ALGORITMOS Y ESTRUCTURAS DE DATOS

Guía 2. Estructuras, Gráficos y Simulaciones

Gráficos y simulaciones

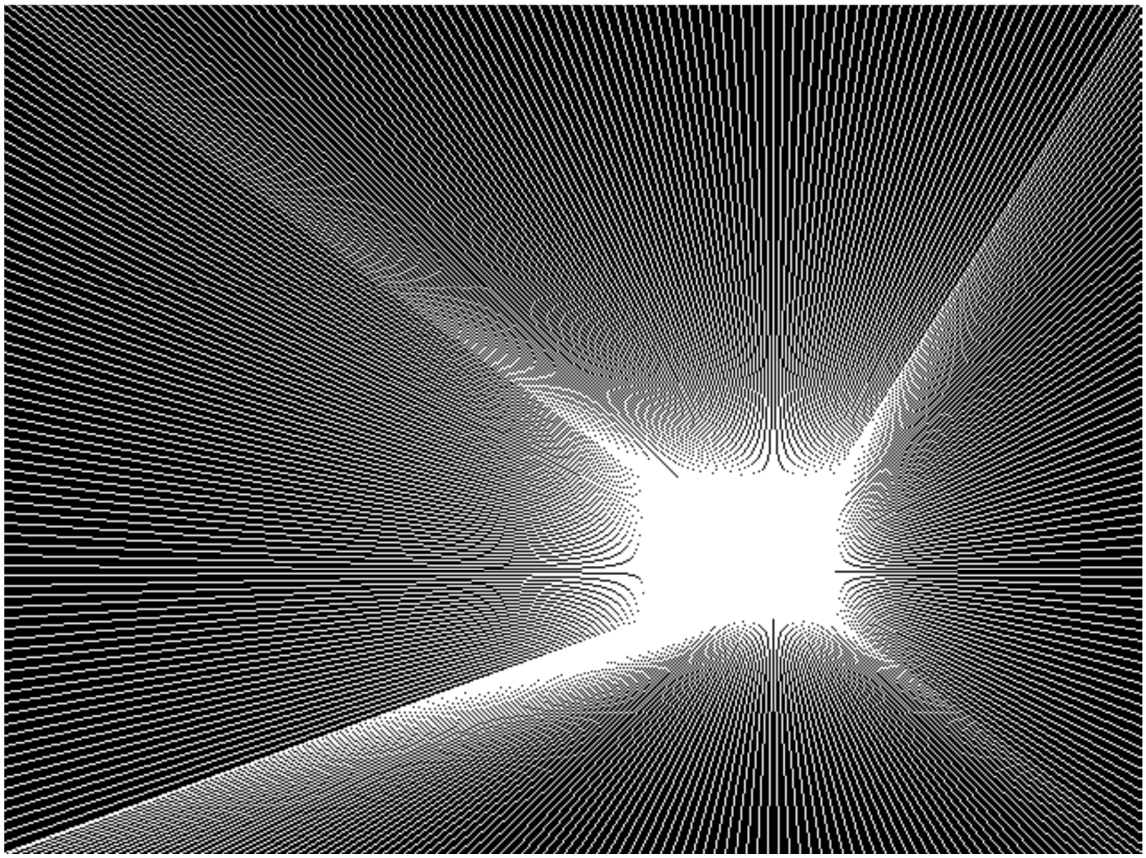
1. Se pide construir un proyecto (la elección del IDE y la plataforma quedan a criterio y gusto del alumno, siendo la recomendación de la cátedra Visual Studio 2017 o superior corriendo en Windows) para la librería gráfica Allegro (http://alleg.sourceforge.net).

Los ejercicios que siguen deben realizarse utilizando la librería gráfica Allegro.

2. Un patrón de Moiré ocurre cuando dos patrones repetitivos pero ligeramente distintos se superponen. También puede ocurrir en gráficas de computadora, por la diferencia entre los pixeles físicos y los puntos matemáticos (los pixeles tienen área finita, los puntos no). Se pide realizar un programa que imprima un patrón de Moiré, cumpliendo con lo siguiente:

a. Debe comenzar con la pantalla en negro.

b. Debe recorrer el perímetro de la pantalla, avanzando de a n pixeles. Debe trazar una línea blanca entre cada punto recorrido y un cierto punto (cx, cy). Los parámetros cx, cy y n se pasan por línea de comando.



