```
//PARÁMETROS
procedure Poner_DeColor_(cantidadAPoner, colorAPoner){
  /*
    PROPÓSITO: Pone **cantidadAPoner** bolitas de color **colorAPoner** en la celda actual.
    PRECONDICIONES: Sin precondiciones.
    PARÁMETROS: cantidadAPoner - Número - Cantidad de bolitas que se ponen en la celda actual.
                colorAPoner - Color - Indica el color de las bolitas que se ponen en la celda actual.
  */
  repeat(cantidadAPoner){
    Poner(colorAPoner)
  }
}

procedure Mover_VecesAl_(cantidadAMover, direcciónAMover){
  /*
    PROPÓSITO: Mover el cabezal **cantidadAMover** veces en dirección al **direcciónAMover** de la celda actual.
    PRECONDICIONES:
      * Debe haber al menos **cantidadAMover** celdas hacia **direcciónAMover** de la celda actual.
    PARÁMETROS: cantidadAMover - Número - Cantidad de veces que se mueve el cabezal de la celda actual.
                direcciónAMover - Dirección - Indica la dirección en la que se va a mover el cabezal.
  */
  repeat(cantidadAMover){
    Mover(direcciónAMover)
  }
}
```

```
procedure Sacar_DeColor_(cantidadASacar, colorASacar){
  /*
    PROPÓSITO: Sacar **cantidadASacar** bolitas de color **colorASacar** de la celda actual.
    PRECONDICIONES:
      * Debe haber al menos **cantidadASacar** bolitas de color **colorASacar** en la celda actual.
    PARÁMETROS: cantidadASacar - Número - Cantidad de veces que se saca una bolita de la celda actual.
                colorASacar - Color - Indica el color de la bolita que se saca en la celda actual.
  */
  repeat(cantidadASacar){
    Sacar(colorASacar)
  }
}

//EXPRESIONES Y TIPOS
procedure SacarTodasLasDeColor_(colorASacar){
  /*
    PROPÓSITO: Sacar todas las bolitas de color **colorASacar** de la celda actual.
    PRECONDICIONES:
      * Debe haber al menos 1 bolita de color **colorASacar** en la celda actual.
    PARÁMETROS: colorASacar - Color - Indica el color de la bolita que se saca en la celda actual.
  */
  repeat(nroBolitas(colorASacar)){
    Sacar(colorASacar)
  }
}
```

```
//ALTERNATIVA CONDICIONAL
procedure Poner_Si_(color, condición){
  /*
    PROPÓSITO: Pone una bolita del color **color** en la celda actual,
    si se cumple **condiciòn**.
    PRECONDICIONES: Sin precondiciones.
    PARÁMETROS:
      * color - Color - Color de la bolita a poner.
      * condiciòn - Booleano - La condiciòn a cumplir para poner la bolita.
  */
  if (condición) {
    Poner(color)
  }
}

procedure Sacar_Si_(color, condiciòn){
  /*
    PROPÓSITO: Saca una bolita del color **color**en la celda actual,
    si se cumple **condiciòn**.
    PRECONDICIONES: Debe haber una bolita de **color** si se cumple **condiciòn**.
    PARÁMETROS:
      * color - Color - Color de la bolita al sacar.
      * condiciòn - Booleano - La condiciòn a cumplir para sacar la bolita.
  */
  if (condición) {
    Sacar(color)
  }
}
```

```
procedure Mover_Si_(dirección, condiciòn){
  /*
    PROPÓSITO: Mueve el cabezal en direcciòn hacia el **dirección** desde la celda actual,
    si se cumple **condiciòn**.
    PRECONDICIONES: Debe haber una celda en direcciòn hacia el **direcciòn**
    si se cumple **condiciòn**.
    PARÁMETROS:
      * dirección - Direcciòn - La direcciòn hacia la cual se debe mover el cabezal.
      * condiciòn - Booleano - La condiciòn a cumplir para mover el cabezal.
  */
  if (condiciòn) {
    Mover(direcciòn)
  }
}

//FUNCIONES
function tieneUnaDeCada(){
  /*
    PROPÓSITO: Indica si hay al menos una bolita de cada color en la celda actual.
    PRECONDICIONES: Sin precondiciones.
    TIPO: Booleano
  */
  return(hayBolitas(Azul)&&hayBolitas(Negro)&&hayBolitas(Rojo)&&hayBolitas(verde))
}
```

```
function esCeldaVacía(){
  /*
    PROPÓSITO: Indica si la celda actual està vacìa.
    PRECONDICIONES: Sin precondiciones.
    TIPO: Booleano
  */
  return((nroBolitas(Azul)+nroBolitas(Negro)+nroBolitas(Rojo)+nroBolitas(verde))==0)
}

function esCeldaConBolitas(){
  /*
    PROPÓSITO: Indica si en la celda actual faltan bolitas de algun color y no esta vacìa.
    PRECONDICIONES: Sin precondiciones.
