3. EXTENSIÓN DEL VOCABULARIO BASICO

En este capítulo se define la forma de enseñarle a Karel nuevas palabras de programación y la técnica de refinamiento paso a paso para resolver problemas.

3.1 INSTRUCCIÓN LOOP

Algunas veces, cuando programamos a Karel, es necesario hacerle repetir una instrucción un número de veces. Anteriormente, la solución de los problemas se efectuó escribiendo una instrucción tantas veces como fuera necesario.

La instrucción loop nos permite simplificar el programa escribiendo esta instrucción una sola vez e indicando el número de veces que se debe repetir, es una ayuda sintáctica para comprimir el código.

La instrucción loop se escribe así:

```
loop (número positivo)
{
  <instrucción>
}
```

Karel repite las instrucciones que están escritas dentro del bloque {…} <número positivo> veces. Si lo que queremos es que repita sólo una instrucción, no podemos omitir el {…}.

Por ejemplo, si queremos que Karel gire a la derecha tres veces, esto lo podemos abreviar utilizando la instrucción loop así:

```
loop(3)
 {
   turnLeft();
 }
```

Veamos el siguiente problema y su solución utilizando la instrucción loop: Karel está en el origen, mirando al este. En todas las esquinas de la calle 1 desde la avenida 1 hasta la 23 hay un pito que Karel debe recoger. Karel termina en la esquina (1,24) con los 23 pitos en su bolsa.

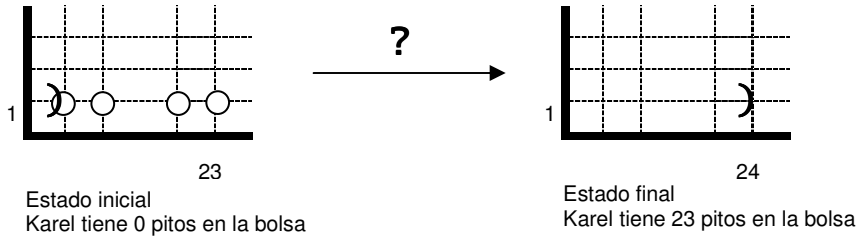


Figura. 76 Posibles estados inicial y final de Karel para la solución del problema utilizando la instrucción loop

Solución: Karel debe repetir el paso básico de recoger un pito y moverse, ya que, se conoce exactamente el número de pitos del mundo y el número de veces que se debe repetir este paso (23). Utilizando la instrucción loop se tiene la siguiente solución:

```
ur_robot karel(1,1,East,0);
loop(23)
 {
   karel.pickBeeper();
   karel.move();
 }
```

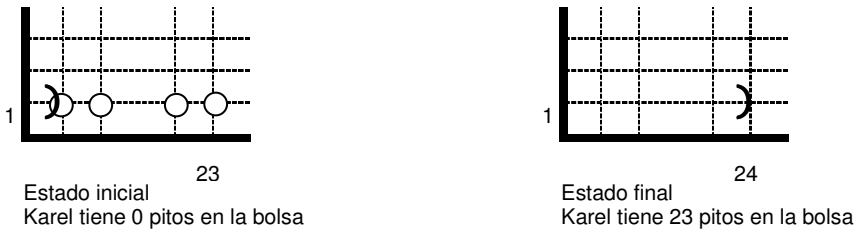


Figura. 77 Posibles estados inicial y final de Karel en la solución del problema utilizando la instrucción loop

3.2 DEFINICIÓN DE NUEVAS INSTRUCCIONES

Karel tiene un mecanismo que le permite aprender, esto nos da la posibilidad de enseñarle nuevas instrucciones adicionales a las primitivas, para que realice acciones más complejas y así ampliarle su vocabulario. Cada instrucción o palabra nueva del vocabulario tiene que estar definida en términos de primitivas o de instrucciones ya aprendidas por Karel. Una instrucción nueva se define dando su nombre y su significado así:

```
class <nombre del robot> : ur_robot
{
    <instrucción>
};
void <nombre de la clase> ::<nuevo método>
{
    <instrucción>
}
```

Donde <instrucción> puede ser una sola declaración o un bloque {…} de instrucciones separadas por ';' (punto y coma).

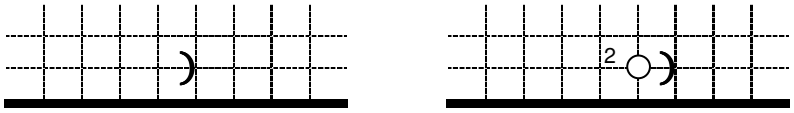
Cuando una nueva instrucción es única, no se puede suprimir los corchetes {...} que delimitan la definición de la instrucción.

Ejemplo, podemos definir la instrucción Gire izquierda como:

```
class zurdo : ur_robot
{
Gire Izquierda ();
};
void Zurdo() :: Gire Izquierda ()
{
    turnLeft();
}
```

Podemos definir una instrucción Pongaysiga para que Karel coloque dos pitos en su esquina y se mueva una cuadra a la derecha, dirección hacia la cual está mirando inicialmente:

```
void Pongaysiga()
      {
           putBeeper();
           putBeeper();
          move();
}
```



Estado inicial
Karek tiene X pitos en la bolsa

Estado final
Karel tiene X-2 pitos en la bolsa

Figura. 78 Posibles estados de Karel en la instrucción Pongaysiga

En un programa la declaración de nuevas instrucciones se coloca entre el comienzo de la clase y el final del bloque de declaraciones.

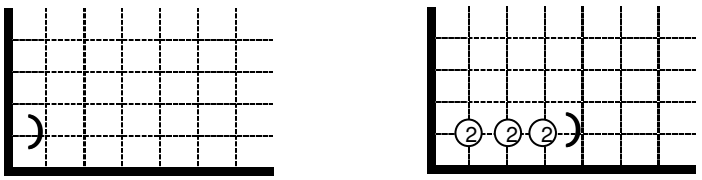
Por ejemplo: entre {después digite class <nombre del robot>: ur_robot y el final del bloque delimitado por};

En un programa la definición de nuevas instrucciones se coloca entre el final del bloque de declaraciones y el comienzo del bloque de ejecución task.