3. Se pide simular la física de una pelota de fútbol (de 0.2m de diámetro) que se encuentra en una "habitación" rectangular de x_m = 4m de ancho e y_m = 3m de alto. Se debe cumplir que:

a. La pelota se encuentra inicialmente en (x0, y0), y tiene velocidad inicial (vx0, vy0). Los parámetros x0, y0, vx0 y vy0 se deben pasar por línea de comando.

b. La pelota cumple las leyes de Newton. Por lo tanto:
   v'y(t) = a = g (constante gravitatoria)
   y'(t) = vy(t)
   v'x(t) = 0
   x'(t) = vx(t)

c. Podemos aproximar estas ecuaciones de la siguiente manera:
   (vy(n) - vy(n - 1)) / Δt = g
   vy(n) = vy(n - 1) + aΔt
   (y(n) - y(n - 1)) / Δt = vy(n)
   y(n) = y(n - 1) + vy(n)Δt

$(vx(n) - vx(n - 1)) / \Delta t = 0$
$vx(n) = vx(n - 1)$
$(x(n) - x(n - 1)) / \Delta t = vx(n)$
$x(n) = x(n - 1) + vx(n)\Delta t$

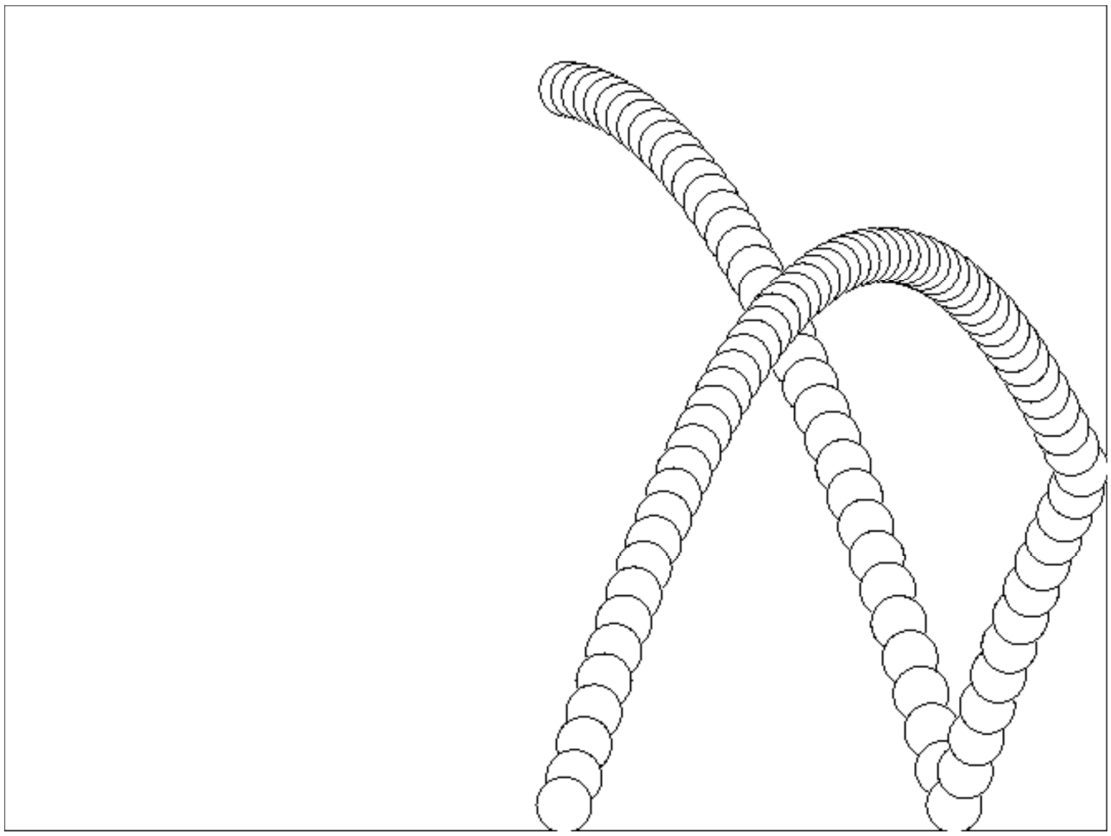
d. Para la simulación calculamos primero vx(n) y vy(n), y luego usamos estos valores para calcular x(n) e y(n). Δt debe valer por defecto 0.1 (segundos), pero ese valor se debe poder cambiar por línea de comando.

e. La habitación se encuentra en un lugar con constante gravitatoria g constante. Por defecto la constante es 9.8 (m/s2), pero es posible cambiarla por línea de comando.

f. Las paredes son parcialmente elásticas: una colisión en una pared hace que el signo de vx(n) o vy(n) se invierta, y su valor se multiplique por una constante de elasticidad. El valor de esta constante es 0.9 por defecto, pero debe poder cambiarse desde la línea de comando.

g. Se pide graficar la simulación en tiempo real. O sea: que un segundo de la simulación corresponda a un segundo real.

h. Lamparita final: se recomienda que las variables sean double, y que se usen las mismas unidades para evitar confusiones. Por ejemplo: metros para distancias y segundos para el tiempo.

