

Pràctiques de laboratori

Problema de les formigues

(1 sessió)

Concurrencia i Sistemes Distribuïts

Introducció

Esta pràctica té com a objectiu analitzar i completar un programa concurrent en el qual s'apliquen diferents condicions de sincronització entre fils, utilitzant algunes de les eines Java proporcionades en la biblioteca *java.util.concurrent*. Quan hagueu conclòs sabreu:

- Utilitzar de forma adequada l'eina *ReentrantLock*.
- Generar *Condicions* associades a un determinat *ReentrantLock* i utilitzar-les de forma apropiada.
- Utilitzar barreres per a la sincronització dels fils.
- Aplicar solucions per al problema dels interbloquejos.

Esta pràctica es realitza en grup, on cada **grup ha d'estar format per dues persones**.

La durada estimada d'esta pràctica és d'una setmana. Recordeu que necessitareu destinar una mica de temps del vostre treball personal per a concloure la pràctica. Utilitzeu per a això els laboratoris docents de l'assignatura i les aules informàtiques del centre en horaris de lliure accés. Podeu utilitzar també el vostre ordinador personal.

Al llarg de la pràctica veureu que hi ha una sèrie d'exercicis a realitzar. Es recomana resoldre'ls i anotar els seus resultats per facilitar l'estudi posterior del contingut de la pràctica.

El problema de les formigues

Es disposa d'un territori en el qual viu un conjunt de formigues. El territori es modela com una matriu rectangular $N \times N$, en què N és un valor a definir per l'usuari (per defecte, val 10). Per la seua banda, cada formiga s'implementa com un fil Java, amb una posició inicial aleatòria dins el territori. Cada formiga es mou lliurement pel territori, movent-se en cada moment a una cel·la contigua a la qual es troba (o siga, amunt, avall, esquerra o dreta). En este exemple, cada formiga realitzarà un total de 3 moviments (steps), i per defecte es llançaran 15 formigues.

Com restricció als moviments de les formigues, es disposa de la següent norma:

- A cada cel·la de la matriu pot haver com a màxim una sola formiga.

Es pretén resoldre este problema utilitzant el concepte de **monitor**. Per a això, el territori actuarà com un monitor, de manera que ofereisca mètodes sincronitzats per a l'actualització dels paràmetres del territori. A més, quan una formiga vullga desplaçar-se a una cel·la i esta es trobe ocupada, s'haurà de suspendre en una variable condició i quedarà allà suspesa fins que siga reactivada per una altra formiga.

Hi ha dues variants per atacar este problema:

- a) Definir una única variable condició associada a tot el territori, on les formigues es suspenen si no poden desplaçar-se a la cel·la desitjada.
- b) Definir una variable condició per a cada cel·la de la matriu territori. D'esta manera, cada formiga es suspèn en la variable condició associada a la cel·la a la que vol desplaçar-se (si està ocupada).

Inicialment, s'ha implementat un monitor utilitzant la sincronització bàsica que ofereix Java mitjançant el *lock intrínsec* associat a la classe *Object*. D'esta manera, els mètodes del territori porten l'etiqueta **synchronized** (que garanteix les operacions de tancament i obertura del *lock* intrínsec a l'entrada i eixida del mètode, respectivament). Així mateix, es disposa d'una única variable condició (implícita) i s'empra l'operació *wait()* per suspendre a un fil a la condició implícita associada al *lock*; i l'operació *notifyAll()* per reactivar a tots els fils suspesos en esta condició implícita.

