Repetición + alternativa condicional

Universidad Nacional de Quilmes

Introducción a la Programación

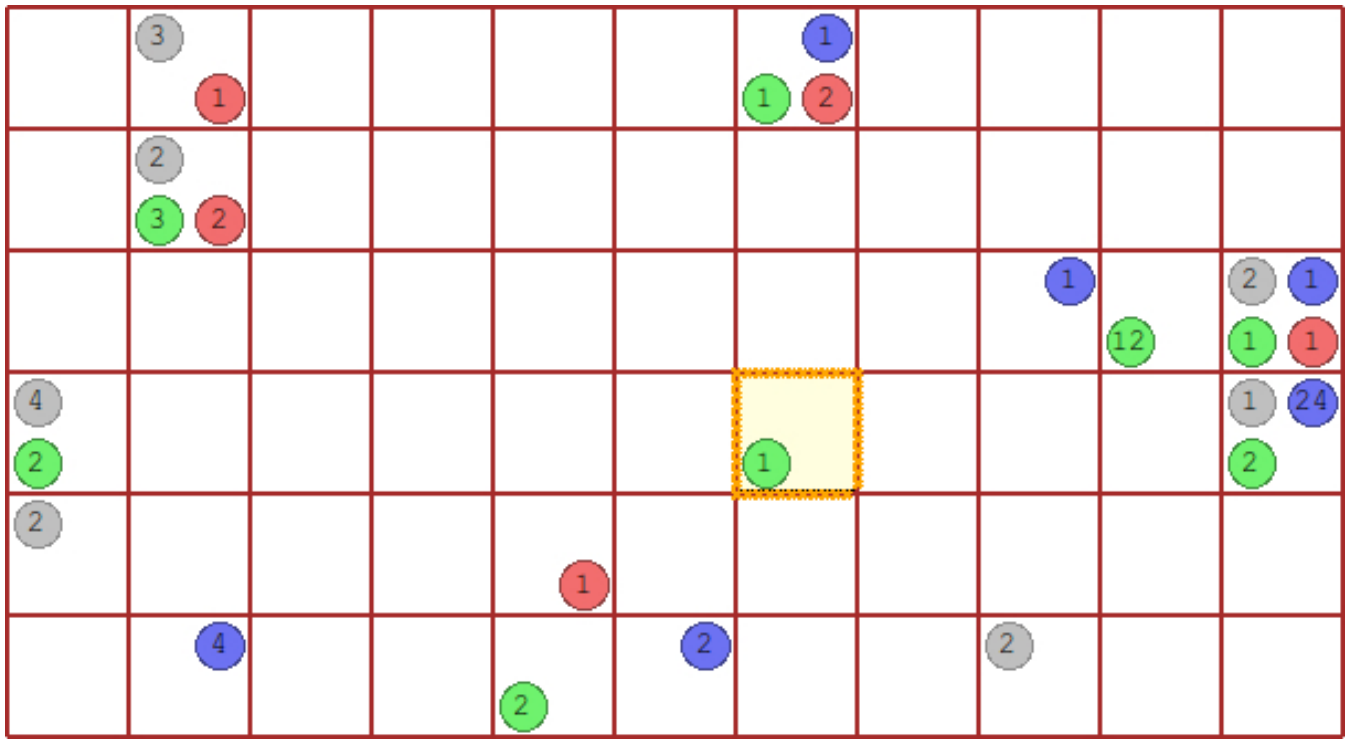
Lo que veremos hoy

1. Repaso

2. Repetición

Repaso (Elementos de Gobstones)

- Tablero: cuadrícula de celdas, que contienen bolitas.
- Cabezal: realiza acciones sobre las celdas del tablero.



