# Задача

Да се напише програма, която позволява интерактивно създаване на сцена, съдържаща произволен брой фигури:

* **Права линия** – задава се с посочване на 2 точки с ляв бутон на мишката – начална точка по X и дължина по Х
* **Елипса** – задава се с посочване на 2 точки с ляв бутон на мишката – център, радиус по Х, а като радиус по Y се задава радиуса по X, умножен по 2
* **Правоъгълник** – задава се с посочване на 2 точки с ляв бутон на мишката – горна лява точка и долна дясна точка
* **Звезда** – задава се с посочване на 2 точки с ляв бутон на мишката – център и радиус на окръжност, в която се вписва звездата, след това се изчисляват пет точки в прозрачна окръжност и се свързват

Да се реализира изтриване на фигура – с десен бутон на мишката или с Ctrl + ляв бутон на мишката в близост до 3 пиксела около линията на съответната фигура. Получената сцена да се съхранява и чете от файл. File🡪New да изчиства сцена, като предложи съхраняване, ако такова не е извършено.

# Реализация

1. Фигура

За всяка фигура се декларира MFC клас, който наследява един базов клас Figure с виртуални методи – метод за изчертаване на фигурата, метод за проверка дали е кликната дадената фигура и метод за записване във файл. Виртуалните методи се имплементират за всеки клас.

1. Изчертаване

Заради базовия клас Figure, които всички фигури наследяват, се използва един от основните принципи на обектно-ориентираното програмиране – полиморфизъм, за да декларираме вектор от тип Figure, в който се съхраняват всички фигури. Така всяка отделна фигура знае как да се изчертае и дали е кликната върху нея, благодарение на виртуалните методи от базовия клас.

Как разбираме коя фигура чертаем? – създаваме променлива mode, която при стойност 0 изчертава Права линия, при стойност 1 изчертава Елипса, при стойност 2 изчертава Правоъгълник, при стойност 3 изчертава Звезда.

1. Съхраняване и четене от текстов файл

Формат на данните в текстовия файл:

“тип\_фигура ˽ променлива1 ˽ променлива2 ˽ променлива3 ˽ ... ˽ променливаN”

Тип фигура – при стойност 0 се чертае права линия, при 1 се чертае елипса, при 2 се чертае правоъгълник.

Програмата работи с множество сцени.

**Файл – Figure.h | Клас - Figure**

class Figure : public CWnd

{

DECLARE\_DYNAMIC(Figure)

public:

// статична променлива за определяне колко пиксела от

// очертанията на дадена фигура да се изтрива

static const int IndentationPixels = 3;

Figure();// празен конструктор

virtual ~Figure();// празен виртуален деструктор

// виртуален метод за изчертаване

virtual void Draw(CDC\* pDC){};

// виртуален метод за определяне дали е кликната фигурата

virtual bool IsClicked(int x, int y) {return false;};

protected:

DECLARE\_MESSAGE\_MAP()

};

**Файл – Line.h | Клас - Line**

#pragma once

#include "Figure.h"

class Line : public CWnd, public Figure // наследяване на базовия клас

{

DECLARE\_DYNAMIC(Line)

private:

int x, y, length;

public:

Line(int x, int y, int length);

virtual ~Line();

void Draw(CDC\* pDC);

bool IsClicked(int x, int y);

void Serialize(CArchive& ar);

void SetX(int x);

void SetY(int y);

void SetLength(int x);

int GetX();

int GetY();

int GetLength();

protected:

DECLARE\_MESSAGE\_MAP()

};

**Файл – Line.cpp | Клас - Line**

#include "stdafx.h"

#include "Graphics.h"

#include "Line.h"

IMPLEMENT\_DYNAMIC(Line, CWnd)

// конструктор

Line::Line(int x, int y, int length)

{

this->SetX(x);

this->SetY(y);

this->SetLength(length);

}

Line::~Line()

{

}

void Line::Draw(CDC\* pDC)

{

pDC->MoveTo(this->x, this->y);

pDC->LineTo(this->x + this->length, this->y);

}

bool Line::IsClicked(int x, int y)

{

// ако е кликнато около 3 пиксела по Х значи върни true

if(abs(y - this->y) <= Figure::IndentationPixels &&

(x >= this->x && x <= (this->length + this->x)))

{

return true;

}

else if(abs(y - this->y) <= Figure::IndentationPixels &&

(x >= (this->length + this->x) && x <= this->x))

