**Содержание**

[1. Введение 4](#_Toc124363558)

[2. Основная часть 6](#_Toc124363559)

[2.1. Цель работы 6](#_Toc124363560)

[2.2. Задачи 6](#_Toc124363561)

[2.3. Техническая постановка и выбор средств реализации. 6](#_Toc124363562)

[2.4. Реализация. 7](#_Toc124363563)

[2.5. Примеры работы. 10](#_Toc124363564)

[3. Заключение 14](#_Toc124363565)

[4. Список литературы 15](#_Toc124363566)

[5. Приложение 16](#_Toc124363567)

1. Введение

С развитием компьютеров и информационных технологий появилось множество механизмов визуализации информации и построения графических объектов. Под видами компьютерной графики подразумевается способ хранения и отображения изображения на плоскости монитора. Графический объект – это любой нарисованный или вставленный объект, который   
можно редактировать и форматировать с помощью инструментов рисования/редактирования. В зависимости от видов компьютерной графики под этим термином понимаются, как пиксели или спрайты (в растровой графике), так и векторные объекты, такие как круг, квадрат, линия, кривая и т.д. (в векторной графике). Существует специальная область информатики, изучающая методы и средства создания и обработки изображений с помощью программно-аппаратных вычислительных комплексов, - компьютерная графика. Компьютерная графика представляет собой одну из современных технологий создания различных изображений с помощью аппаратных и программных средств компьютера [1]. В зависимости от способа формирования изображений компьютерную графику принято подразделять на растровую и векторную. Говоря о графических редакторах, также необходимо отметить, что они, как и цифровые изображения, подразделяются на векторные и растровые. Такие операции группировка, копирование и вставка являются важными в редакторах такого типа, поскольку часто при построении изображений важно сохранять пропорции и иные значения, по которым отстраивается примитив, чтобы увеличить скорость и простоту работы в редакторе.

В рамках данной курсовой работы были реализованы применительно к разработанному векторному редактору операции копирования и вставки объектов.

2. Основная часть

2.1. Цель работы

Реализовать применительно к векторному графическому редактору операции копирования и вставки. Для этого нужно решить ряд задач.

2.2. Задачи

1. Спроектировать корректное копирование выделенных объектов;

2. Реализовать вставку скопированных объектов с дальнейшей возможностью их редактирования

3. Протестировать вставленные объекты.

2.3. Техническая постановка и выбор средств реализации.

Векторный графический редактор реализован с использованием языка программирования C# и платформы пользовательского интерфейса Windows Forms. Для реализации операций копирования и вставки были использованы те же самые инструменты.

Операция «Копировать» должна корректно сохранять координаты, фигуры и их расположение относительно друг друга. Копирование должно происходить без утраты возможности редактирования фигур.

Операция «Вставить» должна полноценно добавлять на поле отрисовки скопированные ранее фигуры, с небольшим смещением от исходных фигур.

Помимо этого, описанные выше операции должны корректно работать с объектами типа «группа».

Реализация копирования и вставки примитивов должно происходить следующим образом: пользователь выбирает объекты, которые желает скопировать, путем нажатия левой кнопки мыши на них (с зажатой клавишей Shift, если их несколько), а затем должен нажать на кнопку «Скопировать» и «Вставить» для отображения.

2.4. Реализация.

Для начала необходимо создать класс CopyController, который включает в себя реализацию методов «Сopy» и «Paste». Данный класс должен являться наследником класса Model, реализованного в течение семестра. Класс CopyController отвечает за создание копии выделенных объектов (с сохранением их Frame и PropList), чтобы при вставке корректно отобразить данные примитивы.

Вставка скопированных фигур происходит с небольшим смещением фрейма, чтобы было удобнее отследить корректную работу операций.

