Algoritmos y Programación I
(75.40/95.14)

Trabajo práctico N°2
`Cyberar-t`

Primer cuatrimestre 2017

Introducción

La mundialmente conocida artista Martha Mijuanin (no confundir con Marta Minujin) esta preparando su nueva exposición. Carente de ideas para nuevas obras, vagó por los bares de San Telmo en busca de inspiración. Una noche, sus pasos la llevaron hasta las puertas de la facultad de ingeniería y ahí tuvo una revelación: "Arrrrte digital, eso es lo que esta en la onda".
Sin perder tiempo se comunico con nosotros en busca de un grupo de ávidos programadores que puedan emprender esta particular tarea.

Consigna

Se pide implementar un generador de fractales[1] circulares, mediante el método de *sandpiles*[2].

Implementación

El programa permite al usuario generar las imágenes. Las mismas se generan en base a las siguientes especificaciones que ingrese el usuario:

- Parámetros obligatorios: El usuario debe ingresar sí o sí estos datos
  - Tamaño del fractal
  - Coordenadas de los montículos y cantidad de arena de cada uno (Nota: Se debe pedir al menos un montículo)
  - Nombre del archivo de salida
- Parámetros opcionales: El usuario puede elegir si ingresar los parámetros o usar una configuración por defecto.
  - Tamaño de celda vertical / Tamaño de celda horizontal: Cantidad de pixels que ocupa cada celda en la imagen generada.
  - Espejado vertical / Espejado horizontal: La cantidad de veces que se debe espejar el fractal vertical y horizontalmente.
  - Colores: El usuario puede elegir si usar los colores por defecto o definir el los colores a usar para representar cada valor posible de la matriz. Se debe implementar de manera que resulte intuitiva para cualquier usuario; por ejemplo, elegir de una lista de colores precargados.

Al generar la obra se debe generar un archivo en formato `.ppm`. De esta manera podremos visualizarlo con varios visores de imágenes como el Image Viewer[3]

[1] https://es.wikipedia.org/wiki/Fractal
[2] https://www.youtube.com/watch?v=1MtEUErz7Gg
[3] El visor de imágenes por defecto de Ubuntu. https://wiki.gnome.org/Apps/EyeOfGnome

Formato PPM [4] [5]

```
P3
<alto> <ancho>
<valor máximo de color>
<rojo> <verde> <azul>
[...]
<rojo> <verde> <azul>
```

Ejemplo de un cuadrado rojo de 2x2 pixels:

```
P3
2 2
255
255 0 0
255 0 0
255 0 0
255 0 0
```

[4] http://netpbm.sourceforge.net/doc/ppm.html
[5] https://en.wikipedia.org/wiki/Netpbm_format

Sandpiles

Los *sandpiles* representan montículos de arena sobre una grilla. Cada montículo puede contener a lo sumo 3 granos de arena; si posee mas, se dice que el montículo se rompe y la arena cae hacia las celdas adyacentes. Las celdas adyacentes son sólo 4: arriba, abajo, izquierda y derecha (las diagonales no son consideradas adyacentes).

Al romperse un montículo, la arena se reparte equitativamente entre las 4 celdas adyacentes, y el excedente de la repartición queda en la celda original. Ejemplo: se tiene un montículo con 5 granos en la celda (1,1) de una grilla de 3x3. Al romperse el montículo, se envía un grano a las celdas (0,1), (2,1), (1,0) y (1,2). El 5to grano, como no se puede repartir equitativamente, queda en la celda (1,1).

Como caso particular, si la celda se encuentra sobre uno de los bordes, los granos que van a las celdas inexistentes se dice que se çaen"de la grilla. Ejemplo: se tiene un montículo con 5 granos en la celda (0,0) de una grilla de 3x3. Al romperse el montículo, se envía un grano a las celdas (-1,0), (1,1), (0,-1) y (0,1). Como las celdas (-1,0) y (0,-1) no existen en la grilla, los granos de arena desaparecen, quedando sólo 3 granos sobre la grilla.

En el TP se debe simular este comportamiento hasta que la grilla quede estable, es decir, no quede **ningún** montículo de 4 o mas granos. Ejemplo:

Se tiene un montículo con 16 granos en la celda (1,1) de una grilla de 3x3. Al romperse el montículo, se envían 4 granos a las celdas (0,1), (2,1), (1,0) y (1,2). Como cada una de estas celdas queda ahora con montículos de 4 granos, éstos deben volver a romperse.

Todos los montículos se deben partir en simultaneo. Esto quiere decir que, si bien el algoritmo se ejecuta de manera secuencial (evaluando de a un montículo por vez), el "partir"de cada montículo no debe afectar a los montículos adyacentes. Ejemplo: Se tienen dos montículos de 16 granos en las únicas 2 celdas de una grilla de 1x2. En la primera iteración, ambos montículos se rompen, enviando 4 a la celda adyacente y tirando el resto. De esta forma, la siguiente iteración verá que en la grilla hay 2 montículos de 4 granos cada uno (y no uno vacío y otro con 20).

