Paintdotnet

Proiect Ingineria Programării 2025

Documentație

Grupa: 1311A
Studenti: Asaftei Rareș-Ștefan, Grigoraș Robert-Constantin, Slabu George-Cristian, Vezeteu Alexandru

---

Software Requirements Specification

for

Paintdotnet

Version 1.0 approved

Prepared by Asaftei Rareș-Ștefan, Grigoraș Robert-Constantin, Slabu George-Cristian and Vezeteu Alexandru

Facultatea de Automatica si Calculatoare, Universitatea "Gheorghe Asachi", Iași

May, 2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Initial Draft	May 2025	Initial document creation	1.0

1. Introduction

1.1 Purpose

This document outlines the software requirements for PaintDotnet v1.0 a simple drawing application which is built using C# and .NET Framework Windows Forms. It specifies all of the functional and non-functional requirements.

The scope of the application is to provide users with basic features of drawing functionalities like pencil drawing, creating shapes, selecting colors, as well as saving or loading images.

This SRS refers to the whole system without restricting the scope to any particular module or subsystem.

1.2 Document Conventions

This document follows IEEE standards for Software Requirements Specifications. The following conventions are used:

- **Bold text** indicates important terms or section headers
- *Italic text* indicates emphasis or references to external documents
- Requirements are numbered sequentially (REQ-1, REQ-2, etc.)
- Priority levels are defined as High, Medium, or Low
- “TBD" indicates information to be determined later

1.3 Intended Audience and Reading Suggestions

This document is intended for:

- **Developers**: Focus on sections 3, 4, and 5 for technical implementation details
- **Project Managers**: Review sections 1, 2, and 5 for project scope and constraints
- **Testers**: Concentrate on sections 4 and 5 for functional and non-functional requirements
- **Users**: Section 2 provides an overview of application capabilities

It is recommended to read the document sequentially, starting with the introduction and overall description before proceeding to detailed requirements.

1.4 Product Scope

PaintDotnet is a desktop drawing application designed to provide basic image editing and creation capabilities. The software aims to:

- Enable users to create digital artwork using various drawing tools

- Provide an intuitive interface for casual and educational use
- Support common image file formats for import and export
- Implement reliable undo/redo functionality using design patterns
- Serve as an educational tool for learning basic digital art concepts
- Provide an alternative to Microsoft Paint

The application is designed for Windows environments and targets users who need simple drawing capabilities without the complexity of professional image editing software.

1.5 References

- IEEE Standard 830-1998 for Software Requirements Specifications
- Microsoft .NET Framework Documentation
- Windows Forms Programming Guidelines
- C# Programming Language Specification

2. Overall Description

2.1 Product Perspective

PaintDotnet is a standalone desktop application developed as a new, self-contained product. It is not part of a larger system but interacts with the Windows operating system for file operations and display rendering.

The application consists of:

- Main drawing canvas
- Tool palette
- Color selection interface
- Menu system for file operations

2.2 Product Functions

The major functions that PaintDotnet must perform include:

- **Drawing Operations**: Pencil tool for freehand drawing
- **Shape Creation**: Rectangle, circle, and line drawing tools
- **Color Management**: Color picker and palette for selecting drawing colors
- **File Management**: New, open, save operations for image files
- **Edit Operations**: Undo functionality with history management
- **Canvas Management**: Clear canvas

2.3 User Classes and Characteristics

Primary Users - Casual Artists and Students

- Frequency of use: Occasional to regular
- Technical expertise: Basic computer skills
- Primary functions: Drawing, basic shape creation, saving work
- Importance: High priority for design decisions

Secondary Users – Educators

- Frequency of use: Regular during teaching activities
- Technical expertise: Moderate computer skills
- Primary functions: Demonstration of digital art concepts
- Importance: Medium priority

2.4 Operating Environment

Hardware Platform:

- Windows-compatible PC
- Minimum 2GB RAM
- 100MB available disk space
- Mouse or touchpad input device
- Display resolution minimum 1024x768

Software Environment:

- Windows 10 or later
- .NET Framework 4.7.2 or later
- Windows Forms runtime support

2.5 Design and Implementation Constraints

- **Technology Constraints**: Must use C# and Windows Forms
- **Pattern Implementation**: Must implement Memento pattern for undo functionality
- **Memory Limitations**: Must manage memory efficiently for image operations
- **Performance Requirements**: Real-time drawing response required
- **File Format Support**: Limited to common image formats (BMP, JPG, PNG)