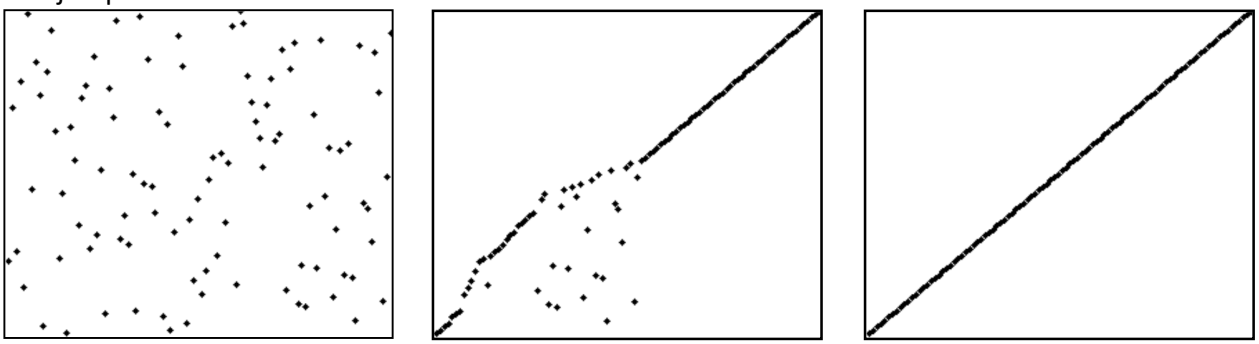


4. Se pide realizar un programa que implemente el algoritmo de ordenamiento bubble sort. El pseudo-código de bubble sort es:

```
bubbleSort(L)
{
  do
  {
    isSwapped = false;
    foreach(i = 0:(length(A) - 2))
    {
      if (L[i] > L[i + 1])
      {
        swap(L[i], L[i + 1])
        isSwapped = true
      }
    }
  } while (isSwapped)
}
```

Para probar el código se deberá generar un arreglo de 256 ints con estos valores: x[0] = 0, x[1]= 1, x[2] = 2, ..., x[255] = 255. Los valores se deberán permutar al azar, para simular que han sido desordenados (averiguar en internet como se pueden permutar valores al azar de manera eficiente). Se debe usar el algoritmo bubble sort para re-ordenar el arreglo. Para cada iteración del algoritmo bubble sort se pide graficar el arreglo, representando índice vs. valor del arreglo en ese índice.

Ejemplo:



(el gráfico del arreglo al comenzar el programa, a medio camino, y al final)

Estructuras y simulaciones

Mini-examen:

1. ¿Qué errores hay en las siguientes declaraciones de C?

```
1. struct Point ( double x, y )
2. struct Point { double x, double y };
3. struct Point { double x; double y }
4. struct Point { double x; double y; };
5. struct Point { double x; double y; }
```

2. ¿Qué errores hay en las siguientes declaraciones de C?

```
A. typedef struct { double x; double y } Point;
B. typedef { double x; double y; } Point;
C. typedef struct { double x; double y; };
D. typedef struct { double x; double y; } Point;
```

3. ¿En qué se diferencian los siguientes programas?

(a)
```
#include <stdio.h>
struct Point { double x; double y; };

int main(void)
{
  struct Point test;
  test.x = .25; test.y = .75;
  printf("[%f %f]\n", test.x, test.y);
  return 0;
}
```

(b)
```
#include <stdio.h>
typedef struct { double x; double y; } Point;

int main(void)
{
   Point test;
   test.x = .25; test.y = .75;
   printf("[%f %f]\n", test.x, test.y);
   return 0;
}
```

(c)
```
#include <stdio.h>
typedef struct { double x; double y; } Point;

int main(void)
{
  Point test = {.25, .75};
  printf("[%f %f]\n", test.x, test.y);
  return 0;   }
```

4. Se pide implementar una función `double getDistanceBetweenPoints(Point p1, Point p2)` que calcule la distancia euclidiana entre dos puntos.

