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

12 de septiembre de 2019
Actividad Formativa

Actividad 04

Threading

Introducción

¡Hostia tío! Luego de tu gran desempeño en ~~destruir~~ distraer a todos los ayudantes del DCC y obtener las pautas de todas las tareas del semestre, *El Profesor* Sergio Dominguez te pide ayuda para asaltar la Distinguida Casa de ~~Papel~~ Comodidades a cargo del ambicioso y corrupto Enzini. Como te podrás imaginar un atraco es una operación delicada y compleja, en donde se pone en riesgo la vida de múltiples personas. Es por esto que es fundamental la coordinación entre los equipos de trabajo para poder tener éxito en la misión. Debido a tus grandes conocimientos de programación y tus habilidades innatas usando *threads*, se te ha asignado la ~~difícil~~ entretenida tarea de hacer una simulación del asalto. Así, "El Profesor" podrá prever todas las situaciones posibles y crear un plan de acción para contrarrestar cualquier percance.

Dentro del equipo de trabajo se han fijado un conjunto de tres reglas inquebrantables: nada de nombres personales, ~~nada de~~ hacerle preguntas a los ayudantes ante cualquier duda y entregar tu código antes de las 16:30. Se ha decidido utilizar nombres de ciudades para identificarse entre los miembros de la pandilla, En tu caso tu has decidido el nombre clave de **Chiloe**, debido a tu gran amor por la isla.

Por último, debes tener en consideración que tu amigo de la infancia **Berlin Pinto** encabezará la misión dentro de la fabrica, por lo que esta en tus manos la vida de tu compañero y que el plan no fracase.

Flujo del asalto

En esta sección, se explica el flujo de la simulación del robo. El flujo principal de este encuentra en el archivo `main.py`, que es el módulo que debe ejecutarse para iniciar la simulación.

El asalto consiste en la creación de dos grupos de secuaces que operan de forma concurrente. Los **excavadores**, que están a cargo de crear el túnel para escapar del lugar, y los **imprimidores**, que están a cargo de imprimir el dinero que robarán. Para modelar ambas entidades, debes implementar la clase `Excavador` y la clase `Imprimidor`, en donde ambas son *threads* (y por lo tanto, heredan de `Thread`).

Mientras estos secuaces realizan sus labores, pueden surgir diversos problemas que los paralicen en su actuar. Cuando ocurre esto, el secuas aproblemado debe esperar a Berlin, para que este se encargue de solucionarle su problema. Pero como Berlin es solo uno, solo es capaz de solucionar un problema a la vez.

La simulación termina cuando el túnel se finaliza de cavar y cuando se logra imprimir un mínimo de dinero que desean robar. Por lo tanto si es que el túnel se termina, pero no se ha alcanzado el dinero mínimo, los excavadores dejan de cavar y esperan a los imprimidores. Pero en el caso contrario, es decir,

se alcanza la cantidad mínima de dinero, pero no se ha terminado el túnel, entonces los imprimidores siguen trabajando hasta que los excavadores terminen.

Para representar el paso del tiempo de un modo más eficiente, se te entrega la función `def reloj(minutos)` dentro del módulo `utils.py`. Esta representará el paso del tiempo que tu le indiques en minutos, pero de manera más rápida y debes usarla en tu implementación. Esta función convertirá la cantidad de minutos ingresada a un equivalente de tal forma que una hora equivalga a solo un segundo durante la simulación.

A continuación se detallan más sobre las clases que debes implementar:

- `class Excavador`: Son tres encargados de generar una vía de escape rápida y segura, sus nombres son Moscu, Denver y Helsinki. En conjunto deberán cavar un túnel de 6000 metros. Cuando finalmente lo logren, deben avisar para terminar de orquestar la salida del banco. Los métodos de esta clase son los siguientes:
  - `def run(self)`: Es el que se encarga del flujo de la actividad del excavador. Cada uno de ellos cavará una cantidad aleatoria[1] entre 50 y 100 metros cada 10 minutos. En este mismo período de tiempo, tendrán un 10 % de probabilidad[2] de que su picota se dañe.
  - `def problema_picota(self)`: es el método encargado de manejar el problema del daño a la picota, deberá esperar a que Berlin esté disponible y luego esperar a que él lo arregle. Berlin toma 5 minutos en arreglar la herramienta para continuar la excavación.
  - `def avanzar(self, metros)`: es el método que suma metros de profundidad de avance por cada excavador al largo total de la instancia túnel. En caso de llegar a la meta, el excavador levanta el aviso para continuar con el escape. Debes tener en cuenta que los tres excavadores trabajan sobre el mismo túnel, y si cavan al mismo tiempo, pueden haber problemas. Debes asegurarte que al sumar el avance de cada uno al túnel, este debe hacerse por **un excavador a la vez**.
- `class Imprimidor`: Los tres encargados de la falsificación de dinero, llamados Nairobi, Tokio y Rio. De ellos depende el éxito o fracaso de la operación, su trabajo es imprimir la mayor cantidad de dinero posible en el tiempo que los otros hacen el túnel de escape. Deben alcanzar un **mínimo** de 10.000.000 € para poder terminar la misión. Sus métodos son los siguientes:
  - `def run(self)`: Es el que se encarga del flujo de la actividad del imprimidor. Cada uno de ellos imprimirá una cantidad aleatoria[3] entre 100.000 € y 500.000 € cada 10 minutos. En este mismo período de tiempo, tendrán un 20 % de probabilidad[4] de que se les acabe el papel de impresión.
  - `def problema_papel(self)`: es el método encargado de manejar el problema de que se acabe el papel. Al igual que sus compañeros, deberán esperar a que Berlin esté disponible y luego esperar a que él lo solucione. Berlin toma 10 minutos en volver a llenar la maquina con papel para continuar con el asalto.
  - `def imprimir(self, dinero)`: Es el método que agrega el dinero impreso de los falsificadores de manera ordenada en la bolsa de dinero. En caso de llegar al mínimo propuesto, levanta el aviso para continuar con el escape si los túneles están listos. **Los imprimidores deben poder seguir falsificando dinero si el túnel no se ha terminado.** Debes tener en consideración que al sumar la cantidad impresa a la bolsa de dinero, esto debe hacerse **solo una persona a la vez**.