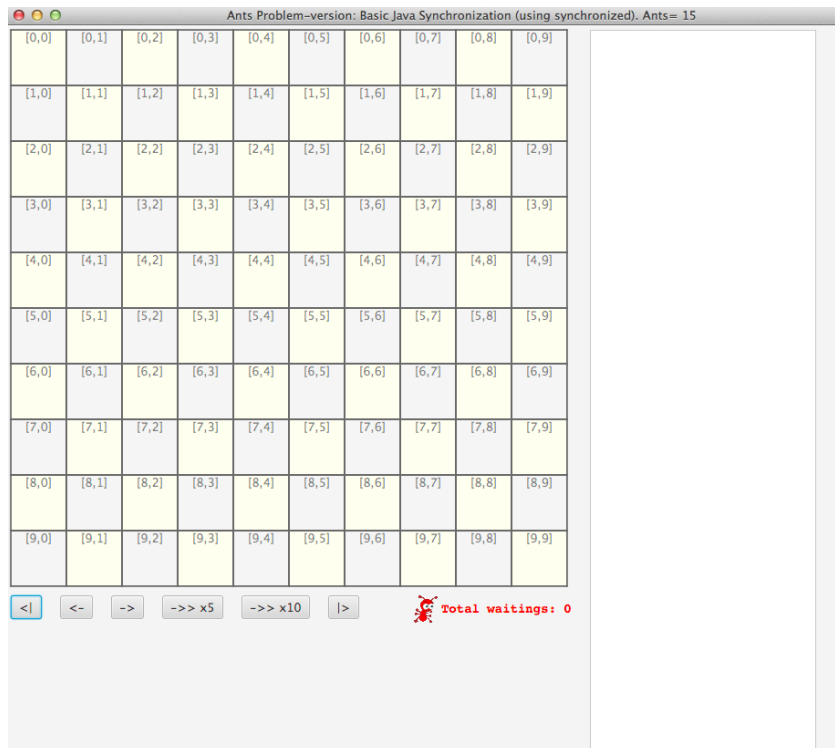
Codi proporcionat

Podeu descarregar el codi necessari per a la pràctica 3 des del lloc Poliformat de l'assignatura (carpeta "Recursos/Materiales para el laboratorio/Práctica 3: Hormigas", fitxer "TheAntsProblem.jar").

El codi proporcionat soluciona els problemes d'exclusió mútua i sincronització condicional, encara que pot presentar interbloquejos entre dos o més formigues quan es produeix entre elles una espera circular.

La classe *TheAntsProblem* conté el mètode principal *main*, que podrà llançar a execució sense arguments. Des d'un terminal, també pot executar directament l'aplicació sense arguments amb la instrucció: *java -jar TheAntsProblem.jar*.

En executar l'aplicació, es mostrarà una pantalla semblant a la següent figura:



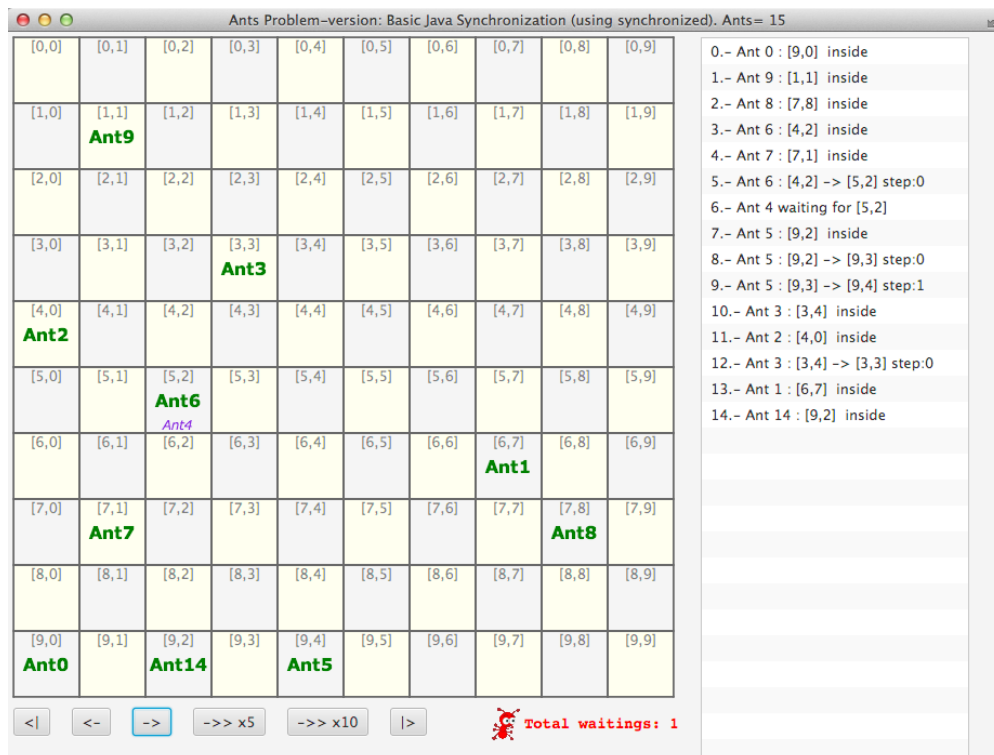
A la part esquerra es mostra el territori, amb cadascuna de les seues cel·les. A la part dreta (inicialment en blanc) s'anirà mostrant l'execució del problema.

