



Introducción a la Programación

Clases teóricas

por Pablo E. “Fidel” Martínez López

6. Repetición condicional



Repaso



- **Programar es comunicar** (con máquinas y personas)
 - Estrategia de solución (división en subtarear)
 - Legibilidad (elección de nombres, indentación)
 - **CONTRATOS:** Propósito, parámetros y precondiciones
- **Programas** (texto con diversos elementos)
 - **Comandos:** describen acciones
 - **Expresiones:** describen información
 - **Tipos:** clasifican expresiones



- **Comandos**

- Primitivos y secuencia
- PROCEDIMIENTOS (con y sin parámetros)
- Repetición simple
- Alternativa condicional



- **Expresiones**

- Valores literales y expresiones primitivas
- Operadores
 - numéricos, de enumeración, de comparación, lógicos
- FUNCIONES (con y sin parámetros)
- Parámetros (como datos)



- **Tipos de datos**

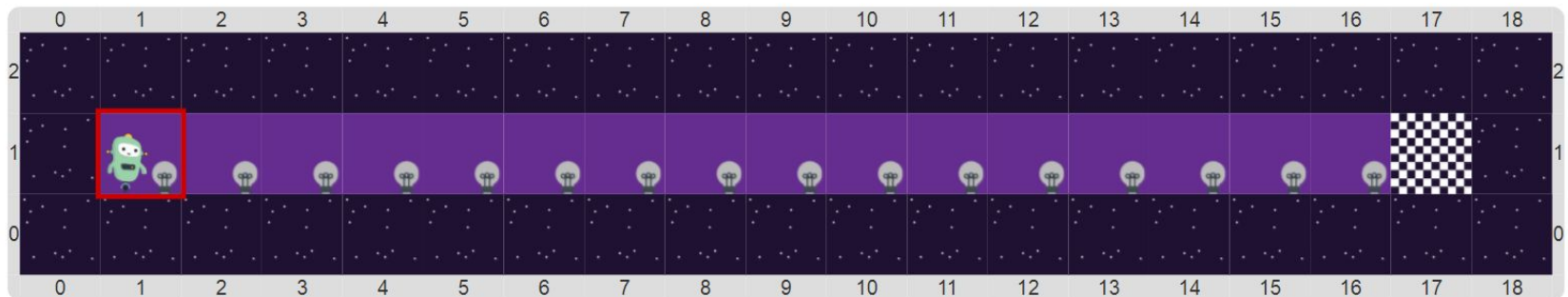
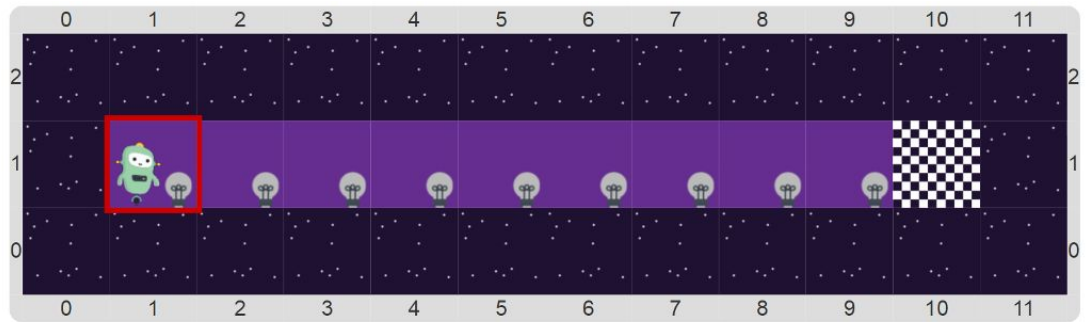
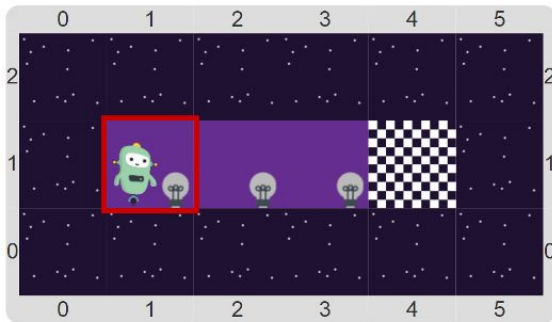
- permiten clasificar expresiones
- en Gobstones, por ahora, son cuatro
 - colores, direcciones, números y valores de verdad
- toda expresión tiene un tipo
- los parámetros deben especificar qué tipo de expresiones aceptan



Repeticiones condicionales

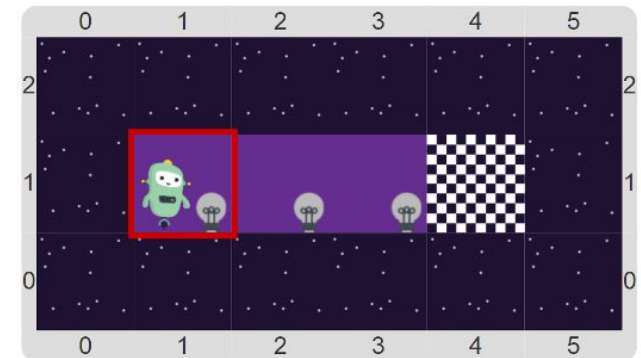
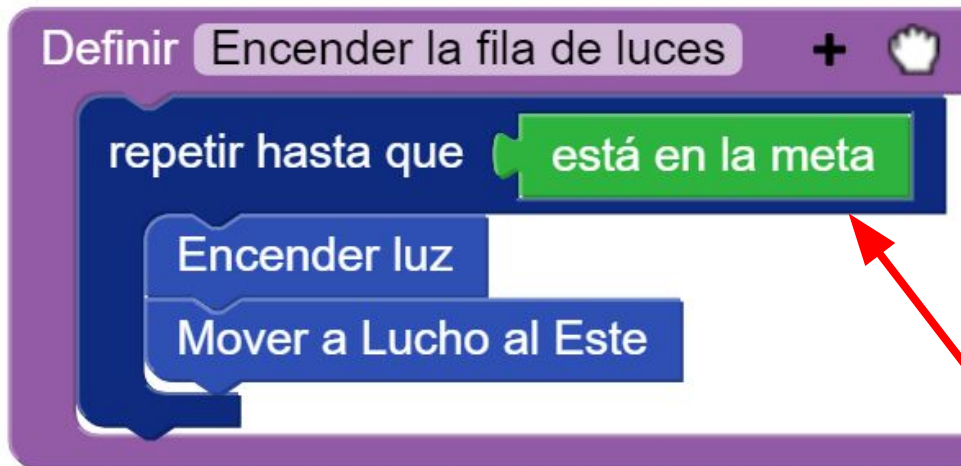


