Guion de prácticas

Proyecto Final - Parte 2

Metodología de la Programación

Grado en Ingeniería Informática

Prof. David A. Pelta

Introducción al guion

A partir de la clase Punto2D y el módulo pintar desarrollados en el guion anterior, se implementará una clase para representar polígonos regulares inscritos en una circunferencia.

Polígonos regulares

Un polígono regular es un polígono en el que:

- Todos los lados tienen la misma longitud.
- Todos los ángulos interiores son de la misma medida.

En un polígono regular podemos distinguir:

- Lados: cada uno de los segmentos que forman el polígono.
- Vértices: cada uno es el punto de unión de dos lados consecutivos.
- Centro: El punto central equidistante de todos los vértices.
- Radio: El segmento que une el centro del polígono con uno de sus vértices.

Una característica de los polígonos regulares es que se pueden trazar inscritos en una circunferencia que tocará cada uno de los vértices del polígono. La Fig. 1 muestra cinco polígonos inscritos en la misma circunferencia.

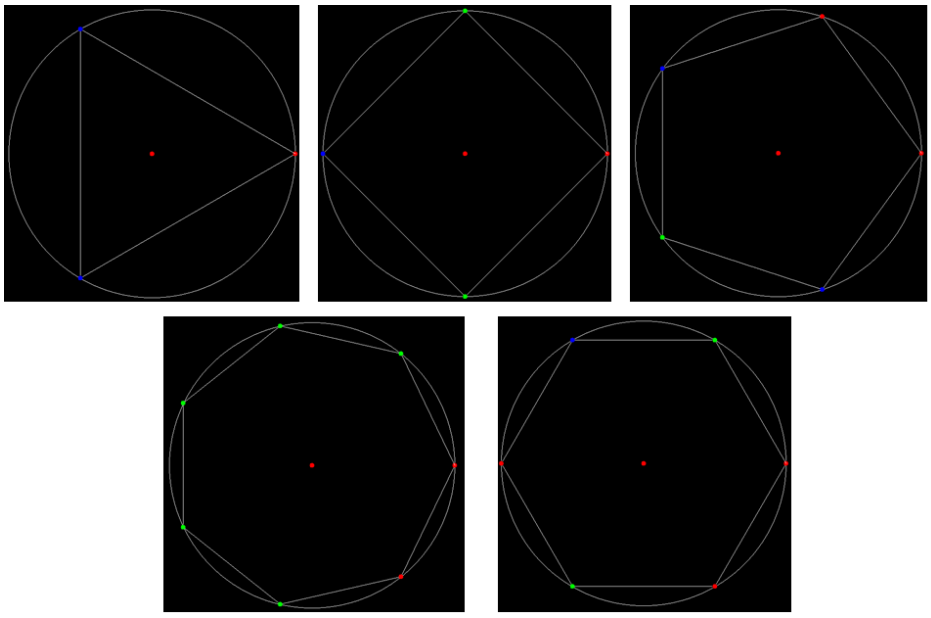
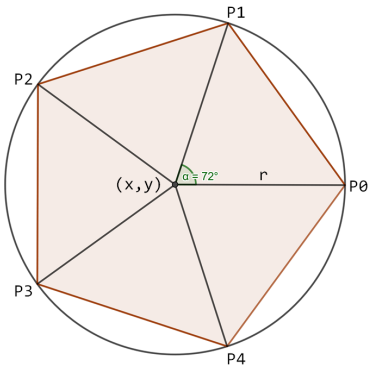


Figura 1: Ejemplos de polígonos regulares inscritos en una circunferencia.

Generación de los vértices

Supongamos que se quiere generar un polígono regular de n lados en una circunferencia con centro en (x, y), y radio r. La Figura derecha muestra un ejemplo para $n = 5$.



Para ello se puede utilizar el siguiente procedimiento. En primer lugar se calcula un ángulo $\alpha = 360°/n$. Luego se genera el primer punto del polígono en $p_0 = (x + r, y)$ (también valdría el $(x, y + r)$).

Para los puntos posteriores debe notarse que el punto p_1 se obtiene al rotar p_0 sobre el centro, α grados. El punto p_2 se obtiene al rotar p_1 sobre el centro, α grados, y así sucesivamente

Alternativamente, el punto p_n se obtiene de rotar p_0 sobre el centro $n \times \alpha$ grados.

