Programación

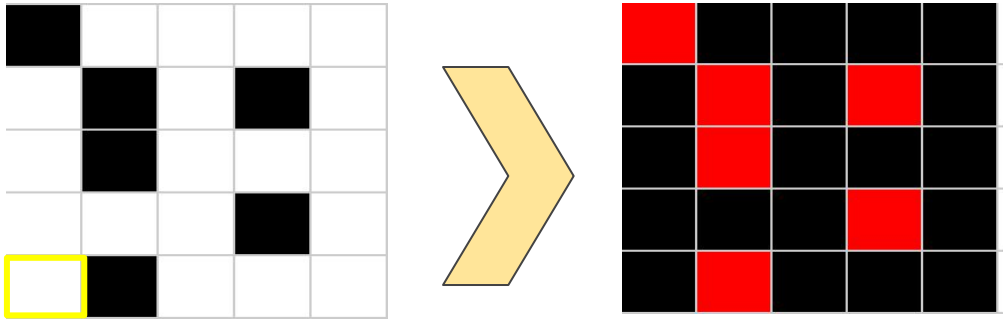
Alternativa Condicional

Universidad Nacional de Quilmes

Pensemos el siguiente problema

A partir del tablero de un juego de 5x5, se solicita pintar de color rojo sólo las celdas negras, y al resto pintarlas de negro. El cabezal debe iniciar en la esquina inferior izquierda del tablero.

Tener en cuenta que ***no conocemos a priori cuáles son las celdas pintadas de negro*** *en el tablero del juego*. Veamos un ejemplo de una posible situación:



Estado inicial del tablero

Estado final del tablero

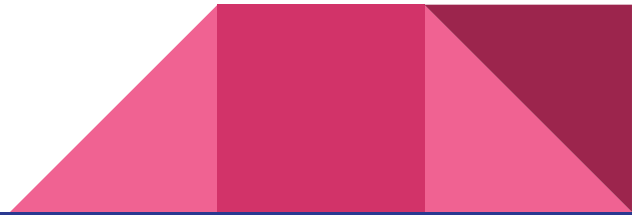
La construcción de un programa o procedimiento que cumpla con la tarea de pintar las celdas del tablero del juego requiere que se puedan ejecutar instrucciones distintas según las celdas sean o no negras, es decir, en base a una condición.

En el lenguaje QDraw es posible escribir un código que realice esa tarea. Pero, para poder hacerlo, requerimos una nueva estructura de control y nuevas primitivas.

Definición

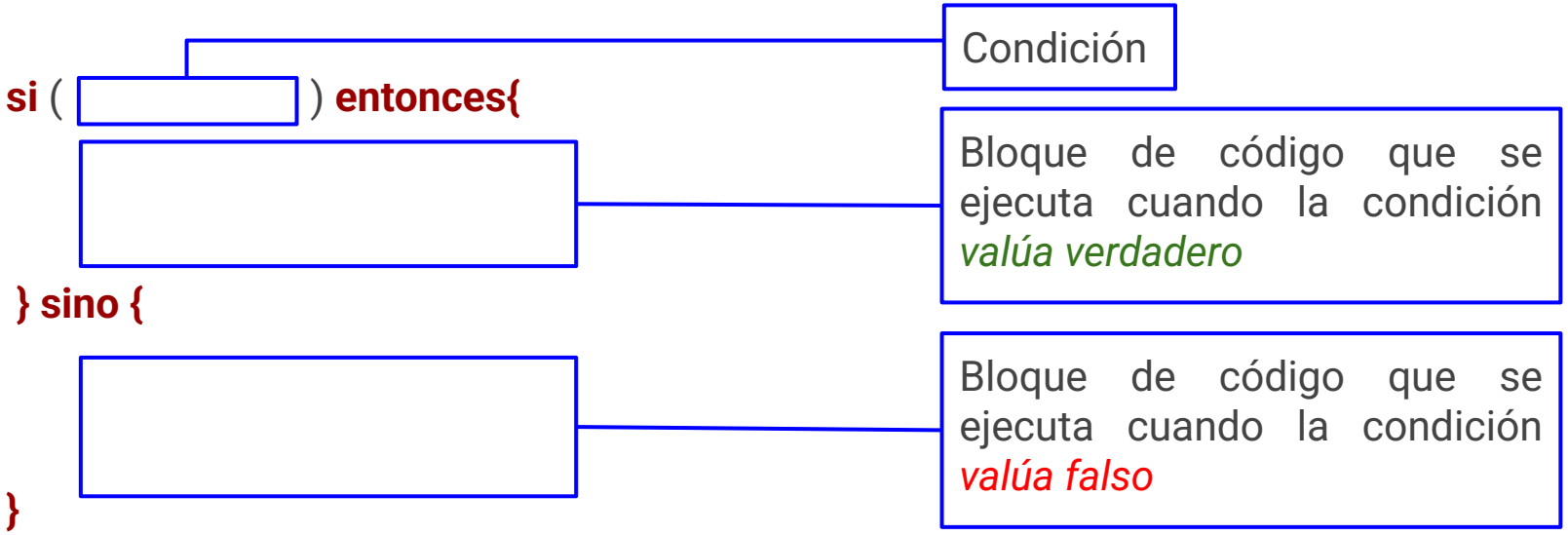
La **alternativa condicional** es una estructura de control que permite al programa determinar, a partir de una **condición**, si un **bloque de código** se **ejecuta** o **no se ejecuta**