- ¿Cómo manejamos escenarios cambiantes, cuando lo que cambia es la distancia a la que está cierto elemento?
 - Se debe **repetir** la acción **hasta que** lleguemos al elemento
- Precisamos una herramienta nueva...





- La **repetición condicional** es una forma de armar comandos que permite hacer eso
 - La **condición** establece cuándo debe dejar de repetirse la acción indicada



¿Cuándo es verdadera esta condición?



- La **repetición condicional** se arma con
 - una expresión de tipo Bool (la **condición** de finalización)
 - en texto, entre paréntesis
 - un grupo de comandos (el **cuerpo** de la repetición)
 - en texto, entre llaves

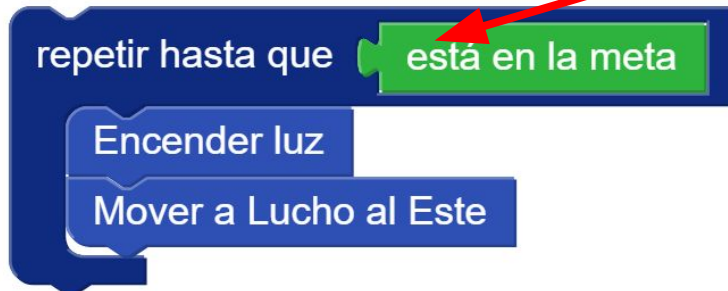


El cuerpo indica la acción a repetir y la condición, cuándo debe finalizar la repetición



- En bloques, usamos la variante ***repetir-hasta-que***
- En texto, usamos la variante ***while*** (*mientras*)
 - Una pregunta cuándo terminar y la otra cuándo seguir
 - Por eso la condición de una aparece negada en la otra (repite *hasta que* terminó, o *mientras NO* terminó)


Comparar repetir
hasta que está en la meta
vs.
*mientras **no*** está en la meta



```
while (not estáEnLaMeta()) {  
    EncenderLuz()  
    MoverALuchoAlEste()  
}
```



- La condición se vuelve a evaluar luego de cada repetición
 - Esto puede llevar a situaciones donde el programa NO TERMINA nunca
 - Es una situación de falla nueva

```
procedure LaBuenaPipa() {  
    /*  */  
    QuerésQueTeCuentoElCuentoDeLaBuenaPipa?()  
    while (respuesta()==respuesta()) {  
        YoNoTeDije-respuesta-TeDijeSiQuerésQueTeCuentoElCuentoDeLaBuenaPipa?()  
    }  
}
```

¿Cuándo termina este juego?
¡La condición siempre es verdadera!

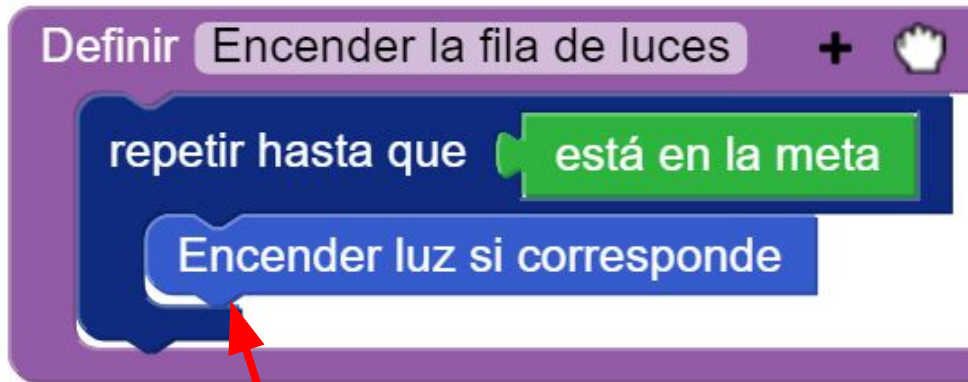


- A esta forma de falla se la considera similar a un BOOM
- La precondition **debe** tener en cuenta estos casos
- Puede darse en condiciones sutiles

¿Cuándo termina **LlegarALaEsquina (Norte, Sur)**?
¿Existe la esquina Norte-Sur?

```
procedure LlegarALaEsquina__(dirección1, dirección2) {  
  /* PROPÓSITO:      llevar el cabezal a la esquina indicada  
  PRECONDICIONES:   las direcciones dadas NO son opuestas ni iguales  
  PARÁMETROS:        dirección1 y dirección2 son Direcciones  
  OBSERVACIÓN:       si las direcciones son opuestas, NUNCA TERMINA  
  */  
  while (puedeMover(dirección1) || puedeMover(dirección2)) {  
    Mover_SiPuede(dirección1) Mover_SiPuede(dirección2)  
  }  
}
```

- La no-terminación puede aparecer porque olvidamos hacer algo, o porque no consideramos todos los casos



¡Olvidamos
mover a
Lucho!