{

return true;

}

return false;

}

void Line::Serialize(CArchive& ar)

{

// записване на обекта в текстов файл

if(ar.IsStoring())

{

CString line;

line.Format(\_T("%d %d %d %d \r\n"), 0, this->x, this->y,

this->length);

ar.WriteString(line);

}

}

void Line::SetX(int x)

{

this->x = x;

}

void Line::SetY(int y)

{

this->y = y;

}

void Line::SetLength(int x)

{

this->length = x - this->x;

}

int Line::GetX()

{

return this->x;

}

int Line::GetY()

{

return this->y;

}

int Line::GetLength()

{

return this->length;

}

**Файл – EllipseFigure.h | Клас - EllipseFigure**

#pragma once

#include "Figure.h"

class EllipseFigure : public CWnd, public Figure // наследяване на базовия клас

{

DECLARE\_DYNAMIC(EllipseFigure)

private:

int centerX, centerY, radius;

public:

EllipseFigure(int x, int y, int r);

virtual ~EllipseFigure();

void Draw(CDC\* pDC);

bool IsClicked(int x, int y);

void Serialize(CArchive& ar);

void SetX(int x);

void SetY(int y);

void SetRadius(int x);

int GetX();

int GetY();

int GetRadius();

protected:

DECLARE\_MESSAGE\_MAP()

};

**Файл – EllipseFigure.cpp | Клас - EllipseFigure**

#include "stdafx.h"

#include "Graphics.h"

#include "EllipseFigure.h"

IMPLEMENT\_DYNAMIC(EllipseFigure, CWnd)

// конструктор

EllipseFigure::EllipseFigure(int x, int y, int r)

{

this->SetX(x);

this->SetY(y);

this->SetRadius(r);

}

EllipseFigure::~EllipseFigure()

{

}

void EllipseFigure::Draw(CDC\* pDC)

{

pDC->Ellipse(this->centerX - this->radius, this->centerY -this->radius\*2,

this->centerX + this->radius, this->centerY + this->radius \* 2);

}

bool EllipseFigure::IsClicked(int x, int y)

{

// проверката дали е клинката върху фигурата не работи коректно

// TODO: fix

if(y - this->radius\*2 <= this->centerY ||

y + this->radius\*2 == this->centerY ||

x - this->radius == this->centerX ||

x + this->radius == this->centerX)

{

return true;

}

return false;

}

void EllipseFigure::Serialize(CArchive& ar)

{

// записване на обекта в текстов файл

if(ar.IsStoring())

{

CString line;

line.Format(\_T("%d %d %d %d \r\n"), 1, this->centerX,

this->centerY, this->radius);

ar.WriteString(line);

}

}

void EllipseFigure::SetX(int x)

{

this->centerX = x;

}

void EllipseFigure::SetY(int y)

{

this->centerY = y;

}

void EllipseFigure::SetRadius(int x)

{

this->radius = this->centerX - x;

}

int EllipseFigure::GetX()

{

return this->centerX;

}

int EllipseFigure::GetY()

{

return this->centerY;

}

int EllipseFigure::GetRadius()

{

return this->radius;

}

**Файл – RectangleFigure.h**

#pragma once

#include "Figure.h"

class RectangleFigure : public CWnd, public Figure//наследяване на базовия клас

{

DECLARE\_DYNAMIC(RectangleFigure)

private:

int topLeftX, topLeftY, downRightX, downRightY;

public:

RectangleFigure(int topLeftX, int topLeftY, int downRightX, int downRightY);

virtual ~RectangleFigure();

void Draw(CDC\* pDC);

bool IsClicked(int x, int y);

void Serialize(CArchive& ar);

void SetTopLeftX(int x);

void SetTopLeftY(int y);

void SetDownRightX(int x);

void SetDownRightY(int y);

int GetTopLeftX();

int GetTopLeftY();

int GetDownRightX();

int GetDownRightY();

protected:

DECLARE\_MESSAGE\_MAP()

};

**Файл – RectangleFigure.cpp | Клас - RectangleFigure**

#include "stdafx.h"

#include "Graphics.h"

#include "RectangleFigure.h"

IMPLEMENT\_DYNAMIC(RectangleFigure, CWnd)

// Конструктор

RectangleFigure::RectangleFigure(int topLeftX, int topLeftY, int downRightX, int downRightY)