[1] El método `randint` de la librería `random` te puede ser de utilidad
[2] El método `uniform` de la librería `random` te puede ser de utilidad
[3] El método `randint` de la librería `random` te puede ser de utilidad
[4] El método `uniform` de la librería `random` te puede ser de utilidad

Se les recomienda que agreguen *prints* del trabajo y avance de cada imprimidor y excavador para poder apreciar en vivo el avance de la simulación. Aquí va un ejemplo:

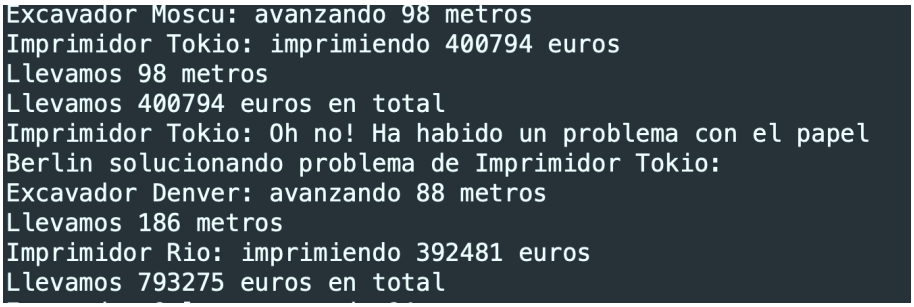

Figura 1: Ejemplo de como deberían avanzar el flujo del juego

Entregables

Para realizar esta simulación, *El Profesor* te entregó cuatro archivos distintos. De estos, **SOLO** debes modificar `excavadores.py`, `imprimidores.py` y `main.py`. En los dos primeros archivos deberás rellenar la información que falta completando las clases `class Excavador` y `class Imprimidor`.

En `main.py` debes crear las estadísticas de cuánto dinero finalmente alcanzaron a imprimir y cuánto tiempo se demorarán en hacer el túnel (información vital para el éxito del atraco) para ello debes completar `def estadisticas`.

Figura 2: Ejemplo de como deberían imprimir las estadísticas

El otro archivo que te entregó es `utils.py` que maneja los eventos de la simulación. Contiene a `def reloj` (explicado en el comienzo), y las clases `class Tunel` y `class BolsaDinero`, que debes **utilizar para guardar los avances de distancia y dinero de ambos equipos**. Recuerda que **NO** debes modificar este archivo.

¡Éxito!

Ojo: Una vez que se alcance el mínimo de dinero y se cave el túnel por completo y la simulación termine, puede que algunos *threads* de secuaces continúen trabajando por un periodo breve. No deben manejar este caso y se considera correcto si la simulación termina. Por ejemplo:

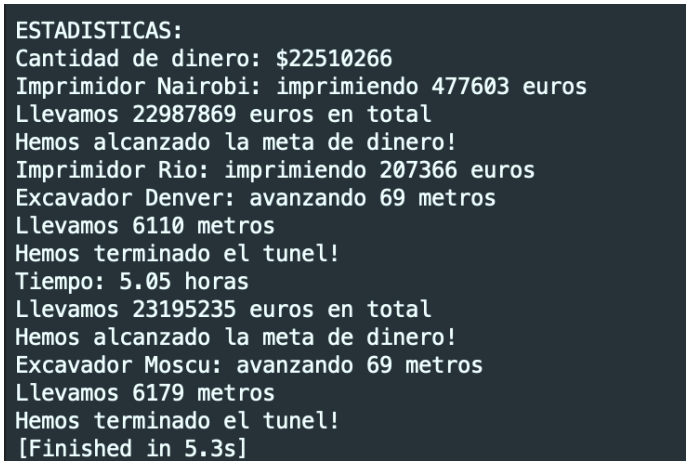

Figura 3: Ejemplo de simulación terminada con secuaces trabajando

Requerimientos

- (2.75 pts) Completar la clase `Excavador` de `excavadores.py`
  - (1.00 pts) Método `run`
  - (1.00 pts) Método `avanzar`
  - (0.75 pts) Método `problema_picota`
- (2.75 pts) Completar la clase `Impresor` de `imprimidores.py`
  - (1.00 pts) Método `run`
  - (1.00 pts) Método `imprimir_dinero`
  - (0.75 pts) Método `problema_papel`
- (0.5 pts) Completar estadísticas de `main.py`

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AC04/`
- **Hora del *push*:** 16:30

Auto-evaluación

Como esta corresponde a una actividad formativa, te extendemos la instancia de responder, después de terminada la actividad, una auto-evaluación de tu desempeño. Esta se habilitará a las **16:50 de jueves 12 de septiembre** y tendrás plazo para responderla hasta las **23:59 del día siguiente**. Puedes acceder al formulario mediante el siguiente enlace:

`https://forms.gle/AtJTVXSWSnzYRZUg9`

El asistir, realizar la actividad y responder la auto-evaluación otorgará como bonificación al alumno 2 décimas para sumar en su mejor actividad sumativa del semestre.