    TIPO: Booleano
  */
  return(not(tieneUnaDeCada)&&not(esCeldaVacía))
}

//REPETICION CONDICIONAL
procedure IrAPrimeraCeldaEnUnRecorridoAl_Y_(dirPrincipal, dirSecundaria){
  /*
    PROPÓSITO: Mueve el cabezal hacia la esquina **dirPrincipal** y **dirSecundaria**.
    PRECONDICIONES: Los paràmetros **dirPrincipal** y **dirSecundaria** no pueden ser
opuestos
    ni iguales.
    PARÀMETROS:
      * dirPrincipal - Direcciòn - Direcciòn principal del recorrido.
      * dirSecundaria - Direcciòn - Direcciòn secundaria del recorrido.
  */
  IrAlBorde(opuesto(dirPrincipal))
  IrAlBorde(opuesto(dirSecundaria))    }
```

```
function haySiguienteCeldaEnUnRecorridoAl_Y_(dirPrincipal, dirSecundaria){
  /*
    PROPÓSITO: Indica si hay celda siguiente en el recorrido hacia **dirPrincipal** o **dirSecundaria**
    PRECONDICIONES:  Los paràmetros **dirPrincipal** y **dirSecundaria** no pueden ser opuestos
    ni iguales.
    PARÀMETROS:
      * dirPrincipal - Direcciòn - Direcciòn principal hacia la cual se mueve el cabezal.
      * dirSecundaria - Direcciòn - Direcciòn secundaria hacia la cual se mueve el cabezal.
    TIPO: Booleano
  */
  return(puedeMover(dirPrincipal)||puedeMover(dirSecundaria))
}

procedure IrASiguienteCeldaEnUnRecorridoAl_Y_(dirPrincipal, dirSecundaria){
  /*
    PROPÓSITO: Mueve el cabezal a la siguiente celda en un recorrido hacia **dirPrincipal**
    y **dirSecundaria**
    PRECONDICIONES:
      *Los paràmetros **dirPrincipal** y **dirSecundaria** no pueden ser opuestos
    ni iguales.
      *Debe haber al menos una siguiente celda en el recorrido.
    PARÀMETROS:
      * dirPrincipal - Direcciòn - Direcciòn principal hacia la cual se mueve el cabezal.
      * dirSecundaria - Direcciòn - Direcciòn secundaria hacia la cual se mueve el cabezal.
  */
  if(puedeMover(dirPrincipal)){
    Mover(dirPrincipal)
  }else{
    IrAlBorde(opuesto(dirPrincipal))
    Mover(dirSecundaria)  }  }
```

```
////VARIABLES
function hayBolitas_Al_(color, dirección){
  /*
    PROPÓSITO:Indica si hay bolitas de color **color** en la celda lindante en direccion
**direccion**
    PRECONDICIONES: Ninguna
    PARAMETROS:
      *dirección - Dirección - La dirección en la cual buscar las bolitas.
      *color - Color - El color de las bolitas a buscar.
    TIPO:Boleeano
  */
  return(puedeMover(dirección)&& tieneBolitas_Al_(color, dirección))
}

function tieneBolitas_Al_(color, dirección){
  /*
    PROPÓSITO:Indica si hay bolitas de color **color** en la celda lindante en direccion
**direccion**
    PRECONDICIONES: Debe haber al menos una celda en direccion **direccion**
    PARAMETROS:
      *dirección - Dirección - La dirección en la cual buscar las bolitas.
      *color - Color - El color de las bolitas a buscar.
    TIPO:Boleeano
  */
  Mover(dirección)
  return(hayBolitas(color))
}
```

```
function mínimoEntre_Y_(primerNumero, segundoNumero){
   /*
      PROPÓSITO:Describe cual es el mínimo entre **primerValor** y **segundoValor**
      PRECONDICIONES: **primerValor** y **segundoValor** deben ser del mismo tipo.
      PARAMETROS:
         *primerNumero - Número - primer numero a evaluar
         *segundoNumero - Número - segundo numero a evaluar
      TIPO:Número
   */
   return( choose primerNumero when primerNumero < segundoNumero 7
               segundoNumero otherwise )
}

function máximoEntre_Y_(primerValor, segundoValor){
   /*
      PROPÓSITO:Describe cual es el máximo entre **primerValor** y **segundoValor**
      PRECONDICIONES: **primerValor** y **segundoValor** deben ser del mismo tipo.
      PARAMETROS:
         *primerNumero - Número - primer numero a evaluar
         *segundoNumero - Número - segundo numero a evaluar
      TIPO:Número
   */
   return( choose primerNumero when primerNumero > segundoNumero
               segundoNumero otherwise )
}
```

```
function distanciaAlBorde_(dirección){
  /*
    PROPÓSITO: Describe la cantidad de celdas que hay entre la celda actual y el borde en direccion **direccion**
    PRECONDICIONES: Sin precondiciones
    PARAMETROS:
      * direccion - Dirección - La direccion hacia la cual se mueve el cabezal.
    TIPO:Numero
    OBSERVACIONES: Recorrido de acumulacion, contando la cantidad de celdas que hay hacia el borde indicado.
