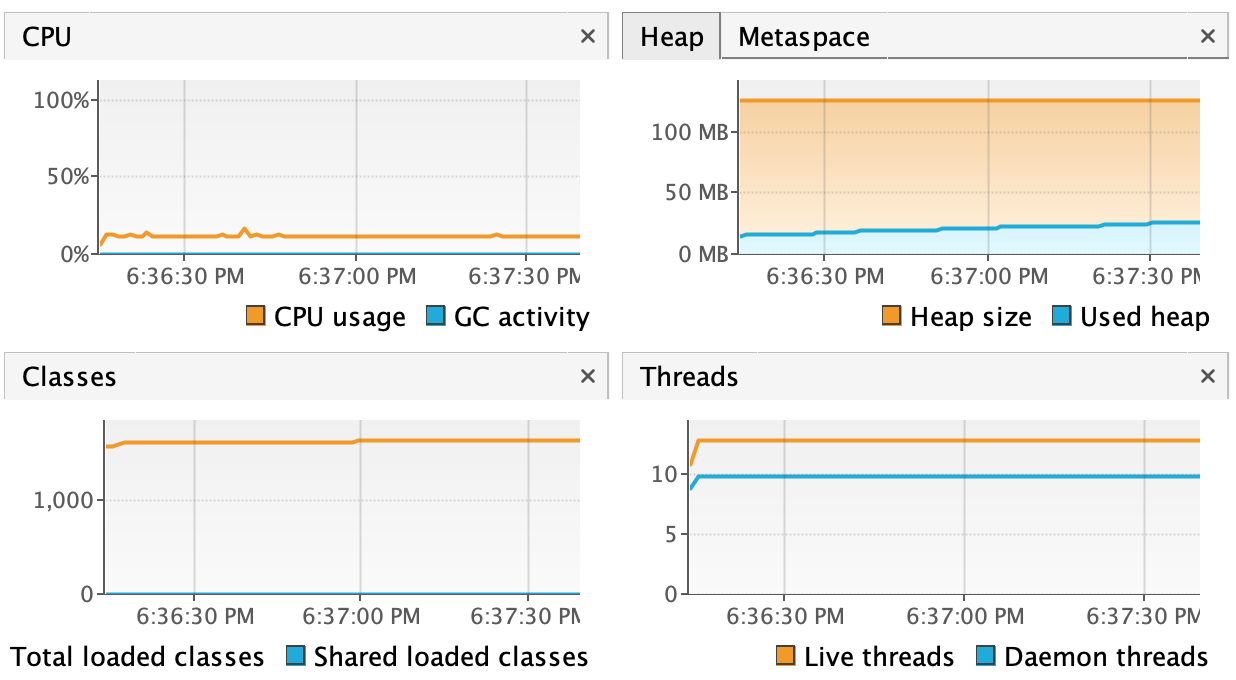
Laboratorio 3

Wilmer Arley Rodríguez Ropero

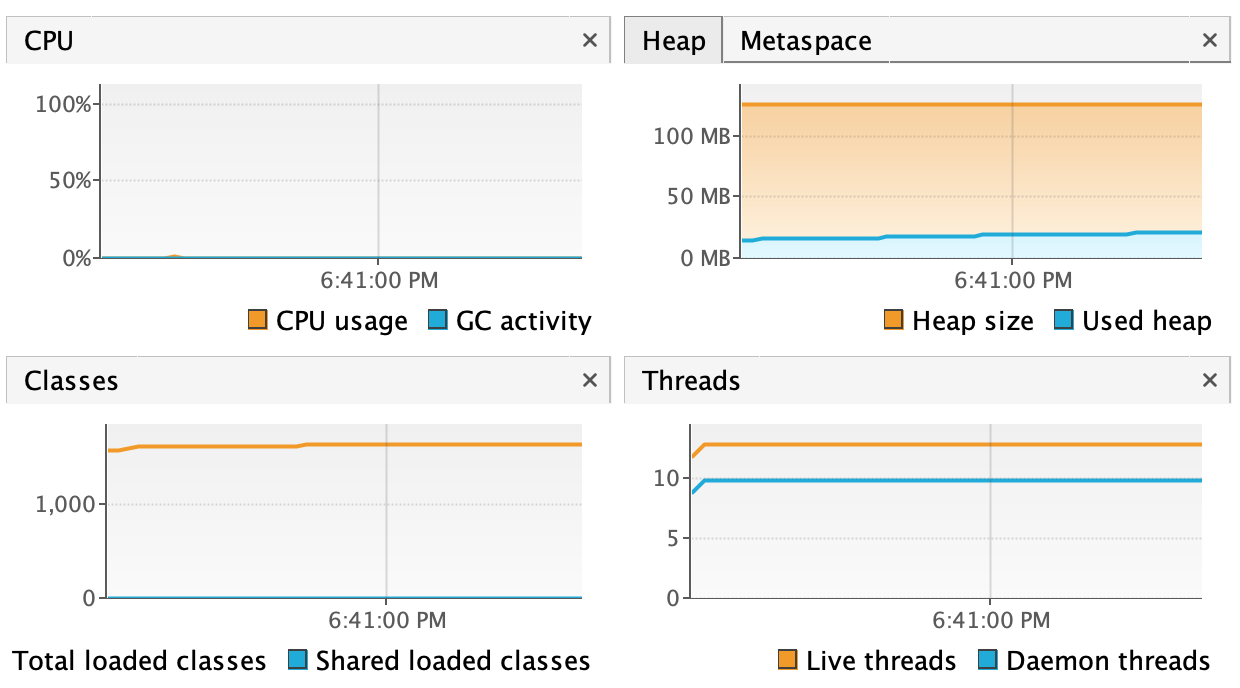
Daniel Sebastián Ochoa Urrego

**Parte 1**



El consumo se debe a que la clase consumidora revisa constantemente a que el productor haya agregado algo a la lista, mientras que el productor agrega elementos cada segundo. Viendo la aplicación, podemos ver que el uso de la CPU ronda un 12%

1. Para hacer que el programa usara menos la CPU hicimos que la clase consumidora esperara el mismo tiempo que productor antes de consumir un numero

