formativa 02

OOP II

Antes de comenzar...

Para esta, y todas las actividades del semestre, como equipo docente esperamos que sigas el siguiente flujo de trabajo:

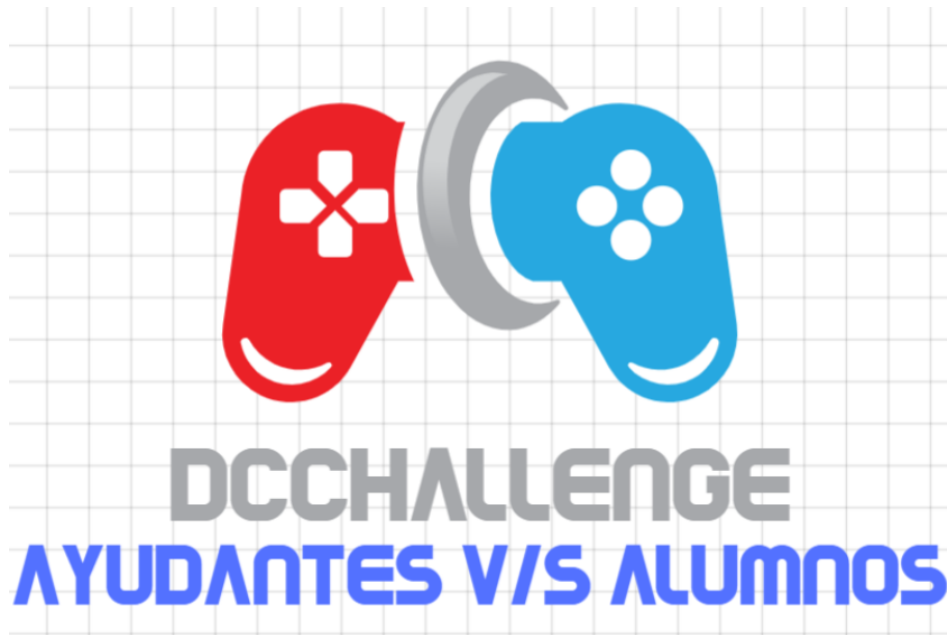
- Lee el enunciado completo, incluyendo las notas. Puedes revisar los archivos subidos a medida que lees, o al final, como te acomode.
- Antes de comenzar a programar, copia y pega todos los archivos de la carpeta AF02 del *Syllabus* y pégalos en la misma carpeta de **tu repositorio personal**.
- Haz `git add`, `git commit` y `git push` de los archivos copiados inmediatamente, para comprobar que el uso de Git esté funcionando correctamente. (**Tip:** Si no recuerdas como utilizar estos comandos, puedes revisar el enunciado de la AC00)
- En caso de encontrar un error, contacta a un ayudante para resolver el problema lo antes posible, en caso de no hacerlo, tu actividad podría no entregarse correctamente.
- Comienza a trabajar en tu actividad en tu repositorio y recuerda subir cada vez que logres un avance significativo (con los mismos comandos de Git de antes).
- Todas las actividades y tareas tienen una fecha y hora de entrega, en la cual automáticamente se recolecta el último *commit* **pusheado** en tu repositorio. Esto no quiere decir que solo se consideran los cambios de ese último *commit*, si no que todos los avances **hasta** ese *commit*. Luego, es importante realices `git push` de todos tus avances, antes de la fecha y hora de entrega. En este caso, es a las 16:50 de hoy.

Introducción

En el DCC cada ciertos semestres, se revive un gran superclásico, las “Olimpiadas DCC”. En su versión anterior, se hizo competir a los alumnos contra los ayudantes en diversas disciplinas deportivas, con la finalidad de poder determinar cual de ambos equipos era el mejor. Este semestre, debido a la contingencia nacional, las disciplinas deportivas no son una opción, así que se definió optar por juegos que propiciaran el hecho de quedarse en casa. (**¡Sé responsable, haz cuarentena!**).

Como el programa planificado quedó obsoleto, necesitamos manos que sean hábiles programando. ¡Te encomendaremos a ti la misión de darle vida al DCCchallenge!

Tu deber será apoyar a la **modelación de entidades** que de vida al Superclásico DCC. Este programa carga dos equipos: ayudantes y alumnos(as); y los hace enfrentarse en cuatro rondas de juegos. Cada ronda corresponde a un tipo de juego distinto: **juegos de cartas, de mesa, de combate y de carreras**.



Jugadores

Casi todo el programa de DCChallenge está implementado, excepto las clases de jugadores que participan en los juegos, y eso debes completar. Todos los participantes, tanto alumnos(as) como ayudantes, comparten una **clase base abstracta** Jugador. Esta se encuentra en el archivo `jugadores.py`. Todas las instancias tienen los siguientes atributos internos:

- `self.nombre (str)`: Representa el nombre del jugador.
- `self.equipo (str)`: Representa si es del equipo de ayudantes o de alumnos.
- `self.especialidad (str)`: Representa la especialidad de jugador: que puede ser cartas, juegos de mesa, de combate, de carreras, jugador inteligente o jugador intrépido.
- `self.energia (int)`: Representa la energía que posee el jugador.
- `self.inteligencia (int)`: Representa la inteligencia del jugador.
- `self.audacia (int)`: Representa la audacia del jugador.
- `self.trampa (int)`: Representa la habilidad de hacer trampa del jugador.
- `self.nerviosismo (int)`: Representa los nervios que tiene el jugador.