Por ejemplo entre {…}; se escriben primero las instrucciones que sólo utilizan primitivas para su definición, luego las que hacen referencia a las primeras y así sucesivamente hasta llegar al bloque principal.

Entre una definición y la siguiente y entre una definición y el bloque de ejecución no debe colocarse el símbolo ‘;’ (punto y coma).

Problema: Uso de la instrucción Pongaysiga. Para el problema del repartidor



Estado inicial
Karel tiene 7 pitos en la bolsa

Estado final
Karel tiene 1 pito en la bolsa

Figura. 79 Posibles estados inicial y final para la solución del problema del repartidor

```
class Repartidor: robot
{
void Pongaysiga();
};
Vid Repartidor: Pongaysiga()
{
              putBeeper();
              putBeeper();
             move();
}
task
{
  Repartidor karel(1,1,East,7);
       loop(3)
        {
          karel.Pongaysiga();
        }
     karel.turnOff();
}
```

Karel comienza a ejecutar el bloque de ejecución. La primera instrucción es Pongaysiga, como ésta no es una primitiva, Karel busca en el diccionario su definición y la ejecuta, obedeciendo una tras otra, cada una de las instrucciones del bloque de definición de Pongaysiga; cuando termina de ejecutar este bloque continúa la ejecución del programa a partir del sitio donde encontró dicha instrucción.

3.3 TÉCNICA DE PROGRAMACIÓN POR REFINAMIENTO PASO A PASO

La técnica por refinamiento paso a paso consiste en dividir un problema grande en problemas más pequeños o subproblemas y éstos a su vez en otros más pequeños, hasta llegar a un nivel en que los problemas sean sencillos y fácilmente solucionables. La solución de un problema es, entonces, la unión de las soluciones de los subproblemas que a su vez pueden estar dadas como la unión de problemas más pequeños.

En el mundo de Karel para resolver problemas complejos utilizando esta técnica se usan las capacidades de definición de nuevas instrucciones para ir dando nombre a los subproblemas que se van identificando. De esta forma se tiene desde un principio, el bloque principal (de ejecución) del programa.

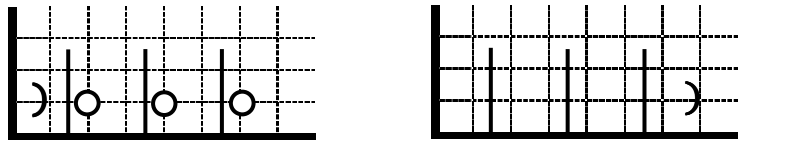
Cada subproblema complejo se resuelve de la misma forma y su solución se convierte en el cuerpo de la instrucción con que se “bautizó” el paso a paso.

La técnica por refinamiento paso a paso se describe así:

1. Hacer un plan general de solución, dividir el problema en subproblemas o pasos.
2. Darle nombre a los diferentes subproblemas que se van identificando, definir las nuevas instrucciones que se van a utilizar
3. Escribir el bloque principal del programa, definiendo los diferentes pasos y asignándoles nombres.
4. Resolver cada subproblema. Si el subproblema es todavía muy complejo, puede ser nuevamente dividido.

Ejemplos de la Técnica de Refinamiento Paso a Paso

Primer caso: Salto de Obstáculos. Karel está en el origen mirando al este. Frente a él hay tres obstáculos cada dos cuadras. Los obstáculos son de dos cuadras de altura; a la derecha de cada obstáculo, sobre la calle 1, se encuentra un pito. Programe a Karel para que salte los tres obstáculos, recoja los tres pitos y termine en la esquina (1,7), mirando al este.



Estado inicial
Karel tiene 0 pitos en la bolsa

Estado final
Karel tiene 3 pitos en la bolsa

Figura. 80 Posibles estados inicial y final de Karel para la solución del problema de obstáculos

Solución: Paso 1: Plan General

En la figura 80 se observa que existen tres obstáculos iguales y para cada uno, Karel debe saltarlo y recoger el pito que está a la derecha. Es lógico, entonces, partir el problema en tres subproblemas iguales:

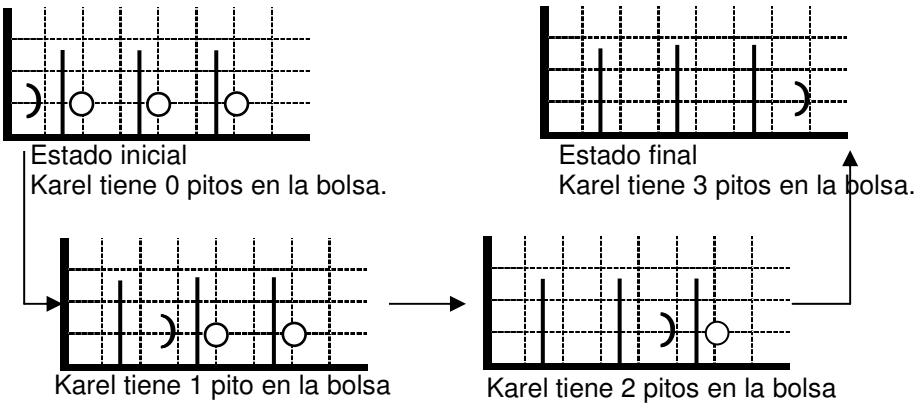


Figura. 81 Posibles estados

Paso 2: Darle nombre a los pasos

Como los tres pasos son iguales, sólo necesitamos definir una nueva instrucción, que vamos a llamar salte-y-recoja, que resuelva el subproblema de saltar un obstáculo y recoger el pito a su derecha.