Baix del territori disposem d'una sèrie de botons que ens permetran avançar o retrocedir en l'execució del problema. Estos botons són:

- Per anar a l'inici de l'execució.
- Per retrocedir un pas de l'execució.
- Per avançar un pas de l'execució.
- Per avançar 5 passos de l'execució.
- Per avançar 10 passos de l'execució.
- Per anar al final de l'execució.

A més, el missatge "Total waitings" ens anirà mostrant el total d'esperes (*waitings*) que realitzen les formigues en cada pas de l'execució. Recordem que una formiga haurà d'esperar si la cel·la a la que vol anar està ocupada.

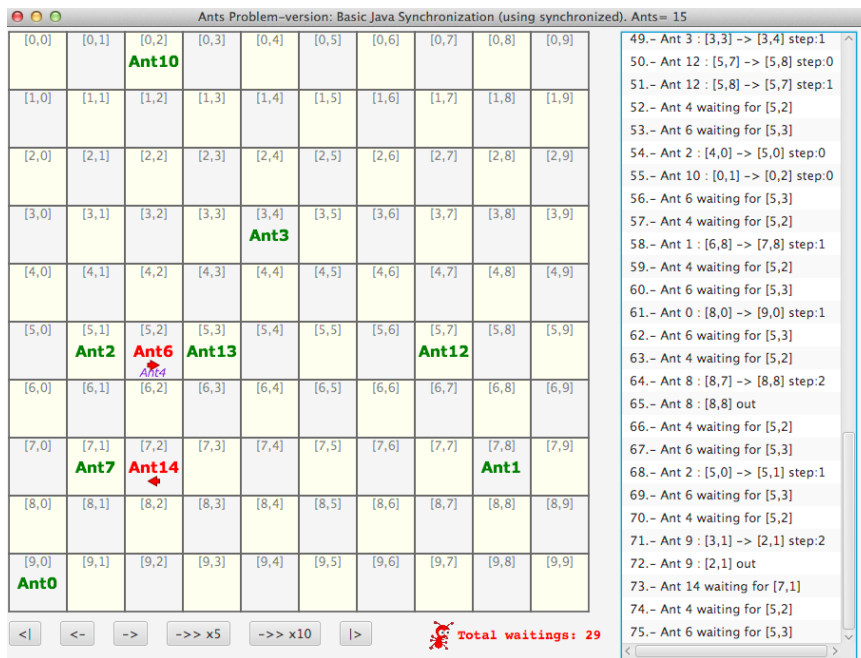
Quan avancem en l'execució, obtenim una pantalla semblant a la següent figura:



Com hem comentat, a la part dreta es van mostrant els passos realitzats en l'execució. En concret, en este exemple s'han realitzat ja 14 passos.


