**ACTIVIDADES PRÁCTICA 3. HORMIGAS.**

Actividad 0 (Sincronización básica en Java).

Lanzaremos varias ejecuciones del código inicial (Terrain 0), con los argumentos por defecto, para obtener distintos resultados sobre su ejecución.

|  |  |  |
| --- | --- | --- |
| **Prueba** | **Total Retries** | **Número de hormigas interbloqueadas** |
| 1 | 26 | 2 |
| 2 | 53 | 6 |
| 3 | 59 | 5 |
| 4 | 16 | 0 |
| 5 | 47 | 4 |
| 6 | 29 | 3 |

Ahora repetiremos las pruebas, utilizando los argumentos 0 (Terrain 0), 8 (Territorio de 8x8 celdas), 10 (10 hormigas en el territorio) y 3 (3 movimientos por hormiga).

|  |  |  |
| --- | --- | --- |
| **Prueba** | **Total Retries** | **Número de hormigas interbloqueadas** |
| 1 | 28 | 2 |
| 2 | 23 | 2 |
| 3 | 29 | 2 |
| 4 | 2 | 0 |
| 5 | 38 | 6 |
| 6 | 2 | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Territorio** | **Ejecución** | **Media Retries** | **Media hormigas interbloqueadas** |
| Terrain 0 | Por defecto | 38.33 | 3.33 |
| {0, 8, 10, 3} | 20.33 | 2 |

Actividad 1 (Sincronización con ReentrantLocks en Java)

Utilizando las herramientas ReentrantLock y Condition proporcionadas en java.util.concurrent, desarrolle Terrain1 (Monitor general con una única variable condición para todo el territorio). Posteriormente realizaremos la misma prueba que en la actividad anterior, pero ahora usando Terrain1.

Lanzaremos varias ejecuciones del código desarrollado (Terrain1), con los argumentos por defecto, para obtener distintos resultados sobre su ejecución.

|  |  |  |
| --- | --- | --- |
| **Prueba** | **Total Retries** | **Número de hormigas interbloqueadas** |
| 1 | 50 | 6 |
| 2 | 7 | 0 |
| 3 | 29 | 2 |
| 4 | 27 | 4 |
| 5 | 39 | 4 |
| 6 | 34 | 3 |

Ahora repetiremos las pruebas, utilizando los argumentos 1 (Terrain 1), 8 (Territorio de 8x8 celdas), 10 (10 hormigas en el territorio) y 3 (3 movimientos por hormiga).

|  |  |  |
| --- | --- | --- |
| **Prueba** | **Total Retries** | **Número de hormigas interbloqueadas** |
| 1 | 11 | 0 |
| 2 | 34 | 3 |
| 3 | 24 | 2 |
| 4 | 1 | 0 |
| 5 | 16 | 2 |
| 6 | 22 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Territorio** | **Ejecución** | **Media Retries** | **Media hormigas interbloqueadas** |
| Terrain 0 | Por defecto | 38.33 | 3.33 |
| {0, 8, 10, 3} | 20.33 | 2 |
| Terrain 1 | Por defecto | 31 | 3.16 |
| {1, 8, 10, 3} | 18 | 1.5 |

Se aprecia una ligera mejora en la cantidad de retries y hormigas interbloqueadas en Terrain1 con respecto a Terrain0. No obstante, no podemos concluir que esa diferencia sea realmente significativa.

En cuanto a ejecuciones acabadas el número de esperas es mayor en el \*reentrantLock\* pero en las que se produce interbloqueo no se aprecia la diferencia.

Actividad 2 (Uso de varias variables condición).

Utilizando las herramientas ReentrantLock y Condition proporcionadas en java.util.concurrent, desarrolle Terrain2 (Monitor general con tantas variables condición como celdas tenga el territorio). Posteriormente realizaremos la misma prueba que en la actividad anterior, pero ahora usando Terrain2.

