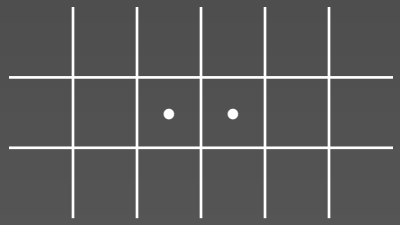
PRINCIPIOS DE PROGRAMACION

Reglas del juego de la Vida

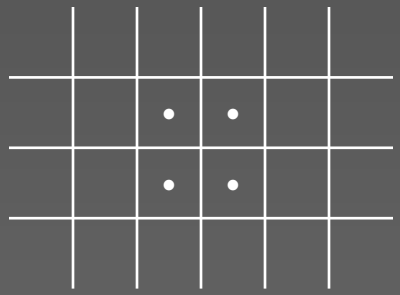
La estructura del juego es una rejilla rectangular abierta, en la cual cada celdilla puede estar ocupada o no por un microorganismo. Las celdillas cambian de acuerdo con las siguientes reglas:

- Los vecinos de una celda son los 8 que la tocan vertical, horizontal o diagonalmente.
- Si una celdilla está viva pero no tiene celdillas vecinas vivas o solamente una viva, muere de soledad en la siguiente generación.
- Si una celdilla está viva y tiene cuatro o más vecinas también vivas, en la siguiente generación muere por hacinamiento.
- Una celdilla viva con dos o tres vecinas vivas permanece viva en la siguiente generación.
- Si una celdilla está muerta, en la siguiente generación recuperará la vida si tiene exactamente tres vecinas, ni una más ni una menos, que ya estén vivas. Todas las otras celdillas muertas permanecen así en la siguiente generación.
- Todos los nacimientos y muertes tienen lugar exactamente al mismo tiempo, de manera que las que mueren ayudan a producir otras; pero no pueden impedir la muerte de otras reduciendo el hacinamiento, ni las que nacen pueden preservar o destruir a las que tienen vida en la siguiente generación.

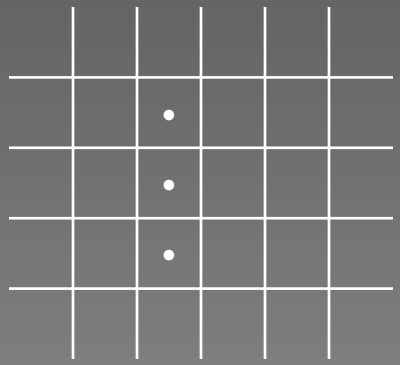
Ejemplos



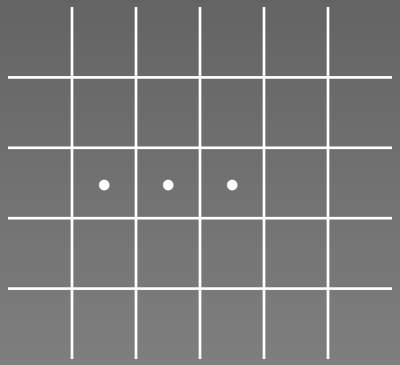
Esta comunidad se extingue en la próxima generación.