Permite elegir dos caminos distintos de acción en base a una condición. La alternativa condicional se agrega al lenguaje QDraw como una nueva estructura de control.



Sintaxis

En QDraw, la sintaxis de una alternativa condicional es así:

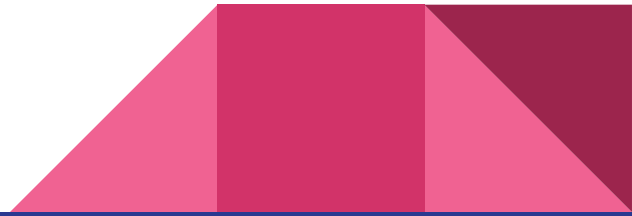


Sintaxis

La **condición** sólo puede tomar *valores booleanos* (o binarios), es decir, verdadero o bien falso. Estos valores son los que permiten la ejecución de ciertos bloques de código. En la condición vamos a encontrar **primitivas booleanas** (que veremos más adelante) y tres conectivas: la **conjunción**, la **disyunción** y la **negación**.

En los **bloques de código** vamos a encontrar las primitivas de movimiento, las que permiten pintar o despintar celdas y/o procedimientos.

Veamos un ejemplo simple de la **alternativa condicional**.



Ejemplo simple

```
procedimiento Marcar() {
/**/
    si (estaPintadaDeNegro?) entonces {
        PintarRojo
    } sino {
        PintarNegro
    }
}
```

Primitiva condicional

Se ejecuta el bloque de código superior *(si)* o el inferior *(sino)* dependiendo del **estado** de la **celda actual**. En este caso, si está pintada de negro, entonces la pinta de rojo y en caso contrario, la pinta de negro.

Un ejemplo más completo

Poco a poco vamos enriqueciendo nuestro lenguaje con un repertorio mayor de instrucciones y estructuras de control. De esta manera, podemos sumar herramientas que, al combinarse, nos permitan resolver problemas más complejos.

```
procedimiento MarcarColumna () {
/* ... */
    repetir 4 veces {
        Marcar()
        MoverArriba
    }
    Marcar()
}
```

Veamos aquí un ejemplo de cómo podemos utilizar el procedimiento anterior como parte de un procedimiento más general y ya conocido, que evalúe todas las celdas de una columna.

Condiciones

Recordemos que las **condiciones** sólo pueden tomar valores booleanos (o binarios), es decir, verdadero o bien falso. Estos valores booleanos son los que permiten discernir entre dos alternativas.

Nuevas primitivas al repertorio del lenguaje

Para consultar si una celda está pintada de algún color o si está vacía, contamos en el lenguaje QDraw, con un conjunto de **primitivas** que, según el estado de la celda, denotan un valor booleano (verdadero/falso). Estas nuevas **primitivas** son las siguientes:

- **estaPintadaDeNegro?**: Denota verdadero si la celda está pintada de **negro**, falso en cualquier otro caso.
- **estaPintadaDeRojo?:** Denota verdadero si la celda está pintada de **rojo**, falso en cualquier otro caso.
- **estaPintadaDeVerde?**: Denota verdadero si la celda está pintada de **verde**, falso en cualquier otro caso.
- **estaVacia**?: Denota verdadero si la celda está vacía, falso en cualquier otro caso

El uso de las conectivas

En el lenguaje QDraw también podemos armar expresiones booleanas al usar conectivas lógicas con las primitivas booleanas. Las tres conectivas lógicas que podemos usar son la **conjunción**, la **disyunción** y la **negación**. Repasemos cada tabla.