- La línia **0.- Ant 0: [9,0] inside** indica que, al pas 0, la formiga 0 s'ha situat en el territori a la cel·la [9,0] i comença la seua execució. Per cada formiga, es tindrà una única línia d'este tipus, amb el missatge "inside". A més, podem veure que la formiga **Ant0** s'ha col·locat correctament en el territori en la cel·la corresponent (està de color verd).
- La línia **5.- Ant 6 [4,2] -> [5,2] step:0** indica que, al pas 5, la formiga 6 s'ha desplaçat de la cel·la [4,2] a la cel·la [5,2], en el seu moviment (*step*) nombre 0. Com hem comentat abans, cada formiga realitzarà un total de 3 moviments (des *step 0* fins *step 2*).
- La línia **6.- Ant 4 waiting for [5,2]** indica que la formiga 4 està esperant que la cel·la [5,2] quede lliure per a col·locar-se en ella. Esta formiga encara no ha estat col·locada en el territori, però casualment la seua casella inicial està a hores d'ara ocupada. Per tant, ha d'esperar. Esta situació s'ha representat al territori amb la formiga en color morat i un text més menut.
 - Important: la formiga 4 no està encara en el territori (encara no tenim el missatge "inside"), pel que no s'incompleix la restricció imposada al territori..

En avançar amb l'execució d'este exemple, obtindrem una pantalla semblant a la següent:



Podem veure que la formiga **Ant14** està remarcada en roig en el territori, ja que està esperant a anar a la cel·la contigua (ocupada per una altra formiga). El mateix li passa a la formiga **Ant6**.

- La línia **73.- Ant 14 waiting for [7,1]** indica que la formiga 14 vol anar a la cel·la [7,1] però esta cel·la està ocupada. Es mostrarà sempre en **roig** el text d'una formiga, situada ja al territori, que estigui esperant que una determinada cel·la quedi lliure. A més, una fletxa roja situada baix de la formiga ens indicarà la cel·la concreta per la qual està esperant.

Si avancem fins al final de l'execució del problema (per exemple, amb el botó ), podem veure una pantalla semblant a la següent:

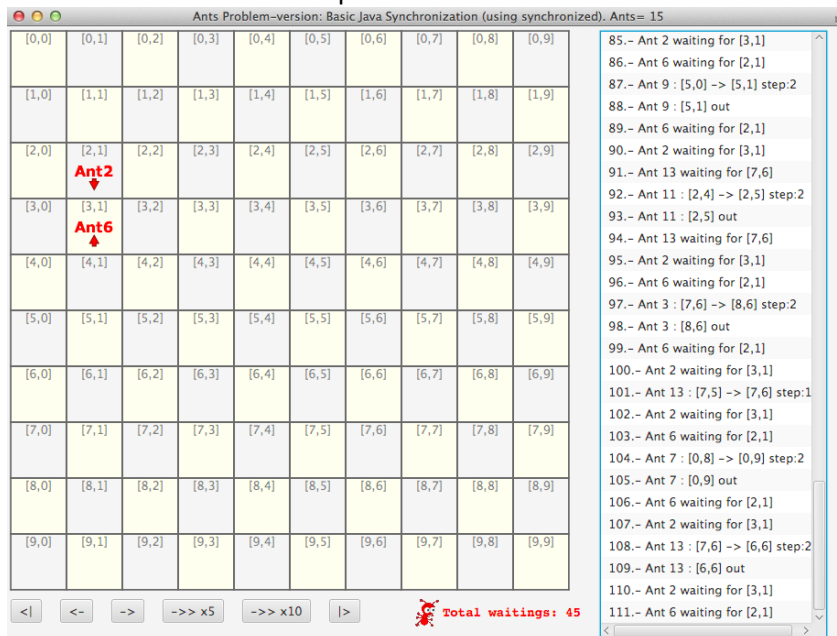


Si tota l'execució ha estat correcta, al final tindrem el missatge de **"done"** i no hi haurà ja formigues al territori, ja que totes elles hauran acabat els seus moviments i hauran eixit del territori. Per exemple:

- La línia **117.- Ant 4: [6,4] out** indica que la formiga 4 ha acabat els seus moviments i allibera la cel·la en la qual estava, eixint així del territori.