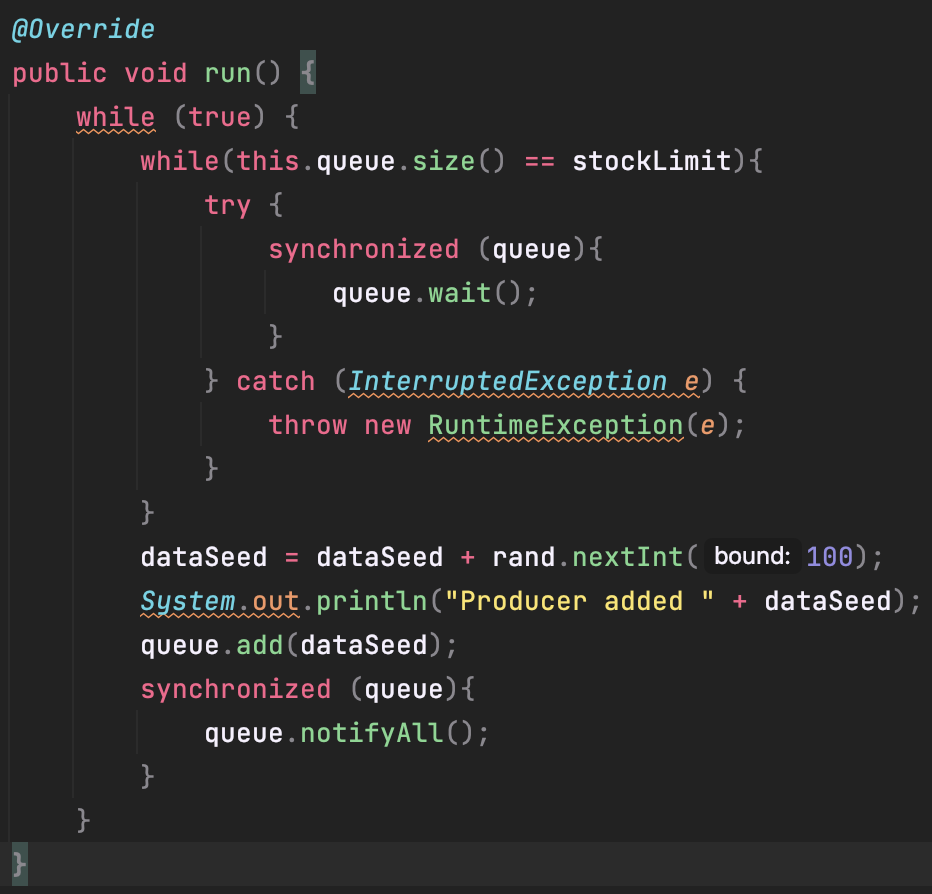


Y si revisamos el uso de la CPU vemos que este bajo a rondar el 0.1%

1. Para añadir esta funcionalidad creamos un atributo que representara el máximo stock en la clase que contiene el main



Y la clase productora la modificamos de la siguiente manera para que tuviera en cuenta el valor límite de producción, y se produjeran elementos en el momento en que se consuman, sin hacer sobre procesamiento de datos



Y la clase consumidora quedo de la siguiente manera haciendo uso de wait/notify



Y al correr el programa y ver su rendimiento en JVisualVM podemos ver como se tiene la misma funcionalidad de antes a un rendimiento mucho mejor

Graphical user interface, application, table, Excel

Description automatically generated

**Parte 2**

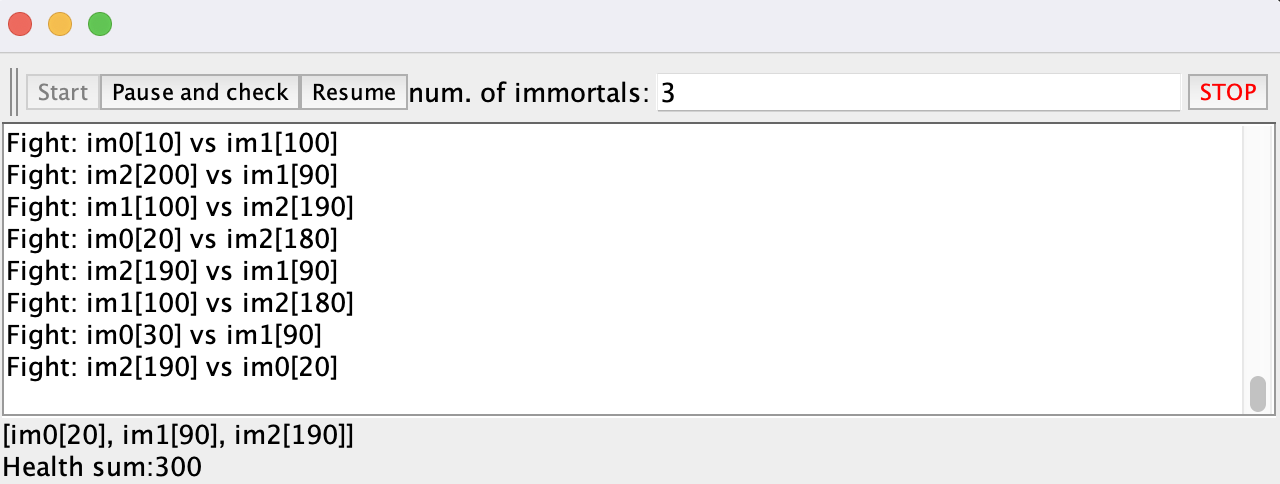
1. Asumiendo que la vida de los inmortales sea siempre de 100, para N jugadores se tendrá que la vida de todos será igual a 100\*N
2. No, el numero varía mucho, a veces sube y a veces baja, además de que no se pausa el juego
3. Con la primera implementación de pause and check no cumple el invariante porque la vida de los inmortales cambia constantemente y no hace que el resultado sea erroneo.
4. Para implementar la pausa creamos lo siguiente en el actionListener del botón



