Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2022-2

15 de Septiembre de 2022
Actividad Sumativa

Actividad Sumativa 2

Threading

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AS2/`
- **Hora del *push*:** 16:40

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

Bienvenido a DCClash of Clans, donde ¡tu aldea está siendo atacada! Uno de tus P.E.K.K.A. se ha rebelado y ha decidido atentar contra tu aldea, tu misión como buen estudiante de programación es crear bárbaros lo más rápido posible para enfrentarte al malvado P.E.K.K.A. y protegerte de sus ataques. Deberás utilizar tus conocimientos de *Threading* para recolectar recursos, construir chozas y entrenar bárbaros, para luego defenderte del ataque nocturno del P.E.K.K.A.

Figura 1: DCClash of Clans

Flujo del programa

El programa consiste en una simulación de un ataque a tu aldea, la cual se divide entre el día y la noche. El flujo del ataque comienza al ejecutar el archivo `main.py`, así se inician los *threads* de las clases `Recolector` y `Constructor` y se da inicio a un nuevo día. Durante el día, los recolectores empiezan a

recolectar monedas de oro, las cuales van depositando en el centro urbano, esto lo harán hasta que se les acabe la energía. En conjunto, trabajan los constructores, estos retirarán monedas del centro urbano para poder construir chozas. Al empezar la noche, las chozas generarán tropas de bárbaros que se enfrentarán al malévolo P.E.K.K.A para poder defender la aldea. En el caso de que no se logre vencer al enemigo esa noche, empezará un nuevo día para volver a prepararse para el ataque. Finalmente, una vez que el P.E.K.K.A haya sido derrotado, tu aldea se salva y se termina el programa.

Importante: Dado que hay varias clases en el programa, **te recomendamos que leas todo el enunciado antes de comenzar a programar.**

Archivos

En el directorio de la actividad encontrarás los siguientes archivos:

Código

- Modificar `trabajadores.py`: Contiene la clase `Recolector` y `Constructor`.
- Modificar `centro_urbano.py`: Contiene la clase `CentroUrbano`.
- Modificar `simulacion.py`: Contiene la clase `Pekka` y se encarga de simular el día y la noche del ataque a la aldea.
- No modificar `main.py`: Este es el archivo principal del programa. Puedes ejecutarlo para probar la simulación.
- No modificar `parametros.py`: Este es el archivo que contiene los parámetros del programa.

Parte 1: Modelación del Centro Urbano 🏛️

En esta parte deberás completar la clase `CentroUrbano` del archivo `centro_urbano.py`, la cual representa la estructura principal para almacenar los recursos y producir los bárbaros.

Centro Urbano

- Modificar `class CentroUrbano`:
  - Modificar `def __init__(self) -> None`: Inicializador de la clase `CentroUrbano`. Contiene la cantidad de oro y chozas. Debes crear los atributos pertinentes para asegurar que los atributos de oro y chozas **no pueden ser accedidos por distinto *threads* a la vez**. Además, la clase posee los siguientes atributos:
    - `self.oro`: `str` que corresponde al oro del centro urbano. Este debe partir con el valor determinado `ORO_INICIAL`.
    - `self.chozas`: `int` que corresponde a la cantidad de chozas del centro urbano. Parte inicialmente con 0 chozas.
  - No modificar `def barbaros(self) -> int`: Esta `property` retorna la cantidad de bárbaros que puede producir el centro urbano según la cantidad de chozas.

Parte 2: Modelación de los trabajadores ⚒️

En esta parte deberás completar las clases `Recolector` y `Constructor` del archivo `trabajadores.py`, las cuales representan a los encargados de recolectar monedas de oro y construir las chozas.

