Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2023-1

20 de Abril de 2023
Actividad

Actividad 2

Threading

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/A2/`
- **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

Al ver cómo en el reino vecino estaban haciendo explotar a pobres tortugas, el benévolo rey **Joaking** decide acudir a su rescate, guiando a sus mejores soldados al reino tortuga para llevarse a todas las sobrevivientes.

Sin embargo, luego de llegar al reino y conversar la situación con su gobernadora **Lily416**, se da cuenta que todo fue un malentendido y las tortugas estaban protegiendo al reino del malvado **Dr. Pinto**, por lo que no necesitan ser rescatadas y, para no desperdiciar el viaje, los monarcas deciden tener una amigable competencia del deporte nacional del reino tortuga, *DCCaptura la tortuga*.

Flujo del programa

El programa consiste en una competencia entre dos equipos que correrán tres carreras de 100 metros, donde a mitad de camino se encuentra una tortuga que debe ser capturada y llevada hasta la meta.

El primer competidor en llegar a los 50 metros captura la tortuga y se ve obligado a correr más cuidadosamente, bajando su velocidad. Si el competidor que no lleva la tortuga alcanza a su rival, intenta robarle la tortuga. El primer corredor que pase la meta con la tortuga en su posesión es el ganador de la carrera, dándole un punto a su equipo. El equipo con más puntos luego de las tres carreras es el ganador de la competencia.

Archivos

Archivos de código

En el directorio de la actividad encontrarás los siguientes archivos con código:

- Modificar `carrera.py`: Aquí encontrarás la definición básica de las clases que debes implementar y completar, junto con un pequeño main que simula una carrera.
- No modificar `main.py`: Aquí encontrarás la instanciación de las distintas entidades y la simulación de las 3 carreras.
- No modificar `test_corredor.py`: Este archivo contiene los *tests* que podrás utilizar para ir viendo si lo desarrollado hasta el momento cumple con lo pedido para los corredores.
- No modificar `test_carrera.py`: Este archivo contiene los *tests* que podrás utilizar para ir viendo si lo desarrollado hasta el momento cumple con lo pedido en la carrera.
- No modificar `test.py`: Este archivo ejecuta todos los *tests* disponibles, es decir, los *tests* de ambos archivos entregados.

Entidades

Modificar `class Corredor`:
Thread que representa a un competidor de la carrera. Este *thread* debe terminar su ejecución en caso que el programa principal finalice. Además, posee la variable de clase `TIEMPO_ESPERA`, `PORCENTAJE_MIN`, `PORCENTAJE_MAX`, `PROBABILIDAD_ROBAR`, e incluye los siguientes métodos:

- Modificar `def __init__() -> None`:
  Inicializador de la clase. Debe asignar los siguientes atributos:

`self.name`	Un `str` que representa el nombre de la persona.
`self.lock_tortuga`	Un `Lock` que representa la instancia de tortuga que debe capturar.
`self.senal_inicio`	Un `Event` que representa señal de inicio de la carrera.
`self.senal_fin`	Un `Event` que representa señal de término de la carrera.
`self.lock_carrera`	Un `Lock` que asegura que solo 1 corredor modifique el estado de la carrera.
`self.tiene_tortuga`	Un `bool` que indica si el competidor tiene la tortuga.
`self.__posicion`	Un `int` que representa la posición del jugador en la carrera, en metros. Es una atributo privado que tiene un valor de 0 por defecto, y cuyo valor no puede superar los 100 metros.
`self.__velocidad`	Un `int` que representa cuánto avanza el corredor por segundo de simulación.
`self.__correr`	Un `bool` que indica si el corredor debe continuar corriendo en la segunda mitad de la carrera.
`self.daemon`	Modificar Un `bool` que determina si el thread es daemon.

- No modificar `def posicion(self) -> int`:
  Property encargada de manejar la posición del corredor.
- No modificar `def velocidad(self) -> int`:
  Property encargada de manejar la velocidad del corredor. La velocidad baja a un 50 % si el corredor lleva una tortuga.

- No modificar `def asignar_rival(self, funcion_notificacion) -> None:`
  Método encargado de guardar una referencia al método que notifica al rival sobre el robo de la tortuga.

- No modificar `def ser_notificado_por_robo(self) -> None:`
  Método encargado de **notificar al corredor actual** que se le robó la tortuga.

- No modificar `def notificar_robo(self) -> None:`
  Método encargado de **notificar al rival** que se le robó la tortuga. Este método se define dentro de `asignar_rival`.

- Modificar `def avanzar(self) -> None:`
  Método que simula el movimiento del corredor. Debe aumentar la posición en una cantidad aleatoria entre el `PORCENTAJE_MIN` % y `PORCENTAJE_MAX` % de la velocidad del jugador, luego imprimir la nueva posición del jugador y, finalmente, duerme el *thread* por `TIEMPO_ESPERA` segundos.
  Hint: Para obtener la parte aleatoria del movimiento puedes utilizar el método `randint` de la librería `random` y luego dividir por 100 para convertir los enteros en porcentajes.