Lanzaremos varias ejecuciones del código desarrollado (Terrain2), con los argumentos por defecto, para obtener distintos resultados sobre su ejecución.

|  |  |  |
| --- | --- | --- |
| **Prueba** | **Total Retries** | **Número de hormigas interbloqueadas** |
| 1 | 4 | 2 |
| 2 | 4 | 2 |
| 3 | 4 | 8 |
| 4 | 2 | 5 |
| 5 | 4 | 7 |
| 6 | 6 | 11 |

Ahora repetiremos las pruebas, utilizando los argumentos 2 (Terrain 2), 8 (Territorio de 8x8 celdas), 10 (10 hormigas en el territorio) y 3 (3 movimientos por hormiga).

|  |  |  |
| --- | --- | --- |
| **Prueba** | **Total Retries** | **Número de hormigas interbloqueadas** |
| 1 | 4 | 2 |
| 2 | 1 | 0 |
| 3 | 5 | 0 |
| 4 | 5 | 3 |
| 5 | 0 | 6 |
| 6 | 2 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Territorio** | **Ejecución** | **Media Retries** | **Media hormigas interbloqueadas** |
| Terrain 0 | Por defecto | 38.33 | 3.33 |
| {0, 8, 10, 3} | 20.33 | 2 |
| Terrain 1 | Por defecto | 31 | 3.16 |
| {1, 8, 10, 3} | 18 | 1.5 |
| Terrain 2 | Por defecto | 4 | 5.83 |
|  | {2, 8, 10, 3} | 2.83 | 2.16 |

En Terrain 2 se producen significativamente menos reactivaciones innecesarias que en Terrain0 y Terrain1; no obstante, el número de hormigas interbloqueadas aumenta ligeramente.

Se aprecia una diferencia significativa entre Terrain0, Terrain1 y Terrain2 debido a que ahora solo despertamos a la hormiga que sabemos que va a poder continuar.

Nota: waiting for representa una hormiga que está esperando.

Actividad 4 (Gestión de interbloqueos).

En esta actividad propondremos una solución al problema de los interbloqueos entre las hormigas, rompiendo alguna de las condiciones de Coffman a excepción de la de “Exclusión mutua”. Para ello, una hormiga realizará un cambio de dirección pasado un determinado tiempo de espera.

En el código, dentro del bucle while del método move(), implementaremos una estructura condicional que se ejecutará solo si “!ocupada[dest.x][dest.y].await(300, TimeUnit.MILLISECONDS)”. Dentro de la misma, se ejecutará v.chgDirr(a) y, posteriormente “dest = v.dest(a)”. Eliminaremos la sentencia v.retry(a) y el resto del código quedará igual.

Esta solución rompe con la condición de Coffman de retención y espera ya que, si una hormiga no puede acceder a la celda solicitada porque está ocupada, elige una nueva celda a la que desplazarse y deja libre la celda que ocupaba. La solución se incluye dentro de las estrategias de prevención de interbloqueos.

En todas las actividades anteriores se podría presentar en algún momento un problema de interbloqueo, si dos o más hormigas llegan a una espera circular.

Para resolver el problema del interbloqueo, se debe romper alguna de las condiciones de Coffman, excepto la de exclusión mutua (es decir, las hormigas deben de seguir respetando la restricción de no estar más de una en la misma celda del territorio).

Se puede resolver de varias formas diferentes:

-Se podría usar los métodos de tipo wait(long timeout, int nanos) o bien await(long timeout, TimeUnit unit) para esperar un máximo de tiempo.

-Se podría permitir que las hormigas pudieron ser "levantadas" del territorio y recolocadas en él posteriormente, para simular por ejemplo la expropiación de un recurso.

-Se podría establecer un lock por cada celda y hacer uso de los métodos tryLock(), tryLock(long timeout, TimeUnit unit) isLocked()...para determinar si una hormiga tiene que reconsiderar la celda a la cual desea ir. Se podría establecer un orden total de acceso en las celdas.