



Universidad Nacional Autónoma de México



Facultad de Ingeniería

División de Ingeniería Eléctrica

Sistemas Operativos

Grupo 06

2025 - 2

Implementación del problema: Intersección de caminos

Alumnos

Eric Ramírez Valdovinos (423095203)

Erick Nava Santiago (320298608)

Profesor

Dr. Gunnar Eyal Wolf Iszaevich

Ciudad Universitaria, Coyoacán, CDMX, 12 de abril de
2025

Proyecto 1 - Ejercicio de sincronización

1. Descripción del Problema

Existe una familia de gatos. Dentro de esta familia se encuentran 5 gatos: 4 hembras y 1 macho. En el lugar donde habitan se les han proporcionado 2 platos donde se les da de comer por medio de un dispensador automático, en cada plato puede comer un gato a la vez.

Todos los gatos comen una porción de alimento cuando tienen hambre, a excepción del macho, el cual, come 2 porciones. El gato macho solo comerá una vez si es que no hay otra hembra que quiera alimentarse, es decir si hay algún plato libre y las demás gatas se encuentran en estado “satisfecho”. Los gatos estarán en tres estados:

- Hambriento
- Comiendo
- Satisfecho
- Enfermo

Es posible que entre algún gato invasor y fulmine con todas las porciones restantes, en dado caso, el dispensador debe emitir una alerta que ahuyente al gato invasor y no rellene los platos hasta que el gato invasor se retire (no es tarea de esta simulación el determinar cómo se diferencia de un gato u otro).

Así pues, el dispensador posee los siguientes estados:

- Alerta
- Cerrado
- Abierto

Para determinar si se encuentra cerrado o abierto debe existir una sincronización entre platos, de tal manera que, si plato 1 y plato 2 se encuentran vacíos, entonces el dispensador pasa a abierto y los platos recargan sus 3 raciones y ambos pasan al estado “Ok”.

Un plato puede únicamente estar vacío o no, por lo que sus estados son:

- Vacío
- Ok

Existe una cola de prioridad de acceso, la cual se va actualizando a medida que los gatos entran al estado hambriento, de esta forma, generamos un ordenamiento de que los actores tengan acceso a los recursos, por supuesto, el gato macho pasará hasta atrás de la cola hasta que solo quedé él.

Caso contrario, si algún gato se encuentra en estado enfermo, siempre tendrá prioridad absoluta sobre cualquiera de los platos, aún si se trata del gato macho.

1.1 Reglas

- Dos gatos no pueden del mismo plato al mismo tiempo
- Para que un plato sea rellenoado, ambos han de estar vacíos
- El gato invasor tiene prioridad por sobre cualquier otro y siempre termina con las porciones
- Las gatas de tipo “HEMBRA” tienen prioridad sobre “MACHO”
- El dispensador rellena los platos al mismo tiempo
- Un gato no debe tener preferencia de un plato sobre otro

2. Lenguaje y Entorno de Desarrollo

Lenguaje: Python

Versión: Python 3.13.2

Entorno: Se desarrolló y testeó en un sistema operativo Windows 10 y IOS Sonoma 14.0 a través de VsCode.

No se requiere instalación de librerías externas.

El programa se dividió en varios archivos para facilitar la legibilidad, a su vez cada archivo considera las clases principales con las que se trabajó, añadiendo para algunas las consideraciones de los estados en los que pudieran encontrarse. Utilizando una clase extra para estos archivos mediante la herramienta enum.

En cuanto el programa realizado se utilizaron las bibliotecas PIL, tkinter, threading y queue.

En listado principalmente estas pues son vitales para el funcionamiento de lo planteado. Threading nos permite trabajar con hilos, PIL para manipular las imágenes de los gatos, tkinter para implementar la interfaz y queue para poder llevar la cola de prioridades planteadas en la programación.

De igual forma utilizamos Enum para facilitarnos los estados que cada objeto requería, random para generar números aleatorios y la función count de intertool. Este último terminó siendo de gran ayuda, puesto que a la hora de pasar una tupla a la cola , la prioridad de dos objetos podría ser del mismo nivel, ocasionando así errores o la detención total del programa, implementar un contador que aumenta por cada entrada permitió hacer el efecto de: la misma prioridad, pasa el primero en entrara a la cola. El decir que usamos estas bibliotecas, de cierta forma, implica el uso del lenguaje de programación python.