2.6 User Documentation

The following documentation will be provided:

- User manual with basic operation instructions
- Quick start guide for common tasks
- Help tooltips within the application

2.7 Assumptions and Dependencies

Assumptions:

- Users have basic Windows operation knowledge
- .NET Framework is available on target systems
- Standard mouse/keyboard input methods are used

Dependencies:

- Windows operating system APIs for file operations
- .NET Framework graphics libraries
- System.Drawing namespace for image manipulation

3. External Interface Requirements

3.1 User Interfaces

Main Application Window:

- Central drawing canvas area
- Tool palette on the right side with drawing tools
- Color selection area below tools and canvas
- Menu bar with File, Edit, and Help menus

Dialog Interfaces:

- File Open/Save dialogs using standard Windows file dialogs
- Color picker dialog for advanced color selection
- About dialog with application information

Interaction Requirements:

- Mouse-driven drawing operations

3.2 Hardware Interfaces

Input Devices:

- Mouse: Primary drawing input, must support click, drag, and release events
- Keyboard: Secondary input for shortcuts and text entry in dialogs
- Display: Must support minimum 16-bit color depth

Storage Devices:

- Local disk access for file save/load operations

3.3 Software Interfaces

Operating System Interface:

- Windows API for file system operations
- Windows GDI+ for graphics rendering
- Windows Forms for user interface components

.NET Framework Dependencies:

- System.Drawing for image manipulation
- System.Windows.Forms for UI components
- System.IO for file operations
- System.Collections.Generic for data structures (Stack for Memento pattern)

3.4 Communications Interfaces

The application operates as a standalone desktop program and does not require network communication interfaces. All operations are performed locally on the user's machine.

4. System Features

4.1 Drawing Tools

4.1.1 Description and Priority

Provides basic drawing capabilities including pencil tool for freehand drawing.
Priority: High

4.1.2 Stimulus/Response Sequences

- User selects pencil tool from palette
- System highlights selected tool
- User clicks and drags on canvas
- System draws continuous line following mouse movement
- User releases mouse button
- System completes drawing operation and saves state for undo

4.1.3 Functional Requirements

REQ-1: The system shall provide a pencil tool that draws continuous lines following mouse movement.

REQ-2: The drawing tool shall use the currently selected color for all drawing operations.

REQ-3: The system shall provide real-time visual feedback during drawing operations.

4.2 Shape Creation

4.2.1 Description and Priority
Enables users to create geometric shapes including rectangles, circles, and lines. **Priority: High**

4.2.2 Stimulus/Response Sequences

1. User selects shape tool (rectangle, circle, or line)
2. System highlights selected tool
3. User clicks starting point on canvas
4. User drags to define shape dimensions
5. System shows preview of shape during drag
6. User releases mouse button
7. System renders final shape and saves state

4.2.3 Functional Requirements

REQ-4: The system shall provide rectangle drawing tool with click-and-drag functionality.
REQ-5: The system shall provide circle drawing tool with center-radius or corner-to-corner definition.
REQ-6: The system shall provide line drawing tool for straight lines between two points.
REQ-7: The system shall show preview of shapes during creation process.

4.3 Color Selection

4.3.1 Description and Priority
Allows users to select colors for drawing operations. **Priority: High**

4.3.2 Stimulus/Response Sequences

1. User clicks on color palette or color picker button
2. System displays color selection interface
3. User selects desired color
4. System updates current color indicator
5. Subsequent drawing operations use selected color

4.3.3 Functional Requirements

REQ-8: The system shall provide a basic color palette with 28 predefined colors.
REQ-9: The system shall provide access to advanced color picker dialog.
REQ-10: The system shall display the currently selected color prominently in the interface.
REQ-11: The system shall apply selected color to all subsequent drawing operations.

4.4 File Operations

4.4.1 Description and Priority

Handles saving and loading of image files. **Priority: High**

4.4.2 Stimulus/Response Sequences

1. User selects File menu option (New, Open, Save)
2. System displays appropriate dialog or performs action
3. For file operations, user navigates to desired location
4. User confirms action
5. System performs file operation and provides feedback

4.4.3 Functional Requirements

REQ-12: The system shall support creating new blank canvas.
REQ-13: The system shall support opening existing image files in BMP, JPG, and PNG formats.
REQ-14: The system shall support saving canvas content to BMP, JPG, and PNG formats.
REQ-15: The system shall provide Save As functionality for specifying new file names and locations.
REQ-16: The system shall warn users about unsaved changes before closing or creating new files.

