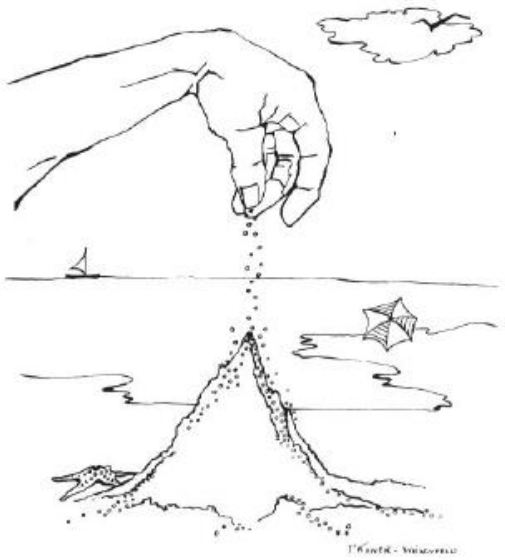
Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Informe Tarea #1:

Pilas de Arena



Fecha: 07 de abril de 2017
Autor: Daniel Araya Poblete
Email: Daniel.arayap1@gmail.com
Curso: CC3001-2
Profesor: Nelson Baloian

Introducción

El fenómeno de la caída de granos de arena en un mismo punto del espacio lleva a preguntarse si existe un patrón de distribución de los granos en el espacio vecino, una vez que deja de ser viable que se depositen en el mismo punto, ya que la gravedad no permite que se depositen uno sobre el otro y caen hacia otros lugares circundantes en una superficie.

Este problema físico aparenta ser complicado de entender usando herramientas de cálculo convencionales que modelen su comportamiento. Es en estos casos cuando se hace conveniente recurrir a herramientas computacionales que, bajo supuestos lo suficientemente aproximados a la realidad, permitirían llegar a un indicio de cómo ocurren estos fenómenos en la realidad.

El problema a resolver en esta tarea corresponde a la distribución espacial 2D de una cantidad de granos de arena apilados en el centro de una matriz potencialmente infinita. El objetivo es poder observar el resultado una vez que todos los granos en cada punto del espacio fueron estabilizados correctamente. Para poder modelar este problema con un programa, el requisito dado es aplicar una regla de estabilidad tal que en cada punto del espacio (i.e en cada celda de la matriz) no se puedan apilar más de 3 granos de arena.

La solución a este problema se realizó mediante un programa que pide un valor entero al usuario (que indica la cantidad de granos de arena), tal que en un estado inicial se ubican todos los granos agrupados en el centro de la matriz. Mediante iteraciones aplicadas en cada punto de la matriz, se evalúa la cantidad de granos que se encuentra en cada punto del espacio y se estabiliza en caso de exceder la cantidad máxima de 3 granos que es el requisito. Es importante definir también qué sucede para valores más grandes entregados por el usuario, y cómo se comporta el programa en tal caso.

En las siguientes páginas de este informe se mostrará con mayor detalle las especificaciones del problema, así como los supuestos que se asumen para resolverlo y los algoritmos utilizados para su solución.

Análisis del Problema

El problema que se presenta, pensándolo en términos prácticos, se trata de estabilizar valores en una matriz de tal manera que, al ejecutar el programa, en ninguna de las celdas se encuentre un valor mayor a 3. De este modo se dirá que la matriz de granos se encuentra "estable" sólo cuando esto ocurra en todas las celdas.

Para inicializar el programa se le pedirá al usuario un valor que, como ya se dijo, indique la cantidad de granos que se pretenden asentar en la matriz; como este valor se pretende estabilizar desde el centro hacia los alrededores del espacio, se debe inicializar una matriz de valores nulo exceptuando la celda central, que es donde se ubicará inicialmente la cantidad de granos dada por el usuario (ver Figura 1).

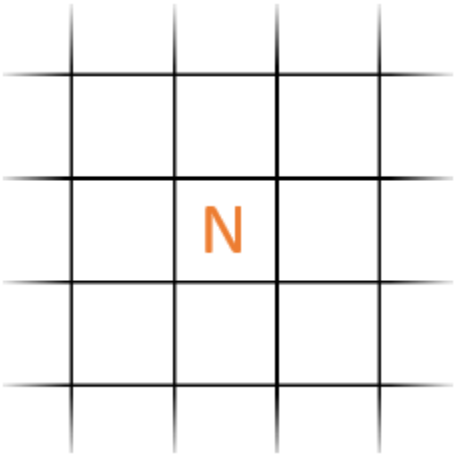


Figura 1. Matriz inicializada con solo un valor N inicial, dado por el usuario, y cero en las demás celdas. Dependiendo del valor de N el tamaño de la matriz será mayor, pero eso se discutirá en la solución de problema.