{

this->SetTopLeftX(topLeftX);

this->SetTopLeftY(topLeftY);

this->SetDownRightX(downRightX);

this->SetDownRightY(downRightY);

}

RectangleFigure::~RectangleFigure()

{

}

void RectangleFigure::Draw(CDC\* pDC)

{

pDC->Rectangle(this->topLeftX, this->topLeftY, this->downRightX, this->downRightY);

}

bool RectangleFigure::IsClicked(int x, int y)

{

// променливи за проверка на дължините на линиите

bool topAndBottomLines = ((x >= this->topLeftX && x <= this->downRightX) || x >= this->downRightX && x <= this->topLeftX);

bool leftAndRightLines = ((y >= this->topLeftY && y <= this->downRightY) || (y >= this->downRightY && y <= this->topLeftY));

// проверка за горната линия на правоъгълника

if(abs(y - this->topLeftY) <= Figure::IndentationPixels && topAndBottomLines)

{

return true;

}

// проверка за долната линия на правоъгълника

if(abs(y - this->downRightY) <= Figure::IndentationPixels && topAndBottomLines)

{

return true;

}

// проверка за лявата линия на правоъгълника

if(abs(x - this->topLeftX) <= Figure::IndentationPixels && leftAndRightLines)

{

return true;

}

// проверка за дясната линия на правоъгълника

if(abs(x - this->downRightX) <= Figure::IndentationPixels && leftAndRightLines)

{

return true;

}

return false;

}

void RectangleFigure::Serialize(CArchive& ar)

{

// записване на обекта в текстов файл

if(ar.IsStoring())

{

CString line;

line.Format(\_T("%d %d %d %d %d \r\n"), 2, this->topLeftX, this->topLeftY, this->downRightX, this->downRightY);

ar.WriteString(line);

}

}

void RectangleFigure::SetTopLeftX(int x)

{

this->topLeftX = x;

}

void RectangleFigure::SetTopLeftY(int y)

{

this->topLeftY = y;

}

void RectangleFigure::SetDownRightX(int x)

{

this->downRightX = x;

}

void RectangleFigure::SetDownRightY(int y)

{

this->downRightY = y;

}

int RectangleFigure::GetTopLeftX()

{

return this->topLeftX;

}

int RectangleFigure::GetTopLeftY()

{

return this->topLeftY;

}

int RectangleFigure::GetDownRightX()

{

return this->downRightX;

}

int RectangleFigure::GetDownRightY()

{

return this->downRightY;

}

BEGIN\_MESSAGE\_MAP(RectangleFigure, CWnd)

END\_MESSAGE\_MAP()

**Файл – StarFigure.h**

#pragma once

#include "Figure.h"

class StarFigure : public CWnd, public Figure//наследяване на базовия клас

{

DECLARE\_DYNAMIC(StarFigure)

private:

int centerX, centerY, radius, x, y;

CPoint star[5];

void SetRadius();

void InitializePoints();

public:

StarFigure(int cx, int cy, int x, int y);

virtual ~StarFigure();

void Draw(CDC\* pDC);

bool IsClicked(int x, int y);

void Serialize(CArchive& ar);

protected:

DECLARE\_MESSAGE\_MAP()

};

**Файл – StarFigure.cpp | Клас - StarFigure**

#include "stdafx.h"

#include <math.h>

#include "Graphics.h"

#include "StarFigure.h"

IMPLEMENT\_DYNAMIC(StarFigure, CWnd)

// Конструктор

StarFigure::StarFigure(int cx, int cy, int x, int y)

{

this->centerX = cx;

this->centerY = cy;

this->x = x;

this->y = y;

this->SetRadius();

this->InitializePoints();

}

// Метод за инициализиране на петте точки на звездата

void StarFigure::InitializePoints()

{

float alpha = 3.1415926 / 2;

for(int i=0; i<5; i++)

{

star[i].x = this->centerX + this->radius \* cos(alpha);

star[i].y = this->centerY - this->radius \* sin(alpha);

alpha += 2 \* 3.1415926 / 5.0;

}

}

StarFigure::~StarFigure()

{

}

void StarFigure::Draw(CDC\* pDC)

{

pDC->MoveTo(star[0].x, star[0].y);

pDC->LineTo(star[2].x, star[2].y);

pDC->LineTo(star[4].x, star[4].y);

pDC->LineTo(star[1].x, star[1].y);

pDC->LineTo(star[3].x, star[3].y);

pDC->LineTo(star[0].x, star[0].y);