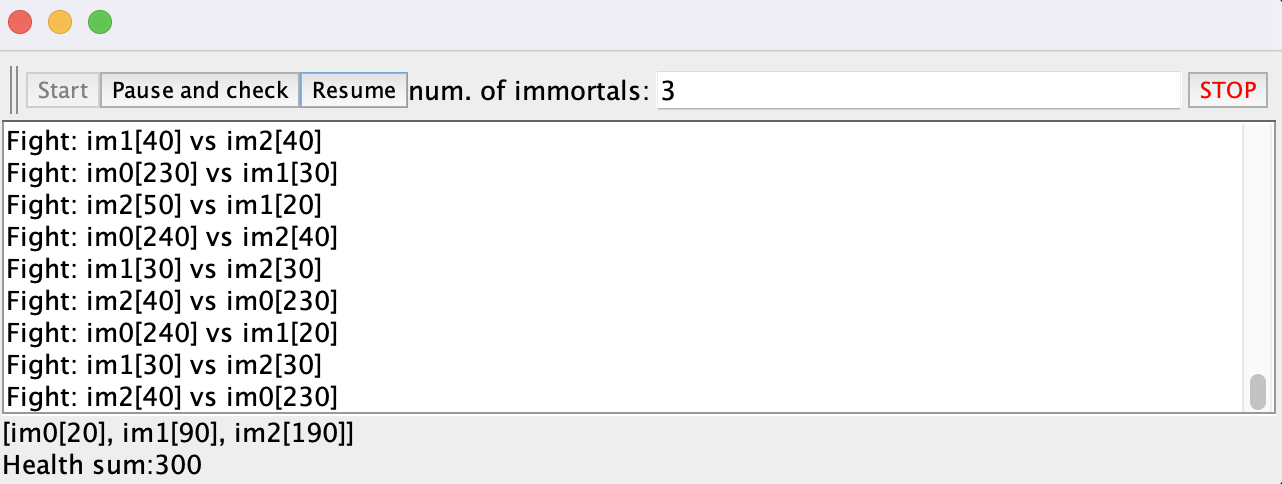
Y en los hilos de los inmortales creamos el método checkPause que revisa si los hilos tienen que parar y realizar lo correspondiente si es así



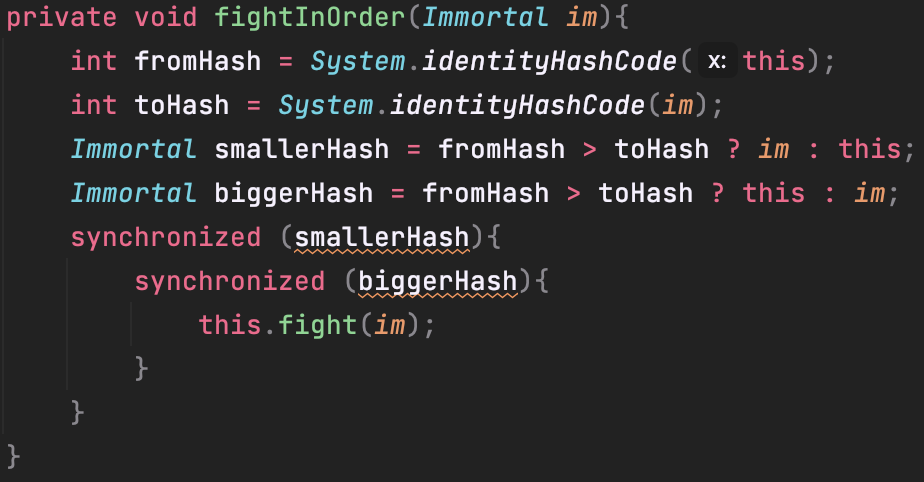
Y al pausar la aplicación podemos ver que los hilos en efecto se detienen y se imprimen su estado