A més, en el missatge "Total waitings" tindrem el nombre total d'esperes (*waitings*) que han realitzat les formigues en este exemple. En este cas, 43 esperes.

No obstant això, podria donar-se el cas que dos o més formigues s'hagueren quedat bloquejades entre si amb una espera circular, produint-se un interbloqueig. En este cas, en arribar al final de l'execució del problema obtindríem un resultat similar al següent:



En este cas, veurem que al final no apareix el missatge "done". I que a més en el territori estan les formigues (en roig) que han quedat interbloquejades. Les fletxes ens mostren clarament esta situació.

Anàlisi del codi

El codi proporcionat implementa diferents classes, que podem classificar en:

- Classes opaques.- Necessàries per al correcte funcionament de l'aplicació, però sense interès per a l'alumne. **No s'han de modificar.**
 - Log, LogItem*
- Classes transparents.- Interessa mirar la seua implementació (almenys algun aspecte de la mateixa). Però **no s'han de modificar.**
 - Ant.* Classe que s'estén de Thread i que implementa a la formiga. En el seu mètode *run()*, la formiga se situa en el territori i realitza fins a 3 moviments. A cada moviment, es desplaça aleatòriament a una cel·la contigua. Finalment, la formiga s'elimina del territori.
 - TheAntsProblem.* És la **classe principal**. D'esta classe interessa conèixer cert codi del mètode *start*, en concret:

- La creació del territori, de mida *tsize*. Esta variable és el primer argument de l'aplicació i permet valors de territori de [2..12], sent el valor per defecte 10.

```
int tsize=integer(arg,0,10,2,12);
// 1st arg: size of Territory (interval [2..12], default 10)
Territory t = new Territory(tsize,log);
```

- La creació i arrencada de les formigues. El nombre de formigues es defineix en la variable *Nants*, que al seu torn és el segon argument de l'aplicació i permet valors de [2..30], sent 15 el valor per defecte.

```
int Nants=integer(arg,1,15,2,30);
//2nd arg: number of ants (interval [2..30], default 15)
Ant[] ants = new Ant[Nants];
for (int i = 0; i < Nants; i++) {
    int x = (int) (rnd.nextDouble() * tsize); //X initial position
    int y = (int) (rnd.nextDouble() * tsize); //Y initial position
    ants[i] = new Ant(i, x, y, t, steps, barrier);
    ants[i].start();
}
```

- Classe a modificar.** - Classe el codi de la qual hem de modificar.
 - Territory.** Classe que actua com un monitor i controla l'accés de les formigues al territori. Les activitats 1 i 2 indiquen què s'ha d'actualitzar en esta classe.

Activitat 0 (Sincronització bàsica en Java)

Llanceu diverses execucions del codi proporcionat i completeu la següent taula, on s'indique per a cada execució:

- el nombre total d'esperes realitzades per les formigues (*total waitings*)
- i la quantitat de formigues que s'han quedat al final bloquejades.

Prova	Total Waitings	Nº formigues interbloquejades
1	11	0
2	64	5
3	58	4
4	60	3
5	43	2
6	8	0

Indiqueu, de mitjana, quantes esperes realitzen en total les formigues, quan:

- a) L'execució acaba de forma adequada, sense interbloquejos.

mitjana d'esperes 9,5

- b) Es produeix un interbloqueig.

mitjana d'esperes 56,25

Activitat 1 (Sincronització amb ReentrantLocks en Java)

Implementeu una nova versió del problema, utilitzant les eines **ReentrantLock** i **Condition** proporcionades a la biblioteca *java.util.concurrent*, que aplique la variant de solució: "a) una única variable condició associada a tot el territori".

