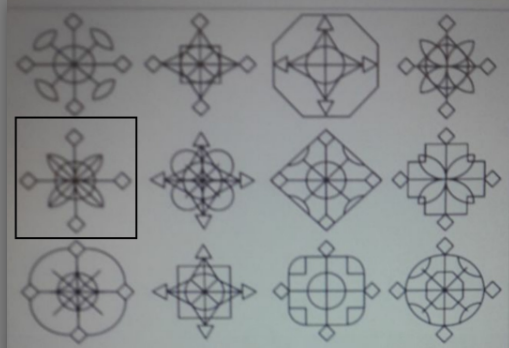
**Лабораторна робота №5**

**Розробка класу для визначення фігури**

**Мета роботи:** оволодіти методами та засобами розробки класів на прикладі малювання фігури

**Завдання:** Розробити класи для малювання фігури



1. **Текст програми**

***Figure.h***

#pragma once

#include <cmath>

#include <SFML/Graphics.hpp>

namespace snow {

   class Figure {

   public:

      Figure();

      Figure(sf::Vector2f coords, float angle = 0, float size = 20);

      void draw(sf::RenderWindow &window);

      void setPosition(sf::Vector2f coords);

      sf::Vector2f getPosition();

      void move(sf::Vector2f offset);

      void setAngle(float angle);

      float getAngle();

      void rotate(float angle);

      void setSize(float size);

      float getSize();

      void updatePoints();

      ~Figure() {}

   private:

      void drawFragment(sf::RenderWindow &window);

      // Рисует общие части фигуры

      void drawGeneralParts(sf::RenderWindow &window);

      sf::VertexArray \_getBezierCoords(sf::Vector2f p1, sf::Vector2f p2, sf::Vector2f p3, int count = 30);

      float \_size;

      sf::Vector2f \_coords;

      float \_angle;

      sf::Vector2f \_p1, \_p2, \_p3, \_p4, \_p5, \_p6, \_p7, \_p8, \_p9;

      // Угол создаваемый точками p2, p0, p6

      static const double \_BETA1;

      // Угол создаваемый точками p6, p0, p5

      static const double \_BETA2;

      // Длина отрезка p0p2

      static const double \_P0P2;

      // Длина отрезка p0p8

      static const double \_P0P8;

      static const double PI;

   };

} //namespace snow

***Figure.cpp***

#include <SFML/Graphics.hpp>

#include <cmath>

#include "Figure.h"

namespace snow {

Figure::Figure():

\_coords(sf::Vector2f(0, 0)),

\_size(10),

\_angle(0)

{

}

Figure::Figure(sf::Vector2f coords, float size, float angle) :

\_coords(coords),

\_size(size),

\_angle(angle)

{

}

void Figure::draw(sf::RenderWindow &window) {

const float angle = float(PI / 2);

const float startAngle = \_angle;

for (int i = 0; i < 4; i++) {

updatePoints();

drawFragment(window);

rotate(angle);

}

setAngle(startAngle);

drawGeneralParts(window);

}

void Figure::drawFragment(sf::RenderWindow &window) {

sf::Vertex part1[] = { \_p1, \_p2, \_p3, \_coords };

sf::Vertex part2[] = { \_coords, \_p4 };

sf::Vertex part3[] = { \_p5, \_p6, \_p7 };

auto arc1 = \_getBezierCoords(\_coords, \_p8, \_p4, 15);

auto arc2 = \_getBezierCoords(\_coords, \_p9, \_p4, 15);

window.draw(part1, 4, sf::LineStrip);

window.draw(part2, 2, sf::Lines);

window.draw(part3, 3, sf::LineStrip);

window.draw(arc1);

window.draw(arc2);

}

void Figure::drawGeneralParts(sf::RenderWindow &window) {

sf::CircleShape circle(\_size);

circle.setOrigin(\_size, \_size);

circle.setPosition(\_coords);

circle.setFillColor(sf::Color::Transparent);

circle.setOutlineThickness(1);

window.draw(circle);