Ahora para estabilizar esta matriz, el supuesto que se asume inicialmente es que cuando una celda tiene más de tres granos (4 o más), ésta distribuye un grano hacia cada uno de sus vecinos, es decir, cada una de las celdas contiguas a ésta, vertical u horizontalmente aumenta en un grano. Otro supuesto que se tendrá en cuenta es que el orden en que los granos se distribuyan en las celdas vecinas no será relevante para ejecutar el algoritmo, es decir, no hace mayor diferencia si la distribución se hace iniciando por la celda que está inmediatamente sobre la central, o si se inicia en la celda de la derecha o la de abajo, etc. Puede observar en el ejemplo del enunciado lo explicado en este segmento (ver Figura 2).

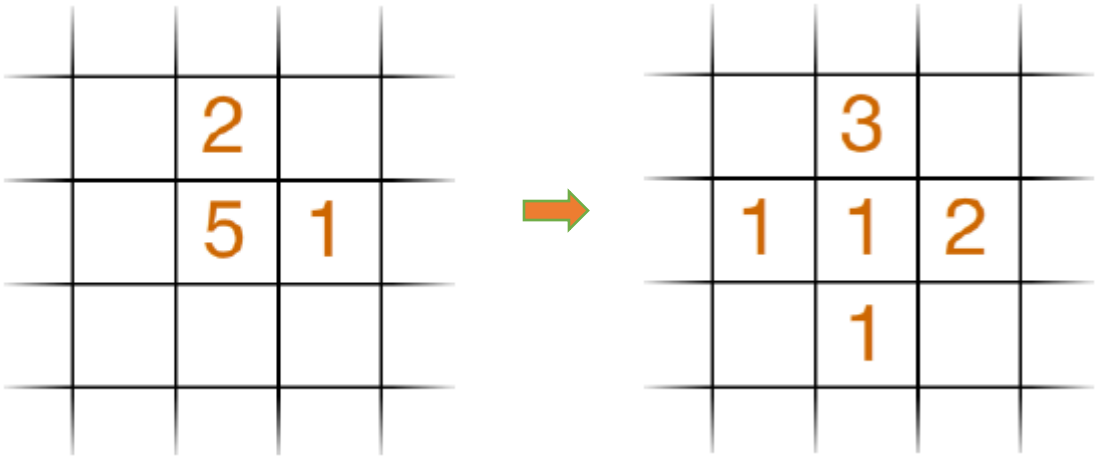


Figura 2. Puede observar que en la matriz de la izquierda el valor central es mayor a 3 granos, por lo que luego de aplicar una regla de equilibrio, a éste valor se le restan 4 granos que se añadirán en las celdas de arriba, abajo, derecha e izquierda de la celda. Queda como resultado la matriz de la derecha estabilizada.

Para la visualización del problema en una ventana, se tiene inicialmente una clase Ventana con un constructor que permite crear una ventana con las mismas dimensiones de largo y ancho, y con la posibilidad de tener un título adecuado (si no se le entrega un título en el input, la ventana tendrá el título "Ventana"); además la clase incluye un método mostrarMatriz, que recibe una matriz de enteros y la dibuja en la ventana vacía creada con el constructor. El método mostrarMatriz entrega un mosaico con los colores negro, amarillo y rojo dependiendo del valor de la celda.

Para comprender de mejor manera el problema se necesita analizar los casos de borde de éste, porque si bien teóricamente la matriz para modelar el problema es infinita, en la implementación que se espera para este programa esto no puede hacerse. En esta tarea se ofrecieron algunas soluciones posibles para el caso de exceder los índices de la matriz durante la implementación del programa. En la tarea presentada en este informe, la solución que se ofreció fue encontrar un valor para el tamaño de la matriz que depende del valor N que se entregue inicialmente, tal que los valores de la matriz nunca alcanzarán esa cota superior de tamaño. Se detallará este paso más profundamente cuando se aborde la solución del problema.

Solución del Problema

Se iniciará esta sección hablando inicialmente del problema de los casos de borde, es decir, los casos en que la cantidad de granos desbordaría la matriz, y por ende ésta no se podría seguir estabilizando ya que alcanzó su tamaño máximo. Para este problema se sugirieron algunas soluciones en el enunciado:

- Encontrar una cota superior para el tamaño de la matriz dado el valor de N;
- Suponer que el borde es un sumidero, cuyas celdas tendrán valor 0 sin importar cuántos granos le lleguen;
- Agrandar la matriz a medida que se requiera más espacio, o copiar directamente la matriz en una de mayor tamaño donde se pueda seguir iterando.