Repaso (Programas y comandos)

- Programa Gobstones: texto que describe cómo transformar un tablero
  - program, comandos, propósito, efecto, documentación, tabulación

```
Propósito:  este programa no hace nada
program {
  1. Poner(Rojo)
  2. Sacar(Rojo)
}
```

- Comandos:
  - Poner(⟨color⟩)
  - Sacar(⟨color⟩)
  - Mover(⟨direccion⟩)
- Colores:
  - Azul
  - Negro
  - Rojo
  - Verde
- Direcciones:
  - Norte ↑
  - Este →
  - Sur ↓
  - Oeste ←

Repaso (Procedimientos simples)

- Procedimientos simples: dan nombre a una secuencia de comandos
  - **procedure** ⟨NombreMayuscula⟩() ⟨bloque⟩

```
Propósito: Dibuja un cuadrado de 2 × 2 con bolitas de cada color
Cabezal: Queda posicionado en la celda actual
procedure CuadradoMulticolor() {
  1. Mover(Este);  PonerUnaDeCada()
  2. Mover(Norte); PonerUnaDeCada()
  3. Mover(Oeste); PonerUnaDeCada()
  4. Mover(Sur);   PonerUnaDeCada()
}
```

- Abstracción: ¿Qué hace `PonerUnaDeCada`?
- Reutilización: ¿Cuántas veces hay que escribir el código para `PonerUnaDeCada`? ⇒ Bibliotecas
- Intercambio: se puede compartir procedimientos (documentación!)

Repaso (División en subtareas)

- Concepto central en la programación
  - Permite reutilización y escalabilidad
- Problema: difícil de aprender
  - Dividir en subproblemas sin importar cómo se resuelven los subproblemas
  - Recordar qué cosas ya hicimos y tratar de reutilizarlas
  - Pensar en unidades de procesamiento que tengan sentido
- Regla del copiar&pegar $\rightarrow$ factorización
  - Si se repite un comportamiento, se extrae

Repaso (Especificación, contratos)

- Problema: intercambio de código para usar como caja negra
- Solución: problemas + especificación
  - Propósito: qué problema resuelve un procedimiento.
  - Precondición: condiciones necesarias para garantizar que el problema puede resolverse
  - Tipos de los parámetros: qué valores puede asumir un parámetro ($\star$)
- Contrato: si se cumple la precondición y los tipos, entonces el procedimiento funciona correctamente
  - Si no se cumple la precondición, el procedimiento puede hacer lo que quiera (incluso BOOM!)
- Permite la construcción de bibliotecas

Repaso (Precondiciones y equivalencia)

- Qué problemas resuelven los siguientes procedimientos
  - Pensar en proposito + precondición.
- ¿Son equivalentes?

```
procedure HacerNada() {
1. Poner(Azul)
2. Sacar(Azul)
}

procedure HacerNadaBis() {
1. Sacar(Azul)
2. Poner(Azul)
}

procedure HacerNadaTri() {
1. Mover(Este)
2. Mover(Oeste)
}
```

- Precondición: condiciones que tienen que satisfacer el cabezal, el tablero, y los parámetros para que el problema tenga solución (*).
- Procedimientos equivalentes: mismo propósito, precondiciones y tipos.

Repaso (Parámetros)

- Formalmente: identificador que permite denotar distintos valores cuando se invoca un procedimiento
- Intuitivamente: cajitas, asteriscos, etc, que reemplazan un valor fijo por una expresión que denota un valor cualquiera
- Motivación estudio: recordar la regla del copy&paste

```
Pone una bolita de color unColor
en la celda actual y
mueve el cabezal en dirección unaDireccion.
Precondición:  el cabezal no se
encuentra en la ultima columna en dirección unaDireccion
procedure PonerYMover(unColor, unaDireccion) {
  1. Poner(unColor)
  2. Mover(unaDireccion)
}
```

Repaso (Invocación)

- Para invocar un procedimiento dentro de otro, escribimos `NombreProc(argumentos)`
- Cada argumento denota un único valor
- La lista de argumentos se separa por comas
- El primer argumento corresponde al primer parámetro, el segundo argumento al segundo parámetro, etc.
  - `PonerYMover(Azul, Este)`, `PonerYMover(Verde, Oeste)`
- Funcionamiento: se interrumpe la ejecución del procedimiento actual y se ejecuta el procedimiento invocado.
  - Cada paramétro pasa a denotar el mismo valor que el argumento correspondiente.
  - Cuando el procedimiento invocado termina, se continúa ejecutando el procedimiento llamador.
- Contrato: si los argumentos y el tablero satisfacen la precondición, entonces el procedimiento cumple su propósito.