Paso 3: Escribir el bloque principal de ejecución

El bloque de ejecución se construye pegando las soluciones de los subproblemas en el orden apropiado, en este caso obtenemos lo siguiente:

```
task
{
/* salto es el nombre de la clase */
 salto karel(1,1,East,0);
 loop(3)
  {
    karel.salte_y_recoja;
  }
 karel.turnOff();
}
```

Paso 4: Resolver los subproblemas

En este caso, sólo hay un subproblema, llamado salte-y-recoja, el cual podemos especificar gráficamente así:

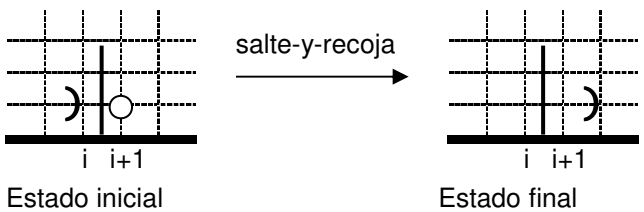


Figura. 82 Posibles estados

Como este problema todavía es un poco complicado para resolverlo con primitivas, podemos volverlo a partir, repitiendo los pasos seguidos para el problema original:

Paso 4.1. Plan general: el problema lo podemos partir en dos subproblemas de la siguiente forma (aunque esta no es la única posibilidad):

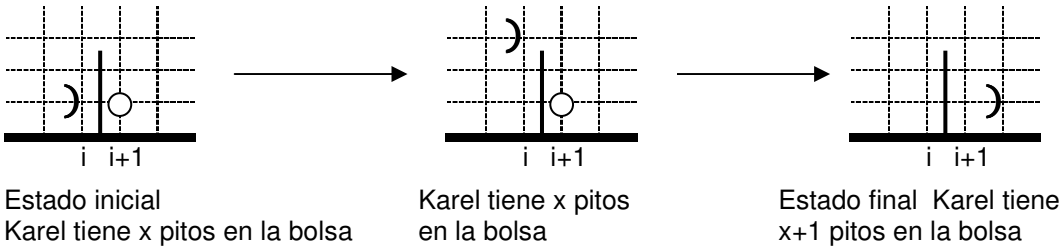


Figura. 83 Posibles estados

Paso 4.2. Nombrar los subproblemas: Como en este caso los dos subproblemas obtenidos son diferentes, debemos darles nombres distintos, por ejemplo, suba y baje respectivamente.

Paso 4.3. Bloque principal de salte-y-recoja: La solución (el cuerpo de su definición) es la unión de suba y baje:

```
void salto::salte_y_recoja()
 {
   suba();
   baje();
 }
```

Paso 4.4. Resolver los subproblemas: Los problemas suba y baje se pueden resolver directamente con primitivas y con la instrucción turnRight() cuya definición se reduce a hacer tres turnLeft()

```
void salto::suba()
 {
   turnLeft();
   move();
   move();
   turnRight();
}


void salto::baje()
 {
   move();
   turnRight();
   move();
   move();
   pickBeeper();
   turnLeft();
   move();
}
```

Para que karel pueda ejecutar correctamente una instrucción no primitiva, debe haberla aprendido antes, teniendo en cuenta esta condición, el programa completo sería el siguiente:

Programa Completo

```
class salto:robot
{
  void turnRight();
  void suba();
  void baje();
  void salte_y_recoja();
};

void salto:turnRight()
{
      turnLeft();
      turnLeft();
      turnLeft();
}

void salto::suba()
{
   turnLeft();
   move();
   move();
   turnRight();
}

void salto::baje()
 {
   move();
   turnRight();
   move();
   move();
```

```
   pickBeeper();
   turnLeft();
   move();
}

void salto::salte_y_recoja()
 {
  suba();
  baje();
}
task
{
/* salto es el nombre de la clase */
  salto karel(1,1,East,0);
  loop(3)
   {
     karel.salte_y_recoja();
   }
   karel.turnOff();
}
```

Segundo caso: Pista de Aterrizaje. En su primer trabajo en la base interestelar, Karel debía iluminar (con pitos) las pistas asignadas a las naves de los marcianos (las diagonales); por descuido iluminó las de los visitantes de Júpiter (verticales y horizontales). Ayude a Karel a cambiar la iluminación:

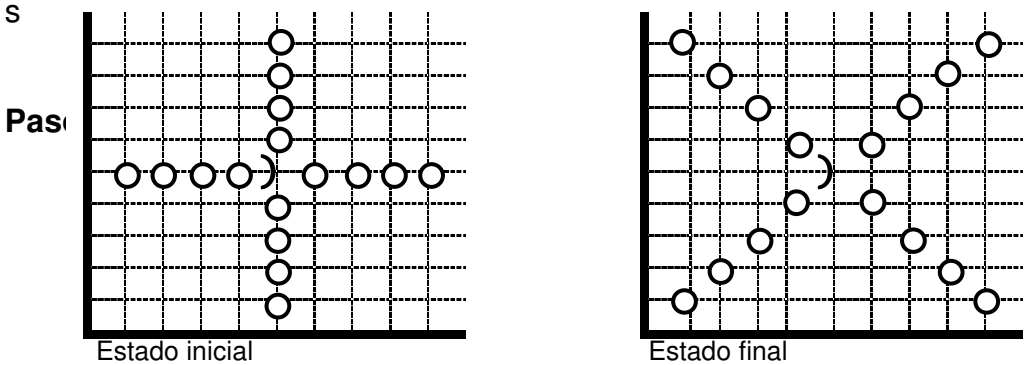


Figura 84 Posibles estados

La idea es dividir el problema en cuatro subproblemas iguales: rotar una fila de

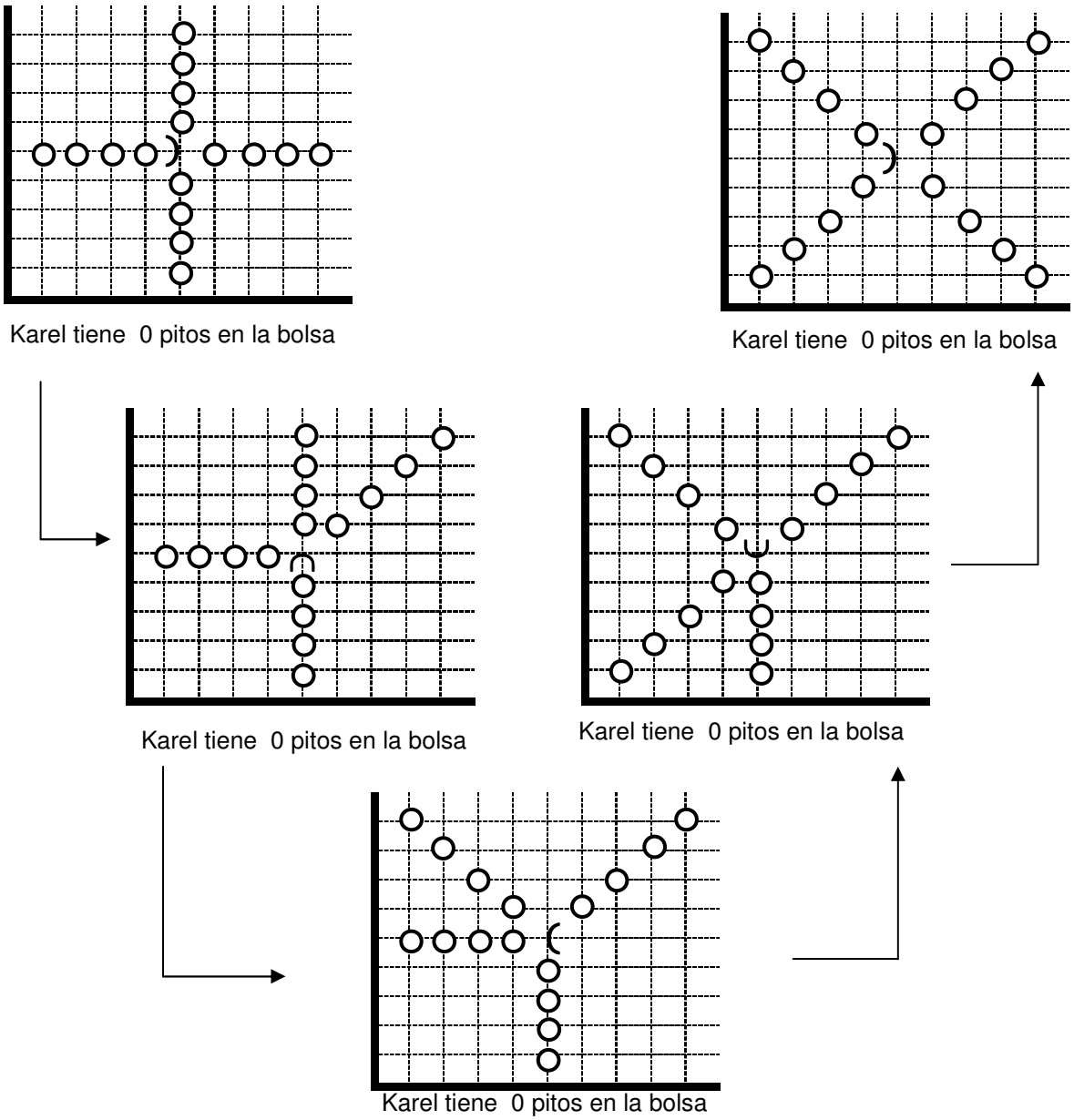


Figura 85 Especificación de los posibles estados

Paso 2: Darle nombre a los pasos

Definimos la instrucción rotar, en la cual Karel comienza mirando hacia la fila que va a recoger y termina mirando hacia la siguiente, después de haber colocado los pitos diagonalmente.

Paso 3: Escribir el bloque principal de ejecución

```
task
{
  /* Pista es el nombre de la clase */
  Pista karel(4,4,East,0);
  loop(4)
  {
    karel.rotar();
  }
  karel.turnOff();
}
```

Paso 4: Resolver los subproblemas

Puesto que rotar es todavía demasiado complicado para resolverlo con primitivas, lo vamos a partir de nuevo:

Paso 4.1. Plan general: El problema se puede dividir en tres subproblemas de la siguiente forma:

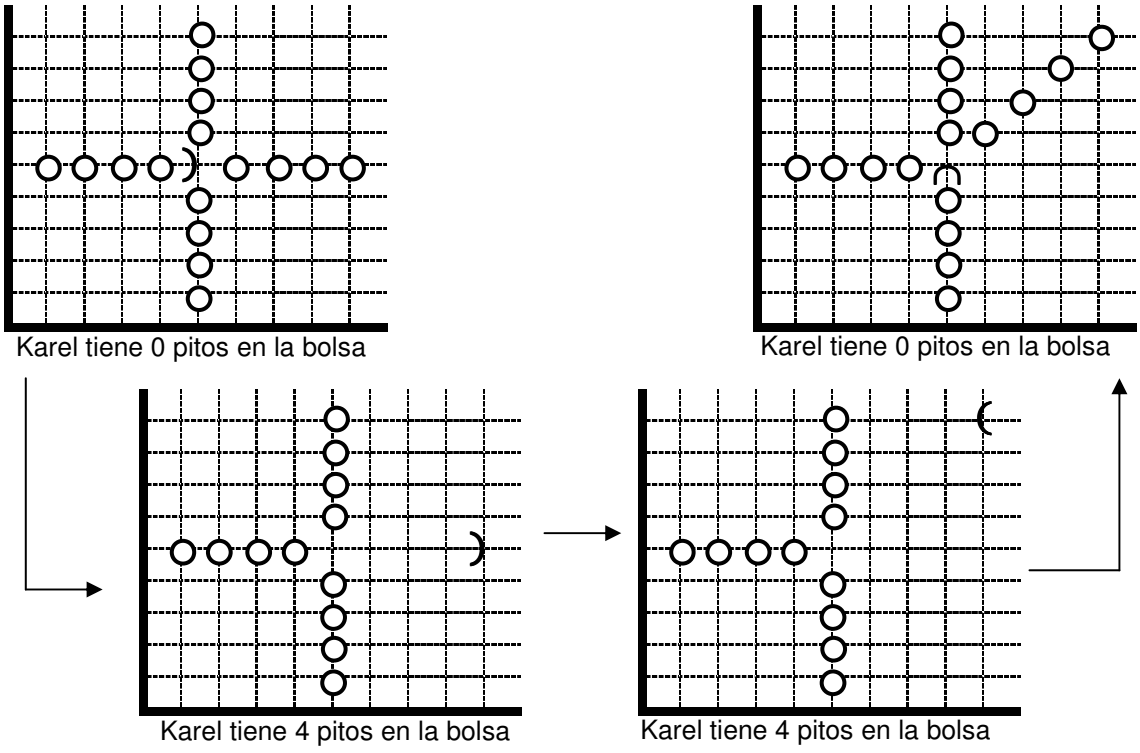


Figura. 86 Estados posibles de los subproblemas

Paso 4.2. Nombrar los subproblemas: cada uno de los tres subproblemas planteados lo vamos a resolver con una nueva instrucción: recoja-recta, vaya - esquina y ponga-diag.