Para esta tarea se escogió la primera opción, de esa manera no se presentarán problema con dimensiones bajo ningún concepto al correr del programa, para ningún valor dado, por grande que éste sea. Para esto se pensó de la siguiente forma: para un valor N, si en cada celda cupiera un solo grano, la dimensiones que debiera tener la matriz es $\sqrt{N}$ x $\sqrt{N}$. Pero en cada celda el valor máximo que puede tener es 3, por lo que $\sqrt{N}$ es una cota superior que puede tener la matriz para evitar llegar al borde de ésta. Con esto, se puede pensar en un tamaño mínimo que debe tener la matriz para que quepan todos los granos de arena. En el programa se usaron estas dimensiones, sumando uno para asegurar que la matriz sea un poco mayor en caso de llegar a su borde. Con estas dimensiones la matriz nunca llega a su límite y se cumplen las condiciones de borde.

Para establecer la estabilidad de los granos, es necesario que ninguna celda sea mayor que 3, y si alguna lo es, ésta distribuye un grano a cada una de sus celdas vecinas y se le quitan 4 granos a la celda “colapsada”. Esto se logra con una condición if (Figura 3):

```
if (m[k][j] >= 4) {
    m[k][j] -= 4;
    m[k][j - 1] += 1;
    m[k - 1][j] += 1;
    m[k + 1][j] += 1;
    m[k][j + 1] += 1;
}
```

Figura 3. Condición utilizada para llegar al equilibrio del sistema.

Éste código evalúa una celda de ubicación (k,j) de la matriz, y si su valor es mayor o igual a 4 ejecuta el algoritmo para “descargar” la celda.

Ahora, es necesario que esta condición se evalúe en cada una de las celdas de la matriz, por lo que fue necesario colocar esta condición dentro de dos ciclos: uno que recorre el primer índice desde 0 hasta $\sqrt{N} + 1$ y otro que recorre el segundo índice de la misma forma (Figura 4):

```
int j=0;
while (j < a) {
    int k=0;
    while (k < a) {
        if (m[k][j] >= 4) {
            m[k][j] -= 4;
            m[k][j - 1] += 1;
            m[k - 1][j] += 1;
            m[k + 1][j] += 1;
            m[k][j + 1] += 1;
        }
     k++;
    }
 j++;
}
```

Figura 4. Ciclo que recorre la matriz m, desde m(0, 0) hasta m($\sqrt{N}+1$, $\sqrt{N}+1$) evaluando la condición de estabilidad para los granos.

De este modo la condición de estabilidad se aplica para cada celda de la matriz. El nuevo problema que surge es que la condición se evaluará únicamente al llegar al centro de la matriz, por lo que muchas celdas en las que al evaluar había ceros, al realizar el balance en la celda central tomarán valores mayores que 3 y no alcanzarán a balancearse. Entonces, si ambos ciclos recorren la matriz una sola vez, ésta no podría balancearse completamente, por lo que fue necesario crear un nuevo ciclo que repita el proceso de barrido de espacio varias veces (ver Figura 5):

```
for(int i=0;i<n;i++) {
    int j=0;
    while (j < a) {
        int k=0;
        while (k < a) {
            if (m[k][j] >= 4) {
                m[k][j] -= 4;
                m[k][j - 1] += 1;
                m[k - 1][j] += 1;
                m[k + 1][j] += 1;
                m[k][j + 1] += 1;
            }
         k++;
        }
     j++;
    }
}
```

Figura 5. Se creó un ciclo que repite los ciclos que recorren los índices de la matriz n veces; con estos ciclos los granos quedan completamente balanceados en la superficie dada.

Modo de Uso y Resultados

Inicialmente el programa le pedirá al lector un número entero, cuando éste introduzca la cantidad, se muestra una ventana con la matriz mostrada en colores. Se realizaron algunas pruebas aplicando un input con distintos valores de N y evaluando el resultado entregado en la ventana "Pilas de Arena" (Figuras 6 a 10):