Ejemplos

El algoritmo a implementar es intrínsecamente costoso en tiempo. Se sugiere probar el código primero con ejemplos pequeños inherentemente de las optimizaciones que cada uno implemente. A continuación se muestran algunos ejemplos que se pueden usar para probar.

Ejemplo N°1

El primer ejemplo consiste en crear un fractal en el centro de la imagen con los siguientes parámetros de entrada:

- Tamaño del fractal: `64x64`
- Tamaño de celda vertical / Tamaño de celda horizontal: `10x10` pixels
- Espejado vertical / Espejado horizontal: `1x1`
- Colores: `"0 0 0", "255 0 255", "255 0 0", "255 255 0"`
- Coordenadas de los montículos y cantidad de arena de cada uno: `32,32,10000`

Figura 1: Salida del ejemplo 1 con un tamaño de 640x640 pixels

El fractal nos muestra una simetría desde el centro de la imagen. Esto se da porque el único sandpile que agregamos fue en la posición 32,32 con un tamaño de fractal de 64x64. El tamaño de la imagen es de 640x640 porque aumentamos 10 veces el tamaño de cada pixel. Los colores que vemos son el negro (0 0 0), el magenta (255 0 255) y el amarillo (255 255 0).

Ejemplo N°2

El segundo ejemplo consiste en crear un fractal doblemente espejado con los siguientes parámetros de entrada:

- Tamaño del fractal: `35x40`
- Tamaño de celda vertical / Tamaño de celda horizontal: `3x3` pixels
- Espejado vertical / Espejado horizontal: `2x2`
- Colores: `"0 0 0", "255 0 255", "255 0 0", "255 255 0"`
- Coordenadas de los montículos y cantidad de arena de cada uno: `10,10,1800 ; 0,0,56`

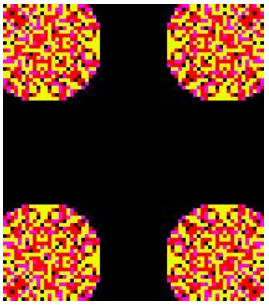

Figura 2: Salida del ejemplo 1 con un tamaño de 210x240 pixels

Como puede observarse, está la misma figura replicada 4 veces pero en una orientación distinta. Esto se debe a que le pedimos que se espeje 2 veces en forma vertical y dos veces en forma horizontal. Por lo que de nuestra imagen original (1er cuadrante) nacen las otras 3.
La figura principal tiene un sandpile en la posición 10,10 de intensidad 1800 y uno en la posición 0,0 de intensidad 56. Puede verse como la simetría principal se da desde el casi el centro del cuadrante (posición 10,10) ya que es la que más intensidad tiene y la que más arena "derramó" hacia sus celdas vecinas. Si se mira con atención en las 4 esquinas puede verse una diferencia respecto del fractal principal. Este corresponde al sandpile en el punto 0,0 que, como su intensidad es solamente 56, se ve con menor detalle.
También se ve que el tamaño final de la imagen (210x240 pixels) está directamente relacionado con los parámetros para la creación. Si nuestra imagen con tamaño de celda=1 y con espejado=1 es de 35x40, luego de aumentarlo en 3 (105x120) y espejarlo tanto horizontal como verticalmente, lo multiplicamos por dos y llegamos a un tamaño final de 210x240 pixels.

Criterios de aprobación

A continuación se describen los criterios y lineamientos que deben respetarse en el desarrollo del trabajo.

Informe

El informe debe consistir en una breve descripción del **diseño** del programa.

Debe recordarse que la etapa de diseño es *anterior a la implementación*, por lo tanto debe describirse, utilizando texto y/o diagramas, cómo se va a estructurar el código para cumplir con las especificaciones de la consigna.

Algunas preguntas que deberían responderse:

- ¿Cómo se va a guardar en memoria los diferentes datos o estados del programa?
- ¿Como se resolvió el problema de generar el fractal en base a los sandpiles?
- ¿Cuales eran las alternativas que se le presentaron y por que eligió esta sobre las otras?

Código

Además de satisfacer las especificaciones de la consigna, el código entregado debe cumplir los siguientes requerimientos:

- El código debe ser claro y legible.
- El código debe estructurarse en funciones y, cuando corresponda, módulos. Las funciones deben definirse de la manera más genérica posible.
- Todas las funciones deben estar adecuadamente documentadas, y donde sea necesario el código debe estar acompañado de comentarios.

Entrega

La entrega del trabajo consiste en:

- El informe y código fuente impresos. Para el código fuente utilizar una tipografía `monoespacio`.
- El informe en formato *PDF*.
- Una versión digital de todos archivos `.py` de código, separados del informe. En el caso de ser más de un archivo, comprimidos en un `.zip`.

El informe impreso debe entregarse en clase. Los dos últimos (PDF y código fuente) deben enviarse a la dirección electrónica `tps.7540rw@gmail.com` con el asunto "TP2 - <*Padrón*>".

Este trabajo práctico se desarrolla en forma **individual**. El plazo de entrega vence el **lunes 8 de mayo de 2017**.