Y al resumir continúan con su ejecución



1. Y al hacer click en el botón también podemos ver como aún no se cumple este invariante.
2. Luego para arreglarlo vimos que otra región critica donde fallaba el programa era en las peleas, se debe asegurar que los inmortales peleen entre ellos sin que se crucen los cálculos en el proceso, para ello implementamos el método fightInOrder



1. Como se tuvo en cuenta el orden de entrada, en un principio, con ayuda de los hashes de cada objeto lock, garantizamos que no ocurran DeadLocks y con esto realizado no tuvimos problema de que el programa se nos detuviera.
2. Probamos con diferentes valores y vemos lo siguiente:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Imagen en blanco y negro

Descripción generada automáticamente con confianza baja

Como podemos ver en ambos casos el invariante se mantiene y no tenemos ningún problema, lo que quiere decir que ya solucionamos el problema de las regiones críticas.

1. Al ejecutar el programa y lo dejamos un tiempo en ejecución, se puede apreciar lo siguiente:

Texto

Descripción generada automáticamente

Varios inmortales están asestando golpes a los que ya están muertos, aunque también tenemos el problema de que los inmortales que en teoría murieron siguen asestando golpes a los demás.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Esto lo podemos ver en lo siguiente donde en teoría el inmortal 0 murió, pero aún sigue asestando golpes y aun puede recuperar vida.

Ya con esto podemos decir que hay una condición de carrera y seria que el thread muera antes de que otro inmortal pueda tomarlo y asestar un golpe, o en caso contrario que muera antes que este aseste un golpe y recupere vida.

Una primera aproximación para poder quitar a los immortals que están muertos es sacándolos de la lista de immortals, sin embargo, esto nos genera dos principales inconvenientes:

a) El primero seria que tendríamos que sincronizar el acceso a la lista y esto haría que el programa fuera muy lenta, ya que los demás threads tendrían que esperar a que la desocupen.

b) El segundo inconveniente es que los hilos que saquemos de la lista seguirán corriendo por lo que podrían seguir haciendo daño a los immortals

-Una segunda idea sería que cada inmortal tenga una bandera que avise a los demás threads cuando este ya esta muerto y a su vez detener el thread para que no siga ejecutando.

Ahora tenemos que pensar en la siguiente pregunta, ¿Cuándo cambia el estado del hilo y finaliza? Ya que como vimos anteriormente el thread podría seguir pegando aun muerto, por lo que debemos detenerlo una vez pierda toda la vida. Esto lo solucionaremos parcialmente creando una bandera y a su vez un método que hace que cambie el estado de un inmortal para que pare el hilo.

Interfaz de usuario gráfica

Descripción generada automáticamente

Ahora el lugar donde debemos cambiar dicho valor va a ser justo después de que el inmortal reciba daño para que pare el hilo y no pueda hacer nada más.

Texto

Descripción generada automáticamente

Como todo esto esta en un bloque sincronizado que toma a los inmortales podemos preguntar antes de que inicien a pelear si ambos están vivos.

Texto

Descripción generada automáticamente

Ahora ejecutamos de nuevo la simulación o podemos ver lo siguiente.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Ahora dejamos un buen tiempo la simulación y vemos que ya no se asestan golpes a los que ya están muertos y además no cambiamos el invariante, lo que quiere decir que nuestra implementación cumple.

11. Para el botón stop lo que hicimos fue que parara toda la simulación y que el usuario de nuevo ponga la cantidad de inmortales deseados e inicie de nuevo todo.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Se puede apreciar que además pusimos un mensaje que avisa de que la simulación se terminó.

Y esta seria su implementación en código.

Texto

Descripción generada automáticamente