}

bool StarFigure::IsClicked(int x, int y)

{

// Проверката работи само за правата линия на звездата

if(abs(y - this->star[4].y) <= Figure::IndentationPixels &&

x >= this->star[1].x && x <= this->star[4].x)

{

return true;

}

return false;

}

void StarFigure::Serialize(CArchive& ar)

{

// записване на обекта в текстов файл

if(ar.IsStoring())

{

CString line;

line.Format(\_T("%d %d %d %d %d \r\n"), 3, this->centerX, this->centerY, this->x, this->y);

ar.WriteString(line);

}

}

void StarFigure::SetRadius()

{

this->radius = sqrt((float)((this->centerX - x) \* (this->centerX - x) +

(this->centerY - y) \* (this->centerY - y)));

}

**Файл – GraphicsDoc.h**

class CGraphicsDoc : public CDocument

{

------------Код-------------

// Деклариране на променливи нужни за обработка

public:

int numberOfObjects, numberOfClicks; // броячи

int x, y; // за временно съхранение на данни

int mode; // променлива за определяне на коя фигура се чертае

std::vector<Figure\*> figures; // вектор от фигури

------------Код-------------

}

**Файл – GraphicsDoc.cpp**

// Деструктор за освобождаване на паметта във вектора

CGraphicsDoc::~CGraphicsDoc()

{

// Използване на итератор за вектор

for(std::vector<Figure\*>::const\_iterator it = figures.begin(); it != figures.end(); it++)

{

delete \*it;

}

figures.clear();

}

// При отваряне на нов документ

BOOL CGraphicsDoc::OnNewDocument()

{

if (!CDocument::OnNewDocument())

return FALSE;

// Зануляване

numberOfClicks = 0;

numberOfObjects = 0;

mode = 0;

x = 0;

y = 0;

// Резервиране на памет във вектора за 50 елемента

figures.reserve(50);

return TRUE;

}

void CGraphicsDoc::Serialize(CArchive& ar)

{

// Деклариране на променлива от тип символен низ

CString line;

if (ar.IsStoring())

{

// Записване във файл

for(int i=0; i<numberOfObjects; i++)

{

figures[i]->Serialize(ar); // Извикване на виртуален метод

}

}

else

{

// Четене от файл

numberOfObjects = 0;

numberOfClicks = 0;

mode = 0;

int x1, y1, m, length, radius;

while(ar.ReadString(line))

{

// Копиране на данни от line в нова променлива заради

// sscanf, който не приема аргумент от тип CString

CT2A ascii(line);

TRACE(\_T("ASCII: %S\n"), ascii.m\_psz);

// За всеки ред се проверява първото число, според него

// се създава съответния обект и се добавя във вектора

// накрая се увеличава броя на обектите

if(line[0] == '0')

{

sscanf(ascii, "%d %d %d %d", &m, &x, &y, &length);

figures.push\_back(new Line(x, y, length+x));

numberOfObjects++;

}

else if(line[0] == '1')

{

sscanf(ascii, "%d %d %d %d", &m, &x, &y, &radius);

figures.push\_back(new EllipseFigure(x, y, radius+x));

numberOfObjects++;

}

else if(line[0] == '2')

{

sscanf(ascii, "%d %d %d %d %d", &m, &x, &y, &x1, &y1);

figures.push\_back(new RectangleFigure(x, y, x1, y1));

numberOfObjects++;

}

else if(line[0] == '3')

{

sscanf(ascii, "%d %d %d %d %d", &m, &x, &y, &x1, &y1);

figures.push\_back(new StarFigure(x, y, x1, y1));

numberOfObjects++;

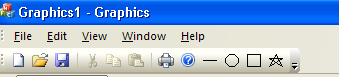
}

}

}

}

**Toolbar**



Създадени са икони в главния toolbar за избиране на коя фигура да се изчертава (По ред: Права линия, Елипса, Правоъгълник, Звезда). При избиране с мишката на някоя от иконите променливата mode си променя стойността и може да се започне да се чертае съответната фигура.