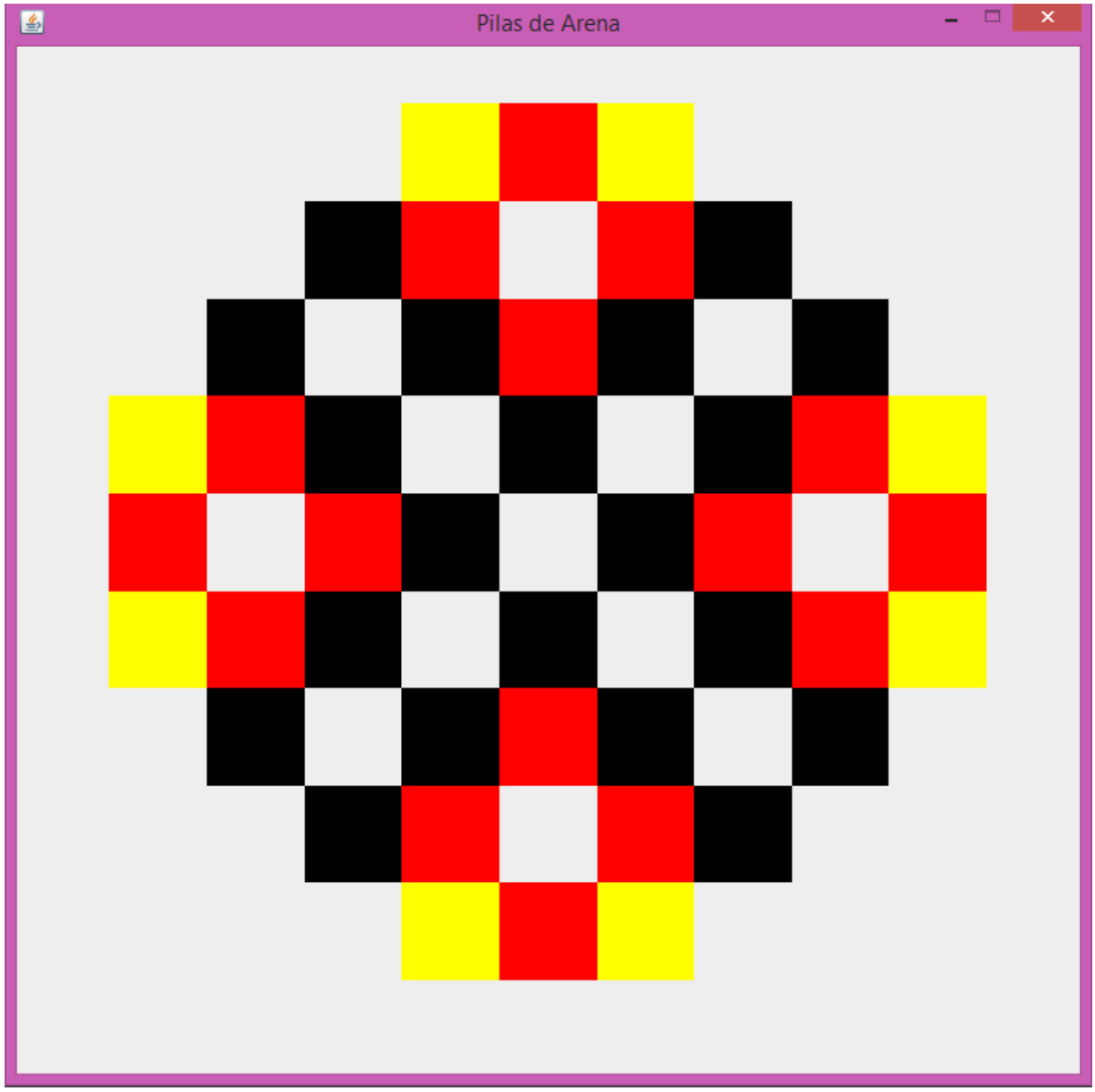


Figura 6. Resultado de aplicar el programa con un valor N=100.