El trabajo no fue en tiempo compartido, se comunicaban los cambios por medio de github, se revisaba y se aprobaba un commit hacia el repositorio acordado para subir nuestros trabajos de cursos. Cada uno de los miembros contaba con distintos sistemas operativos, los cuales fueron ios y windows 10.

Trabajamos usando clases, una por archivo.

2.1 Ejecución

Ubicar el programa en el directorio correspondiente, manteniendo la estructura de archivos y ejecutar el archivo main.py.

Ejecutar el programa mediante la línea de comandos:

```
python main.py
```

3. Estrategia de Sincronización

3.1 Mecanismos de sincronización

3.1.1 Exclusión mutua

Al utilizar el self.lock de la biblioteca threading garantizamos que no haya más de un gato peleando por el recurso al mismo tiempo, de esta forma, los actores esperan para utilizar los platos que son contenedores del recurso compartido.

Las condiciones de carrera son manejadas a través de esta implementación, así nos aseguramos de simular correctamente las porciones de comida disponibles y que no haya un número de porciones -1, por ejemplo.

3.1.2 Coordinación para refill de platos

Los platos no se coordinan directamente entre ellos, si no mediante el objeto Dispensador el cual actúa como intermediario entre estos.

El dispensador monitorea los estados de los dos platos para que cuando detecta que sus porciones estén en 0, este cambie a estado “ABIERTO” y los platos se rellenen.

3.2 Lógica de operación

3.2.1 Actores del sistema

Los gatos (macho, hembra e invasor), mediante la cola de prioridades y el acceso con exclusión mutua de los platos, el gato macho y hembra tienen un orden para consumir los recursos de forma ordenada. El gato invasor solo compite por llegar primero y si bien su presencia está manejada en el código, las reglas del sistema están diseñadas para que su anarquía se tome como prevista

3.2.2 Recursos compartidos

Los platos albergan el recurso compartido por los gatos. El dispensador administra y suministra recursos a los platos, existen una serie de prioridades para el acceso a los platos con el recurso.

3.2.3 Invasor

El agente invasor (gato invasor) es un detonante inesperado que fulmina toda la comida disponible y pone en alerta al dispensador, por lo que el sistema se bloquea hasta que manualmente se remueve de la “casa”, es decir se presiona el botón de sacar gato

3.2.4 Funcionamiento general

Los gatos inician satisfechos todos, eventualmente les tendrá que dar hambre y buscarán comida en los platos, si les da hambre y el plato está ocupado, la cola de prioridades entra en acción y ordena el acceso al recurso por parte de los gatos. Cuando ya no queda comida en ninguno de los platos, el dispensador se abre y rellena ambos platos.

3.3 Estado compartido

Los estados que puede tener un plato, aunque no estén representados directamente en el código, si modifican directamente los estados de los gatos porque al no tener recurso, necesitan a volver a suministrar alimento para que los gatos continúen con su vida.

Los estados del dispensador dependen de la presencia del invasor y las porciones disponibles en ambos platos.

3.4 Descripción algorítmica del avance de cada hilo/proceso

En el main se inicializan hilos de la siguiente forma:

```
# Botones de control

def iniciar_simulacion():

    threading.Thread(target=dispensador.gestion, daemon=True).start()

    threading.Thread(target=dispensador.run, daemon=True).start()

    for gato in gatos:

        gato.start()

    actualizar_estados()
```

Aquí se logra ver que se utilizan dos hilos, los cuales van a estar directamente ligados a los procesos run y gestión del dispensador. En el dispensador, gestión se encarga de la lógica necesaria para que la cola de prioridades funcione correctamente, este método se ejecuta en loop infinitamente, esto porque esta evaluando todo el tiempo el estado de la cola.

```
def gestion(self):
```

```

while True:

    if not self.cola.empty(): #en caso de que la cola no este vacia

        prioridad, _, gato = self.cola.get()      #se saca el gato con mayor
prioridad, el _ es el numero de ingreso, esto por si empatan en prioridad

        print(f'{gato.nombre} volverá a intentar comer')#mensaje

        gato.comer()                      #se le manda a comer otra ves al gato

        if gato.tipo == TipoGato.INVASOR:

            self.detectarGatoInvasor()

            with self.lock:

                if self.cola.empty():

                    self.retiradaGatoInvasor()

                    print("Gato invasor neutralizado")

    sleep(3)