- Modificar `def intentar_capturar_tortuga(self) -> None:`
  Método que simula el intento de capturar la tortuga. Debe intentar adquirir el `lock` de la tortuga, pero **NO** debe quedarse esperando a que se libere el `lock` si no logra obtenerlo. En caso de obtener el `lock`, debe actualizar el atributo `tiene_tortuga` e imprimir un mensaje avisando que se logró la captura.
  Hint 1: El método `acquire()` de la clase `Lock` recibe como parámetro el booleano *blocking*, que le indica si debe quedarse esperando a que se libere el lock (`True`) o si debe intentarlo solo una vez y seguir su ejecución (`False`).
  Hint 2: El método `acquire()` de la clase `Lock` retornar `True` si el *lock* fue adquirido correctamente y `False` en caso contrario.
  Hint 3: Para más información sobre `Lock` y `acquire`, puedes revisar la documentación de `Lock`.

- Modificar `def perder_tortuga(self) -> None:`
  Método que simula la pérdida de la tortuga si esta es robada. El corredor debe actualizar su atributo `tiene_tortuga`, liberar el `lock` de la tortuga, e imprimir un mensaje informando que perdió la tortuga.

- Modificar `def robar_tortuga(self) -> bool:`
  Método que simula el intento de robar la tortuga. Debe lograr robar la tortuga con una probabilidad de `PROBABILIDAD_ROBAR` %. Si se logra el robo:

  - Se debe notificar al rival que perdió la tortuga.
  - El corredor debe obtener el `lock`, asegurándose de que el *lock* se obtiene correctamente.
  - El corredor debe actualizar su atributo `tiene_tortuga`.
  - Se debe imprimir un mensaje informando el resultado del robo y se debe retornar `True`.

  Si falla el robo:

- Se debe retornar `False`.

- Modificar `def correr_primera_mitad(self) -> None:`
  Método encargado de simular la primera mitad de la carrera. Mientras el corredor esté a menos de 50 metros de la meta, el corredor simplemente se dedica a avanzar.

- Modificar `def correr_segunda_mitad(self) -> bool:`
  Método encargado de simular la segunda mitad de la carrera. Mientras el corredor deba correr (`self.__correr == True`), primero se debe adquirir el `lock_carrera`, si este ya está tomado, el corredor tendrá que esperar que este sea liberado.

  Mientras tenga el `lock_carrera` deberá verificar:

  1. Si la señal de fin de la carrera fue levantada: debe retornar `False`.
  2. Si la posición del corredor es mayor o igual a 100 y tiene la tortuga: debe avisar que la carrera terminó, liberar el *lock* de la tortuga (`lock_tortuga`) y retorna `True`.
  3. Si el corredor no tiene la tortuga: intentará robar la tortuga de su adversario.

  Después de haber verificado todo lo anterior, el corredor avanzará en carrera.

- Modificar `def run(self) -> None:`
  Método encargado de la ejecución del *Thread*.

  1. Al iniciarse la ejecución del *thread*, el corredor debe esperar hasta que se avise el inicio de la carrera.
  2. Luego, debe correr la primera mitad de la carrera.
  3. Tras superar los 50 metros, debe intentar solo 1 vez capturar la tortuga.
  4. Finalmente, debe correr la segunda mitad de la carrera.

Modificar `class Carrera`:
Thread que representa una carrera entre 2 corredores. El programa principal debe esperar que este *thread* finalice para terminar con su programa. Además, posee los siguientes métodos:

- Modificar `def __init__(self, corredor_1, corredor_2, senal_inicio, senal_fin) -> None`:
  Inicializador de la clase. Debe asignar a los corredores como rivales y además definir los siguientes atributos:

`self.senal_inicio`	Un `Event` que representa señal de inicio de la carrera.
`self.senal_fin`	Un `Event` que representa señal de término de la carrera.
`self.corredor_1`	Un `Corredor` que corresponde uno de los competidores de la carrera.
`self.corredor_2`	Un `Corredor` que corresponde uno de los competidores de la carrera.
`self.daemon`	Modificar Un `bool` que determina si el thread es daemon.

- Modificar `def empezar(self) -> str:`
  Método encargado de iniciar el hilo de ejecución de la Carrera. Este método debe quedarse activo hasta que el método `run` finalice. Luego, debe retornar el nombre del jugador que tenga la tortuga.

- Modificar `def run(self) -> None:`
  Método encargado de la ejecución del *thread*. En primer lugar, debe iniciar los *threads* de ambos corredores. Luego, debe avisar el inicio de la carrera. Finalmente, debe esperar hasta que algún jugador indique el fin de la carrera.

No modificar `class Simulación`:
Clase encargada de instanciar 2 equipos, con 3 corredores cada uno, para simular 3 carreras y decidir un equipo ganador. **No** necesitas modificar esta clase.

Notas

- Recuerda que la ubicación de tu entrega es en tu **repositorio personal**. Verifica que no estés trabajando en el **Syllabus**.
- Recuerda que esta evaluación presenta corrección **automatizada**. Si entregas un código que se cae al momento de correr los *tests*, será evaluado con 0 puntos.
- Se recomienda completar la actividad en el orden del enunciado.
- Si aparece un error inesperado, ¡léelo! Intenta interpretarlo y/o buscarlo en google.