**Файл – GraphicsView.cpp**

// Вмъкване на класовете

#include "GraphicsDoc.h"

#include "GraphicsView.h"

#include "Line.h"

#include "EllipseFigure.h"

#include "RectangleFigure.h"

#include "StarFigure.h"

// Метод за изчертаване на всички обекти

void CGraphicsView::OnDraw(CDC\* pDC)

{

CGraphicsDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

if (!pDoc) return;

// Избиране на прозрачна четка

pDC->SelectStockObject(HOLLOW\_BRUSH);

// Чертаене на всички обекти

for(int i=0; i<pDoc->numberOfObjects; i++)

{

// Извикване на виртуалния метод Draw за всяка фигура

pDoc->figures[i]->Draw(pDC);

}

}

// Метод при натискане на ляв бутон с мишката за изчертаване

void CGraphicsView::OnLButtonUp(UINT nFlags, CPoint point)

{

CGraphicsDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

if (!pDoc) return;

if(nFlags == 0x0008)

{

// Ако Ctrl бутон е натиснат извикване на метода за изтриване

OnRButtonDown(nFlags, point);

}

else

{

// Добавяне на обекти

if(pDoc->numberOfClicks % 2 == 0)

{

// Ако броят на кликанията е четно число се

// задват началните точки на дадената фигура

pDoc->x = point.x;

pDoc->y = point.y;

}

else

{

// Ако броят на кликанията е нечетно число се

// проверява коя фигура се чертае в момента и се

// създава нов обект от съответната фигура, след

// което се добавя във вектора

if(pDoc->mode == 0)

{

// Права линия

pDoc->figures.push\_back(new Line(pDoc->x, pDoc->y, point.x));

}

else if(pDoc->mode == 1)

{

// Елипса

pDoc->figures.push\_back(new EllipseFigure(pDoc->x, pDoc->y, point.x));

}

else if(pDoc->mode == 2)

{

// Правоъгълник

pDoc->figures.push\_back(new RectangleFigure(pDoc->x, pDoc->y, point.x, point.y));

}

else if(pDoc->mode == 3)

{

// Звезда

pDoc->figures.push\_back(new StarFigure(pDoc->x, pDoc->y, point.x, point.y));

}

// Увеличаване на броя на обектите

pDoc->numberOfObjects++;

// Преизчертаване на екрана с новодобавената фигура

Invalidate();

}

// Увеличаване броя на кликанията

pDoc->numberOfClicks++;

}

CView::OnLButtonUp(nFlags, point);

}

// Метод при натискане на десен бутон с мишката за изтриване на даден обект

void CGraphicsView::OnRButtonDown(UINT nFlags, CPoint point)

{

CGraphicsDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

if (!pDoc) return;

// Проверката дали е кликнато върху някой обект

// и ако е кликнато премахване на обекта от вектора

// и намаляване на броя на обектите

for(int i=0; i<pDoc->numberOfObjects; i++)

{

if(pDoc->figures[i]->IsClicked(point.x, point.y))

{

pDoc->figures.erase(pDoc->figures.begin() + i);

pDoc->numberOfObjects--;

Invalidate();

}

}

CView::OnRButtonDown(nFlags, point);

}

// Метод при натискане на Права линия от toolbar-a

void CGraphicsView::OnButtonMode0()

{

CGraphicsDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

if (!pDoc) return;

// Промяна на стойността на mode за изчертаване на Права линия

pDoc->mode = 0;

Invalidate();

}

// Метод при натискане на Елипса от toolbar-a

void CGraphicsView::OnButtonMode1()

{

CGraphicsDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

if (!pDoc) return;

// Промяна на стойността на mode за изчертаване на Елипса

pDoc->mode = 1;

Invalidate();

}

// Метод при натискане на Правоъгълник от toolbar-a

void CGraphicsView::OnButtonMode2()

{

CGraphicsDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

if (!pDoc) return;

// Промяна на стойността на mode за изчертаване на Правоъгълник

pDoc->mode = 2;

Invalidate();

}

// Метод при натискане на Звезда от toolbar-a

void CGraphicsView::OnButtonMode3()

{

CGraphicsDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

if (!pDoc) return;

// Промяна на стойността на mode за изчертаване на Звезда

pDoc->mode = 3;