5. Se pide implementar una función `double getAngleBetweenPoints(Point p1, Point p2)` que calcule el ángulo del segmento entre dos puntos. Tomaremos como referencia el cero en el "norte", y ángulos α crecientes en el sentido de las agujas del reloj, $0 \leq \alpha < 360$.

6. Se pide implementar una función `Point translatePoint(Point p, double distance, double angle)` que traslade un punto p por una distancia igual a `distance`, y a un ángulo igual a `angle`. El ángulo está definido igual que en el ejercicio 5.

7. Se pide implementar una función `bool isPointEqual(Point p1, Point p2)` que determina si dos puntos son coincidentes. Cuando se usan números de coma flotante no se puede simplemente verificar la igualdad con el operador igual, sino que se debe verificar que la distancia sea menor que cierto valor, por ejemplo con $\varepsilon = 0.0000001$. ¿Por qué? ¿Se podrá reciclar alguna función anterior?

8. Se pide definir un tipo de datos `Rect` para rectángulos paralelos a los ejes de un sistema de coordenadas cartesiano. El rectángulo debe estar representado por los puntos inferior izquierdo y superior derecho, usando el tipo de datos `Point` definido antes.

9. Se pide implementar una función `double getRectArea(Rect r)` que calcule el área del rectángulo.

10. Se pide implementar la función `bool isPointInRect(Point p, Rect r)` que determina si el punto p está dentro de o en el borde del rectángulo r.

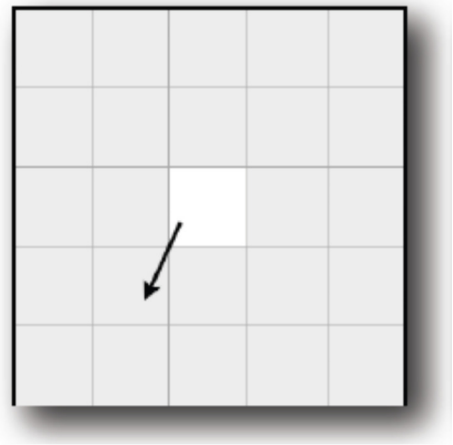
11. Se pide implementar un módulo (con archivo .h asociado) que integre todas las declaraciones y funciones de los ejercicios 4 a 10.

Para entregar:

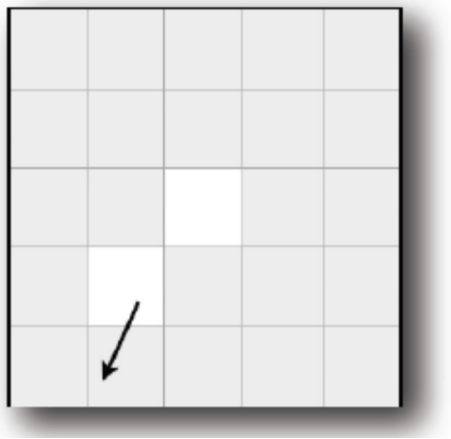
12. Este ejercicio se debe resolver sin emplear objetos ni clases. El objetivo del TP es que el alumno pueda observar cómo se estructura un programa correctamente para luego pasar a emplear objetos y clases. Por favor no utilicen los recursos de objetos y clases en este ejercicio.

Se pide simular un robot limpia-pisos. Veamos un ejemplo para explicar qué es lo que vamos a hacer (en el ejemplo el piso es de 25 baldosas, 5 x 5 unidades):

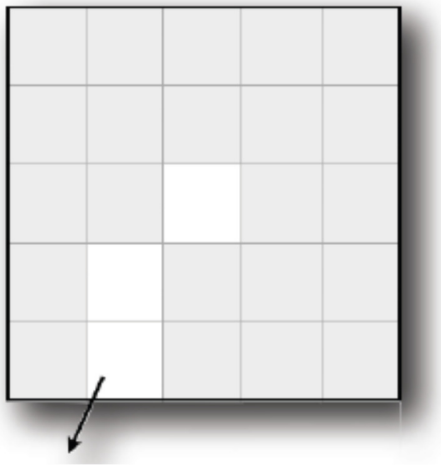
a. Tiempo *t* = 0. El robot se encuentra en la posición (2.1, 2.2), mirando a 205 grados (en el sentido de las agujas del reloj, desde el "norte"). El robot limpia la baldosa en la que está.