```

Por otra parte el método run se ve de la siguiente manera:

```
def run(self):
```

```

while True: #esto hace que sea un loop sin fin

    if self.estado != EstadoDispensador.ALERTA: #en caso de
que no este en alerta


        a=len(self.platos)                      #cantidad de
platos

        for plato in self.platos:                 #verificara
la comida de cada plato

            if plato.comida_disponible < 1:       #si no hay
comida en un plato se restara a 'a'

                a = a-1

            if a == 0 :

```

```
        self.dispensar()                      #si ya no
hay comida en los dos platos se rellenan

        self.estado=EstadoDispensador.CERRADO    # una
vez rellenado se cerrra el dispensador

        sleep(5)
```

el cual, a grandes rasgos está evaluando la todo el tiempo el estado del dispensador, de ser necesario llama a otros métodos para el cumplimiento del planteamiento.

Por parte de los gatos, todo el tiempo están comiendo de los platos, para que éstos no intervengan entre si se utiliza un semáforo con bloqueo a 1, representado que los platos solo tiene un cupo para ser utilizados de forma simultánea. De igual forma los gatos hacen uso de hilos para funcionar de forma independiente.

4. Ejemplos / pantallazos

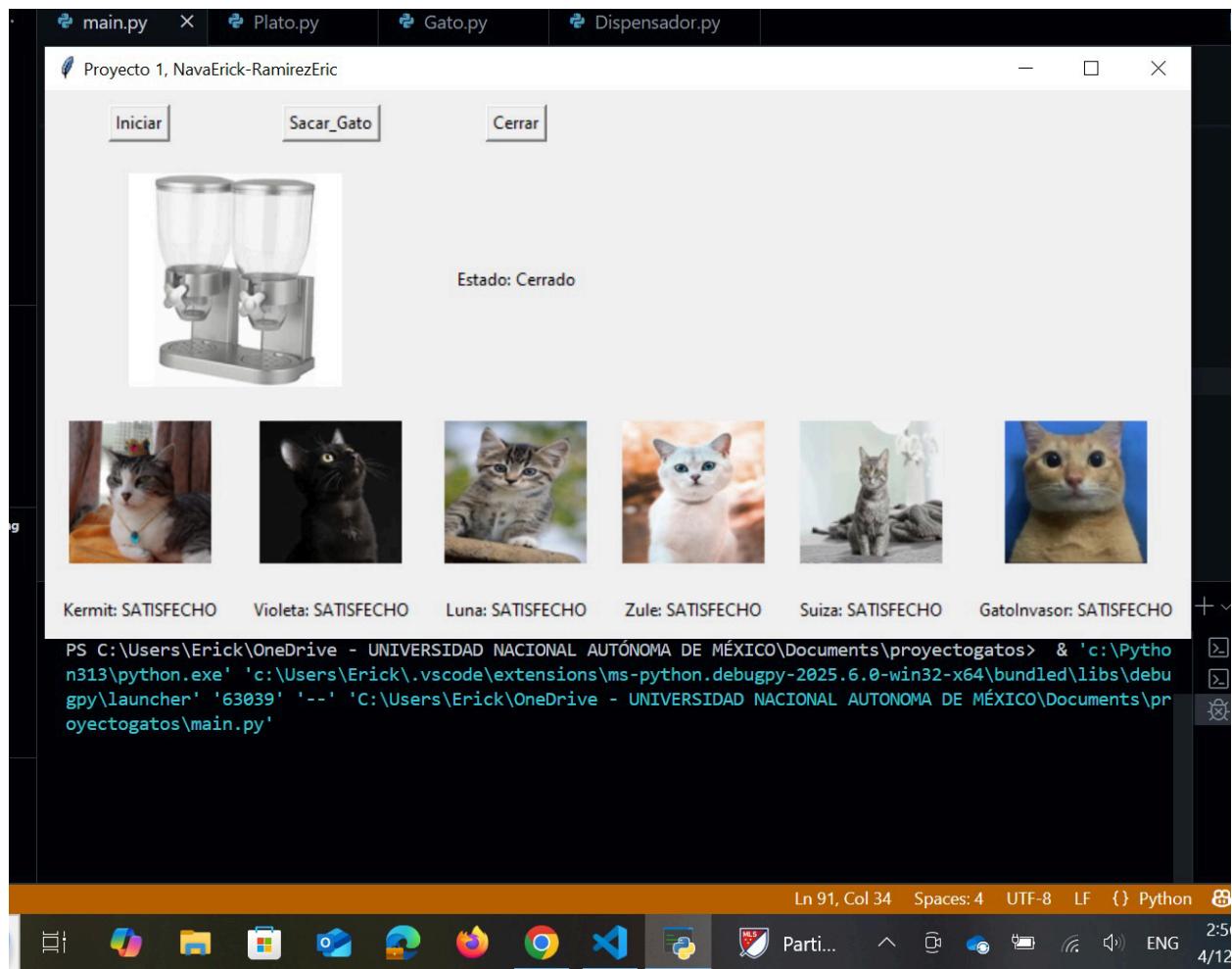


Imagen 1. Interfaz inicial.