Per a això, substituïu en la classe **Territory** de manera apropiada el *lock intrínsec* associat a la classe *Object* (i, per tant, l'ús de l'etiqueta *synchronized*) per la utilització d'un *ReentrantLock* amb una única variable *Condition* associada.

A més, a la classe **Territory** canvieu la descripció del territori per:

`String description="ReentrantLock with 1 Condition"`

Llanceu diverses execucions del nou codi implementat i compareu els seus resultats amb el codi original, per a condicions similars del problema. Per a això, completeu la taula següent.

Prova	Total Waitings	Nº formigues interbloquejades
1	6	2
2	16	0
3	52	3
4	117	6
5	37	2
6	47	3

I calculeu, de mitjana, quantes esperes realitzen en total les formigues, quan:

a) L'execució acaba de forma adequada, sense interbloquejos.

16

b) Es produeix un interbloqueig.

51.8

Qüestió 1.1: Si utilitzem les mateixes grandàries de territori i nombre de formigues, podem veure diferències significatives entre utilitzar el *lock intrínsec* (*synchronized*) o *ReentrantLock*? Per exemple, es produeixen menys reactivacions "innecessàries" de fils?

Enquant a execucions acabades el nombre desperes es major en el reentrantLock pero en les que es produeix interbloqueig no s'aprecia la diferencia.

Activitat 2 (Ús de diverses variables Condition)

Implementeu una nova versió del problema, utilitzant les eines **ReentrantLock** i **Condition** proporcionades a la biblioteca *java.util.concurrent*, que aplique la variant de solució: "b) una variable condició per cada cel·la de la matriu".

Per això, en la classe **Territory** utilitzeu un únic *ReentrantLock* associat al territori i creeu tantes variables condició *Condition* com cel·les tinga este territori.

A més, a la classe **Territory** canvieu la descripció del territori per:

String description="ReentrantLock with multiple Conditions"

Llanceu diverses execucions del nou codi implementat i compareu els seus resultats amb el codi de l'Activitat 1, per a condicions similars del problema.

Per a això, completeu la taula següent:

Prova	Total Waitings	Nº formigues interbloquejades
1	8	5
2	5	2
3	7	3
4	8	7
5	3	0
6	8	3

I calculeu, de mitjana, quantes esperes realitzen en total les formigues, quan:

a) L'execució acaba de forma adequada, sense interbloquejos.

b) Es produeix un interbloqueig.

Qüestió 2.1: Si utilitzem les mateixes grandàries de territori i nombre de formigues, podem veure diferències significatives entre utilitzar una única condició associada a tot el territori (Activitat 1) o bé una condició per cada cel·la (Activitat 2)? Per exemple, es produeixen menys reactivacions "innecessàries" de fils?

si, hi ha una diferencia significativa degut a que ara sols despertem a la formiga que sabem que va a poder continuar.

Qüestió 2.2: Si utilitzem una condició per cada cel·la, podríem obtenir la següent seqüència de línies per pantalla? Què representen cadascuna de les línies amb "waiting for"?

6.- Ant 4 : [9,4] inside
7.- Ant 5 : [2,7] inside
8.- Ant 4 waiting for [9,3]
9.- Ant 6 : [7,9] inside
10.- Ant 7 : [2,3] inside
11.- Ant 7 : [2,3] -> [3,3] step:0
12.- Ant 8 : [8,2] inside
13.- Ant 9 : [3,6] inside
14.- Ant 10 : [0,8] inside
15.- Ant 11 : [2,1] inside
16.- Ant 12 : [2,3] inside
17.- Ant 4 waiting for [9,3]