Repaso (alcance de parámetros)

- ¿Qué relación hay entre el parámetro `unColor` de PonerDos y PonerCuatro?

```
// Pone dos bolita de color unColor sin mover el cabezal
procedure PonerDos(unColor) {
  1. Poner(unColor)
  2. Poner(unColor)
}

// Pone cuatro bolitas de color unColor sin mover el cabezal
procedure PonerCuatro(unColor) {
  1. PonerDos(unColor)
  2. PonerDos(unColor)
}
```

- Alcance de un parámetro: bloque en el que es válido
- Nombre y apellido de un parámetro
  - "unColor de PonerDos" vs. "unColor de PonerCuatro"

Repetición simple: motivación

- ¿Cómo hacemos para poner 4 bolitas rojas?
- ¿Cómo hacemos para poner 10 bolitas rojas?
- ¿Cómo hacemos para poner 10000 bolitas rojas?

```
Propósito:  Agrega 10000 bolitas de color Rojo
procedure PonerDiezMilRojas() {
1. repeat(10000)
2. {
3.       Poner(Rojo)
4. }
}
```

Repetición simple: forma general

Repetición simple (repeat)

repeat ⟨numero⟩
    ⟨bloque a repetir⟩

donde,

- ⟨numero⟩ denota un número n cualquiera

- El **repeat** se puede escribir en los mismos lugares que los comandos
  - Bloque: secuencia de comandos y repeticiones (*)
- Funcionamiento: se ejecuta ⟨bloque a repetir⟩ tantas veces como el número n. Si $n \leq 0$, entonces no hace nada.

Repeat y parámetros

- Hacer un procedimiento `PonerVeinte(unColor)` que ponga veinte bolitas de color `c` en la celda actual.
- Hacer un procedimiento `PonerTreinta(unColor)` que ponga treinta bolitas de color `c` en la celda actual.
- Hacer un procedimiento `PonerMil(unColor)` que ponga 1000 bolitas de color `c` en la celda actual.

- Hacer un procedimiento `PonerN(unColor, unaCantidad)` que ponga `unaCantidad` bolitas de color `unColor` en la celda actual.
- ¿Qué hace `PonerN(unColor, -10)`?

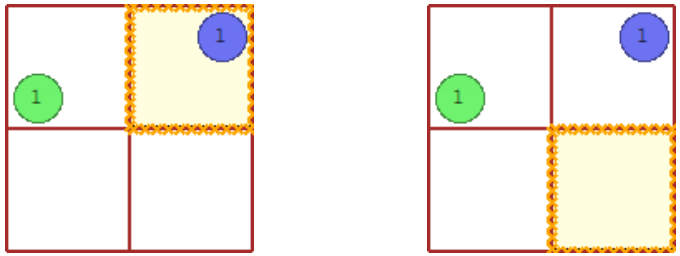
- Cualquier parámetro (u expresión) que denote un número puede usarse como cantidad de una repetición simple.

