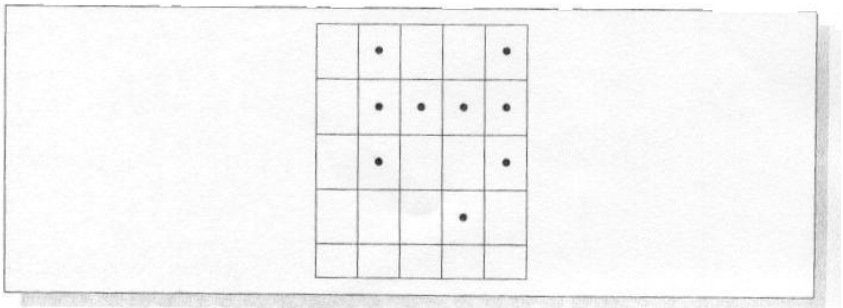
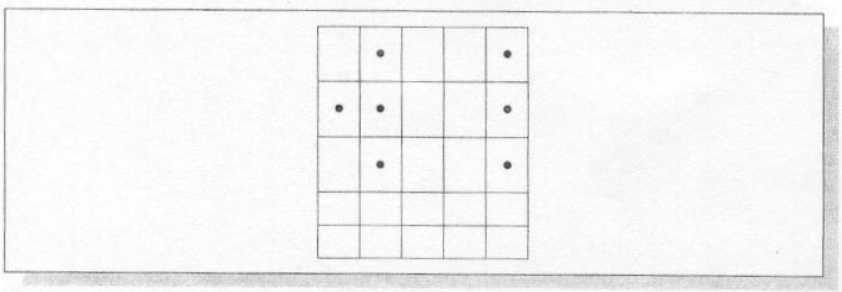
9.9
(P556)

Write a program for playing Life (a common program that simulates cellular automata). Given a two-dimensional array of cells in which each cell is either dead or alive, the values for the next generation of cells are based on the following rules. Any living cell that is adjacent to exactly two or three other living cells continues to live. Any living cell that is adjacent to any less than two or any greater than three other living cells will die. Any dead cell that is adjacent to exactly three other living cells comes to life. For example, if the first generation was a five-by-five array with the live cells filled with dots as shown,



then the next generation would appear as follows:



Using the initial configuration shown previously, compute the next four generations.

小考11

Output:

```
Generation 1
-*--*
-****
-*--*
---*-
-----

Generation 2
-*--*
**--*
-*--*
-----
-----

Generation 3
**---
*****
**---
-----
-----

Generation 4
*--*-
---*-
*--*-
-----
-----

Generation 5
-----
--***
-----
-----
-----
```

```
(deffacts MAIN::initial
  (cell (row 1) (column 1) (status -))
  (cell (row 1) (column 2) (status *))
  (cell (row 1) (column 3) (status -))
  (cell (row 1) (column 4) (status -))
  (cell (row 1) (column 5) (status *))
  (cell (row 2) (column 1) (status -))
  (cell (row 2) (column 2) (status *))
  (cell (row 2) (column 3) (status *))
  (cell (row 2) (column 4) (status *))
  (cell (row 2) (column 5) (status *))
  (cell (row 3) (column 1) (status -))
  (cell (row 3) (column 2) (status *))
  (cell (row 3) (column 3) (status -))
  (cell (row 3) (column 4) (status -))
  (cell (row 3) (column 5) (status *))
  (cell (row 4) (column 1) (status -))
  (cell (row 4) (column 2) (status -))
  (cell (row 4) (column 3) (status -))
  (cell (row 4) (column 4) (status *))
  (cell (row 4) (column 5) (status -))
  (cell (row 5) (column 1) (status -))
  (cell (row 5) (column 2) (status -))
  (cell (row 5) (column 3) (status -))
  (cell (row 5) (column 4) (status -))
  (cell (row 5) (column 5) (status -))
  (dimensions (rows 5) (columns 5))
  (current-generation 0)
  (last-generation 4)
)
```

```
(defmodule MAIN (export deftemplate ?ALL)

   (deftemplate MAIN::cell (slot row) (slot column) (slot status))
   (deftemplate MAIN::dimensions (slot rows) (slot columns))

   (defrule MAIN::life-init
    (last-generation ?l)
    ?f <- (current-generation ?c&:(<= ?c ?l))
    =>
    (retract ?f)
    (assert (current-generation (+ ?c 1)))
    (focus PRINT COMPUTE-NEIGHBORS NEXT-GENERATION))

(defmodule PRINT (import MAIN deftemplate ?ALL)

(defmodule COMPUTE-NEIGHBORS (import MAIN deftemplate cell)
                             (export deftemplate neighbor-sum))

   (deftemplate COMPUTE-NEIGHBORS::neighbor
                             (slot row) (slot column) (multislot live-cell))

   (deftemplate COMPUTE-NEIGHBORS::neighbor-sum
                             (slot row) (slot column) (slot value))

(defmodule NEXT-GENERATION (import MAIN deftemplate cell)
        (import COMPUTE-NEIGHBORS deftemplate neighbor-sum))
```

請到雲端學院
下載事實資料
cell.txt

```
(defrule PRINT::start-print
  (current-generation ?g)
  =>
  (assert (print-row 1))
  (assert (print-column 1))
  (printout t crlf "Generation " ?g crlf))

(defrule PRINT::cell-print
  (dimensions (rows ?nr) (columns ?nc))
  ?f <- (print-column ?c&:(<= ?c ?nc))
  (print-row ?r&:(<= ?r ?nr))
  (cell (row ?r) (column ?c) (status ?s))
  =>
  (printout t ?s)
  (retract ?f)
  (assert (print-column (+ 1 ?c))))
```

```
(defrule COMPUTE-NEIGHBORS::make-neighbors
  (cell (row ?x) (column ?y) (status *))
  =>
  (assert (neighbor (row (- ?x 1)) (column (- ?y 1)) (live-cell ?x ?y)))
  (assert (neighbor (row (- ?x 1)) (column ?y) (live-cell ?x ?y)))
  (assert (neighbor (row (- ?x 1)) (column (+ 1 ?y)) (live-cell ?x ?y)))
  (assert (neighbor (row ?x) (column (- ?y 1)) (live-cell ?x ?y)))
  (assert (neighbor (row ?x) (column (+ 1 ?y)) (live-cell ?x ?y)))
  (assert (neighbor (row (+ 1 ?x)) (column (- ?y 1)) (live-cell ?x ?y)))
  (assert (neighbor (row (+ 1 ?x)) (column ?y) (live-cell ?x ?y)))
  (assert (neighbor (row (+ 1 ?x)) (column (+ 1 ?y)) (live-cell ?x ?y))))

(defrule COMPUTE-NEIGHBORS::cleanup-neighbors
  ?f1 <- (neighbor (row ?x) (column ?y) (live-cell $?))
  (not (cell (row ?x) (column ?y) (status ?)))
  =>
  (retract ?f1))

(defrule COMPUTE-NEIGHBORS::create-neighbor-sum
  (cell (row ?x) (column ?y) (status ?))
  =>
  (assert (neighbor-sum (row ?x) (column ?y) (value 0))))
```

```
(defrule NEXT-GENERATION::continue-life
  ?a <- (cell (row ?x) (column ?y) (status *))
  ?b <- (neighbor-sum (row ?x) (column ?y) (value 2|3))
  =>
  (retract ?a ?b)
  (assert (cell (row ?x) (column ?y) (status *))))
```