Диаграмма состояний

Пользователь выбрал примитив(ы) и нажал на кнопку «Копировать»

CopyController

Пользователь нажал на кнопку «Вставить»

CopyController

Отображение

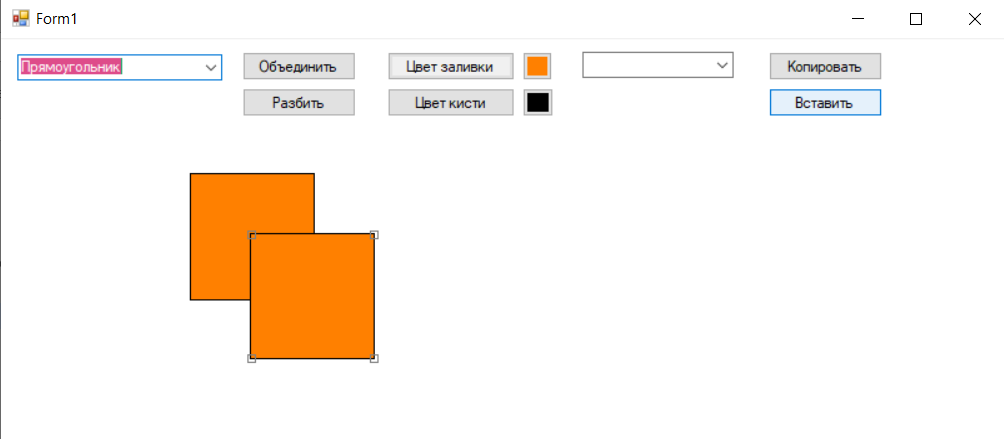
2.5. Примеры работы.

Примеры копирования и вставки:

1. Копирование оранжевого прямоугольника.

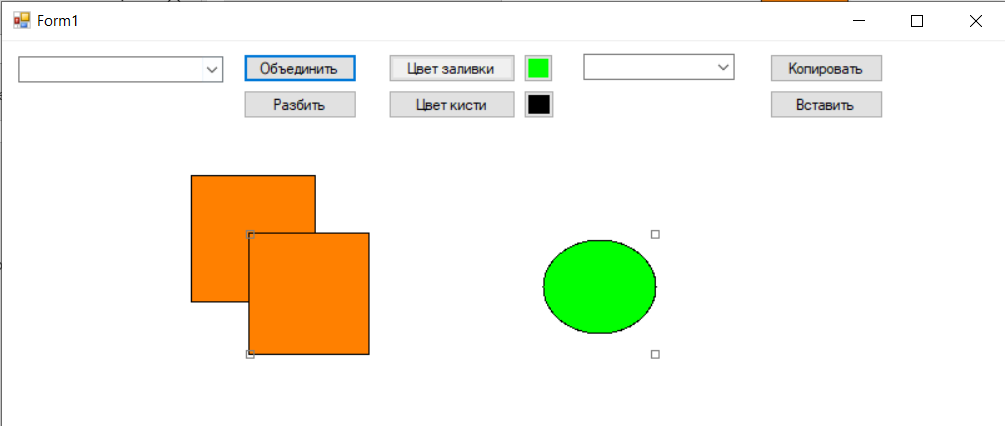


1. Вставка оранжевого прямоугольника.

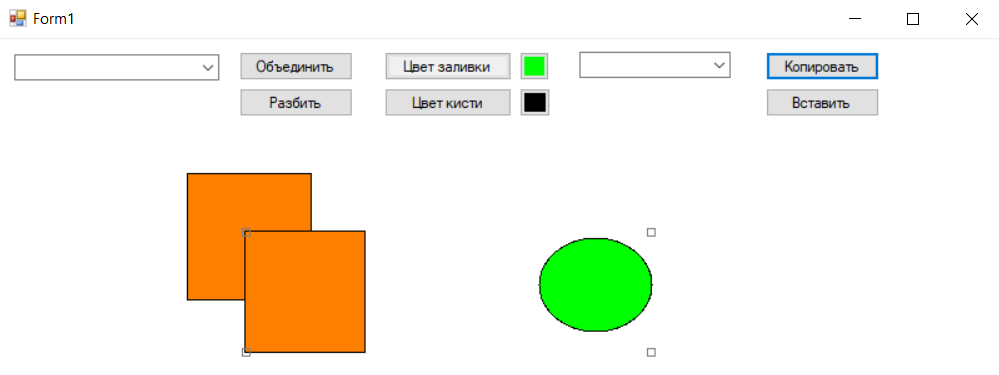


Чтобы скопировать несколько объектов, было принято решение реализовать данную операцию с помощью «Объединение».