Formalizando I: valores y expresiones

- Valores: colores, direcciones, números.
- Expresión: cadena de símbolos (i.e., palabras) que denota un valor (*)
- Expresiones literales (o atómicas): denotan directamente un valor
  - Colores: `Azul`, `Negro`, `Rojo`, `Verde`
  - Direcciones: `Norte`, `Este`, `Sur`, `Oeste`
  - Números (enteros): ..., `-3`, `-2`, `-1`, `0`, `1`, `2`, `3`, ...
- Expresiones no literales: denotan valores indirectamente
  - identificadores: e.g. parámetros
  - operaciones: `1 + 2`, `7 - 8`, `minColor()`, `maxColor()`, `nroBolitas(Azul)`, etc
- Evaluar: determinar qué valor denota una expresión
- Cualquier expresión se puede utilizar en lugar de un literal

Formalizando II: expresiones y operaciones

- Distintas expresiones pueden denotar el mismo valor
  - e.g. `1`, `5 - 4`, `-2 + 3`, `nroBolitas(Azul)`
- La misma expresión puede denotar valores distintos en instantes diferentes
  - `nroBolitas(Azul)`



Operaciones

- denotan valores a partir del estado del tablero y los parámetros
- no modifican el estado del tablero
- se escriben en minúscula y tienen una lista de argumentos

Formalizando II: operaciones

Algunas operaciones con colores: *unColor* de tipo color

- `minColor()`: mínimo color en orden alfabético (`Azul`)
- `maxColor()`: máximo color en orden alfabético (`Verde`)
- `siguiente(`*c*`)`: color siguiente a *c* en orden alfabético
  - `siguiente(maxColor())` denota `minColor()`
- `previo(`*unColor*`)`: color anterior a *unColor* en orden alfabético
  - `previo(minColor())` denota `maxColor()`
- `nroBolitas(`*unColor*`)`: cantidad de bolitas de color *unColor* en la celda actual

Formalizando II: operaciones

Algunas operaciones con direcciones: *unaDireccion* de tipo dirección

- `minDir`(): mínima dirección en el sentido del reloj (`Norte`)
- `maxDir`(): máxima dirección en el sentido del reloj (`Oeste`)
- `siguiente`(*unaDireccion*) : *direcciónsiguienteaunaDireccion* en sentido del reloj.
  - `siguiente`(maxDir()) denota `minDir`()
- `previo`(*unaDireccion*): dirección anterior a *unaDireccion* en sentido del reloj
  - `previo`(`minDir`()) denota `maxDir`()
- `opuesto`(*unaDireccion*): dirección cardinal opuesta a *unaDireccion*

Formalizando II: operaciones

Algunas operaciones con números: *puntos* y *resto* son de tipo números

- *puntos* + *resto*: denota la suma de *puntos* y *resto*
- *puntos* − *resto*: denota la resta de *puntos* y *resto*
- *puntos* ∗ *resto*: denota la multiplicación *puntos* y *resto*
- *puntos* `div` *resto*: denota la división entera de *puntos* dividido *resto*
- *puntos* `mod` *resto*: denota el resto de la división entera de *puntos* dividido *resto*
- *puntos* `^` *resto*: denota el resultado de elevar *puntos* por *resto*, i.e., $puntos^{resto}$
- −*puntos*: denota 0 − *puntos*

Formalizando II: operaciones

- ¿Qué hace el siguiente procedimiento?

```
procedure AdivinarComportamiento() {
  1. Poner(minColor())
  2. Mover(siguiente(minDir())
  3. Poner(siguiente(minColor())
  4. Mover(minDir())
  5. Poner(previo(maxColor())
  6. Mover(previo(minDir())
  7. Poner(maxColor())
  8. Mover(opuesto(minDir())
}
```

- Recordar: cada vez que se requiera un valor se puede usar una expresión
  - Los literales son sólo una forma de expresión

Más ejemplos motivadores

- ¿Qué hace el procedimiento `MoverN(Este, 5)`?
- ¿Cuál es el propósito de `MoverN`?
- ¿Cuál es su precondición?

```
procedure MoverN(unaDireccion, unaCantidad) {
1. repeat(unaCantidad)
2. {
3.       Mover(unaDireccion)
4. }
}
```

- Recordar: para describir la cantidad podemos usar expresiones!