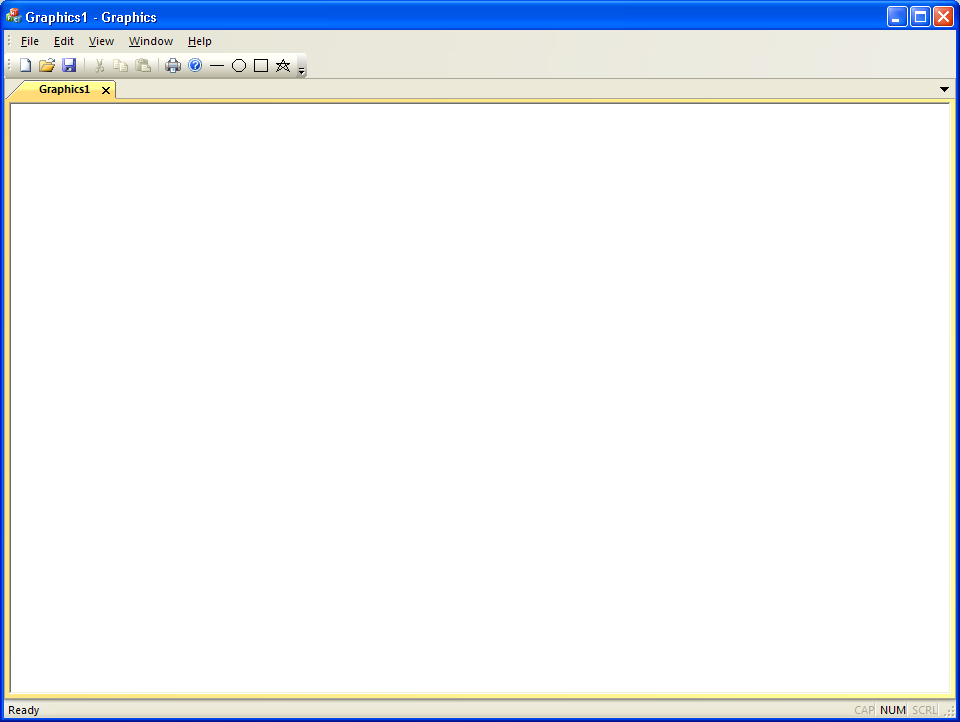
Invalidate();