Además implementan los métodos:

- `def __str__(self)`: Este método será común para todas las especializaciones de jugadores y **se entrega implementado**. Tiene como propósito retornar un *string* que representa amigablemente al jugador. Algunos ejemplos son: "Alumno(a) Juan (cartas)" y "Ayudante Daniela (mesa)"
- `def __repr__(self)`: Este método será común para todas las especializaciones de jugadores y **se entrega implementado**. Tiene como propósito retornar un *string* que representa detalladamente al jugador exponiendo todos sus atributos listados anteriormente.
- `def enfrentar(self, tipo_de_juego, enemigo)`: Este es un método **abstracto** que deben implementarlo todas las especializaciones de jugadores y **debes completarlo**. Recibe `tipo_de_juego (str)`, que es un juego que siempre podrá jugar el jugador correspondiente, y `enemigo` (instancia

de Jugador) que es contra quien se enfrenta. Se produce un enfrentamiento entre la instancia de jugador que desafía a `enemigo`. Primero, imprime en pantalla un mensaje que indica que el jugador desafía a `enemigo` a un juego de `tipo_de_juego`. Por ejemplo:

```
1 "Ayudante Pedro (mesa): ¡Desafio a Alumno(a) Juan (mesa) a un juego de mesa!"
```

Luego se ejecuta el método correspondiente al juego, dependiendo de la especialización del jugador, donde se decide el ganador del juego. En cada juego varía la forma de decidir su ganador (especificado más adelante). Este método (`enfrentar`) debe retornar un *boolean* (`True` o `False`) que indica si el jugador gana o no el enfrentamiento.

Especializaciones

Cada participante tiene una especialidad en cierto tipo de juego, lo cual especifica ciertos atributos y define comportamiento diferenciado. **Estas especializaciones se deben modelar mediante clases que hereden de Jugador**, cuyas bases también puedes encontrar en el archivo `jugadores.py`.

A continuación detallamos el comportamiento de las clases que encontrarás en el archivo `jugadores.py`, a las cuales les deberás completar el método `__init__` y los métodos según cada caso.

Ten en cuenta que las clases son instanciadas en el archivo `main.py` (ya implementado), utilizando una función en `cargado.py` desde un archivo CSV, y a cada una se le entregará los argumentos: `nombre` (`str`), `equipo` (`str`), `especialidad` (`str`) y `energia` (`str`), en ese orden. Estos atributos se deben asignar manteniendo los tipos provenientes de los archivos (*string*), con la excepción de `energia` que en realidad es un `int`. Los otros atributos de `Jugador` tendrás que definirlos según las especificaciones que se muestran a continuación.

Al mismo tiempo, cada especialización contiene al menos un método propio específico para ejecutar un juego contra un enemigo y determinar quien gana. Se debe hacer uso de estos métodos específicos para la implementación de `def enfrentar` que se mencionó antes.

Las clases de especialización de `Jugador` son las siguientes:

- `class JugadorMesa`: Esta clase representa a los jugadores que son especialistas en los juegos de mesa. Los cuales poseen una audacia, una habilidad de trampa e inteligencia equivalente a 3. Mientras que su nerviosismo se calcula como:

```
1 self.nerviosismo = min(self.energia, random.randint(0, 3))
```

Adicionalmente, este jugador posee el método `def jugar_mesa(self, enemigo)` y que ejecuta cuando se enfrenta a alguien. Este recibe la instancia del jugador al cual se está enfrentando para revisar su atributo `nerviosismo`. Si el nerviosismo del enemigo es mayor que el del jugador el método debe retornar `True`, indicando que ganó, y en caso contrario `False`.

- `class JugadorCartas`: Esta clase representa a los jugadores que son especialistas en los juegos de cartas. Los cuales poseen una audacia, una habilidad de trampa y nerviosismo equivalente a 3. Mientras que su inteligencia se calcula como:

```
1 self.inteligencia = self.energia * 2.5
```

Adicionalmente, este jugador posee el método `def jugar_cartas(self, enemigo)` y que ejecuta cuando se enfrenta a alguien. Este recibe la instancia del jugador al cual se está enfrentando, para revisar su atributo `inteligencia`. Si la inteligencia del enemigo es menor que la del jugador la función debe retornar `True`, indicando que ganó, y en caso contrario `False`.