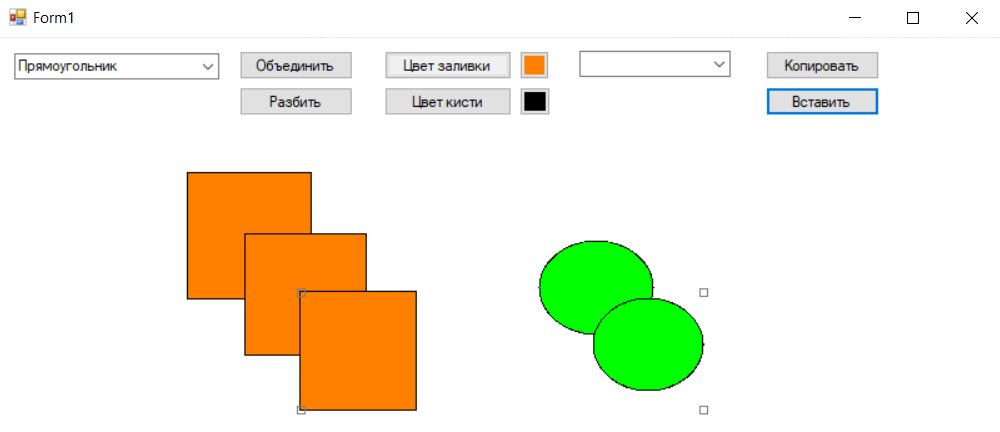
1. Группировка прямоугольника и эллипса.



1. Копирование группы, состоящей из прямоугольника и эллипса.



1. Вставка группы, состоящей из прямоугольника и эллипса.



Как видно, копирование и вставка реализованы успешно.

3. Заключение

В ходе курсовой работы реализовано следующее: реализованы операции копирования и вставки примитивов и групп примитивов. Копирование и вставка корректно осуществляется. В целом, операции полностью выполняют возложенный функционал.

4. Список литературы

1. Мишутина И.Е., Кудрина М.А. Разработка векторного графического редактора // Материалы Международной конференции и молодежной школы. Федеральное государственное автономное образовательное учреждение высшего образования «Самарский государственный аэрокосмический университет имени академика С.П. Королева (Национальный исследовательский университет)». – 2015 – C.329-332.

5. Приложение

**Класс CopyController:**

class CoppyController

{

private Factory factory;

private Scene scene;

private GrPropChannel grPropChannel;

private GrController grController;

public IFactory Factory { get { return factory as IFactory; } }

public IGrProperties GrProperties { get { return grPropChannel as IGrProperties; } }

public IGrController GrController { get { return grController as IGrController; } }

public GraphSystem graphSystem;

public Store store;

public Model()

{

store= new Store();

scene = new Scene(graphSystem, store);

factory = new Factory(store, scene.Selections);

grPropChannel = new GrPropChannel(factory);

grController = new GrController(scene);

}

}

public GraphItem CreateItemCopy(int x, int y, Figure figure)

{

Frame frame = figure.Frame;

PropList propList = figure.PropList;

if (Object.ReferenceEquals(figure.GetType(), new Line(frame, propList).GetType()))

{

var line = new Line(frame, propList);

return line;

} else

{

var rectangle = new GraphicItems.Rectangle(frame, propList);

return rectangle;

}

return null;

}

public GraphItem CreateItem(int x, int y)

{

Frame frame;

PropList propList;

switch (itemType)

{

case ItemType.Line:

frame = new Frame(x, y, x+50, y+50);

propList = new PropList();

propList.Add(ContourProp.Copy());

var line = new Line(frame, propList);

return line;

case ItemType.Rectangle:

frame = new Frame(x, y, x+50, y+50);

propList = new PropList();

propList.Add(ContourProp.Copy());

propList.Add(FillProp.Copy());

var rectangle = new GraphicItems.Rectangle(frame, propList);

return rectangle;

}

return null;

}

public void CreateAndGrabItemCopy(int x, int y, Figure it)

{

GraphItem item = CreateItemCopy(x, y, it);

if (item != null)

{

Store.Add(item);

SelectionsController.SelectAndGrab(item, x, y);

}

}

public void CreateAndGrabItem(int x, int y)