Conjunción

p	q	p ∧ q
V	V	V
V	F	F
F	V	F
F	F	F

Disyunción

p	q	p v q
V	V	V
V	F	V
F	V	V
F	F	F

Negación

p	¬p
V	F
F	V

Veamos ejemplos con conectivas

Podría ser que necesitemos ejecutar la misma instrucción para dos estados distintos de las celdas. En esos casos, podemos usar las conectivas.

En ambos ejemplos las condiciones son verdaderas en los mismos **casos (equivalentes):** cuando las celdas están pintadas de verde o de rojo.

```
si (estaPintadaDeVerde? v estaPintadaDeRojo?) entonces {
     Limpiar
} sino {
     PintarNegro
}
```

```
si (¬estaPintadaDeNegro? ^ ¬estaVacia?) entonces {
     Limpiar
} sino {
     PintarNegro
}
```

Algo más sobre equivalencias en las condiciones

Como una celda no puede estar pintada de dos colores, o estar vacía y pintada a la vez, vamos a encontrar en el lenguaje QDraw que algunas expresiones booleanas, que no son equivalentes desde el punto de vista de la lógica, son equivalente en QDraw, como ocurre en el ejemplo anterior.
Otros ejemplos de expresiones booleanas equivalentes en QDraw son:

(¬estaVacia?) = (estaPintadaDeNegro? v estaPintadaDeRojo? v estaPintadaDeVerde?)

(estaVacia? v estaPintadaDeVerde?) = (¬(estaPintadaDeNegro? v estaPintadaDeRojo?))

Tener en cuenta que:

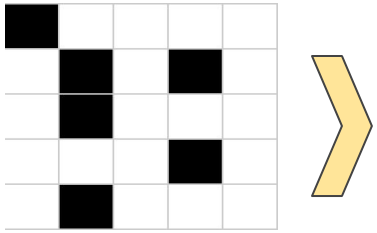
(estaVacia? v estaPintadaDeVerde?) ≠(¬estaPintadaDeNegro? v ¬estaPintadaDeRojo?)

Alternativa Condicional

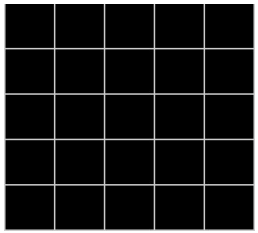
Forma acotada

Analicemos el siguiente código

Supongamos que con un procedimiento se marcan (pintan) con color negro todas las celdas de un cuadrado, ignorando aquellas que ya están pintadas de ese color. Contamos con el siguiente procedimiento:



Estado inicial del cuadrado



Estado final del cuadrado

```
procedimiento MarcarVacia() {
/**/
     si (estaVacia?) entonces {
          PintarNegro
     } sino {

     }
}
```

¿Notan algo raro?

Forma Acotada - Caso I

¡Exacto! Tenemos un bloque de código al que no le damos ningún uso.

Cuando necesitemos ejecutar una acción mediante sólo una alternativa, el bloque de código que no se utiliza lo debemos **eliminar por completo**, simplificando así el código y su lectura. En este caso, la alternativa eliminada es aquella cuya condición **valúa falso** (bloque del "**sino**"). Como lo podemos ver en el ejemplo.

```
procedimiento MarcarVacia() {
/**/
    si (estaVacia?) entonces {
        PintarNegro
    }
}
```

Forma Acotada - Caso II

¿Qué pasa si el bloque de código en desuso es aquel cuya condición **valúa verdadero**?

Mmm...

```
procedimiento MarcarNegro() {
/**/
      si (estaPintadaDeNegro?) entonces {

      } sino {
            PintarNegro
      }
}
```

En este caso no se puede simplemente eliminar el bloque como en la situación anterior. Lo que conviene hacer es cambiar la pregunta **negando** la **condición original**, e invertir el orden de los bloques de la alternativa condicional.