18.- Ant 3 : [3,9] -> [2,9] step:0
19.- Ant 4 waiting for [9,3]
20.- Ant 6 : [7,9] -> [7,8] step:0
21.- Ant 4 waiting for [9,3]
22.- Ant 10 : [0,8] -> [1,8] step:0
23.- Ant 4 waiting for [9,3]
24.- Ant 1 : [6,0] -> [7,0] step:0
25.- Ant 1 : [7,0] -> [7,1] step:1
26.- Ant 14 : [9,3] -> [8,3] step:0
27.- Ant 0 waiting for [3,3]
28.- Ant 4 : [9,4] -> [9,3] step:0
29.- Ant 1 : [7,1] -> [8,1] step:2
30.- Ant 1 : [8,1] out
31.- Ant 0 waiting for [3,3]

No, degut a que la formiga 4 s'activa i es suspen moltes vegades i en aquesta implementacio sols despertem una formiga quan sabem que va a poder continuar.

waiting for representa una formiga que esta esperant.

ACTIVITATS OPTATIVES

A continuació s'indiquen diverses activitats que l'alumne pot realitzar de forma optativa. **Estes activitats no seran avaluades en el test associat a esta pràctica.**

Activitat 3 (Ús de barreres per a la sincronització de fils)

Implementeu una nova versió del problema, utilitzant com a base el codi de qualsevol de les activitats anteriors, en la qual les formigues esperen a que totes elles estiguen col·locades en una cel·la (o estiguen esperant a ser col·locades, si la cel·la està ocupada) abans de fer el primer moviment. Per això, utilitzeu un dels dos tipus de barrera (*CyclicBarrier* o *CountDownLatch*) que s'ofereixen a la biblioteca *java.util.concurrent*.

En la classe **Territory** canvieu la descripció del territori per:
String description="Using Barriers"

Qüestió 3.1: Quin tipus de barrera heu utilitzat? Quins avantatges li ofereix, davant de l'altre tipus de barrera? [CountDownLatch](#), ya que el problema de la *CyclicBarrier* es que una vegada totes les formigues situades, al lliurarles la barrera es reinicia i torna a esperar.

Qüestió 3.2: Si utilitzem barreres d'acord al que indica esta activitat, podríem obtindre la següent seqüència de línies per pantalla? Per què?

```
0.- Ant 0 : [8,0] inside
1.- Ant 11 : [2,6] inside
2.- Ant 13 : [8,5] inside
3.- Ant 14 : [5,0] inside
4.- Ant 12 : [8,2] inside
5.- Ant 10 : [8,7] inside
6.- Ant 9 waiting for [5,0]
7.- Ant 12 : [8,2] -> [8,1] step:0
8.- Ant 7 : [0,8] inside
9.- Ant 6 : [3,1] inside
10.- Ant 14 : [5,0] -> [4,0] step:0
11.- Ant 5 : [7,2] inside
```

Activitat 4 (Gestió de interbloquejos)

En totes les activitats anteriors podria presentar-se en algun moment un problema d'interbloqueig, si dos o més formigues arriben a una espera circular. En esta activitat es pretén que l'alumne proposi una solució per al problema dels interbloquejos entre les formigues, de manera que es trenqui alguna de les condicions de Coffman, excepte la d'exclusió mútua (és a dir, les formigues han de seguir respectant la restricció de no estar més d'una en la mateixa cel·la del territori).

Esta activitat es pot abordar de diverses formes possibles. A continuació indiquem algunes guies de com abordar-ho:

- Es podria fer ús de mètodes de tipus *wait(long timeout, int nanos)* o bé *await(long timeout, TimeUnit unit)* per esperar un màxim de temps.
- Es podria permetre que les formigues pogueren ser "alçades" del territori i recol·locades en ell posteriorment, per simular per exemple l'expropiació d'un recurs.
- Es podria establir un *lock* per cada cel·la i fer ús dels mètodes *tryLock()*, *tryLock(long timeout, TimeUnit unit)*, *isLocked()*...per determinar si una formiga ha de reconsiderar la cel·la a la qual desitja anar. Es podria establir un ordre total d'accés a les cel·les.