}

# Изпълнение

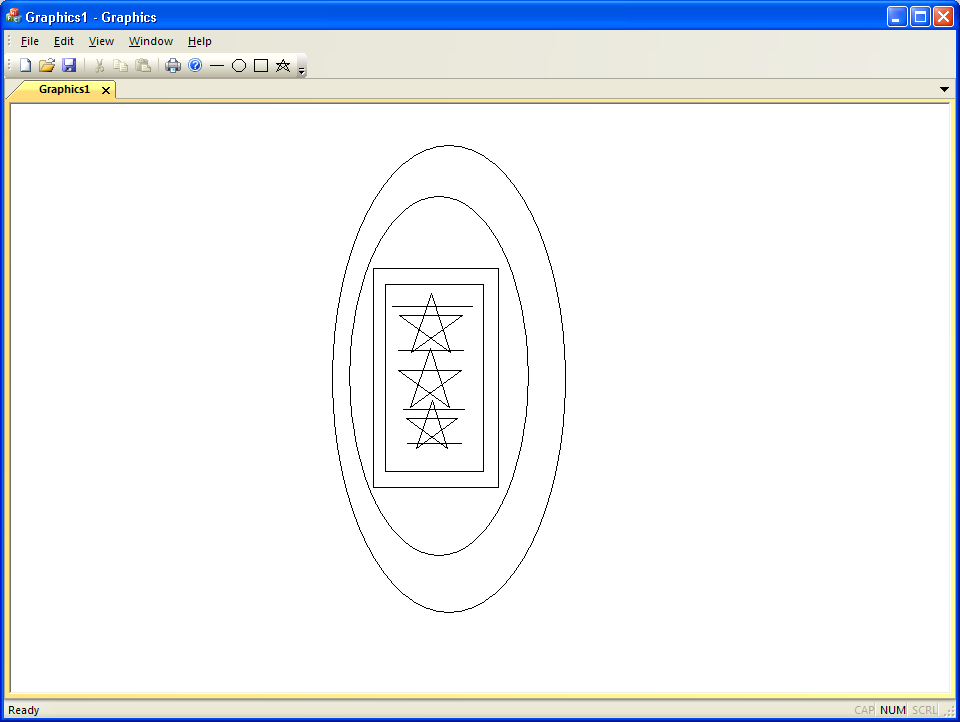
**Тест 1**

Стартиране на програмата за първи път



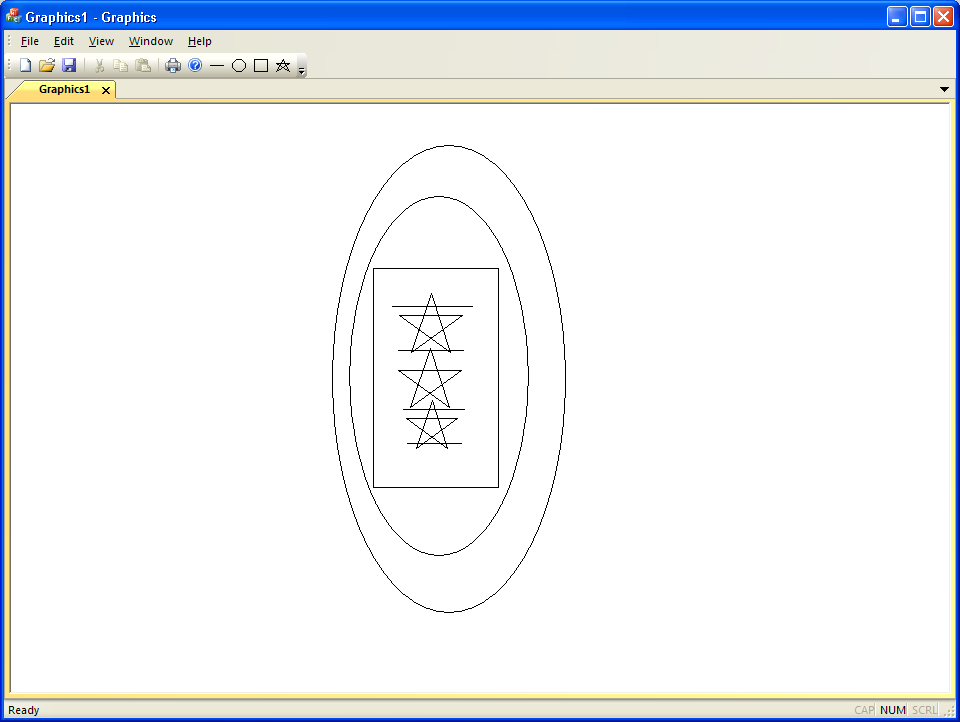
**Тест 2**

Избиране на фигура от toolbar-а и изчертаване



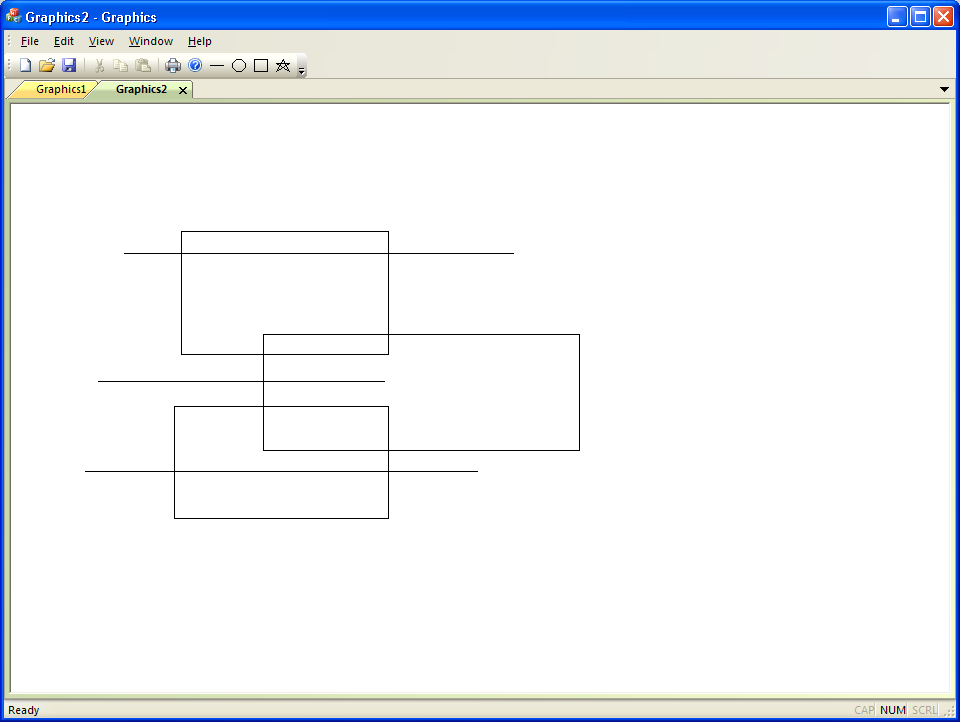
**Тест 3**

Изтриване на фигура с натискане на десен бутон на мишката