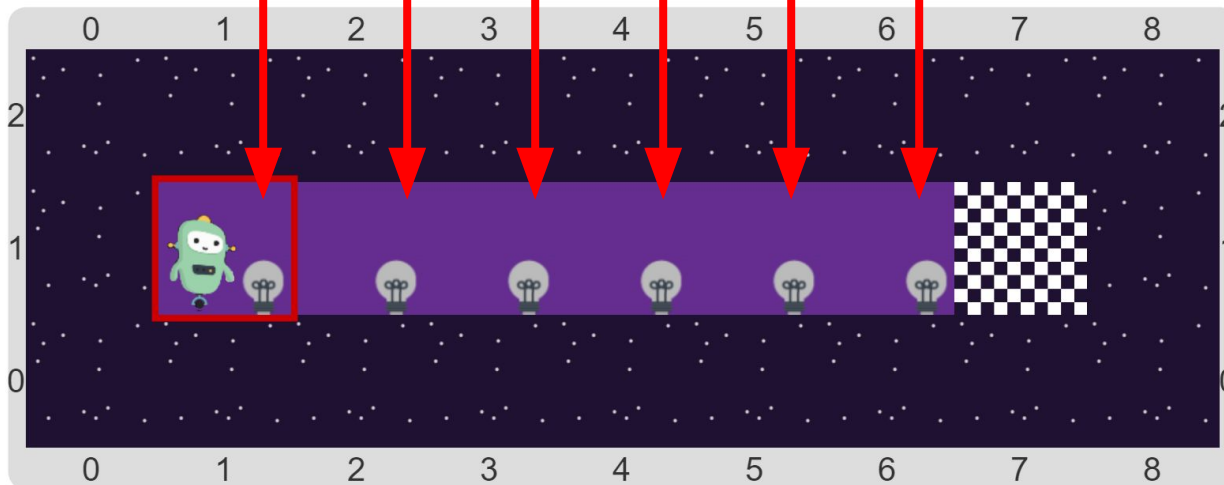
BOOM

La ejecución del programa
demoró más de 3000ms.

- ¿Cómo asegurar que una repetición condicional termina?
 - Hay muchas técnicas
 - Nosotros usaremos una simple: la idea de **recorrido**
 - Se basa en una secuencia finita de “elementos”

Secuencia de 6
celdas con luz

“No pregunto cuántos son,
sino que vayan saliendo”





Recorridos



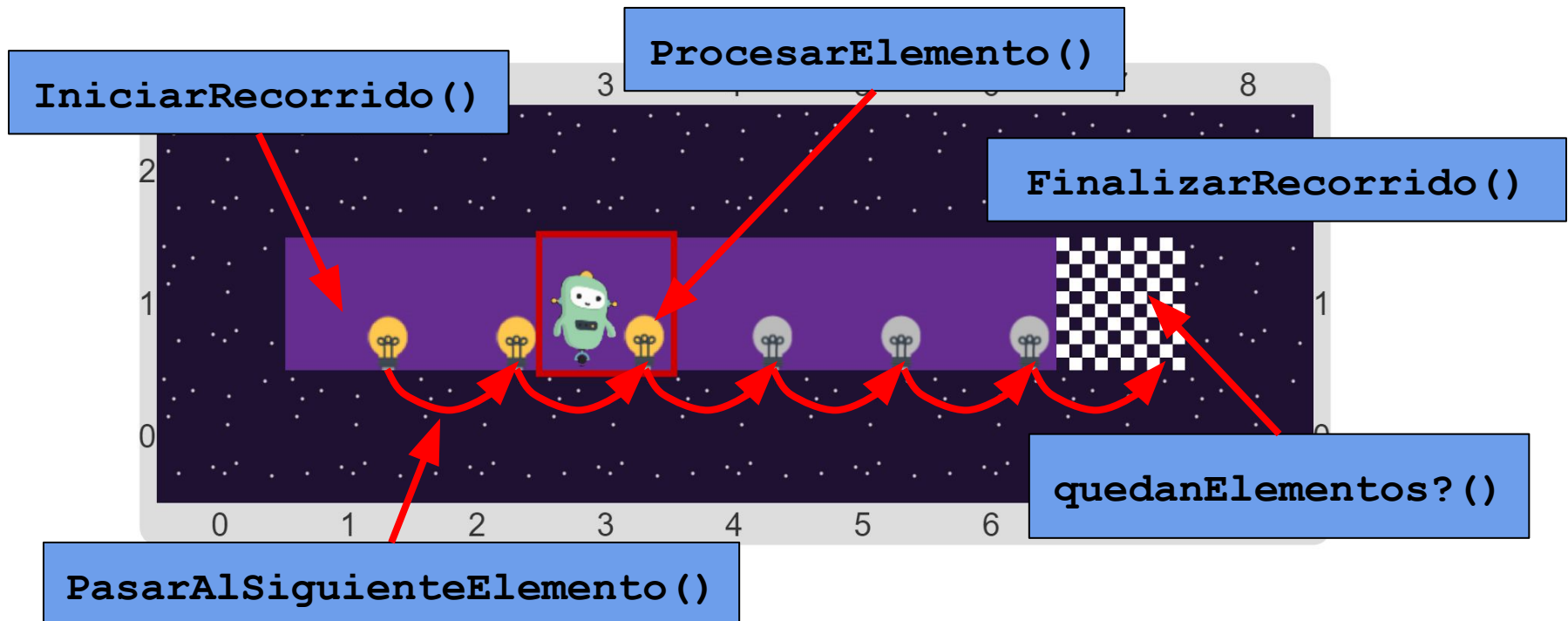
- Un **recorrido** es una forma de dividir en subtareas
 - en problemas con una secuencia finita de “elementos”
 - para asegurar que todos los elementos son procesados
- Involucra definir 5 subtareas (los nombres pueden variar)

```
procedure RecorridoGenérico() {  
    /*  
        PROPÓSITO:  
        * procesar todos los elementos de una  
        secuencia de elementos determinada  
        PRECONDICIÓN: según el problema  
    */  
    IniciarRecorrido()  
    while (quedanElementosParaProcesar()) {  
        ProcesarElementoActual()  
        PasarAlSiguienteElemento()  
    }  
    FinalizarRecorrido()  
}
```