Ejemplo de expresiones

- ¿Qué hace el procedimiento `Duplicar(Azul)`?
- ¿Cuál es el propósito de `Duplicar`?
- ¿Cuál es su precondición?

```
procedure Duplicar(unColor) {
 1. PonerN(unColor, nroBolitas(unColor))
}
```

- Recordar: cualquier expresión puede usarse como argumento
- Ejercicio: escribir la función `SacarTodas` invocando a `SacarN`

Más ejemplos motivadores

- ¿Qué hace el procedimiento `RepartirBolitas(Este, Azul)`?
- ¿Cuál es el propósito de `RepartirBolitas` (cabezal!)?
- ¿Cuál es su precondición?

```
procedure RepartirBolitas(unaDireccion, unColor) {
1. repeat(nroBolitas(unColor))
2. {
3.       Mover(unaDireccion)
4.       Poner(unColor)
5. }
}
```

Formalizando III: argumentos y expresiones

- Formalmente, cada comando u operación tiene una lista de parámetros, que son simplemente identificadores
  - `Poner`(⟨color⟩), **procedure** PonerN(unColor,unaCantidad) ⟨bloque⟩
  - `nroBolitas`(⟨color⟩), `n + m`, **repeat**(⟨unNumero⟩) ⟨bloque⟩
- Al invocar un comando u operación, se debe incluir una lista con tantos argumentos como parámetros tenga el comando u operación invocado
  - Bien: `PonerN(Azul, 3) nroBolitas(Rojo)`, **repeat**(4) ⟨bloque⟩.
  - Mal: `PonerN(Azul) nroBolitas(Rojo, Azul)`, **repeat** ⟨bloque⟩
- Cada argumento debe ser una expresión que puede no ser literal
- La unica restricción es que el tipado sea correcto; clase que viene
  - Bien: `siguiente(Rojo)`, PonerN(`Azul`, `nroBolitas(Verde) + 2`)
  - Bien: **repeat**(`nroBolitas(Azul)div 12`) ⟨bloque⟩
  - Mal: `nroBolitas(Poner(Rojo))`, `Poner(Poner(Verde))`
- Recíprocamente, las expresiones sólo pueden usarse como argumentos.
  - Mal: **repeat**(1){ `Azul`}, **repeat**(1){ `nroBolitas(Azul)` }

Formalizando III: parámetros y expresiones

- Dentro de un procedimiento, cada párametro es una expresión
  - Puede usarse como argumento y sólo como argumento
  - Denota un valor desconocido hasta que el procedimiento se ejecuta.

```
//unColor es un parámetro y se usa como cualquier expresión
procedure Duplicar(unColor) {
  1. PonerN(unColor, nroBolitas(unColor))
}
```

- Cuando el procedimiento es invocado, el parámetro denota el valor correspondiente a su argumento durante esa invocación.
  - Durante la invocación de `Duplicar(Azul)`, su parámetro `unColor` denota el valor `Azul`
  - Cuando se ejecute luego la invocación de `PonerN`, su parámetro `unaCantidad` denotara la cantidad de bolitas azules de la celda actual.

Repetición indexada: motivación

- ¿Cómo hacemos para poner 1 bolita azul en la celda actual, 2 en la celda siguiente al este, 3 en la celda a dos de distancia al este, y 4 en la celda a distancia 3 al este?
- ¿Y si queremos hacerlo 10 veces?
- ¿Y en el caso de que queramos 10000 veces?

```
Propósito:  Hace una progresión de 10000 bolitas azules
Cabezal:  a 10000 celdas al este
Precondición:  Hay al menos 10000 celdas al Este de la celda actual
procedure ProgresionDiezMil() {
1. foreach i in [1..10000]
2. {
3.       PonerN(Rojo, i)
4.       Mover(Este)
5. }
}
```

Repetición indexada: forma general

Repetición indexada (foreach)

foreach ⟨indice⟩ **in** ⟨rango⟩
    ⟨bloque a repetir⟩

donde,

- ⟨indice⟩ es un identificador
- ⟨rango⟩ describe un rango $[n..m]$ (a ser definido)