Recolector

- Modificar `class Recolector`: Esta clase debe heredar de `Thread`. Se encarga de modelar a los recolectores que recogen monedas de oro y luego las ingresan al centro urbano.
  - Modificar `def __init__(self, nombre: str, centro_urbano: CentroUrbano) -> None`: Inicializador de la clase `Recolector`, recibe un nombre y el centro urbano correspondiente. En caso que la simulación finalice de forma abrupta, se requiere que este `Thread` también finalice. Por lo tanto, **debes indicar que este *thread* es *daemon***. Además, la clase posee los siguientes atributos:
    - `self.nombre`: `str` que corresponde al nombre del recolector.
    - `self.energia: int` que indica la energía disponible del recolector.
    - `self.oro: int` que indica el oro acumulado por recolector.
    - `self.centro_urbano: CentroUrbano` corresponde al centro urbano donde se ingresará el oro recolectado.
  - No modificar `def run(self) -> None`: Este método ejecuta el *thread* de la rutina del recolector. Se debe llamar a los métodos `trabajar`, `ingresar_oro` y `dormir`, en ese orden.
  - No modificar `def log(self, mensaje: str) --> None`: Este método se encarga de imprimir el siguiente string: `f"El recolector {self.nombre}: {mensaje}"`.
  - Modificar `def trabajar(self) -> None`: Debes usar el método `self.log()` para imprimir un mensaje que indique que el recolector ha comenzado a trabajar. Luego, mientras el recolector tenga energía disponible, se extrae `ORO_RECOLECTADO` unidades de oro, sumando este valor a `self.oro`. Cada vez que se extrae oro, se debe utilizar el método `self.log()` para indicar que se recolectó `ORO_RECOLECTADO` monedas de oro. Luego, debes restarle 1 a la energía disponible del recolector. Finalmente, para simular el tiempo de recolección, el recolector debe esperar `TIEMPO_RECOLECCION` unidades de tiempo.
  - Modificar `def ingresar_oro(self) -> None`: Este método se encarga de ingresar el oro al centro urbano. Primero, debes incrementar el oro en `centro_urbano` en `self.oro` unidades. Luego, debes hacer que el oro acumulado por el recolector sea igual a 0, para reportar 2 mensajes más con el método `self.log()`. El primero debe indicar que el recolector depositó el oro, y el segundo debe indicar la cantidad existente de oro en el centro urbano.

    Debes asegurarte de que sólo un recolector a la vez pueda ingresar oro al centro urbano.
  - No modificar `def dormir(self) -> None`: Este método se encarga de mandar a dormir al recolector. Se utiliza el método `self.log()` para indicar que el recolector se ha ido a dormir[1].

[1] Para siempre.

Constructor

- Modificar `class Constructor`: Esta clase hereda de `Thread`. Se encarga de crear las chozas donde viven los bárbaros.

  - Modificar `def __init__(self, nombre: str, centro_urbano: CentroUrbano) -> None`: Inicializador de la clase `Constructor`, recibe el nombre y el centro urbano correspondiente. En caso que la simulación finalice de forma abrupta, se requiere que este `Thread` también finalice. Por lo tanto, **debes indicar que este *thread* es *daemon***.

    Además, la clase posee los siguientes atributos:

    - `self.nombre`: `str` que corresponde al nombre del recolector.
    - `self.centro_urbano: CentroUrbano` que corresponde al centro urbano donde se extrae el oro y construyen las chozas.

  - No modificar `def run(self) -> None`: Este método ejecuta la rutina de trabajo del constructor. En esta rutina, mientras el constructor pueda retirar oro mediante el método `self.retirar_oro`, utilizará el método `self.log` para indicar que está construyendo una choza. Luego, se espera una cantidad de tiempo `TIEMPO_CONSTRUCCION` y finalmente, construye la choza mediante el método `self.construir_choza()`. Cuando ya no pueda continuar con esta rutina, se utilizará el método `self.log` para imprimir un mensaje indicando que el constructor ha finalizado el trabajo de este día.

  - No modificar `def log(self, mensaje: str) --> None`: Este método se encarga de imprimir el siguiente string: `f"El constructor {self.nombre}: {mensaje}"`.

  - Modificar `def retirar_oro(self) -> bool`: Este método se encarga de verificar que el centro urbano tenga `ORO_CHOZA` o más unidades de oro disponible.