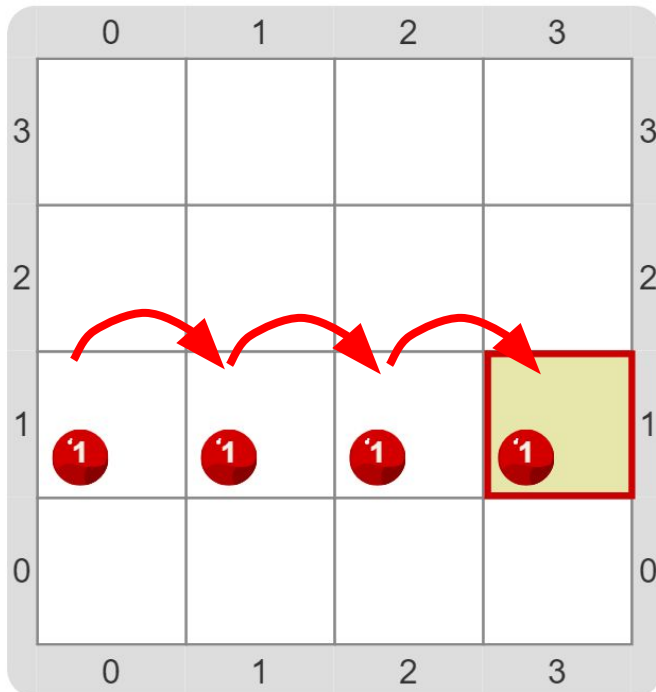
Muchos
problemas
involucran
secuencias de
elementos



- Un **recorrido** es una forma de dividir en subtarear
 - en problemas con una secuencia finita de “elementos”
 - para asegurar que todos los elementos son procesados
- Involucra definir 5 subtarear (los nombres pueden variar)



- El recorrido más simple es procesar las celdas de una fila (o una columna)
 - los “elementos” son las celdas, que están una al lado de otra
 - **PasarAlSiguienteElemento** es simplemente moverse a la celda lindante en la dirección dada



- El recorrido más simple es procesar las celdas de una fila (o una columna)
 - los “elementos” son las celdas, que están una al lado de otra
 - **PasarAlSiguienteElemento** es simplemente moverse a la celda lindante en la dirección dada

```
procedure PonerBolitasEnTodaLaFila() {  
    /* PROPÓSITO: poner una bolita roja en cada celda  
       de la fila actual. No importa dónde queda el  
       cabezal  
       PRECONDICIÓN: ninguna (es una operación total)  
    */  
    IrAlBorde(Oeste)           // IniciarRecorrido()  
    while (puedeMover(Este)) {  // quedanElementosParaProcesar()  
        Poner(Rojo)            // ProcesarElementoActual()  
        Mover(Este)             // PasarAlSiguienteElemento()  
    }  
    Poner(Rojo)                // FinalizarRecorrido()  
}
```

Observar la
estructura de
recorrido

- Otro recorrido sencillo es procesar un camino simple
 - cada parte del camino tiene dos partes del camino lindantes
 - **PasarAlSiguienteElemento** es moverse a la celda lindante que es parte del camino y no se visitó


```
procedure SalirDelLaberintoComiendoElQueso() {  
  /* PROPÓSITO: sacar al ratón del laberinto, comiendo el queso  
    que encuentre por el camino  
  PRECONDICIONES:  
    * hay un escenario correctamente representado { }  
  OBSERVACIÓN:  
    * los elementos a recorrer son las posiciones del  
    camino hasta la salida, indicadas por las flechas  
  */  
  
  while (not estoyEnLaSalida()) {  
    ComerElQuesoSiHay()  
    AvanzarUnPasoSiguiendoLaFlecha()  
  }  
}
```

// NO HAY IniciarRecorrido()
// quedanElementosSinProcesar()
// ProcesarElementoActual()
// PasarAlSiguienteElemento()
// NO HAY FinalizarRecorrido()

El código
para pasar al
siguiente
elemento
requiere más
trabajo



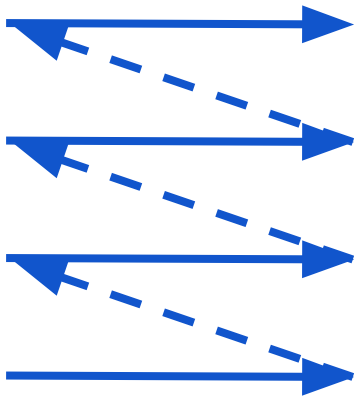
- Otro recorrido sencillo es procesar un camino simple
 - cada celda tiene una antes y una después
 - **PasarAlSiguienteElemento** es moverse a la celda lindante que es parte del camino y no se visitó

```
procedure AvanzarUnPasoSiguiendoLaFlecha() {  
    /*  */  
    if (laFlechaApuntaAlNorte()) {  
        SacarLaFlecha()  
        MoverAlRatónAl_(Norte)  
    }  
    elseif (laFlechaApuntaAlEste()) {  
        SacarLaFlecha()  
        MoverAlRatónAl_(Este)  
    }  
    elseif (laFlechaApuntaAlSur()) {  
        SacarLaFlecha()  
        MoverAlRatónAl_(Sur)  
    }  
    elseif (laFlechaApuntaAlOeste()) {  
        SacarLaFlecha()  
    }  
}
```