Esta comunidad nunca cambiará

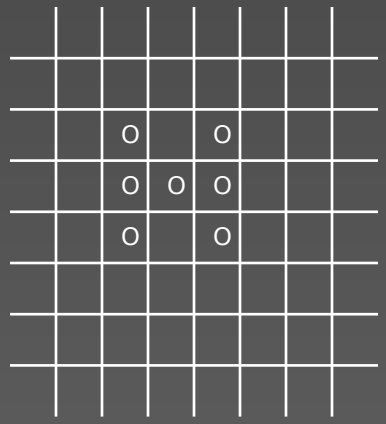


Esta comunidad se alternará por todas las generaciones

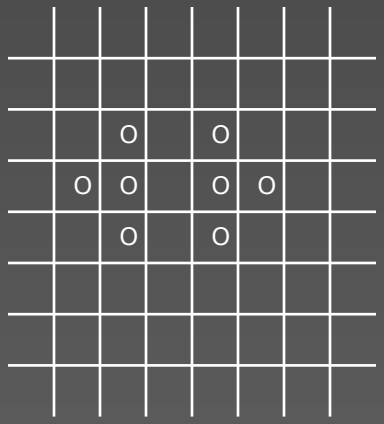


Secuencia de un comunidad

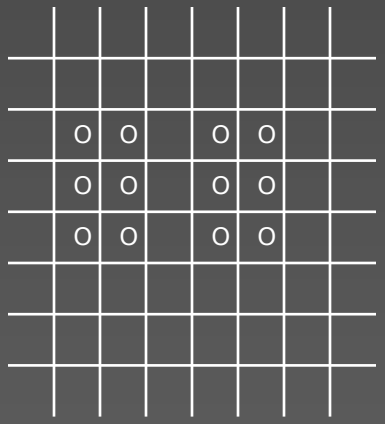
1



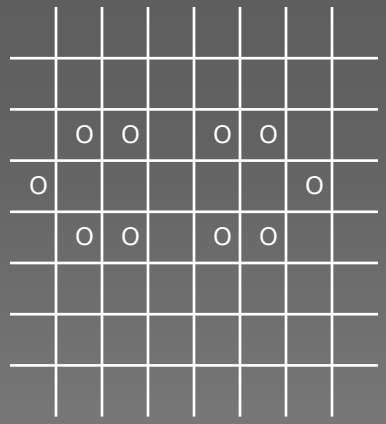
2



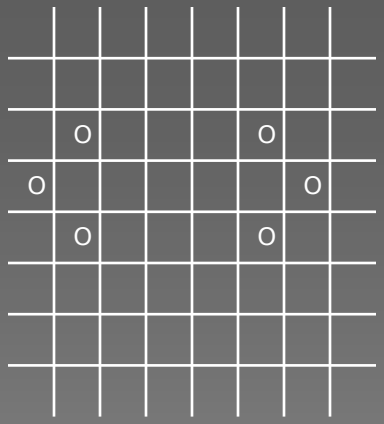
3



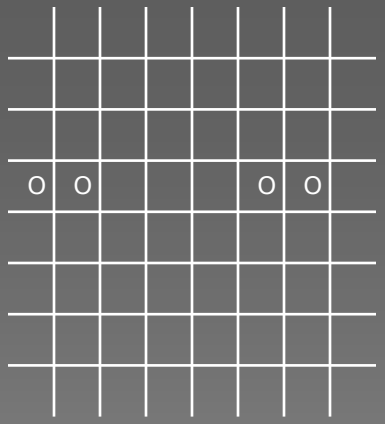
4



5



6



La solución

Inicializar un arreglo llamado MAPA, que contenga la configuración inicial de las celdillas vivas.

Determinar cuantas generaciones abarcará el juego.

Repetir los siguientes pasos para el número de generaciones deseado:

Para cada celdilla del arreglo haga esto:

Cuente el número de vecinos vivos de la celdilla. Si el recuento es 0, 1, 4, 5, 6, 7 o 8, ponga la celdilla correspondiente en otro arreglo llamado NUEVOMAPA para que esté muerta; si el conteo es 3, se pone la celdilla correspondiente para que este viva; y si el conteo es 2, se pone la celdilla correspondiente para que sea la misma que la del arreglo MAPA.

Copiar el arreglo NUEVOMAPA en el arreglo MAPA.

Imprimir el arreglo MAPA para el usuario.

La solución

```
Programa VIDA;
Filas = 50; Columnas = 60;
Mapa, nuevomapa = grilla( filas, columnas)

InicializoVariables;
Generacion = 0;
EscribeMapa;
DESDE generacion = 1 HASTA ultimageneracion
        DESDE i = 1 HASTA filas
                DESDE j = 1 HASTA columnas
                        vecinos = CuentaVecinos( i, j )
                        SI vecinos IGUAL A
                                0, 1 : NUEVOMAPA( i, j ) = muerta
                                2 :    NUEVOMAPA( i, j ) = mapa( i, j )
                                3 :    NUEVOMAPA( i, j ) = viva
                                4, 5, 6, 7, 8 : NUEVOMAPA( i, j ) = muerta
        mapa = nuevomapa
        MuestraMapa
FIN
```

Principios de programación

- Nombres: importancia del nombre de variable y del nombre de los subprogramas.

Regla de programación

Siempre se impone a las variables y subprogramas o funciones con mucho cuidado y se explican de manera detallada.

1. Se deben elegir nombres significativos y que sugieran claramente la finalidad del subprograma.
2. Las variables que se usan poco deben ser nombres sencillos, una sola letra puede ser una elección idónea.
3. Servirse de prefijos o sufijos comunes para asociar nombres pertenecientes a la misma categoría general. Ej. Los archivos utilizados, podrían llamarse:

   ArchivoEntrada ArchivoTransaccion ArchivoTotal
   ArchivoSalida ArchivoRechazo

Principios de programación

4. Evitar errores de ortografía deliberados y los sufijos carentes de significado cuya única finalidad es acuñar nombres diferentes.

Ej. índice indice indc indicee indice2 indice3

5. Evitar nombres elegantes cuyo significado tiene poca o nula relación con el problema.

Ej. **while** TV **in** empeño **do** estudiar;

if not tengosueño **then** jugar **else** siesta;

6. No escoger nombres que tengan una grafia semejante o que por otros motivos se confundirían fácilmente.
7. No se recomienda usar las letras l, 0, O en variables sólas.

Ej. l := 1; x :=1; x := l; x := O;

Principios de programación

- Documentación y formato: Es conveniente tanto para programas pequeños, y mucho más para programas extensos, hacerse el hábito de documentar el programa. Es más una de las tareas más interesantes es la de documentar antes de iniciar el programa.