¿Les resulta familiar?

```
procedimiento MarcarNegro() {
/**/
si (¬estaPintadaDeNegro?) entonces {
            PintarNegro
      }
}
```

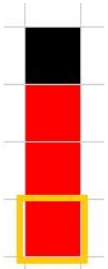
No tenemos bloques en desuso y seguimos cumpliendo con el propósito

Veamos un ejemplo aplicando la disyunción

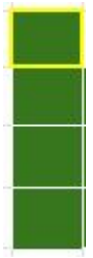
```
procedimiento MarcarVerde() {
/*PROPÓSITO: marcar de verde la celda que sea negra o roja. */
      si(estaPintadaDeNegro? v estaPintadaDeRojo?) entonces {
            PintarVerde
      }
}
```

Notar que al no haber desplazamientos, el procedimiento no tiene **precondición**. ¡No hay peligro de BOOM!

```
procedimiento MarcarColumna() {
/*PROPÓSITO: marcar de verde la columna cuyas celdas sean negra o roja. El
cabezal finaliza en el extremo superior de la columna.
PRECONDICIÓN: El cabezal inicia en la base de la columna, y debe haber 3 celdas
hacia arriba.*/
      repetir 3 veces {
            MarcarVerde()
            MoverArriba
      }
      MarcarVerde()
}
```



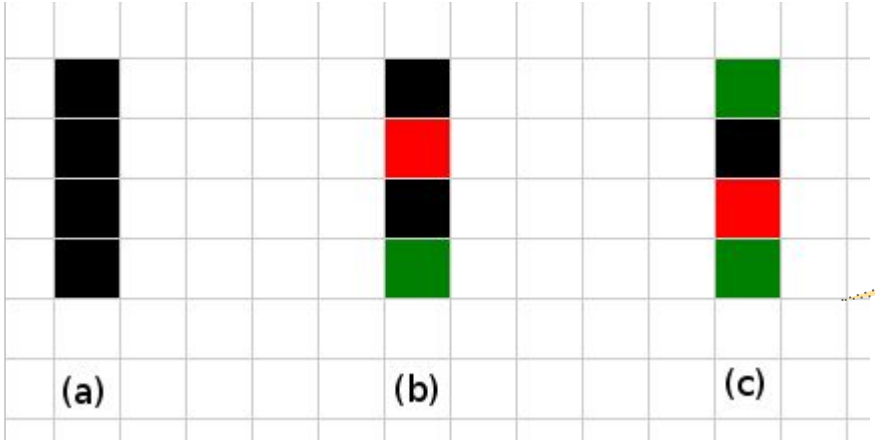
Ejemplo inicial



Ejemplo final

Analicemos mejor

Evaluar el procedimiento **MarcarColumnas()** para las siguientes situaciones.
Responder a la pregunta planteada:



¿Cómo queda el estado final de las columnas en cada caso, luego de haberse ejecutado el procedimiento?

Anidaciones: No son buena práctica

```
procedimiento MarcarVerde() {
      si(estaPintadaDeNegro?) entonces {
            si (estaPintadaDeRojo?) entonces {
                  PintarVerde
            }
      }
}
```

¡ANIDAR! No es una buena práctica en la alternativa condicional. El código no es legible y es propenso a errores. Además, este código **no funciona**.

```
procedimiento MarcarVerde() {
      si (estaPintadaDeNegro? v estaPintadaDeRojo?) entonces {
            PintarVerde
      }
}
```

Solución: utilizar la conectiva correspondiente en base al propósito.

Ejemplos que no aplican buenas prácticas

```
procedimiento MarcarColor() {
        si(estaPintadaDeNegro?) entonces {
                PintarVerde
        }
        si (estaVacia?) entonces {
                PintarRojo
        }
}
```

¡ALTERNATIVAS CONDICIONALES CONSECUTIVAS! No es una buena práctica, al igual que en repeticiones. El código no es legible y es propenso a errores.

```
procedimiento MarcarColor(){
        si(estaPintadaDeNegro?) entonces {
                PintarVerde
        }sino {
                si (estaVacia?) entonces {
                        PintarRojo
                }
        }
}
```

¡ALTERNATIVAS CONDICIONALES ANIDADAS! En un principio parecería una solución, pero nuevamente esta versión no es la acertada en términos de buenas prácticas.

Solución aplicando buenas prácticas

```
procedimiento MarcarColor(){
      si (estaPintadaDeNegro?) entonces {
            PintarVerde
      } sino {
            MarcarRojo()
      }
}
procedimiento MarcarRojo() {
      si (estaVacia?) entonces {
            PintarRojo
      }
}
```

Solución: descomponer el problema. Acá vemos una posible solución, pero todo dependerá del propósito.

Esta solución permite cumplir con el propósito de una manera más legible al dividir el problema en 2 partes más sencillas, donde cada una resuelve un problema específico.

¿Estructuras de control anidadas? ¡No!