El código
para pasar al
siguiente
elemento
requiere más
trabajo

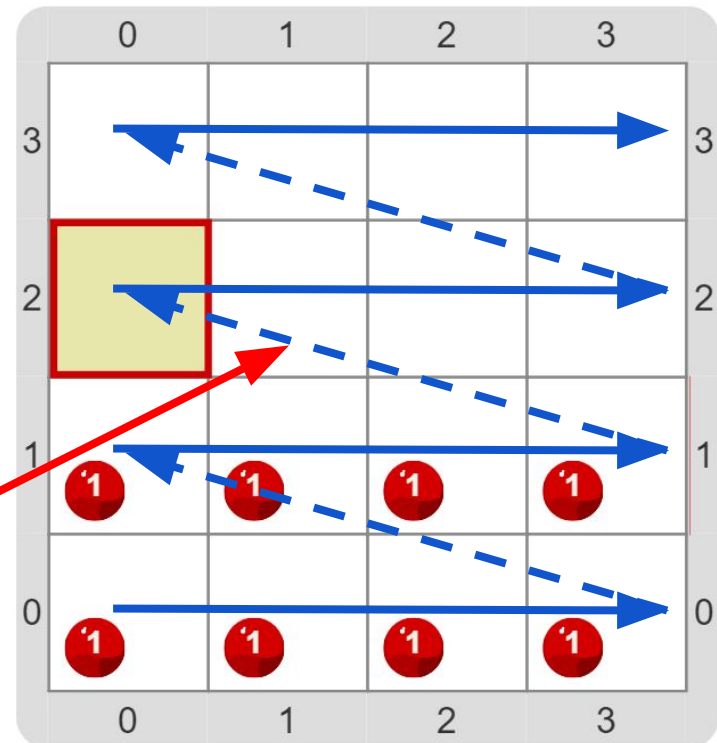


- Un recorrido más complicado es recorrer todas las celdas del tablero, en cierto orden
 - **PasarAlSiguienteElemento** debe determinar si sigue en la misma fila, o empieza con otra



Recorrido por celdas,
al Este y al Norte

¡La celda siguiente a la última
de una fila, es la primera de la
fila siguiente!





- Un recorrido más complicado es recorrer todas las celdas del tablero, en cierto orden
 - **PasarAlSiguienteElemento** debe determinar si sigue en la misma fila, o empieza con otra

```
procedure PonerUnaBolitaEnCadaCeldaDelTablero() {  
    /*  
        PROPÓSITO: poner una bolita en cada celda del tablero  
                   y dejar el cabezal en la esquina NorEste  
        PRECONDICIÓN: ninguna (es una operación total)  
        OBSERVACIÓN: se arma como un recorrido sobre todas las  
                   celdas del tablero, hacia el Este y el Norte  
    */  
    IrAlBorde(Sur) IrAlBorde(Oeste)    // IniciarRecorrido()  
    while (puedeMover(Este)            // quedanElementos-  
        || puedeMover(Norte)) {       // SinProcesar()  
        Poner(Rojo)                   // ProcesarElemento()  
        PasarASiguienteCeldaDelTablero() // PasarAlSiguiente()  
    }  
    Poner(Rojo)                       // FinalizarRecorrido()  
}
```

Acá
también
pasar al
siguiente
elemento
requiere
algo de
trabajo



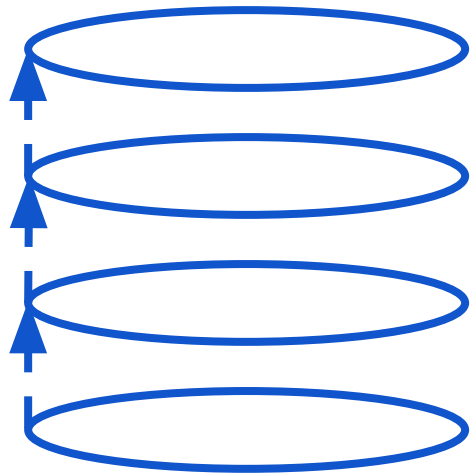
- Un recorrido más complicado es recorrer todas las celdas del tablero, en cierto orden
 - **PasarAlSiguienteElemento** debe determinar si sigue en la misma fila, o empieza con otra

```
procedure PasarASiguienteCeldaDelTablero() {  
  /*  
    PROPÓSITO: pasar a la siguiente celda en un  
    recorrido por celdas en dirección Este y Norte  
    PRECONDICIÓN:  
    * la celda actual NO es la esquina NorEste  
    OBSERVACIONES:  
    * es una de las operaciones del recorrido  
  */  
  if (puedeMover(Este))  
  then { Mover(Este) }  
  else {  
    // Puede mover al Norte, por la precondición  
    Mover(Norte) IrAlBorde(Oeste)  
  }  
}
```

Acá
también
pasar al
siguiente
elemento
requiere
algo de
trabajo

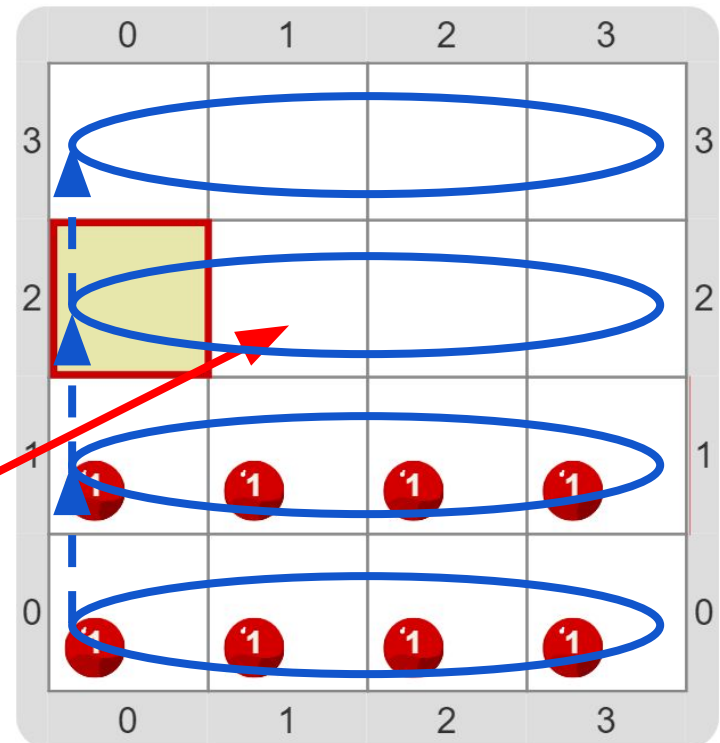


- Los elementos a recorrer no tienen por qué ser celdas
 - Podemos hacer un recorrido por filas, o por columnas



Recorrido por filas
al Norte

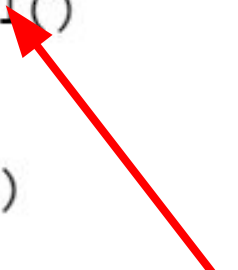
¡Cada elemento es
TODA una fila!