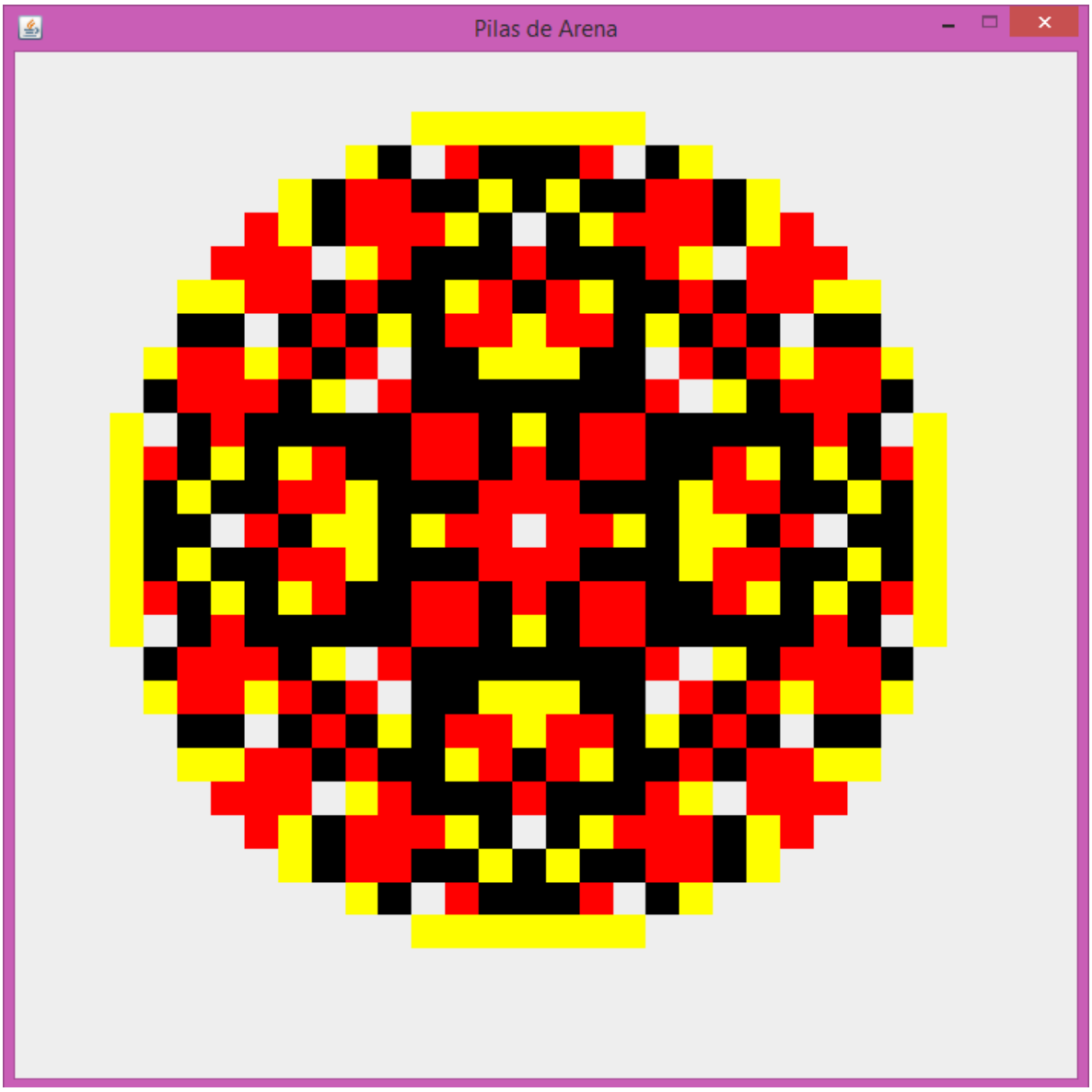


Figura 7. Resultado de aplicar el programa con un valor N=1000.