    - En caso de tener disponible esa cantidad, se retira `ORO_CHOZA` de oro del centro urbano. Luego, se usa el método `self.log` para indicar la cantidad restante en el centro urbano. Finalmente, se retorna `True` para indicar que se retiró oro exitosamente.
    - En caso de no tener dicha cantidad, se usa el método `self.log` para indicar que el constructor no fue capaz de encontrar suficiente oro en el centro urbano. Luego, se retorna `False` para indicar que no se puede retirar oro.

    Recuerda que debes asegurarte que sólo un *thread* acceda al oro del centro urbano a la vez.

  - Modificar `def construir_choza(self) -> None`: Este método se encarga de crear las chozas. Debe sumar 1 a la cantidad de chozas en el centro urbano. Luego, utiliza el método `self.log` para indicar que agregó una choza y el número total de chozas.

    Recuerda que debes asegurarte que sólo un constructor pueda construir una chozas a la vez.

Parte 3: La batalla contra el P.E.K.K.A. ⚔️

En esta parte deberás completar la clase `Simulación` del archivo `simulacion.py`, la cual simula la batalla contra el P.E.K.K.A.

P.E.K.K.A.

- No modificar `class Pekka`:
  - No modificar `def __init__(self, centro_urbano: CentroUrbano) --> None`:
    Inicializador de la clase Pekka, recibe el centro urbano correspondiente y establece los atributos del P.E.K.K.A.
  - No modificar `def recuperar_vida(self) --> None`:
    Método que incrementa la vida del P.E.K.K.A. en `RECUPERACION_VIDA_PEKKA` unidades. Luego imprime cuanta vida recuperó y la vida total.

Simulación

- Modificar `class Simulacion`:
  - No modificar `def __init__(self) --> None`: Inicializador de la clase Simulacion. Se encarga de instanciar el centro urbano y el P.E.K.K.A.
  - Modificar `def nuevo_dia(self) --> None`: Método que ejecuta el día de la simulación. Este debe instanciar 2 recolectores y 2 trabajadores, y debe hacer que estos empiecen a ejecutar sus hilos de ejecución de forma paralela. **El método debe terminar cuando los *threads* hayan terminado su ejecución**. Finalmente, se debe imprimir un mensaje que indique que ha terminado el día.
  - No modificar `def nueva_noche(self)--> None`: Método que ejecuta la noche de la simulación. Se encarga de simular el combate entre bárbaros y el P.E.K.K.A.
  - No modificar `def iniciar(self)--> None`: Este método se encarga de dar inicio a la simulación. Mientras el P.E.K.K.A. tenga vida positiva, debes llamar a los métodos `nuevo_dia` y `nueva_noche`, en ese orden.

Notas

- La recolección de la actividad se hará en la rama principal (main) de tu repositorio.
- Recuerda verificar que **no** estés trabajando en el Syllabus. Debes hacer la entrega en tu repositorio personal.
- Se recomienda completar la actividad en el orden del enunciado.
- Si aparece un error inesperado, ¡léelo! Intenta interpretarlo.
- Siéntete libre de agregar nuevos `print` en cualquier lugar de tu código para encontrar errores. Es una herramienta muy útil.
- Puedes probar tu código ejecutando el archivo `main.py`.
- Recuerda hacer `git pull` antes de comenzar a trabajar para asegurarte de que tu repositorio esté sincronizado con GitHub.

Requerimientos

- (0.50 pts) Modelación de Centro Urbano
  - (0.50 pts) Modelar correctamente la Clase Centro Urbano.
- (3.00 pts) Modelación de Trabajadores
  - (1.00 pts) Modelar correctamente la Clase Recolector.
  - (2.00 pts) Modelar correctamente la Clase Constructor.
- (1.50 pts) Modelación de Simulación
  - (1.50 pts) Implementa correctamente el método `nuevo_dia`.
- (1.0 pts) Manejo de concurrencia
  - (0.5 pts) Maneja correctamente la concurrencia de la simulación para el manejo del oro.
  - (0.5 pts) Maneja correctamente la concurrencia de la simulación para la construcción de chozas.