- Los elementos a recorrer no tienen por qué ser celdas
 - Podemos hacer un recorrido por filas, o por columnas

```
procedure PintarCadaCeldaDelTablero() {  
  /* PROPÓSITO: poner una bolita en cada celda del tablero  
    y dejar el cabezal en la esquina NorEste  
    PRECONDICIÓN: ninguna (es una operación total)  
    OBSERVACIÓN: se arma como un recorrido por filas  
  */  
  IrAlBorde(Sur) // IniciarRecorrido()  
  while (puedeMover(Norte)) { // quedanElementos()  
    PintarFilaActual() // ProcesarElemento()  
    Mover(Norte) // PasarAlSiguiente()  
  }  
  PintarFilaActual() // FinalizarRecorrido()  
}
```



¡Acá, **ProcesarElemento** también involucra un recorrido!

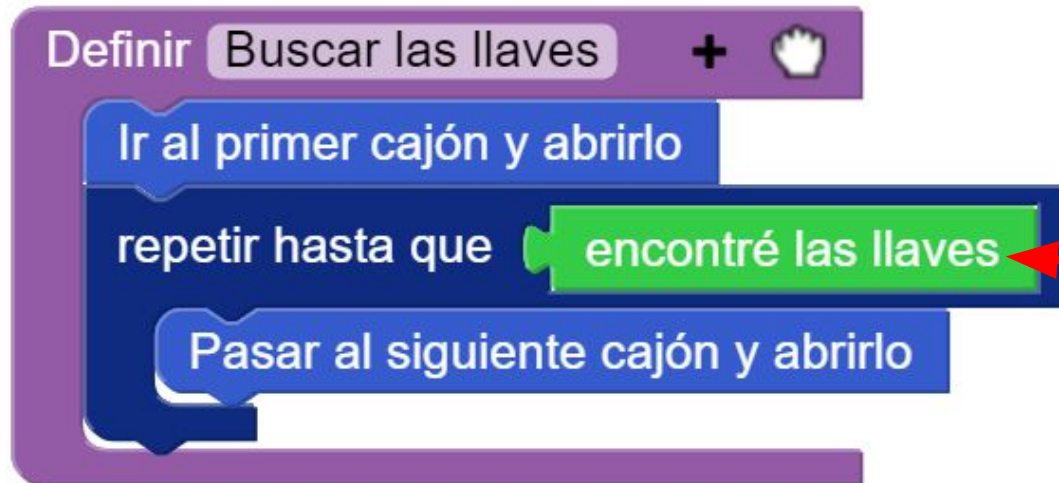
- Los elementos a recorrer no tienen por qué ser celdas
 - Podemos hacer un recorrido por filas, o por columnas

```
procedure PintarFilaActual() {  
  /* PROPÓSITO:  
    * poner una bolita en cada celda de la fila actual  
    y dejar el cabezal en el borde Este de la fila  
  PRECONDICIÓN: ninguna (es una operación total)  
  OBSERVACIÓN: se arma como un recorrido sobre todas  
    las celdas de la fila  
  */  
  IrAlBorde(Oeste)           // IniciarRecorrido()  
  while (puedeMover(Este)) { // quedanElementos?()  
    Poner(Rojo)              // ProcesarElemento()  
    Mover(Este)              // PasarAlSiguiente()  
  }  
  Poner(Rojo)               // FinalizarRecorrido()  
}
```

¡Acá, **ProcesarElemento** también involucra un recorrido!



- En ocasiones, el recorrido debe terminar antes
 - Hablamos de un **recorrido de búsqueda**
 - Se detiene cuando se encontró lo que se buscaba
 - ¿Qué pasa si no está lo que buscamos?

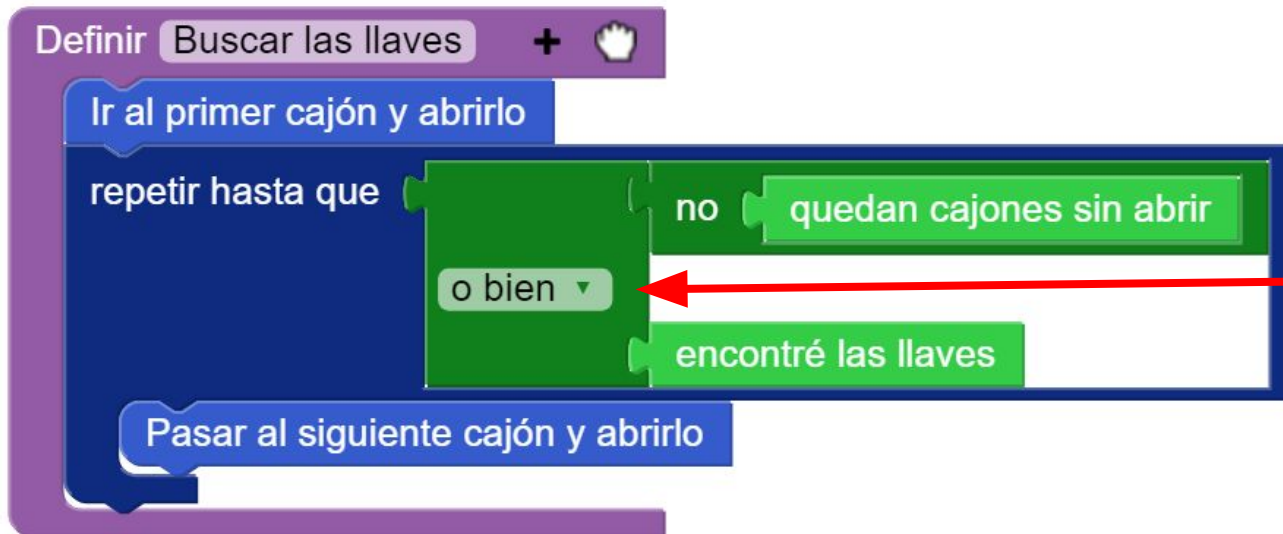


¡Si las llaves no están, va a explotar cuando no haya más cajones!