- El **foreach** se puede escribir en los mismos lugares que los comandos
  - Bloque: secuencia de comandos y repeticiones (*)
- Funcionamiento: por cada elemento del rango, se ejecuta ⟨bloque a repetir⟩

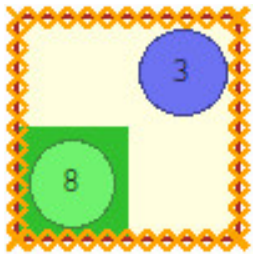
Formalizando: rangos

- Informalmente: lo que se escribe al final del `foreach`
- Formalmente: secuencia ordenada (menor a mayor) y finita de valores contiguos

- En GOBSTONES se escribe [⟨primero⟩..⟨ultimo⟩]
  - ⟨primero⟩ y ⟨ultimo⟩ son expresiones
  - Si ⟨ultimo⟩ es anterior a ⟨primero⟩, entonces el rango es vacío
  - **foreach** ejecuta ⟨bloque a repetir⟩ una vez por cada elemento del rango

Ejercitando rangos

- Indicar los valores (en orden) de los siguiente rangos
  - `[1..10]`:
  - `[-3..3]`:
  - `[10..1]`:
  - `[nroBolitas(Azul)..nroBolitas(Verde)]`:



Índice: motivación

- Suponga `DibujarLinea(longitud, deColor, hacia)` donde el cabezal queda en la celda actual
- ¿Qué hace el procedimiento Escalera(5, `Azul`)?
- ¿Cuál es el propósito de Escalera(longitud, deColor)?

```
procedure Escalera(longitud, deColor) {
1. foreach i in [1..longitud]
2. {
3.       DibujarLinea(i, deColor, Este)
4.       Mover(Norte)
5. }
}
```

- Los índices son expresiones que denotan los valores del rango
- Como cualquier expresión se pueden usar como argumentos y SOLO como argumentos

Formalizando IV: índices

- **foreach** ejecuta ⟨bloque a repetir⟩ una vez por cada elemento del rango
- Esta ejecución es en orden
  - Primer repetición: corresponde al primer elemento del rango
  - Segunda repetición: corresponde al segundo elemento del rango
  - *n*-ésima repetición: corresponde al *n*-ésimo elemento del rango
  - Última repetición: corresponde al último elemento del rango
- Índice: denota el valor del rango correspondiente
  - Escribiendo su identificador se accede al valor denotado

Rangos de colores

- ¿Qué hace el siguiente procedimiento?
- ¿Cuál es su precondición?

```
procedure ArcoIris() {
1. foreach unColor in [Azul..Verde]
2. {
3.       PonerYMover(unColor, Este)
4. }
}
```

- ¡Los colores también pueden ser índices!
  - Orden alfabético
  - Usar `[minColor()..maxColor()]` para recorrer todos los colores

Rangos de direcciones

- ¿Qué hace el siguiente procedimiento?
- ¿Cuál es su precondición?

```
procedure DibujarCuadradadito(color) {
1. foreach dir in [Norte..Oeste]
2. {
3.       Mover(dir)
4.       Poner(color)
5. }
}
```

- ¡Las direcciones también pueden ser índices!
  - Orden en el sentido de las agujas del reloj desde `Norte`
  - Usar `[minDir()..maxDir()]` para recorrer todas las direcciones

Repetición indexada (Resumen)

- La repetición indexada ejecuta ⟨bloque a repetir⟩ para cada elemento de un rango [⟨primero⟩..⟨ultimo⟩]
- ⟨primero⟩ y ⟨ultimo⟩ son expresiones
- El identificador ⟨indice⟩ es un expresión que denota el elemento del rango correspondiente a la repetición actual
- El rango puede ser numérico, de colores, o de direcciones
- El concepto de bloque se extiende para incluir **foreach**
- El tema de valores y expresiones lo repasamos la clase que viene