Clase PoliReg

Se pretende diseñar una clase que permita trabajar con polígonos regulares y se propone la siguiente representación.

```
class PoliReg{
private:
 Punto2D *vertices; // un array de puntos para los vertices
 int N;             // nro. de lados/vertices del poligono
 Punto2D centro;    // el centro de la circunferencia
 float radio;         // y su radio
 static const int MAX_VERT; // lims para el nro de vertices
 static const int MIN_VERT;
 static const float MAX_RADIO; // lims para el valor del radio
 static const float MIN_RADIO;
```

Cada objeto contiene los vértices del polígono, así como los datos de la circunferencia que permitió generarlo. Además, la clase establece límites para el máximo/mínimo número de vértices/lados y para el valor del radio. De esta manera, podremos ampliar/reducir el número de vértices, expandir/reducir el área del polígono, rotar el polígono alrededor del centro, etc.

Para inicializar las constantes de la clase, incluya las siguientes sentencias en el fichero `PoliReg.cpp`

```
const float PoliReg::MAX_RAD = 600.0;
const float PoliReg::MIN_RAD = 15.0;
const int PoliReg::MAX_VERT = 200;
const int PoliReg::MIN_VERT = 3;
```

Respecto a los métodos, y además de los `set/get` adecuados, la clase debe proveer:

- `PoliReg()`: Constructor por defecto. Genera un polígono con $N = MIN\_VERT$ elementos, con centro en la posición $(0, 0)$ y radio MIN_RADIO.

- `PoliReg(int nroLados, const Punto2D & centro, int r)`: Constructor con parámetros. Genera un polígono de `nroLados` elementos, inscrito en una circunferencia de radio r y origen en `centro`.
- `void AgregaVertice()`: reconstruye `vertices`, generando un nuevo polígono regular con 1 vértice más. No se permitirán polígonos con más de MAX_VERTS vértices.
- `void EliminaVertice()`: reconstruye `vertices`, generando un nuevo polígono regular con 1 vértice menos. No se puede dejar el polígono con menos de MIN_VERTS vértices.
- `void Expande(int pct)`: incrementa el radio en un porcentaje `pct` y recalcula los vértices. El radio no deberá ser mayor que MAX_RADIO
- `void Contrae(int pct)`: reduce el radio en un porcentaje `pct` y recalcula los vértices. El radio no deberá ser mayor que MIN_RADIO
- `double Perimetro()`: calcula y devuelve el perímetro del polígono.
- `void Rotar(int rads)`: rota el polígono sobre su centro en $rads$ radianes.
- `bool Colision(const PoliReg & otro)`: devuelve `true` si los polígonos colisionan. Para ello, debe comprobar que las correspondientes circunferencias no se intersecan (debe utilizar los respectivos centros y radios). Si no hay colisión, devuelve `false`.
- `~PoliReg()`: Destructor de la clase.

Si considera que alguno de estos métodos debe ser `const`, agréguelo.

Además de estos métodos, puede agregar los que considere oportunos. En este clase hay al menos tres métodos privados que le pueden ser de utilidad para simplificar la implementación y evitar la repetición de código. Estos métodos son:

1. `ReservaMemoria()`: reserva memoria para el vector `vertices`, usando el valor N.
2. `LiberaMemoria()`: libera la memoria ocupada por el vector `vertices`.
3. `GeneraVertices()`: utilizando el `centro`, el `radio` y `N`, genera el vector de vértices.

Analice las implicaciones de métodos tales como `SetRadio`, `SetNroVertices`, `SetCentro`. ¿Sería necesario que estuvieran disponibles?

Pintar Poligono

Al igual que la Clase `Punto2D`, la clase `PoliReg` tampoco conoce nada del `MiniWin` y por tanto, no tiene sentido que la clase provea métodos para "dibujar" un objeto.

Para pintar un objeto de la Clase `PoliReg` extienda las funciones disponibles en el módulo `pintar` desarrollado en el guion anterior.

Pruebas

Al finalizar la implementación de esta parte del proyecto final, tendrá 4 módulos disponibles: `MiniWin, pintar, Punto2D y PoliReg`. Como además tendrá un programa principal (`main.cpp`) estará manejando proyectos de Netbeans con hasta 9 ficheros y comprobará la utilidad de disponer de un entorno de desarrollo.

Uno de los aspectos fundamentales de este proyecto final es el uso correcto de clases con memoria dinámica (que hasta el momento solo es posible parcialmente). Para verificar la implementación de la Clase `PoliReg` en este aspecto, no se necesita `MiniWin` ni `pintar`.

Tiene disponible en PRADO un fichero `main.cpp` donde se recogen pruebas básicas para comprobar el funcionamiento del código. Verifique que `valgrind` indica que no existen errores en el manejo de la memoria dinámica.

1. Instrucciones para la entrega

Se publicarán en PRADO.

Importante

Estas instrucciones deben complementarse con las indicaciones dadas durante las sesiones de práctica.