La precondition debería pedir que las llaves estén...



- En ocasiones, el recorrido debe terminar antes
 - Hablamos de un **recorrido de búsqueda**
 - Se detiene cuando se encontró lo que se buscaba
 - ¿Qué pasa si no está lo que buscamos?

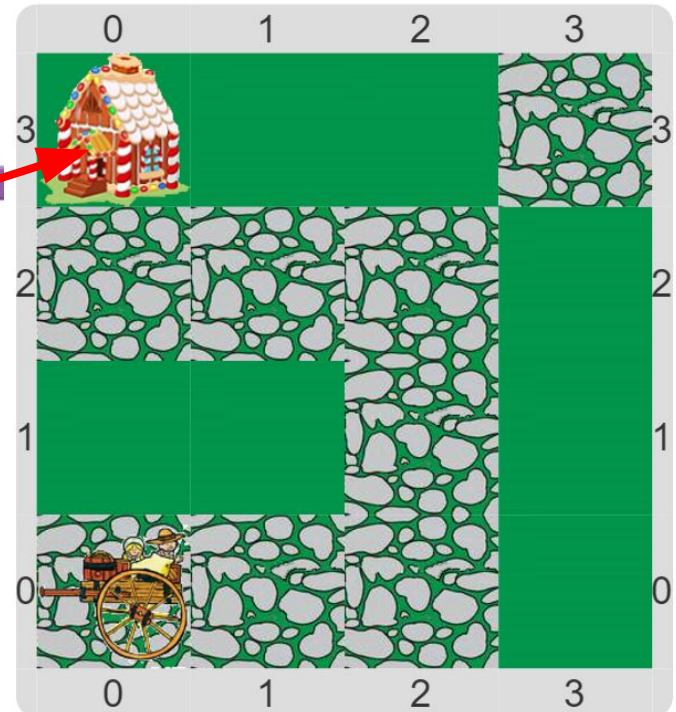
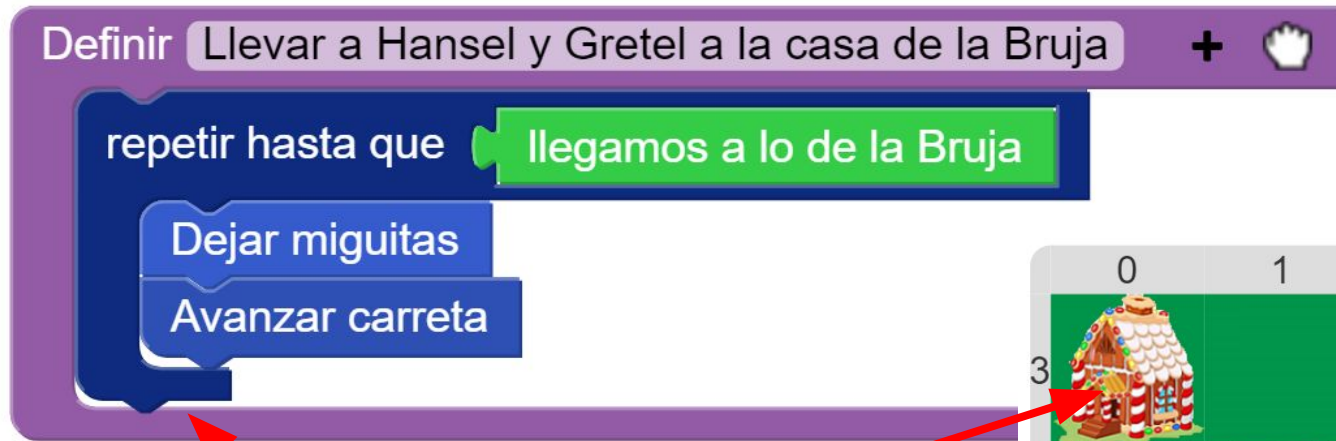


Ahora no falla, porque si no hay llaves, frena al final

...debe controlarse que queden cajones



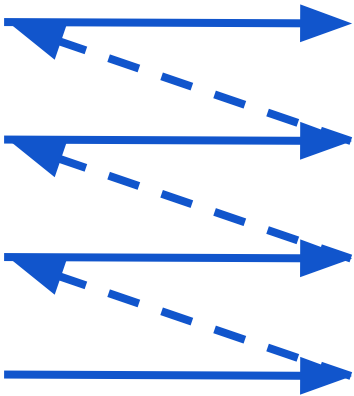
- ...y en otros recorridos, no hace falta ver casos de borde



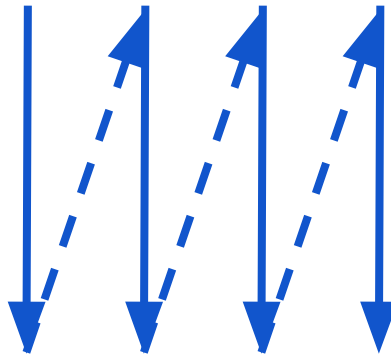
En la casa de la bruja no
hay que dejar miguitas



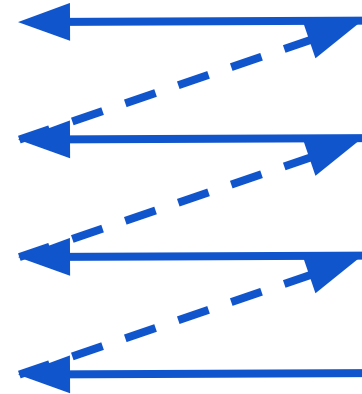
- El recorrido por celdas no tiene por qué ser solamente al Este y al Norte
 - ¿Cómo hacer que no sea siempre para el mismo lado?
 - ¡Parámetros!