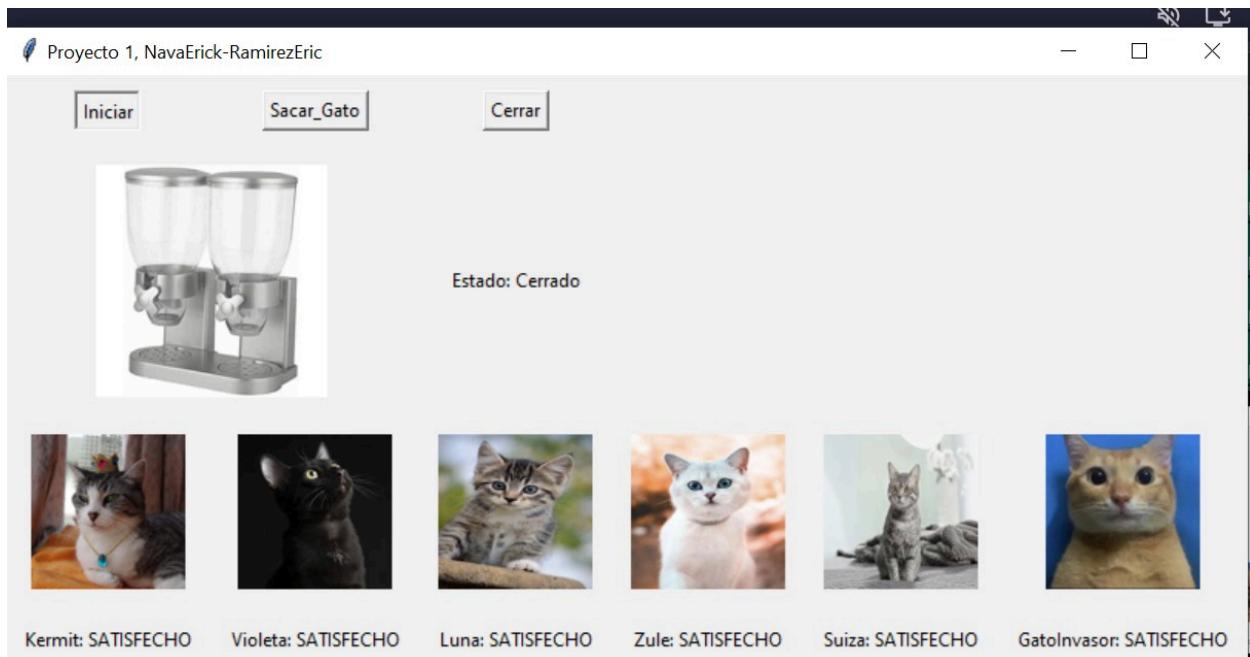


Imagen 2. Presionando Iniciar

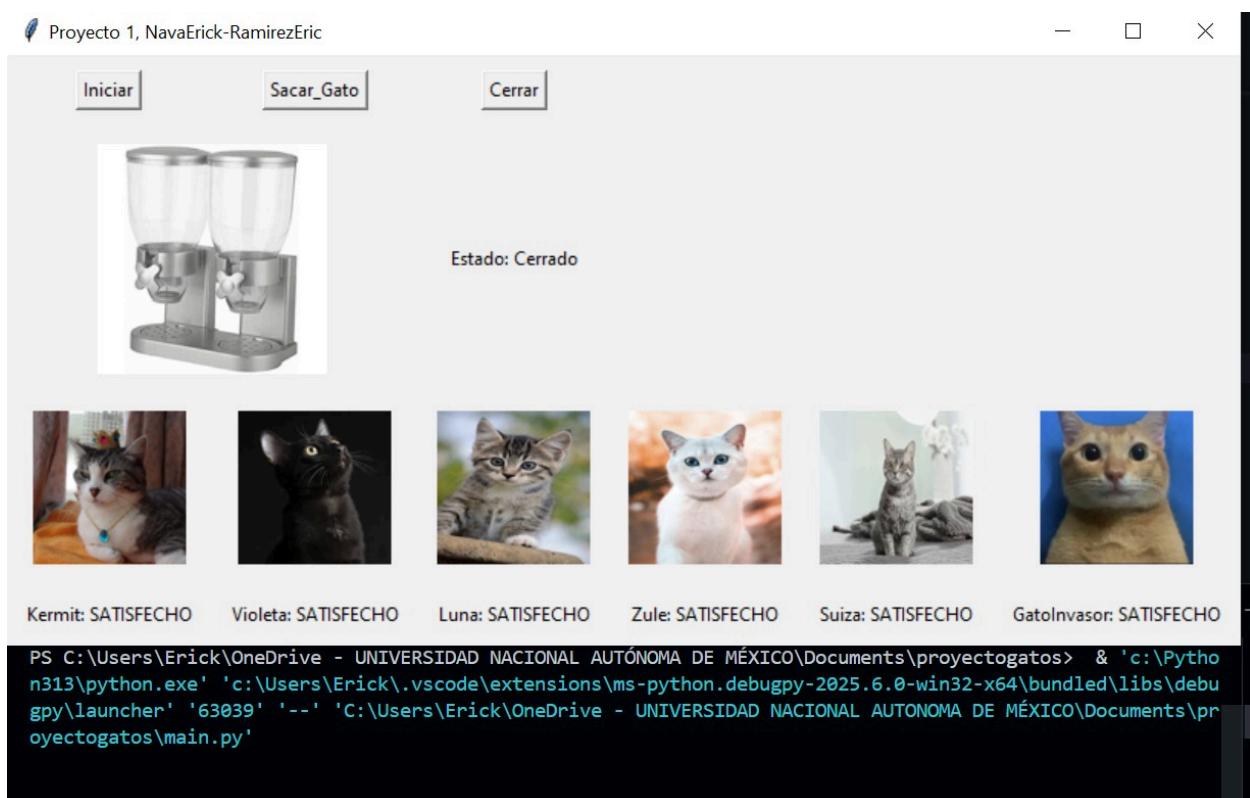


Imagen 3. Despu s de presionar Iniciar, tardar  un momento en reflejarse (modificable en el c digo).