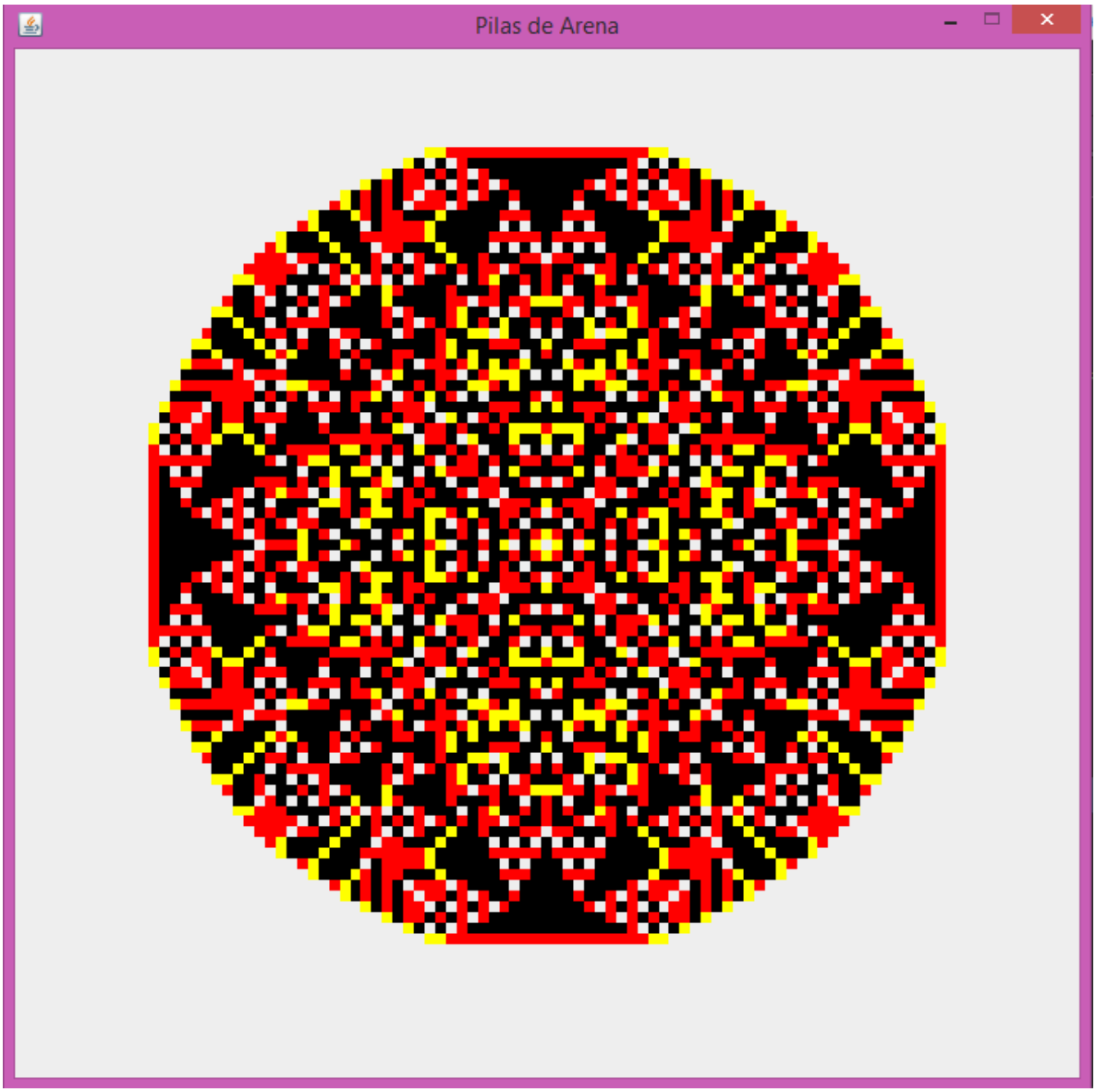


Figura 8. Resultado de aplicar el programa con un valor N=10000.