Al Este y
al Norte




Al Sur y
al Este




Al Oeste
y al Norte




- ¿Cómo saber qué parametrizar? Técnica de los recuadros

```
procedure PintarTodasLasCeldas() {  
    /*  */  
    IrAlBorde(Norte)  
    IrAlBorde(Oeste)  
    while (quedanCeldas()) {  
        Poner(Rojo)  
        PasarASiguienteCelda()  
    }  
    Poner(Rojo)  
}
```


Primero mueve al Sur, y
si no puede, al Este

```
procedure PasarASiguienteCelda() {  
    /*  */  
    if (puedeMover(Sur)) { Mover(Sur) }  
    else { Mover(Este) IrAlBorde(Norte) }  
}
```


```
function quedanCeldas() { /*  */  
    return (puedeMover(Sur) || puedeMover(Este))  
}
```




- ¿Cómo saber qué parametrizar? Técnica de los recuadros

```
procedure PintarTodasLasCeldas() {  
    /*  */  
    IrAlBorde(Norte)  
    IrAlBorde(Oeste)  
    while (quedanCeldas()) {  
        Poner(Rojo)  
        PasarASiguienteCelda()  
    }  
    Poner(Rojo)  
}
```




Primero mueve al Sur, y
si no puede, al Este

```
procedure PasarASiguienteCelda() {  
    /*  */  
    if (puedeMover(Sur)) { Mover(Sur) }  
    else { Mover(Este) IrAlBorde(Norte) }  
}
```






```
function quedanCeldas() { /*  */  
    return (puedeMover(Sur) || puedeMover(Este))  
}
```






- ¿Cómo saber qué parametrizar? Técnica de los recuadros

```
procedure PintarTodasLasCeldas() {  
  /*  */  
  IrAlBorde()  
  IrAlBorde()  
  while (quedanCeldas()) {  
    Poner(Rojo)  
    PasarASiguienteCelda()  
  }  
  Poner(Rojo)  
}
```

¡Las direcciones
pueden ser otras!

```
procedure PasarASiguienteCelda() {  
  /*  */  
  if (puedeMover() { Mover() }  
  else { Mover() IrAlBorde() }  
}
```

```
function quedanCeldas() { /*  */  
  return (puedeMover() || puedeMover())  
}
```



- Un recorrido por celdas con parámetros para la dirección

```
procedure PintarTodasLasCeldasEnRecorrido__(  
    /**/  
    IrAlBorde(opuesto()) // Ir al inicio del recorrido  
    IrAlBorde(opuesto())  
    while (quedanCeldasEnRecorrido__(, ) {  
        Poner(Rojo)  
        PasarASiguienteCelda__(, )  
    }  
    Poner(Rojo) // Caso de borde  
}
```

¿Cómo pasar los parámetros a las subtarefas?

```
procedure PasarASiguienteCelda__(  
    /**/  
    if (puedeMover()) { Mover() }  
    else { Mover() IrAlBorde(opuesto()) }  
}
```

```
function quedanCeldasEnRecorrido__(  
    return (puedeMover() || puedeMover() ) { /**/  
}
```




- Un recorrido por celdas con parámetros para la dirección

```
procedure PintarTodasLasCeldasEnRecorrido__(dirPrincipal, dirSecundaria) {  
    /**/  
    IrAlBorde(opuesto(dirPrincipal)) // Ir al inicio del recorrido  
    IrAlBorde(opuesto(dirSecundaria))  
    while (quedanCeldasEnRecorrido__(dirPrincipal, dirSecundaria)) {  
        Poner(Rojo)  
        PasarASiguienteCelda__(dirPrincipal, dirSecundaria)  
    }  
    Poner(Rojo) // Caso de borde  
}
```

¡Más parámetros!

```
procedure PasarASiguienteCelda__(dirPpal, dirSec) {  
    /**/  
    if (puedeMover(dirPpal)) { Mover(dirPpal) }  
    else { Mover(dirSec) IrAlBorde(opuesto(dirPpal)) }  
}
```

```
function quedanCeldasEnRecorrido__(dirPpal, dirSec) { /**/  
    return (puedeMover(dirPpal) || puedeMover(dirSec))  
}
```



- Un recorrido por celdas con parámetros para la dirección

```
procedure PintarTodasLasCeldasEnRecorrido__(dirPrincipal, dirSecundaria) {  
    /**/  
    IrAlBorde(opuesto(dirPrincipal))    // Ir al inicio del recorrido  
    IrAlBorde(opuesto(dirSecundaria))  
    while (quedanCeldasEnRecorrido__(dirPrincipal, dirSecundaria)) {  
        Poner(Rojo)  
        PasarASiguienteCelda__(dirPrincipal, dirSecundaria)  
    }  
    Poner(Rojo) // Caso de borde  
}
```

```
procedure PasarASiguienteCelda__(dirPpal, dirSec) {  
    /**/  
    if (puedeMover(dirPpal) { Mover(dirPpal) }  
        else { Mover(dirSec) IrAlBorde(opuesto(dirPpal)) }  
}
```

```
function quedanCeldasEnRecorrido__(dirPpal, dirSec) { /**/  
    return (puedeMover(dirPpal) || puedeMover(dirSec))  
}
```




Cierre



- ***Repetición condicional***

- Una forma de repetición para cuándo no se sabe la cantidad de veces que hay que repetir
- Tiene un **cuerpo** de comandos a repetir, y una **condición** que establece cuándo terminar
- En bloques tiene la forma **repetir-hasta-que**
- En texto tiene la forma **mientras (while)**
- ¡Puede suceder que la repetición no termine nunca!
 - Esto es equivalente a hacer BOOM



- ***Recorridos***

- Una forma de controlar la repetición condicional sugiriendo cómo dividir en subtareas
- Se basa en una secuencia finita de “elementos”
- Sugiere 5 subtareas
 - `IniciarRecorrido()`
 - `quedanElementosParaProcesar()`
 - `ProcesarElementoActual()`
 - `PasarAlSiguienteElemento()`
 - `FinalizarRecorrido()`
- Las tareas pueden ponerse en procedimientos o definirse directamente con comandos sueltos