Regla de programación

El tiempo de lectura de los programas es mucho más largo que el de escritura. Haga que la lectura sea fácil.

1. Se pone un prólogo al inicio de cada subprograma.
2. Cuando cada variable se declare se le pone una explicación, o mejor, el nombre de la variable es evidente.
3. Se introduce cada sección importante del programa con un comentario breve que explica su finalidad.
4. Se indica el final de esta sección, en caso de no ser obvio.
5. No se hacen comentarios que repitan lo que hace el programa.

   Ej. `cont := cont + 1;  { aumentar el contador en 1 }`
6. Mantener la documentación actualizada al modificar un programa.

Principios de programación

- Refinamiento y modularidad: No son las computadoras sino las personas quienes resuelven los problemas. La clave para resolver el problema central es dividir este problema en varios problemas más pequeños.

Regla de programación

No dejar que los árboles impidan ver el bosque.

Esto se denomina **refinamiento descendente**, y es la clave para que programas extensos funcionen.

Cada subprograma deberá ejecutar sólo una tarea para hacerlo bien.

Cada subprograma debe poder ser descripto de manera sencilla.

Cada subprograma deberá ocultar algo.

Los subprogramas tienen sus propias tareas a realizar y sólo deben informar lo que sea necesario a los programas que lo llaman.

Ver ejercicios.

Principios de programación

- Codificación, prueba y refinamiento ulterior: Si bien este título va junto podemos analizarlo por separado, para interpretar todo su significado.

La codificación es el proceso de escribir un algoritmo en la sintaxis correcta al lenguaje seleccionado.

La prueba es el proceso consistente en correr el programa en los datos muestra seleccionados para encontrar los errores, si es que los hay.

Para realizar un refinamiento ulterior recurrimos a los subprogramas todavía no escritos y repetimos estos pasos.

Para refinar se puede usar el método de los CABOS.

EJ. Proceso Inicializar()

Proceso EscribeMapa()

Funcion CuentaVecino(fila, columna):valor

Ejemplo: Función CuentaVecino()

```
Funcion CuentaVecino( i, j )
SI i = 1 ENTONCES xInferior = 1 SINO xInferior = i – 1
SI i = Filas ENTONCES xSuperior = Filas SINO xSuperior = i + 1
SI j = 1 ENTONCES yInferior = 1 SINO yInferior = j – 1
SI j = Columnas ENTONCES ySuperior = Columnas SINO ySuperior = j + 1
Contar = 0
DESDE x = xInferior HASTA xSuperior
        DESDE y = yInferior HASTA ySuperior
            SI mapa( x, y ) = viva ENTONCES contar = contar + 1
SI mapa( i, j ) = viva ENTONCES contar = contar - 1

CuentaVecino = contar
```

Principios de prueba de Programación

```
Funcion CuentaVecino( i, j )
SI i = 1 ENTONCES xInferior = 1 SINO xInferior = i – 1
SI i = Filas ENTONCES xSuperior = Filas SINO xSuperior = i + 1
SI j = 1 ENTONCES yInferior = 1 SINO yInferior = j – 1
SI j = Columnas ENTONCES ySuperior = Columnas SINO ySuperior = j + 1
Contar = 0
DESDE x = xInferior HASTA xSuperior
        DESDE y = yInferior HASTA ySuperior
            SI mapa( x, y ) = viva ENTONCES contar = contar + 1
SI mapa( i, j ) = viva ENTONCES contar = contar - 1

CuentaVecino = contar
```

Principios de prueba de programación

Regla de programación

La calidad de los datos de prueba es más importante que la cantidad.

1. Método de caja negra.
   El usuario trata al programa como una caja negra, le importa entregar valores de entrada y ver su resultado.
   Los criterios mínimos que nos guiarán al escoger los datos de prueba son:
   Valores fáciles, valores típicos realistas, valores extremos, valores ilegales.
2. Método de caja de cristal.
   El usuario desea probar todas las partes del sistema funcionando, para ello deberá generar todos los valores de entrada posible para que todas las partes del programa se prueben.
   En el caso de cuenta vecino, se deberán probar valores de conteo que den 0, 1, 2, 3, 4, 5, 6, 7, 8
3. Método de caja de Pandora
   Nos abstenemos de hacer pruebas y dejamos esa tarea para el cliente.

Algunas otras reglas de programación

Casi todos los programadores pasan 90% del tiempo realizando el 10% de las instrucciones. Encuentre ese 10% y concéntrese en mejorar allí la eficiencia.

Procure que los algoritmos sean lo más simples posibles.

Analice las exigencias de tiempo y espacio al seleccionar un algoritmo.

Nunca tema volver a empezar desde el principio. Es posible que el segundo intento sea más breve y fácil.

Cerciórese de que entiende completamente el problema. Si se debe cambiar los términos del mismo explique exactamente lo que usted ha hecho.

Comenzar de nuevo suele ser más fácil que parchar un programa viejo.