}

void Figure::updatePoints() {

\_p1.x = float(\_coords.x + \_size \* cos(\_angle + PI / 2) \* 8 / 3);

\_p1.y = float(\_coords.y + \_size \* sin(\_angle + PI / 2) \* 8 / 3);

\_p2.x = float(\_coords.x + \_P0P2 \* \_size \* cos(\_angle + \_BETA1));

\_p2.y = float(\_coords.y + \_P0P2 \* \_size \* sin(\_angle + \_BETA1));

\_p3.x = float(\_coords.x + 2 \* \_size \* cos(\_angle + PI / 2)),

\_p3.y = float(\_coords.y + 2 \* \_size \* sin(\_angle + PI / 2));

\_p4.x = float(\_coords.x + 2 \* \_size \* cos(\_angle + PI / 4));

\_p4.y = float(\_coords.y + 2 \* \_size \* sin(\_angle + PI / 4));

\_p5.x = float(\_coords.x + 2 \* \_size \* cos(\_angle));

\_p5.y = float(\_coords.y + 2 \* \_size \* sin(\_angle));

\_p6.x = float(\_coords.x + \_P0P2 \* \_size \* cos(\_angle + \_BETA2));

\_p6.y = float(\_coords.y + \_P0P2 \* \_size \* sin(\_angle + \_BETA2));

\_p7.x = float(\_coords.x + \_size \* cos(\_angle) \* 8 / 3);

\_p7.y = float(\_coords.y + \_size \* sin(\_angle) \* 8 / 3);

\_p8.x = float(\_coords.x + \_P0P8 \* \_size \* cos(\_angle + PI / 2));

\_p8.y = float(\_coords.y + \_P0P8 \* \_size \* sin(\_angle + PI / 2));

\_p9.x = float(\_coords.x + \_P0P8 \* \_size \* cos(\_angle));

\_p9.y = float(\_coords.y + \_P0P8 \* \_size \* sin(\_angle));

}

void Figure::setPosition(sf::Vector2f coords) {

\_coords.x = coords.x;

\_coords.y = coords.y;

}

sf::Vector2f Figure::getPosition() {

return sf::Vector2f(\_coords);

}

void Figure::move(sf::Vector2f offset) {

setPosition(\_coords + offset);

}

void Figure::setAngle(float angle) {

\_angle = angle;

}

float Figure::getAngle() {

return \_angle;

}

void Figure::rotate(float angle) {

setAngle(\_angle + angle);

}

void Figure::setSize(float size) {

if (size < 1) size = 1;

\_size = size;

}

float Figure::getSize() {

return \_size;

}

////////////////////////////////////////////////////////////

/// \brief Возвращает массив точек кривой Безье для 3 контрольных точек

///

/// \param p1, p2, p3 Соответственно 1, 2, 3 контрольные точки

/// \param count Количество точек из которых будет состоять кривая

////////////////////////////////////////////////////////////

sf::VertexArray Figure::\_getBezierCoords(sf::Vector2f p1, sf::Vector2f p2, sf::Vector2f p3, int count) {

if (count < 3) count = 3;

sf::VertexArray vertices(sf::LineStrip, count);

for (int i = 0; i < count; i++) {

float t = float(i) / (count - 1);

vertices[i].position = ((1.f - t) \* (1.f - t)) \* p1

+ (2.f \* (1.f - t) \* t) \* p2

+ (t \* t) \* p3;

}

return vertices;

}

const double Figure::PI = 3.14159265358979323846;

const double Figure::\_BETA1 = atan(7);

const double Figure::\_BETA2 = PI / 2 - Figure::\_BETA1;

const double Figure::\_P0P2 = 1.f / 3.f \* sqrt(50);

const double Figure::\_P0P8 = sqrt(2);