  */
  cantidadHaciaElBorde := 0
  while(puedeMover(direccion)){
    cantidadHaciaElBorde := cantidadHaciaElBorde + 1
  }
 return(cantidadHaciaElBorde)
}

function coordenadaX(){
  /*
    PROPÓSITO: Describe el numero de celda, con respecto a la fila de la celda actual, en la que se encuentra el cabezal
    PRECONDICIONES: Sin precondiciones
    TIPO:Numero
  */
  cantidadHaciaOeste := 0
  while(puedeMover(Oeste)){
    cantidadHaciaOeste := cantidadHaciaOeste + 1
  }
 return(cantidadHaciaOeste)
}
```

```
function coordenadaY(){
   /*
      PROPÓSITO: Describe el numero de celda, con respecto a la columna de la celda actual, en la que se encuentra
       el cabezal.
      PRECONDICIONES: Sin precondiciones
      TIPO:Numero
   */
   cantidadHaciaSur := 0
   while(puedeMover(Sur)){
      cantidadHaciaSur := cantidadHaciaSur + 1
   }
  return(cantidadHaciaSur)
}

function nroFila(){
   /*
      PROPÓSITO: Describe la cantidad de filas que hay en el tablero.
      PRECONDICIONES: Sin precondiciones
      TIPO:Numero
   */
   cantidadHaciaEste := 0
   IrAlBorde(Oeste)
   while(puedeMover(Este)){
      cantidadHaciaEste := cantidadHaciaEste + 1
   }
  return(cantidadHaciaEste)
}
```

```
function nroColumna(){
   /*
      PROPÓSITO: Describe la cantidad de columnas que hay en el tablero.
      PRECONDICIONES: Sin precondiciones
      TIPO:Numero
   */
   cantidadHaciaNorte := 0
   IrAlBorde(Sur)
   while(puedeMover(Norte)){
      cantidadHaciaNorte := cantidadHaciaNorte + 1
   }
  return(cantidadHaciaNorte)
}

function nroVacía(){
   /*
      PROPÓSITO: Describe la cantidad de celdas vacias que hay en el tablero.
      PRECONDICIONES: Sin precondiciones
      TIPO:Numero
   */
   IrAPrimeraCeldaEnUnRecorridoAl_Y_(Este,Norte)
   cantidadSinBolitas := 0
   while(haySiguienteCeldaEnUnRecorridoAl_Y_(Este,Norte)){
      cantidadSinBolitas:= cantidadSinBolitas + unoSi_CeroSino(esCeldaVacía())
      IrASiguienteCeldaEnUnRecorridoAl_Y_(Este,Norte)
   }

  return(cantidadSinBolitas:= cantidadSinBolitas + unoSi_CeroSino(esCeldaVacía()))
}
```

```
function cantidadDeCeldasConBolitasDeColor_(color){
  /*
    PROPÓSITO: Describe la cantidad de celdas con al menos una bolita de color **color** que hay en el tablero.
    PRECONDICIONES: Sin precondiciones
    TIPO:Numero
  */
  IrAPrimeraCeldaEnUnRecorridoAl_Y_(Este,Norte)
  cantidadConBolitas := 0
  while(haySiguienteCeldaEnUnRecorridoAl_Y_(Este,Norte)){
    cantidadConBolitas:= cantidadConBolitas + unoSi_CeroSino(not esCeldaVacía())
    IrASiguienteCeldaEnUnRecorridoAl_Y_(Este,Norte)
  }

 return(cantidadConBolitas:= cantidadConBolitas + unoSi_CeroSino(not esCeldaVacía()))
}

function nroBolitasTotalDeColor_(color){
  /*
    PROPÓSITO: Describe la cantidad de bolitas de color **color** que hay en el tablero.
    PRECONDICIONES: Sin precondiciones
    TIPO:Numero
  */
  IrAPrimeraCeldaEnUnRecorridoAl_Y_(Este,Norte)
  cantidadDeBolitas := 0
  while(haySiguienteCeldaEnUnRecorridoAl_Y_(Este,Norte)){
    cantidadDeBolitas:= cantidadDeBolitas + cantidadDeBolitasEnLaCeldaActual()
    IrASiguienteCeldaEnUnRecorridoAl_Y_(Este,Norte)
  }

 return(cantidadDeBolitas:= cantidadDeBolitas + cantidadDeBolitasEnLaCeldaActual())    }
```

```
function cantidadDeBolitasEnLaCeldaActual(){
   /*
      PROPÓSITO:Describe la cantidad de bolitas en la celda actual
      PRECONDICIONES: Sin precondiciones.
      TIPO: Numero
   */
   return((nroBolitas(Azul)+nroBolitas(Negro)+nroBolitas(Rojo)+nroBolitas(verde))
}

function unoSi_CeroSino(condicion){
   /*
      PROPÓSITO: Describe 1 cuando se cumple una condiciòn o 0 en caso contrario
      PRECONDICIONES: Sin precondiciones
      TIPO:Numero
   */
  return(choose 1 when condicion
                0 otherwise)}
```