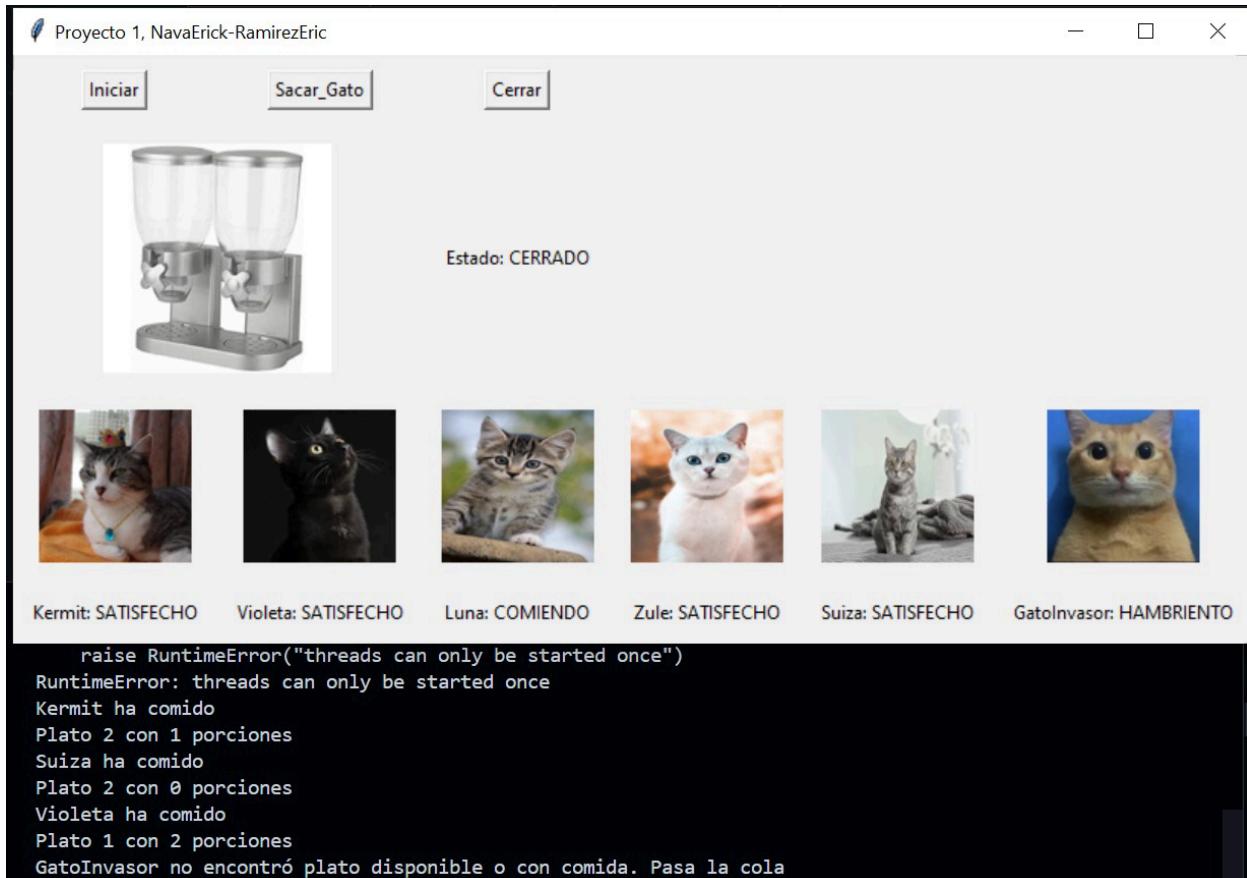


Imagen 4. La terminal dará más especificaciones de los platos. La interfaz mostrará solo a los gatos y el dispensador.