} // namespace snow

***Main.cpp***

#include <iostream>

#include <SFML/Graphics.hpp>

#include "Figure.h"

enum Keys {

Left = sf::Keyboard::A,

Top = sf::Keyboard::W,

Right = sf::Keyboard::D,

Bottom = sf::Keyboard::S,

RotatePlus = sf::Keyboard::Right,

RotateMinus = sf::Keyboard::Left,

ScalePlus = sf::Keyboard::Up,

ScaleMinus = sf::Keyboard::Down,

};

void controlFigure(snow::Figure &figure, float frameTime);

int main() {

sf::ContextSettings settings;

settings.antialiasingLevel = 4;

sf::RenderWindow window(sf::VideoMode(1000, 600), "Figure", sf::Style::Default, settings);

window.setFramerateLimit(180);

snow::Figure f1(sf::Vector2f(250, 250), 50, 0.2f);

snow::Figure f2(sf::Vector2f(50, 250), 20, 0.6f);

snow::Figure f3(sf::Vector2f(600, 350), 20, 0.f);

sf::Clock clock;

float frameTime;

while (window.isOpen()) {

frameTime = clock.restart().asMilliseconds();

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) {

window.close();

}

else if (event.type == sf::Event::Resized) {

sf::FloatRect visible(0, 0, float(event.size.width), float(event.size.height));

window.setView(sf::View(visible));

}

}

controlFigure(f1, frameTime);

window.clear();

f1.draw(window);

f2.draw(window);

f3.draw(window);

window.display();

}

return 0;

}

void controlFigure(snow::Figure &figure, float frameTime) {

typedef sf::Keyboard::Key Key;

float step = 0.3f \* frameTime;

float angle = 0.005f \* frameTime;

float scale = 0.05f \* frameTime;

if (sf::Keyboard::isKeyPressed((Key)Keys::Left)) {

figure.move(sf::Vector2f(-step, 0));

}

if (sf::Keyboard::isKeyPressed((Key)Keys::Right)) {

figure.move(sf::Vector2f(step, 0));

}

if (sf::Keyboard::isKeyPressed((Key)Keys::Top)) {

figure.move(sf::Vector2f(0, -step));

}

if (sf::Keyboard::isKeyPressed((Key)Keys::Bottom)) {

figure.move(sf::Vector2f(0, step));

}

if (sf::Keyboard::isKeyPressed((Key)Keys::ScalePlus)) {

figure.setSize(figure.getSize() + scale);

}

if (sf::Keyboard::isKeyPressed((Key)Keys::ScaleMinus)) {

figure.setSize(figure.getSize() - scale);

}

if (sf::Keyboard::isKeyPressed((Key)Keys::RotateMinus)) {

figure.rotate(-angle);

}

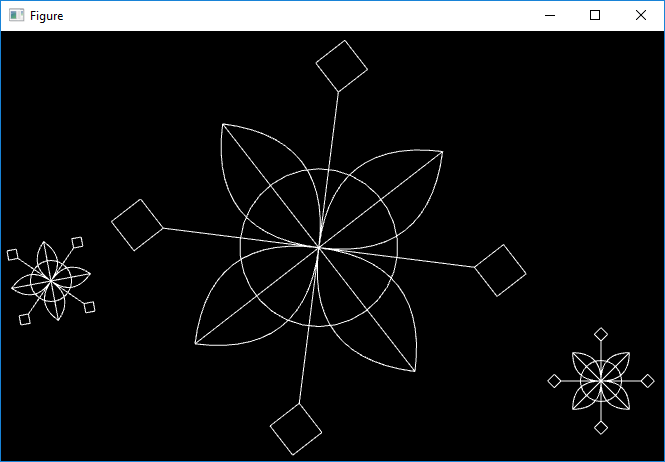
if (sf::Keyboard::isKeyPressed((Key)Keys::RotatePlus)) {

figure.rotate(angle);

}

}

1. **Результат виконання програми**



**Висновок:** при виконанні лабораторної роботи я оволодів методами та засобами представлення послідовності виконання дій програми на прикладі розробки комп’ютерної гри «Pacman».