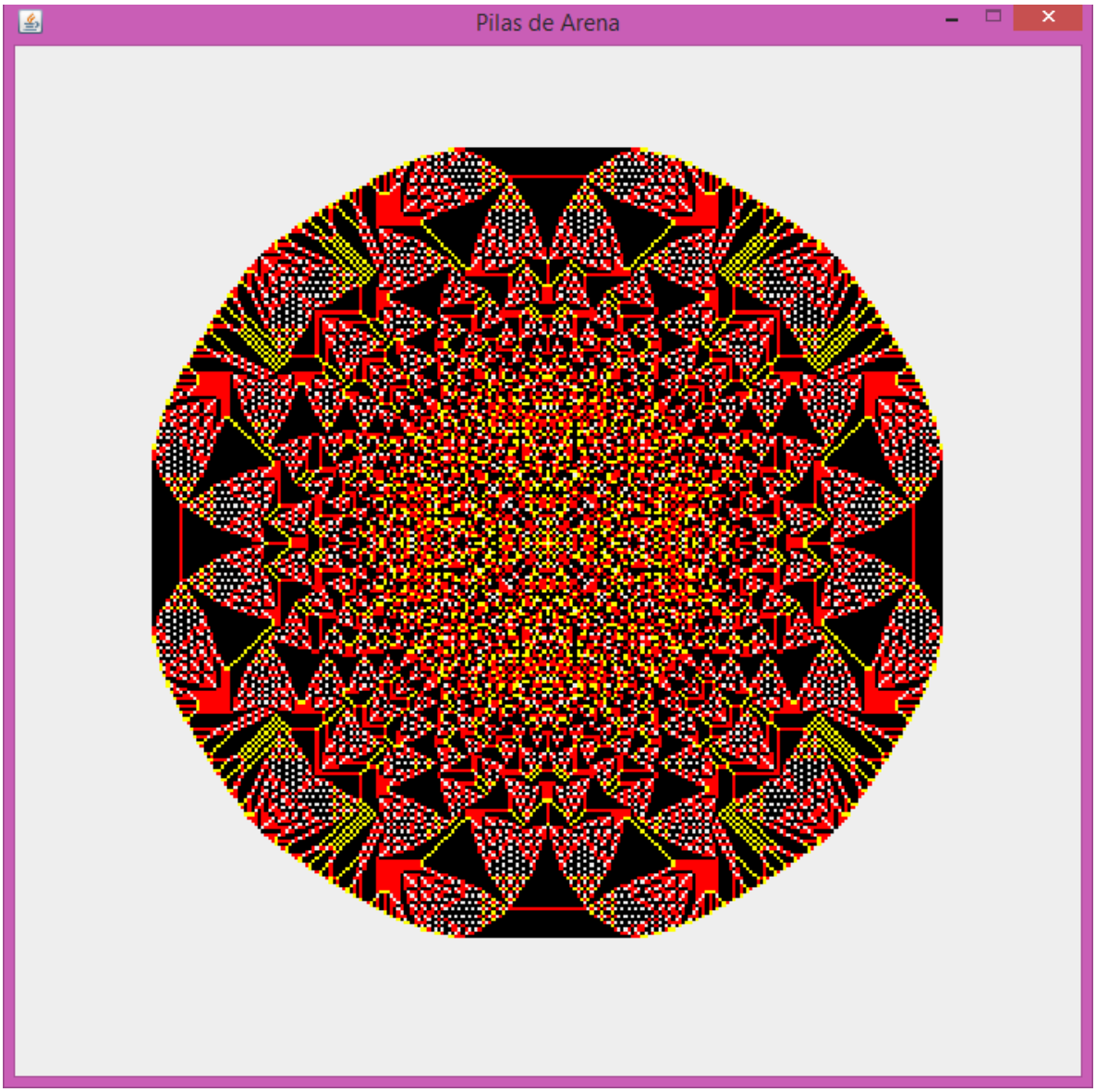


Figura 9. Resultado de aplicar el programa con un valor N=100000.