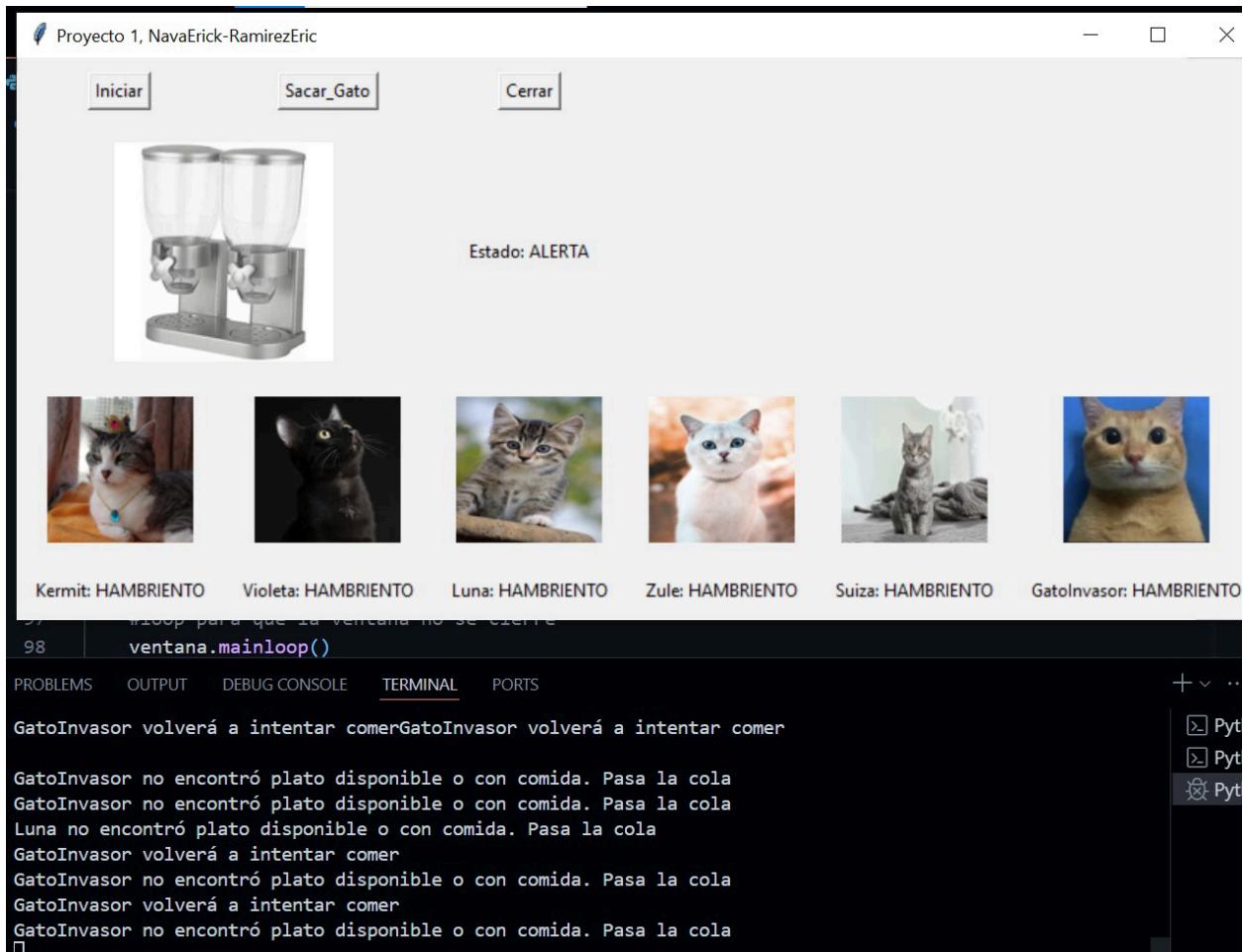


Imagen 5. En caso de alerta por gato invasor. Se bloqueará el dispensador, por lo tanto, ni los platos, ni los gatos tendrán acceso al recurso.