- `class JugadorCombate`: Esta clase representa a los jugadores que son especialistas en los juegos de combate. Los cuales poseen una inteligencia, una habilidad de trampa y nerviosismo equivalente a 3. Mientras que su audacia se calcula como:

```
1 self.audacia = max(self.energia, random.randint(3, 5))
```

Adicionalmente, este jugador posee el método `def jugar_combate(self, enemigo)` y que ejecuta cuando se enfrenta a alguien. Este recibe la instancia del jugador al cual se está enfrentando para revisar su atributo `audacia`. Si la audacia del enemigo es menor que la del jugador la función debe retornar `True`, indicando que ganó, y en caso contrario `False`.

- `class JugadorCarreras`: Esta clase representa a los jugadores que son especialistas en los juegos de carreras. Los cuales poseen una audacia, inteligencia y nerviosismo equivalente a 3. Mientras que su habilidad de trampa se calcula como:

```
1 self.trampa = self.energia * 3
```

Adicionalmente, este jugador posee el método `def jugar_carrera(self, enemigo)` y que ejecuta cuando se enfrenta a alguien. Este recibe la instancia del jugador al cual se está enfrentando para revisar su atributo `trampa`. Si la trampa del enemigo es menor que la del jugador la función debe retornar `True`, indicando que ganó, y en caso contrario `False`.

- `class JugadorInteligente`: Esta clase representa a los jugadores que son especialistas tanto en los juegos de mesa como en los juegos de cartas. Poseen una audacia y trampa equivalente a 3, mientras que su nerviosismo se comporta al igual que un `JugadorMesa` y su inteligencia igual a un `JugadorCartas`. Adicionalmente, este jugador posee los métodos del `JugadorMesa` y del `JugadorCartas`, que dependiendo del enfrentamiento utilizarán el método que corresponda.
- `class JugadorIntrepido`: Esta clase representa a los jugadores que son especialistas tanto en los juegos de carreras como en los juegos de combate. Poseen un nerviosismo e inteligencia equivalente a 3, mientras que su audacia se comporta al igual que un `JugadorCombate` y su trampa igual a un `JugadorCarreras`. Adicionalmente, este jugador posee los métodos del `JugadorCombate` y del `JugadorCarreras`, que dependiendo del enfrentamiento utilizarán el método que corresponda.

Combate

Como mencionamos, el resto del programa está implementado, por lo que solo debes completar el archivo `jugadores.py`. Los archivos `main.py`, `cargado.py` y `combate.py` asumen que se completó la implementación especificada para los jugadores. A continuación se detalla el orden de ejecución del programa completo:

- El archivo principal de ejecución es `main.py`. Este comienza cargando los jugadores utilizando la función `cargar_jugadores` de `cargado.py`.
- La función `cargar_jugadores` crea dos diccionarios para almacenar las instancias de jugadores de tipo ayudante y tipo alumno, separados por tipo de juego que juega cada uno. **Hace uso de las clases definidas en `jugadores.py` para instanciar los objetos** y luego clasificarlos a partir de sus atributos. También revisa que las instancias tengan los atributos especificados.
- Luego, en `main.py` se ejecuta la función `imprimir Equipos` que imprime todas las instancias creadas en el paso anterior. **Puedes verificar que los valores de los atributos de cada instancia correspondan en esta parte.**

- Luego, en `main.py`, se inicializa el puntaje de cada equipo en 0, y se hacen cuatro llamadas a la función `jugar` (definida `combate.py`) que simula una ronda de DCChallenge.
- La función `jugar` imprime los puntajes actuales de cada equipo, y luego elige a un jugador al azar de cada equipo que sea capaz de jugar el tipo de juego de la ronda. **Se hace uso del método `enfrentar de una de las instancias`** y a partir de su resultado se determina quien gana el juego para aumentar el puntaje del equipo correspondiente.
- Finalmente, en `main.py`, se llama a la función `anunciar = _ganadores`, que imprime el equipo ganador a partir de su puntaje.

Notas

- En caso de empate en un enfrentamiento se asume que ambos jugadores pierden, por lo que el método debe retornar `False`.
- Se dejaron comentarios en el código implementado para explicar que se realiza en cada paso.

Requerimientos

Este puntaje es solo referencial. Al ser una actividad formativa, no hay nota directa asociada.

- (6.00 pts) Creación de las clases
 - (1.00 pts) Clase `JugadorMesa` bien implementada.
 - (0.50 pts) implementar correctamente el `__init__`.
 - (0.50 pts) implementar correctamente los métodos.
 - (1.00 pts) Clase `JugadorCartas` bien implementada.
 - (0.50 pts) implementar correctamente el `__init__`.
 - (0.50 pts) implementar correctamente los métodos.
 - (1.00 pts) Clase `JugadorCombate` bien implementada.
 - (0.50 pts) implementar correctamente el `__init__`.
 - (0.50 pts) implementar correctamente los métodos.
 - (1.00 pts) Clase `JugadorCarrera` bien implementada.
 - (0.50 pts) implementar correctamente el `__init__`.
 - (0.50 pts) implementar correctamente los métodos.
 - (1.00 pts) Clase `JugadorIntrepido` bien implementada.
 - (1.00 pts) Clase `JugadorInteligente` bien implementada.

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AF02/`
- **Hora del *push*:** 16:50