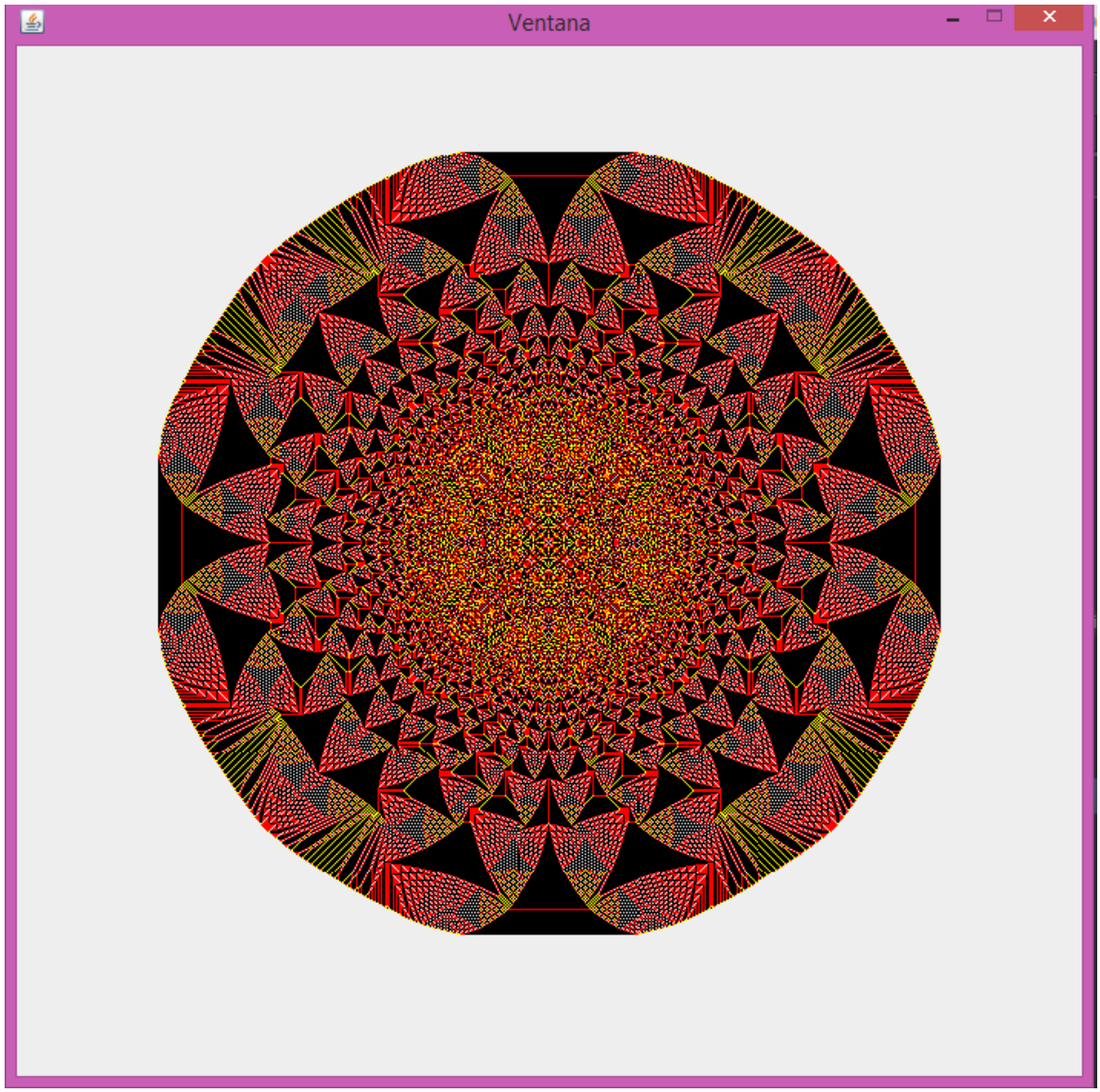


Figura 10. Resultado de aplicar el programa con un valor N=500000.

Discusión y conclusión

Se puede observar que a medida que aumenta el valor de N aumenta la cantidad de celdas en la matriz, ya que la función del programa es crear una matriz de tamaño $\sqrt{N}$+1 x $\sqrt{N}$+1, donde el número de celdas dependerá del N inicial. De este modo al aumentar el valor de N, aumenta la cantidad de celdas por lo que se tiene un mejor detalle de la distribución de los granos (cabe destacar que cada color indica un número de granos para cada celda, donde un valor de 0 granos corresponde al color blanco, 1 = amarillo, 2 = rojo y finalmente 3 = negro).

El comando if en este programa es necesario para testear la condición de estabilidad del sistema, aunque al insertar esto en un ciclo while, para un número grande (ejemplo N=1000000) el programa demorará un tiempo mayor en aplicar la condición en un mismo punto. Aplicar la condición dentro de este ciclo, repetirá la condición $a = \sqrt{N}$+1 veces, y estas a repeticiones se realizan a veces debido al segundo while, por lo que en total el algoritmo que recorre todos los índices de la matriz es de orden N.

Ahora, para verificar que la condición de estabilidad se cumpla en toda la matriz, este recorrido por la matriz debe realizarse varias veces, por lo que fue necesario insertar un nuevo ciclo que recorre la matriz N veces por todas las celdas. En total, el programa completo toma un tiempo de orden N^2 en realizar los recorridos, por lo que este algoritmo resulta ser bastante ineficiente en términos de tiempo.

Para mejorar el programa (principalmente su eficiencia) se pensó en crear un método aparte del programa main() que implementará el algoritmo de estabilidad de forma recursiva, y así insertar un solo ciclo for que repita el procedimiento en el programa. De este modo, con la recursividad en un método estático se minimiza la cantidad N^2 de repeticiones que realiza este programa en su implementación. Otra mejora pudo haber sido iniciar el proceso del recorrido de la matriz desde el centro, que es donde inicialmente se encuentra el valor N y así ahorrarse la evaluación en las celdas "esquinas" de la superficie, ya que éstas son las últimas en rellenarse.

Finalmente se puede concluir que, si bien la intuición inicial indica realizar un procedimiento o aplicar el programa de cierta manera, siempre es conveniente plantear una solución alternativa que podría funcionar de mejor manera o bien revisar el código en busca de implementaciones que podrían mejorar la aplicación de éste, tanto en eficacia como en eficiencia. Algunos algoritmos recursivos hubiesen mejorado de manera significativa el código inicial haciéndolo más veloz en su realización, y no se tomaron en cuenta debido a que se prefirió la idea original sin ahondar en las alternativas.

Anexo: Código fuente

```
package sample;
import java.util.Scanner;
public class PilaArena {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n;
        System.out.println("Introduzca un número entero: ");
        n = sc.nextInt();
        int a = (int) Math.sqrt(n) + 1;//Se define un tamaño minimo que
debe tener la matriz
        int m[][] = new int[a][a];
        int pm = (a - 1) / 2;
        m[pm][pm] = n;//se inicializa la matriz con el dato n en la celda
central
        for(int i=0;i<n;i++) {//ciclo que repite el recorrido n veces
            int j=0;
            while (j < a) {//ciclos de recorrido de la matriz
                int k=0;
                while (k < a) {
                    if (m[k][j] >= 4) {//condición de estabilidad
                        m[k][j] -= 4;
                        m[k][j - 1] += 1;
                        m[k - 1][j] += 1;
                        m[k + 1][j] += 1;
                        m[k][j + 1] += 1;
                    }
                 k++;
                }
             j++;
            }
        }
        Ventana v = new Ventana(700, "Pilas de Arena");
        v.mostrarMatriz(m);
    }
}
```