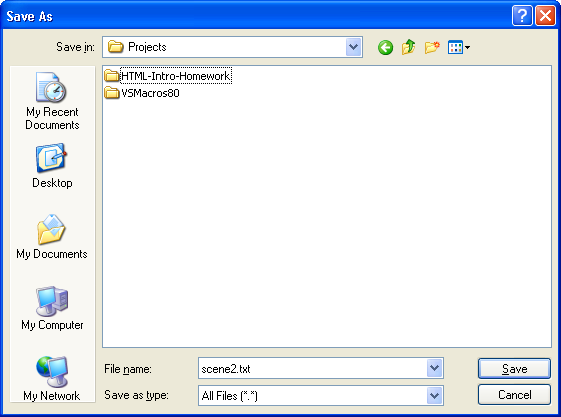
**Тест 4**

Отваряне на нова сцена и изчертаване



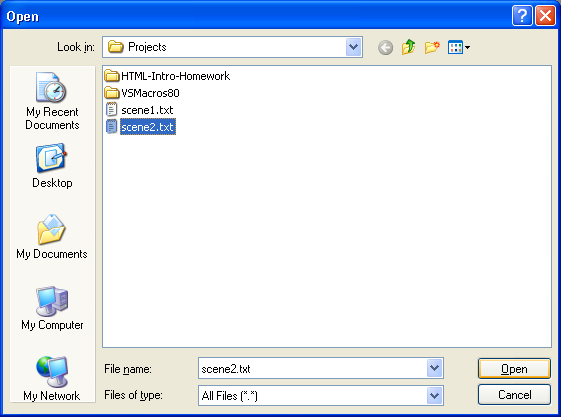
**Тест 5**

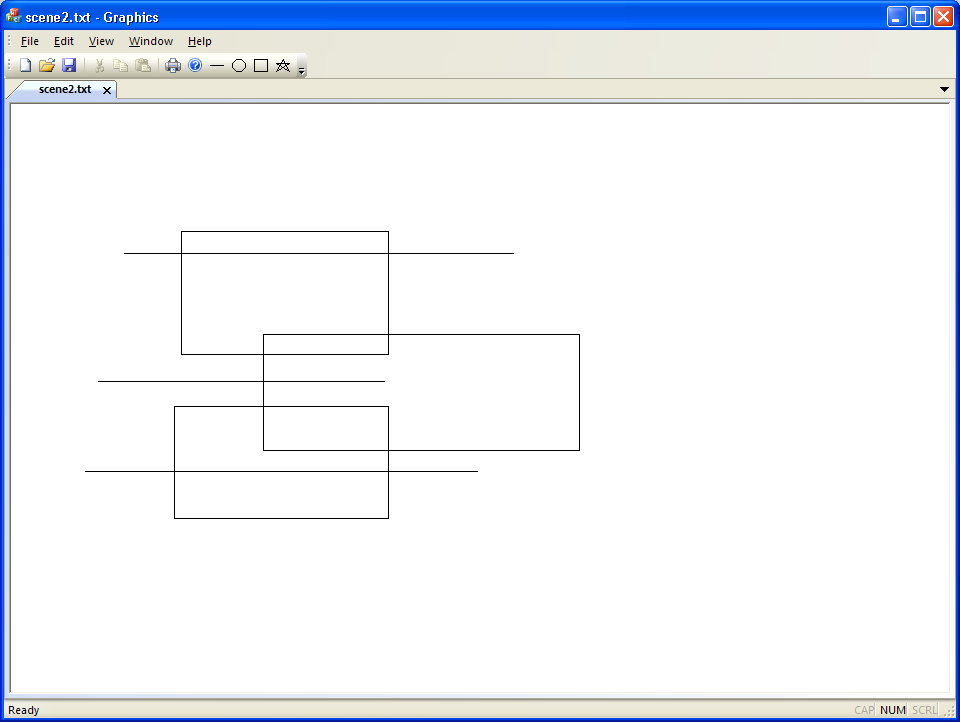
Записване в текстов файл на сцена



**Тест 6**

Затваряне на сцената и отваряне на запазения файл





**Тест 7**

Изтриване на фигура с Ctrl + ляв бутон на мишката