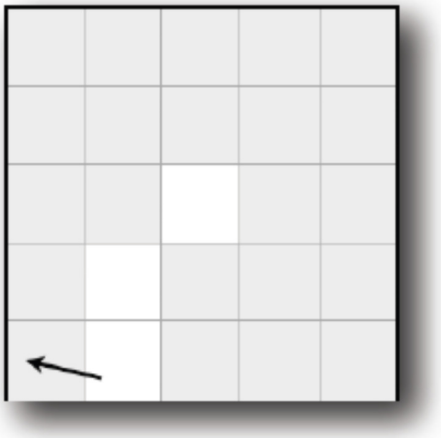
b. Tiempo *t* = 1. El robot se mueve una unidad, en la dirección que estaba mirando, a la posición (1.7, 1.3), y limpia otra baldosa.



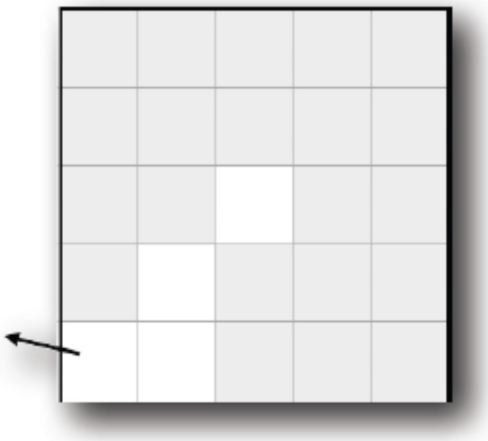
c. Tiempo *t* = 2. El robot se mueve otra unidad en la misma dirección, a la posición (1.2, 0.4), y limpia otra baldosa.



d. Tiempo *t* = 3. El robot no puede moverse en la misma dirección porque choca con la pared. Por eso cambia su dirección al azar. En este ejemplo, a 287 grados.



e. Tiempo *t* = 4. El robot se mueve una unidad en la nueva dirección hasta (0.3, 0.7), limpiando otra baldosa.



Los detalles de la simulación son:

- La habitación es rectangular, de largo *h* y ancho *w*. El tamaño de la habitación es pasado por línea de comandos y es tal que h <= 100 y w <=70.
- El piso está dividido en baldosas de una unidad por una unidad. Las baldosas pueden estar limpias o sucias. Inicialmente todo el piso está sucio.
- Hay *n* robots limpiando en simultáneo. La cantidad de robots se pasa por línea de comando.
- Para simplificar las cosas asumiremos que los robots son puntos y se pueden cruzar sin interferirse.
- La posición de un robot se describe con dos números reales, (x, y), y su dirección con un ángulo real *d* ($0 \leq d < 360$). La baldosa en que se encuentra es (floor(x), floor(y)).
- Inicialmente los robots se encuentran dentro de la habitación, en posiciones y direcciones al azar.
- La simulación avanza de a una unidad de tiempo ("*tick*"). En cada unidad de tiempo un robot avanza una unidad de distancia desde donde se encuentra, en la dirección en la que está.
- Cuando un robot choca contra una pared conserva su posición anterior, y selecciona una nueva dirección al azar.
- Cuando un robot pisa una baldosa, se considera que limpió esa baldosa.
- El programa termina cuando todas las baldosas quedaron limpias.
- Se pide desarrollar un programa que le permita elegir al usuario entre dos modos a ingresar por línea de comandos:
    1. Modo 1: El programa mostrará en tiempo real como se va limpiando el piso; graficando las baldosas limpias y sucias, a los robots y sus direcciones.
    2. Modo 2. El programa realizará un gráfico de tiempo medio de demora en la limpieza del pido en función de cantidad de robots para el tamaño de piso dado. De esta forma comenzará con n = 1 y continuará iterando hasta que el tiempo medio para n y n+1 sea menor a 0.1. Por cada step en n (cada incremento en n) se deberán correr 1000 simulaciones para estimar el tiempo medio que demora limpiar el piso con dicha cantidad de robots.

Notas:

- El programa recibe por línea de comandos, el tamaño del piso (Width y Heigh), la cantidad de robots (Robots) y el modo de la simulación (Modo).
- En Modo 2 se ignora el parámetro Robots (en el caso de recibirlo).
- El programa en el Modo 1 informa la cantidad de ticks necesarios para limpiar el piso según los parámetros recibidos. En el Modo 2 muestra el gráfico según se describe arriba.
- Es sumamente útil re-usar el módulo del ejercicio 11.

- Recuerden separar ideas separadas en módulos separados. Reducir dependencias, para el código quede poco ligado, sea fácil de entender y se pueda reciclar. Y “as simple as possible”.