Paso 4.3. Bloque principal de rotar:

```
Void pista::rotar()
{
recoja_recta();
vaya_esquina();
ponga_diag();
}
```

Paso 4.4. Resolver los subproblemas: para resolver la subtarea que colocar la diagonal de pitos, vamos a definir una instrucción adicional:

```
Void pista::diag1()
 {
    putBeeper();
    move();
    turnLeft();
    move();
    turnRight();
 }
```

Coloca uno de los pitos de la diagonal y queda preparado para colocar el siguiente. De esta forma ponga-diag se reduce a:

```
Void pista::ponga_diag()
 {
    loop(4)
     {
        diag1();
         turnRight();
     }
 }
```

Las instrucciones recoja-recta y vaya-esquina se pueden resolver directamente de la siguiente manera:

```
Void pista::recoja_recta()
 {
   loop(4)
    {
```

```
      move();
      pickBeeper();
    }
 }
```

```
Void pista::vaya_esquina()
{
  turnLeft();
    loop(4)
     {
       move();
       turnLeft();
    }
}
```

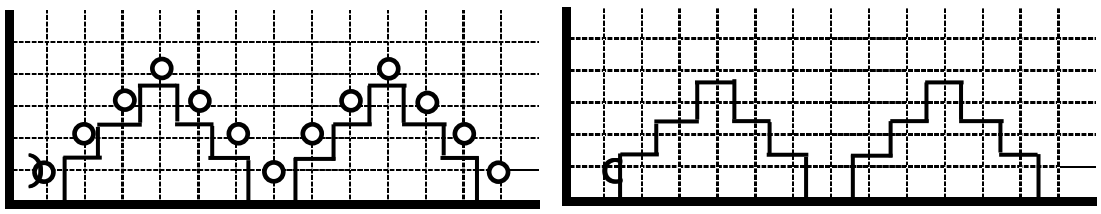
Programa Completo

```
Class pista : robot
  {
     void turnRight();
     void diag1();
     void ponga_diag();
     void recoja_recta();
     void vaya_esquina();
     void rotar()
 };

void pista::turnRight()
 {
    turnLeft();
    turnLeft();
    turnLeft();
 }

void pista::diag1()
 {
   putBeeper();
   move();
   turnLeft();
   move();
   turnRight();
}

void pista::ponga_diag()
 {
    loop(4)
     {
        diag1();
         turnRight();
     }
 }
void pista::recoja_recta()
 {
     loop(4)
```

```
    {
      move();
      pickBeeper();
    }
}

void pista::vaya_esquina()
{
    turnLeft();
    loop(4)
     {
       move();
       turnLeft();
     }
}

void pista::rotar()
{
  recoja_recta();
  vaya_esquina();
  ponga_diag();
}

task
{
    /* Pista es el nombre de la clase */
  Pista karel(4,4,East,0);
  loop(4)
   {
     karel.rotar();
   }
     karel.turnOff(); }
```

3.4. EJERCICIOS PROPUESTOS

3.4.1. El pastor Karel ve que se avecina una tormenta y desea recoger a sus ovejas (pitos) que se encuentran en las dos colinas de la región y volver a la casa (origen).

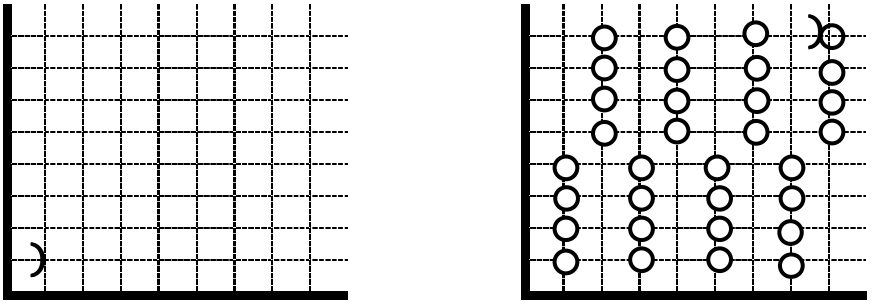


Posible estado inicial
Karel tiene 0 pitos en la bolsa

Estado final
Karel tiene 13 pitos en la bolsa

Figura. 87 Estados inicial y final del ejercicio recogiendo ovejas

3.4.2. El pobre Karel tuvo que ir a la guerra y se encuentra en el frente de batalla; su jefe le acaba de encargar la difícil tarea de poner minas (pitos) en el campo. Programe a Karel para que, partiendo del origen, coloque las 32 minas en la forma que se describe a continuación (Nota: obviamente Karel no debe volver a pasar por una esquina ya minada porque explotaría).

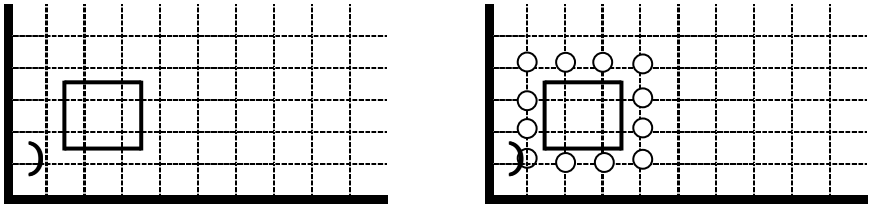


Estado inicial
Karel tiene 32 pitos en la bolsa

Estado final
Karel tiene 0 pitos en la bolsa

Figura. 88 Estados inicial y final del ejercicio colocar minas

3.4.3. Programe a Karel para que coloque baldosines (pitos) alrededor de su piscina cuadrada. Debe partir y terminar en el origen, mirando hacia el este.