{

GraphItem item = CreateItem(x, y);

if (item != null)

{

Store.Add(item);

SelectionsController.SelectAndGrab(item, x, y);

}

}

}

**Класс Model**

namespace TeorProg2.Model

{

class Model : IModel

{

private Factory factory;

private Scene scene;

private GrPropChannel grPropChannel;

private GrController grController;

public IFactory Factory { get { return factory as IFactory; } }

public IGrProperties GrProperties { get { return grPropChannel as IGrProperties; } }

public IGrController GrController { get { return grController as IGrController; } }

public GraphSystem graphSystem;

public Store store;

public Model()

{

store= new Store();

scene = new Scene(graphSystem, store);

factory = new Factory(store, scene.Selections);

grPropChannel = new GrPropChannel(factory);

grController = new GrController(scene);

}

}

class Factory:IFactory

{

Store Store;

private SelectionsController SelectionsController;

public ISelection Selection { get { return SelectionsController as ISelection; } }

public ItemType itemType { get; set; }

public ContourProp ContourProp { get; set; } = new ContourProp(Color.Black, 2);

public FillProp FillProp { get; set; } = new FillProp(Color.Empty);

public Factory(Store store, SelectionStore selections)

{

Store= store;

SelectionsController= new SelectionsController(selections, store, this);

}

public GraphItem CreateItem(int x, int y)

{

Frame frame;

PropList propList;

switch (itemType)

{

case ItemType.Line:

frame = new Frame(x, y, x+50, y+50);

propList = new PropList();

propList.Add(ContourProp.Copy());

var line = new Line(frame, propList);

return line;

case ItemType.Rectangle:

frame = new Frame(x, y, x+50, y+50);

propList = new PropList();

propList.Add(ContourProp.Copy());

propList.Add(FillProp.Copy());

var rectangle = new GraphicItems.Rectangle(frame, propList);

return rectangle;

case ItemType.Ellipse:

frame = new Frame(x, y, x + 50, y + 50);

propList = new PropList();

propList.Add(ContourProp.Copy());

propList.Add(FillProp.Copy());

var ellipse = new GraphicItems.Ellipse(frame, propList);

return ellipse;

}

return null;

}

public void CreateAndGrabItem(int x, int y)

{

GraphItem item = CreateItem(x, y);

if (item != null)

{

Store.Add(item);

SelectionsController.SelectAndGrab(item, x, y);

}

}

public Group NewGroup(List<GraphItem> items)

{

Group group = new Group(items);

for (int i = 0; i < items.Count; i++)

{

Store.Remove(items[i]);

}

Store.Add(group);

return group;

}

}

class Store: List<GraphicItems.GraphItem>

{

public void Delete(List<GraphItem> items)

{

foreach (var item in items)

{

this.Remove(item);

}

}

public GraphItem GetItem(int x, int y)

{

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

{

if (this[i].InBody(x, y))

{

return this[i];

}

}

return null;

}

}

class GrController:IGrController

{

public Scene scene;

public GraphSystem graphSystem;

public GrController(Scene scene)

{

this.scene = scene;

}

public void SetPort(Graphics graphics, int width, int height)

{

graphSystem = new GraphSystem(graphics);

scene.GraphSystem = graphSystem;

}

public void Repaint()

{

scene.Repaint();

}

}

class Scene

{

public Store Store;

public GraphSystem GraphSystem { get; set; }

public SelectionStore Selections = new SelectionStore();

public Scene(GraphSystem GraphSystem, Store store)

{

Store= store;

this.GraphSystem = GraphSystem;

}

public void Repaint()

{

GraphSystem.Clear();

foreach (var item in Store)

{

item.Draw(GraphSystem);

}

Selections.Draw(GraphSystem);

}

}

class GrPropChannel:IGrProperties

{

public IContourProps ContourProps { get; }

public IFillProps FillProps { get;}

Factory factory;

public GrPropChannel(Factory factory)

{

this.factory = factory;

this.ContourProps = factory.ContourProp;

this.FillProps = factory.FillProp;

}

public void ApplyProps()

{

factory.ContourProp.Color = ContourProps.Color;

factory.ContourProp.Width = ContourProps.Width;

factory.FillProp.Color = FillProps.Color;

}

}

}