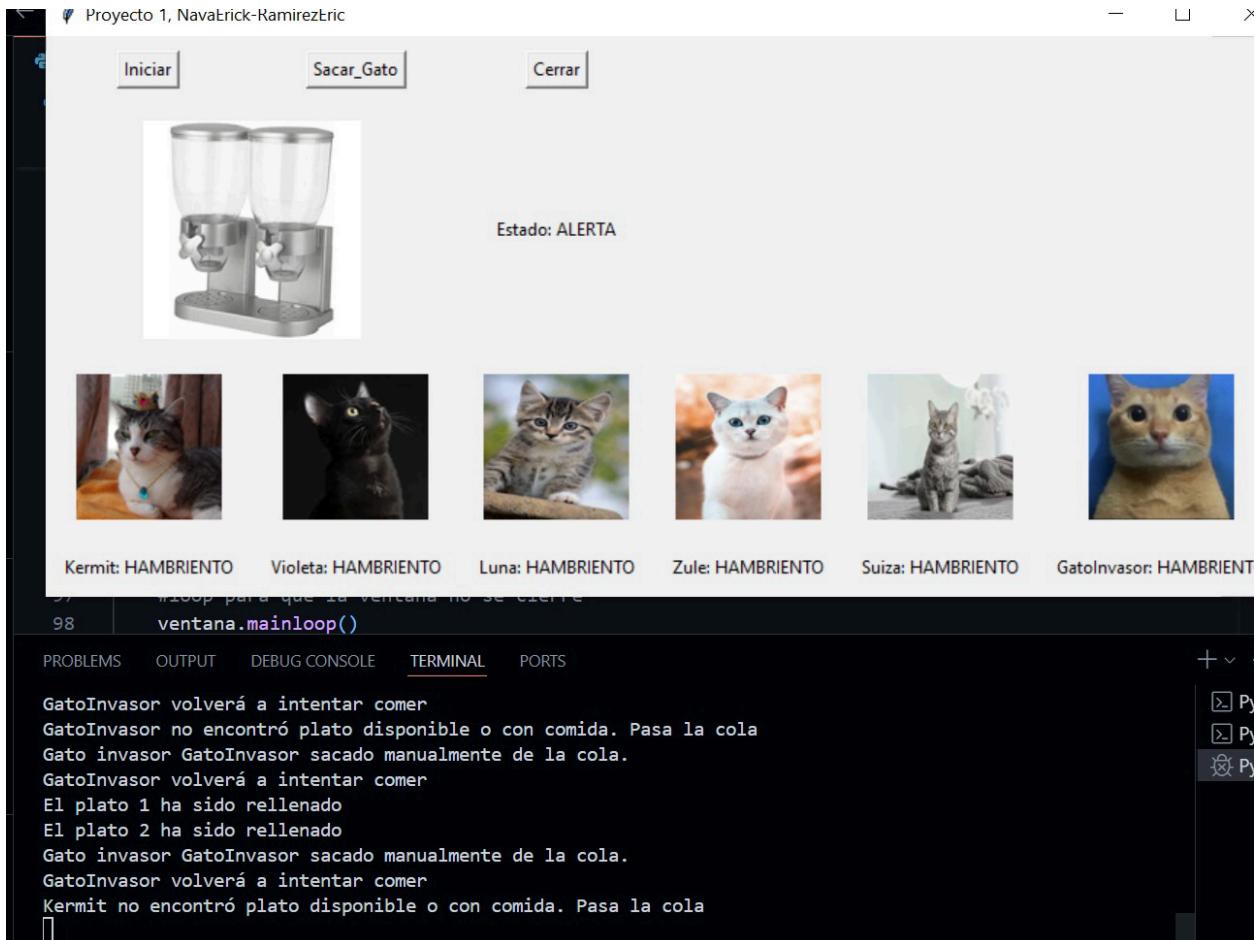


Imagen 6. Al presionar sacar gato, el dispensador volverá a trabajar con normalidad, la presencia del invasor termina.

5. Dudas y mejoras

Con todo el tiempo el cual fue justo para desarrollar un buen proyecto, creo que nos volvimos a quedar cortos en cuánto a la implementación de una solución sencilla y elegante para el problema planteado, el cual no consideramos que haya sido de gran complejidad. La realidad es que aún y con el tiempo extra por las situaciones extra curriculares (paro de actividades) no dedicamos el tiempo necesario para sacar un trabajo del que estemos 100% orgullosos. Creemos que se aprecia sobre todo en la interfaz gráfica presentada. Sigue siendo nuestro punto a mejorar (y con creces).

Considerando que el tiempo de vida de cada proceso en el sistema, es decir cada gato en la casa, por así decirlo, es equivalente a la duración de la ejecución, sin duda se podrían implementar mejoras para hacer que tengan un fin natural o para añadirle complejidad y originalidad al planteamiento, hacer que en algún punto por alguna situación los tiempos de acción de algún hilo en particular cambiarán o desapareciera por completo, no solo asignar prioridades distintas a cada proceso (gato) y que estas cambiarán aleatoriamente en función de la situación en la que se encontrara la simulación.

Otra mejora podría ser implementar un conteo de tiempo (reloj integrado el dispensador) y así registrar los “refills” del dispensador y los consumos de los gatos para poder administrar de mejor manera el

alimento total del dispensador, de esta forma entrarían en juego otros factores, como la frecuencia de consumo, cantidad total de recursos y su administración (comida de gatos), algoritmos para identificar momentos del día de mayor consumo, etc.

Nos quedamos con la idea de que nuestro planteamiento fue original, se implementó con la lógica correcta, pero no podemos evitar pensar que debe de existir un mecanismo para compartir los recursos de manera más elegante que utilizando semáforos y locks.

Material consultado

GeeksforGeeks. (2025, March 12). *What is Priority Queue | Introduction to Priority Queue.*

GeeksforGeeks. <https://www.geeksforgeeks.org/priority-queue-set-1-introduction/>

Graphical User Interfaces with Tk. (n.d.). Python Documentation.

<https://docs.python.org/3/library/tk.html>

Haider. (2023, June 21). *Semáforos en Python.* Delft Stack.

<https://www.delftstack.com/es/howto/python/python-semaphore/>

threading — *Thread-based parallelism.* (n.d.). Python Documentation.

<https://docs.python.org/es/3/library/threading.html>