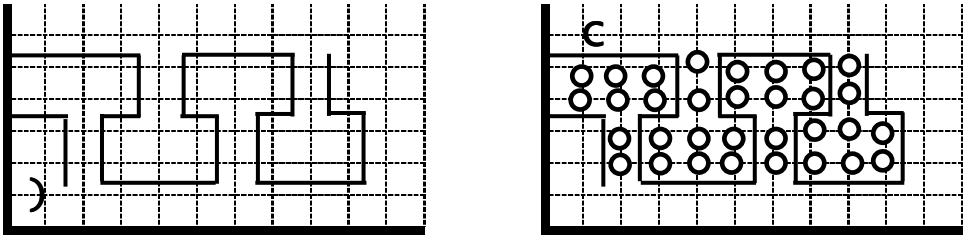


Estado inicial
Karel tiene 12 pitos en la bolsa

Estado final
Karel tiene 0 pitos en la bolsa

Figura. 89 Estados inicial y final del ejercicio colocando baldosines

3.4.4. Karel trabaja en un hotel y debe lavar los tapetes de las 4 habitaciones del primer piso; para cada metro cuadrado (esquina) gasta un balde de agua con jabón (pito). Programe a Karel para que lave los tapetes. A continuación se describe el problema con la planta del piso. (Nota: observe que todos los cuartos son iguales, tanto los que dan hacia el sur como los que tienen la entrada por el norte).

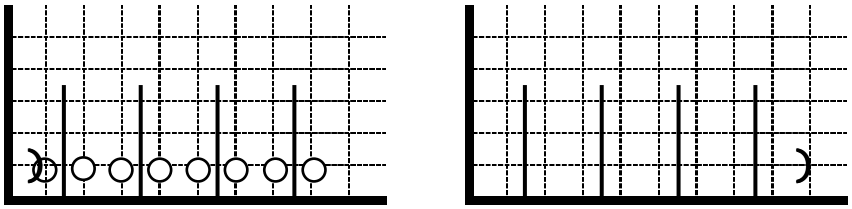


Estado inicial
Karel tiene 40 pitos en la bolsa

Estado final
Karel tiene 8 pitos en la bolsa

Figura. 90 Estados inicial y final del ejercicio lavar tapetes

3.4.5 Modifique el programa de los obstáculos presentados en la teoría, para que resuelva el siguiente problema:

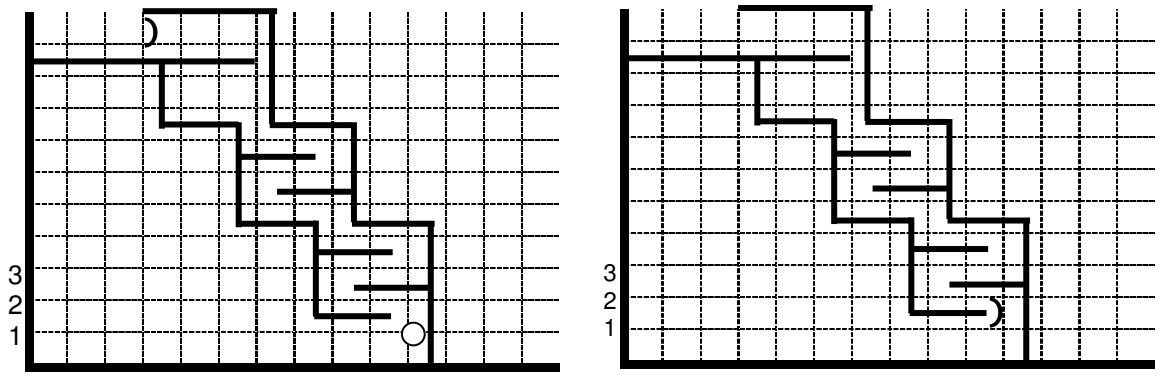


Estado inicial
Karel tiene 0 pitos en la bolsa

Estado final
Karel tiene 8 pitos en la bolsa

Figura. 91 Estados inicial y final del ejercicio obstáculos

3.4.6 Programe a Karel para que recoja el pito que se encuentra al final del laberinto que se muestra a continuación (Ayuda: note que el laberinto no es completamente irregular).



Estado inicial
Karel tiene 0 pitos en la bolsa

Estado final
Karel tiene 1 pito en la bolsa

Figura. 92 Estados inicial y final del ejercicio laberinto

3.4.7 Karel, el jardinero del parque VERDE debe abrir un camino en forma de cruz para que los peatones pueden atravesar el parque por dentro. Para esto debe talar algunos árboles de la manzana del parque. Programe a Karel para

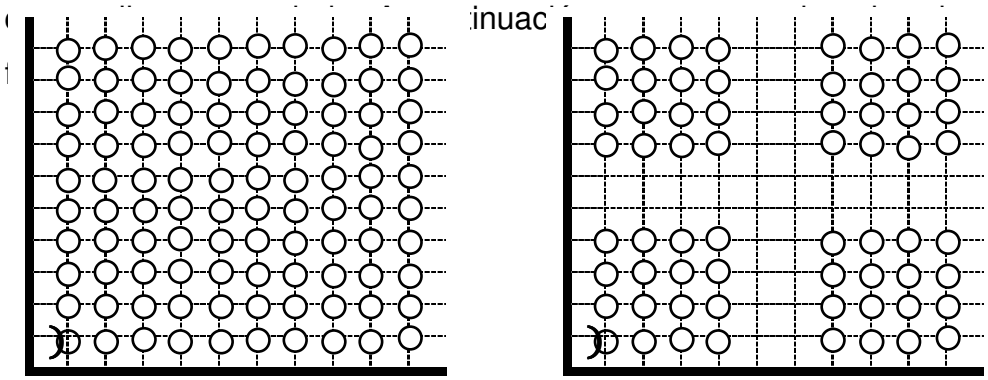


Figura. 93 Estados inicial y final del ejercicio del jardinero

3.4.8 Programe a Karel para que recoja la diagonal de 5 pitos que va de sureste a noroeste y construya con estos pitos una diagonal que parta del origen y se extienda hacia el nordeste.