Caso I

```
procedimiento MarcarColores() {
      repetir 2 veces {
            si (estaPintadaDeNegro?) entonces {
                  PintarVerde
            }
            MoverArriba
      }
}
```

Caso II

```
procedimiento MarcarColores(){
      si (estaPintadaDeNegro?) entonces {
            repetir 2 veces {
                  MarcarColor()
            }
      }
}
```

Anidar diferentes estructuras de control tampoco es una buena práctica.
No se entiende qué se desea hacer.

Solución a la anidación de estructuras de control

```
procedimiento MarcarColores(){
      repetir 2 veces {
            MarcarVerde()
            MoverArriba
      }
}
procedimiento MarcarVerde() {
      si (estaPintadaDeNegro?) entonces {
            PintarVerde
      }
}
```

Caso I

```
procedimiento MarcarColores(){
      si (estaPintadaDeNegro ?) entonces {
            MarcarDosColores()
      }
}
procedimiento MarcarDosColores() {
      repetir 2 veces {
            MarcarColor()
      }
}
```

Caso II

Recordemos que…

```
procedimiento IrACopaArbol(){
      repetir 4 veces {
            repetir 2 veces {
                  MoverArriba
            }
      }
}
```

```
procedimiento IrACopaArbol(){
      repetir 8 veces {
            MoverArriba
      }
}
```

Recordemos que **anidar** estructuras de repetición tampoco es una buena práctica. En casos como estos, se debe unificar la cantidad de repeticiones en una sola.

¿Estructuras de control consecutivas? ¡Tampoco!

Caso I

```
procedimiento MarcarColores() {
      repetir 2 veces {
            MoverArriba
      }
      si (estaPintadaDeNegro?) entonces {
            PintarVerde
      }
}
```

Caso II

```
procedimiento MarcarColores(){
      si (estaPintadaDeNegro?) entonces {
            PintarVerde
      }
      repetir 2 veces {
            MoverArriba
      }
}
```

Diferentes estructuras de control **consecutivas** tampoco es una buena práctica. El código es poco legible, poco mantenible y propenso a errores.

Solución a las estructuras de control consecutivas

```
procedimiento AvanzarDosPasos() {
    repetir 2 veces {
        MoverArriba
    }
}

procedimiento MarcarVerde() {
    si (estaPintadaDeNegro?) entonces {
        PintarVerde
    }
}
```

En casos como estos, lo mejor es **descomponer (subdividir en tareas)** cada una de las estructuras de control en un procedimiento y luego invocar los procedimientos definidos en el orden que corresponda.

Solución a las estructuras de control consecutivas

Caso I

```
procedimiento MarcarColores() {
        AvanzarDosPasos()
        MarcarVerde()
}
```

Caso II

```
procedimiento MarcarColores() {
        MarcarVerde()
        AvanzarDosPasos()
}
```

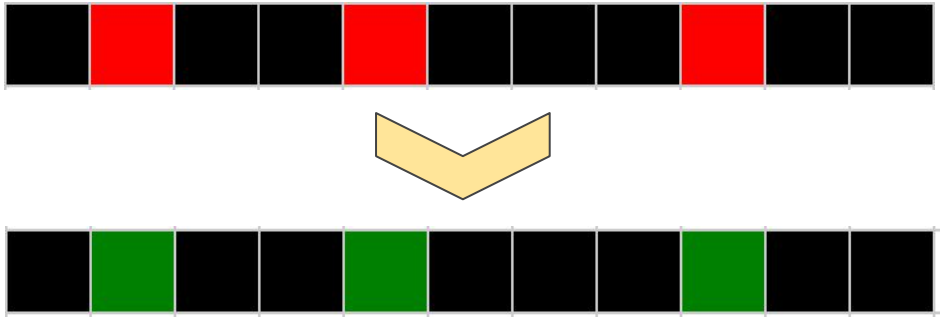
Resumiendo...

- La alternativa condicional permite elegir en base a una condición
- Las condiciones pueden tomar valores o bien verdaderos o bien falsos
- En las condiciones podemos utilizar operadores lógicos como negación, conjunción y disyunción
- Utilizar la versión acotada en lugar de dejar bloques vacíos
- La alternativa condicional no se puede anidar ni utilizar de manera consecutiva, al igual que la repetición simple
- Tampoco es legible anidar varias ni distintas estructuras de control, ni colocarlas de manera consecutiva.

Ejercicio para calentar

Dada una fila de celdas pintadas de negro con algunas de ellas pintadas de rojo. Se solicita marcar (pintar) con verde solamente las celdas rojas.

Estado inicial de la fila



Estado final de la fila

Para reflexionar...

"Estudiar no es un acto de consumir ideas, sino de crearlas y recrearlas"