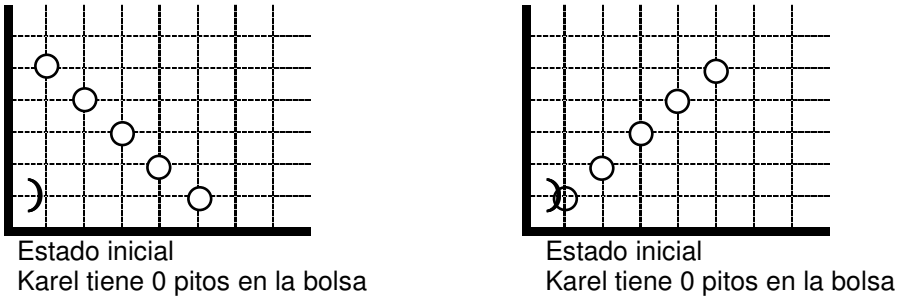


Figura. 94 Estados inicial y final del ejercicio diagonal sureste a noreste

3.4.9 Programe a Karel para que, comenzando en el origen y mirando al este con 6 pitos en su bolsa, coloque todos sus pitos en una línea recta de pendiente 1/2 a partir del origen. Resuelva el problema usando refinamiento paso a paso.

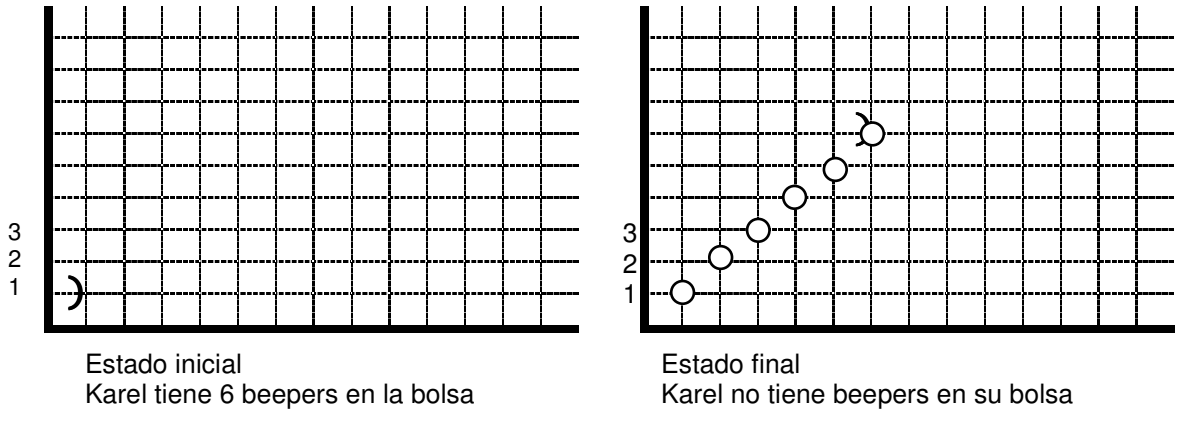


Figura. 95 Estados inicial y final del ejercicio línea recta de pendiente

3.4.10 Haga un programa para que Karel recoja todos los pitos de su mundo. Estos se encuentran cubriendo una escalera, como se muestra a continuación:

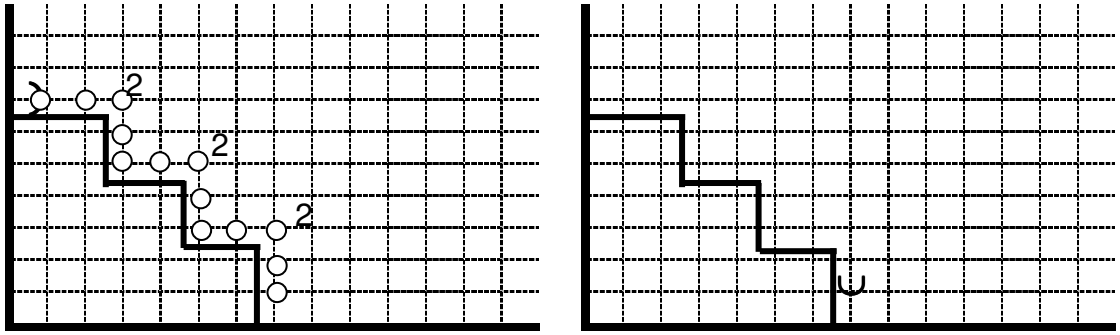


Figura. 96 Estados inicial y final del ejercicio recoger pitos de la escalera

Fíjese que en las esquinas de los escalones hay dos pitos y en las otras esquinas que rodean la escalera sólo hay uno. Karel parte de la calle 7 avenida 1 mirando al este y debe terminar en la calle 1 avenida 7 mirando al sur. Resuelva el problema usando refinamiento paso a paso.

3.4.11 Programe a Karel que comenzando en el origen y mirando al norte no tiene pitos en su bolsa, recoja todos los frutos del sembrado de lulo que se muestra en la siguiente figura (cada lulo es un pito) y regrese al origen. Resuelva el problema usando refinamiento paso a paso.

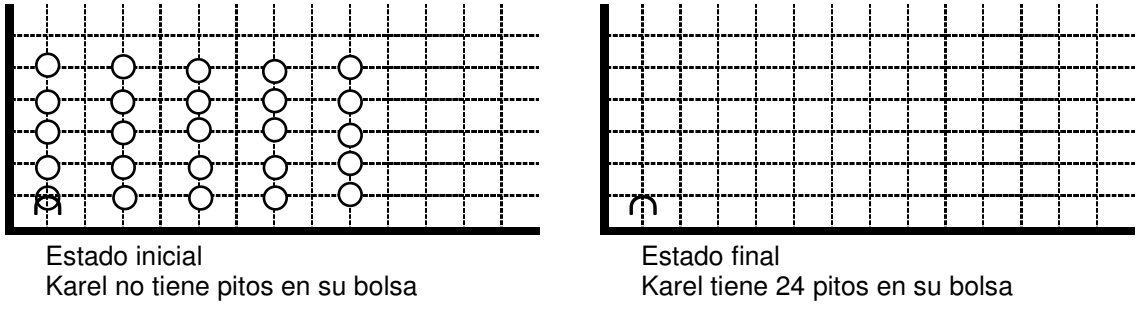
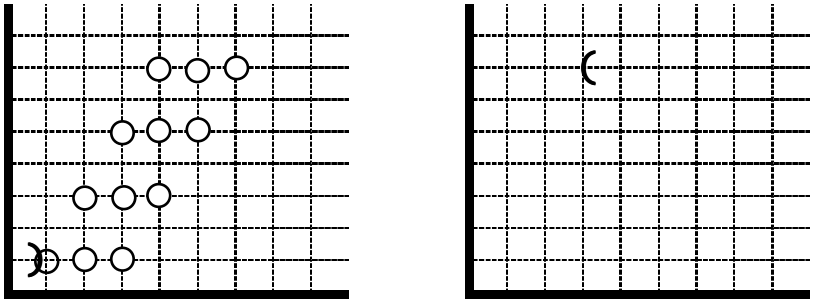


Figura. 97 Estados inicial y final del ejercicio recoger frutos

3.4.12 Programe a Karel que partiendo del origen y mirando al este, recoja las 12 alcachofas (pitos) de su sembrado y termine en la esquina (7,3), mirando al oeste. Resuelva el problema usando refinamiento paso a paso.

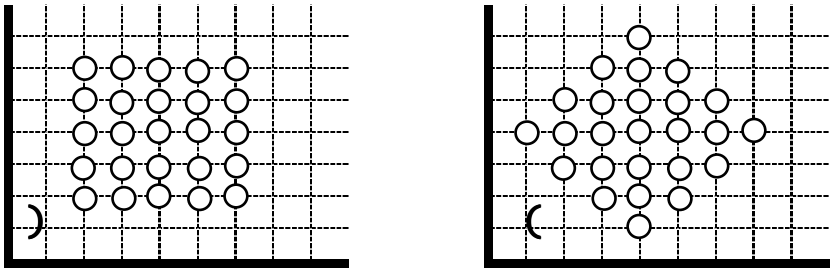


Estado inicial
Karel tiene 0 pitos en la bolsa

Estado final
Karel tiene 12 pitos en la bolsa

Figura. 98 Estados inicial y final del ejercicio recoger alcachofas

3.4.13 Karel tiene una huerta de lechugas (pitos) cuadrada (de 5 por 5), que va de la esquina (2,2) a la (6,6). Por problemas de riego quiere distribuir las lechugas formando un diamante. Programe a Karel para que resuelva su tarea partiendo del origen en dirección este y terminando en esa misma posición. Ayuda: mueva el mínimo número de lechugas posibles.



Estado inicial
Karel tiene 0 pitos en la bolsa

Estado final
Karel tiene 0 pitos en la bolsa

Figura. 99 Estados inicial final del ejercicio huerta de lechugas

3.4.14. Programe a Karel que comenzando en la esquina (2,1) y mirando al este, con 27 pitos en su bolsa, siembre 27 matas de lulo como se indica en la figura (cada semilla de lulo es un pito) y luego regrese al origen. Resuelva el problema usando refinamiento paso a paso.

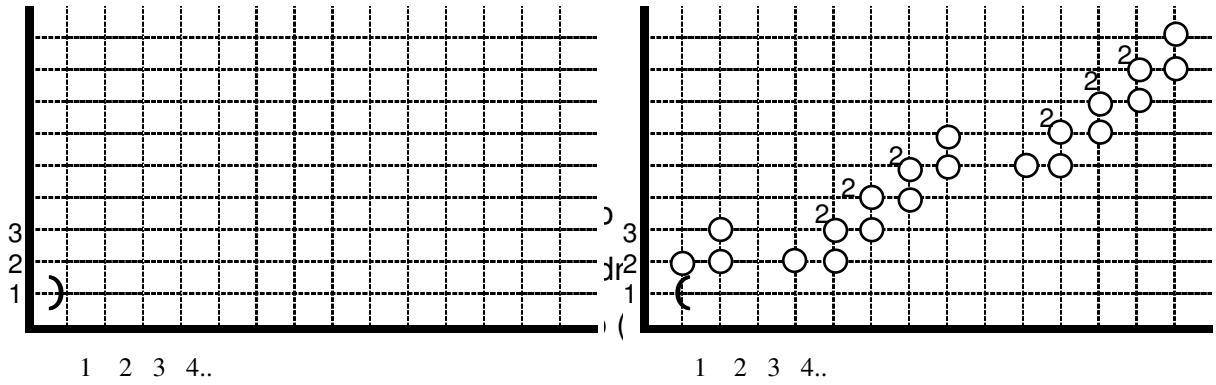


Figura. 100 Estados inicial y final del ejercicio matas de lulo

3.4.15 Karel se encuentra en un campo de trigo, montando a caballo (este caballo se mueve como el caballo de ajedrez). Programe Karel para que, partiendo del origen, recoja todos los atados de trigo (pitos) que se encuentran en el rectángulo de 12 pitos que se muestra en la figura.

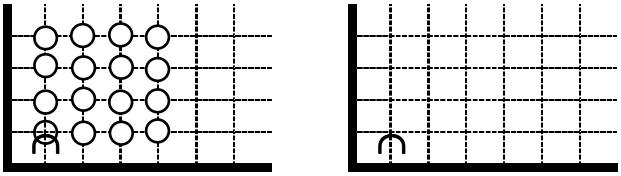


Figura. 101 Estados inicial y final del ejercicio campo de trigo