4.5 Undo Operation

4.5.1 Description and Priority

Implements undo functionality using the Memento design pattern. **Priority: High**

4.5.2 Stimulus/Response Sequences

1. User performs drawing operation
2. System automatically saves current state using Memento pattern
3. User triggers undo operation (Edit menu)
4. System restores previous state from history
5. Canvas is updated to show previous state

4.5.3 Functional Requirements

REQ-17: The system shall implement Memento pattern for state management as shown in provided code.
REQ-18: The system shall automatically save canvas state after each significant user action.
REQ-19: The system shall support undo operation to revert to previous states.
REQ-20: The system shall maintain undo history for at least 20 operations.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

PERF-1: Drawing operations must provide real-time response with maximum 50ms delay between mouse movement and visual update.

PERF-2: File save operations for typical images (up to 2MB) must complete within 5 seconds.

PERF-3: Application startup time must not exceed 3 seconds on minimum system requirements.

PERF-4: Undo operations must complete within 100ms regardless of canvas complexity.

5.2 Safety Requirements

SAFE-1: The system shall automatically save backup files to prevent data loss during unexpected shutdowns.

SAFE-2: The system shall validate file formats before attempting to open files to prevent crashes.

SAFE-3: Memory usage shall be monitored to prevent system instability from excessive image history.

5.3 Security Requirements

SEC-1: The system shall only access files explicitly selected by the user through standard file dialogs.

SEC-2: The system shall not automatically execute or interpret file content beyond image data.

SEC-3: File operations shall respect Windows file system permissions and security settings.

5.4 Software Quality Attributes

Usability:

- Interface elements shall follow Windows UI conventions
- Common operations shall be accessible through both menus and shortcuts
- Visual feedback shall be provided for all user actions

Reliability:

- Application shall handle common error conditions gracefully
- File operations shall include error checking and user notification
- Memory management shall prevent crashes from resource exhaustion

Maintainability:

- Code shall follow object-oriented design principles
- Design patterns (Memento) shall be properly implemented
- Code shall include appropriate comments and documentation

Portability:

- Application shall run on Windows 10 and later versions
- Dependencies shall be limited to standard .NET Framework components

5.5 Business Rules

BR-1: Only one drawing tool may be active at a time.

BR-2: Color selection applies to all subsequent drawing operations until changed.

BR-3: Undo history is cleared when a new file is created or opened.

BR-4: File save operations preserve image quality according to selected format capabilities.

6. Other Requirements

Legal Requirements:

- The application shall respect intellectual property rights for any included resources
- Usage shall comply with educational institution policies

Internationalization:

- Interface design should accommodate future localization

Installation Requirements:

- Installation shall not require administrator privileges beyond initial setup

Appendix A: Glossary

Canvas: The main drawing area where users create and edit images

Memento Pattern: A behavioral design pattern that provides the ability to restore an object to its previous state (undo functionality)

Originator: The object whose state needs to be saved (the canvas/image in this context)

Caretaker: The object responsible for keeping track of multiple states (undo history manager)

GDI+: Graphics Device Interface Plus, Microsoft's API for 2D graphics

Appendix B: Analysis Models

The application architecture follows these key patterns:

Memento Pattern Implementation:
- Originator class manages the current canvas state
- Memento class stores canvas snapshots
- Caretaker class manages undo history using a Stack data structure

Model-View-Controller Architecture:
- Model: Image data and application state
- View: Windows Forms interface components
- Controller: Event handlers and business logic

Appendix C: To Be Determined List

- **TBD-1:** Maximum canvas size limitations
- **TBD-2:** Additional file format support (GIF, TIFF)
- **TBD-3:** Advanced drawing tools (spray brush, eraser)
- **TBD-4:** Redo functionality implementation
- **TBD-5:** Image filters and effects
- **TBD-6:** Print functionality requirements
- **TBD-7:** Clipboard operations (copy/paste)

Diagrame UML:

Diagrama de clase

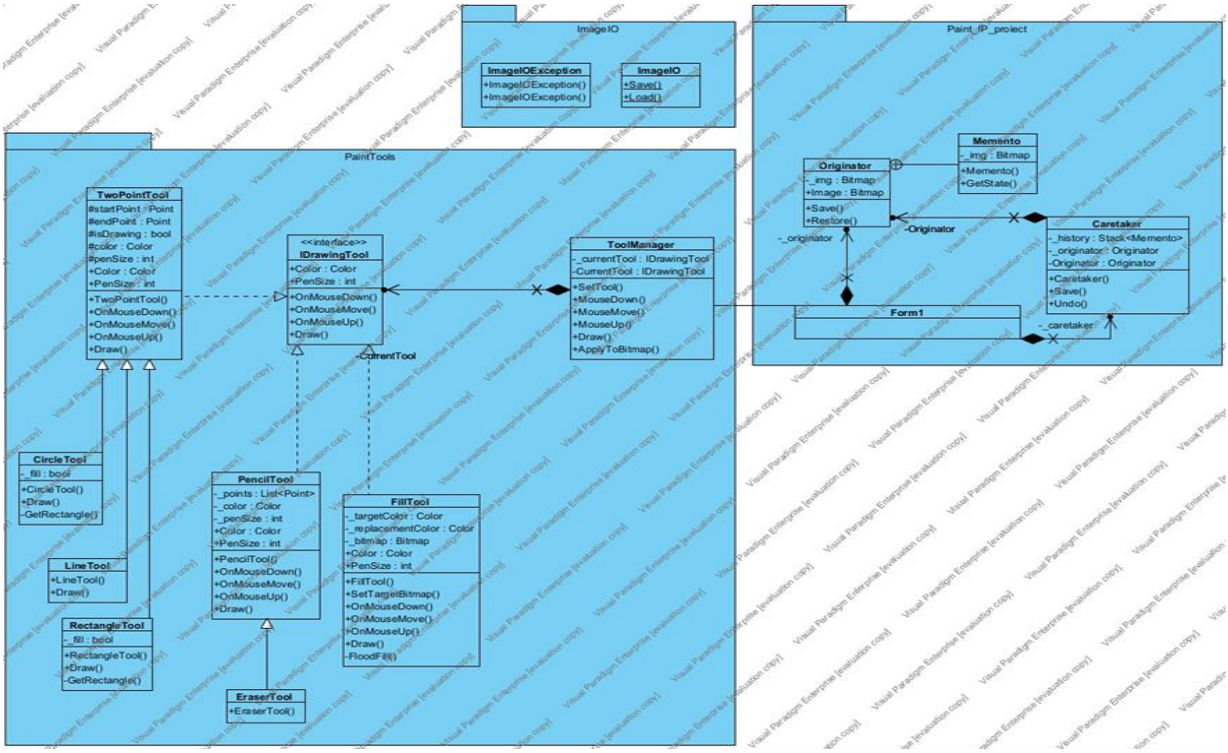


Diagrama de secvență:

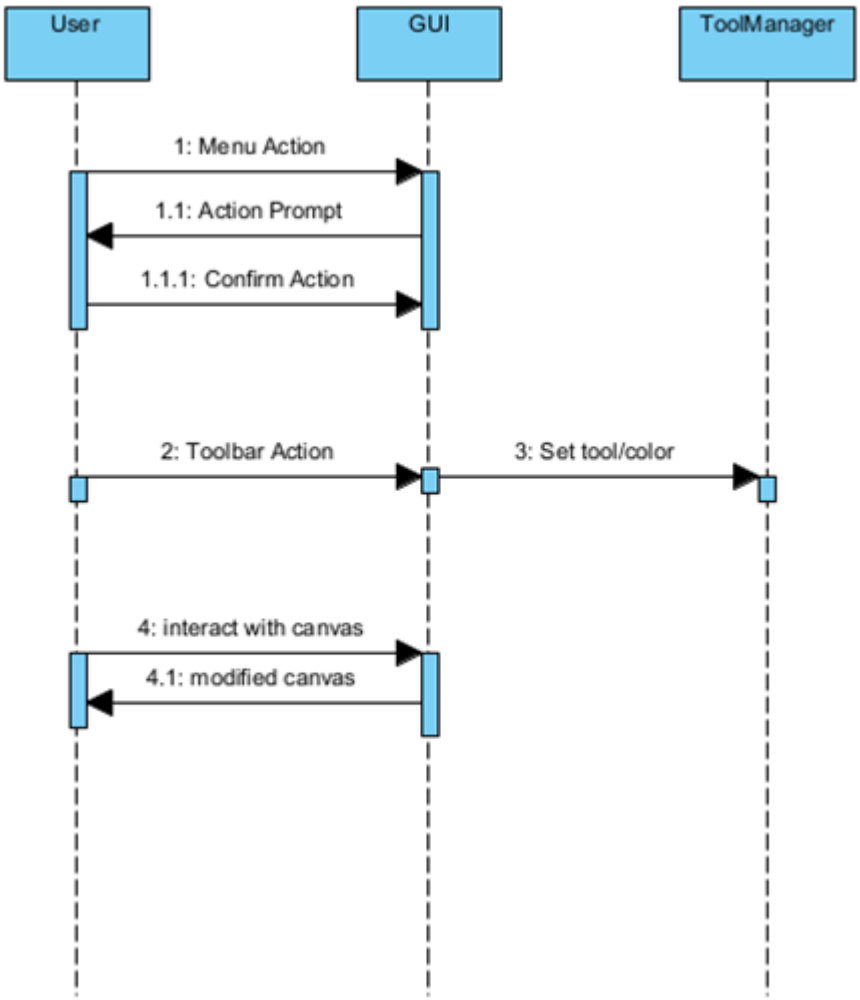
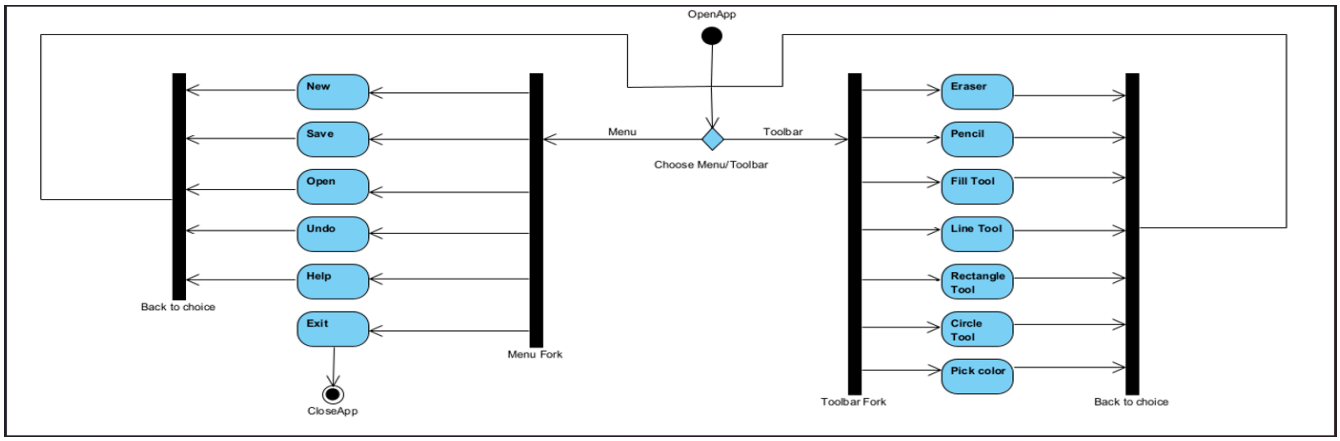
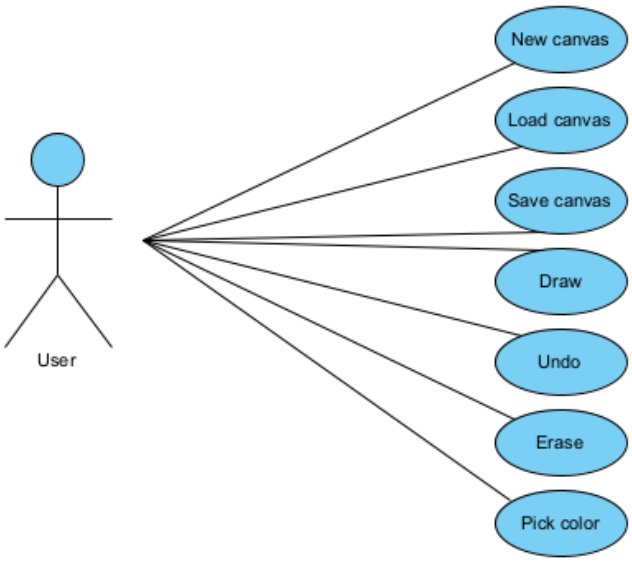


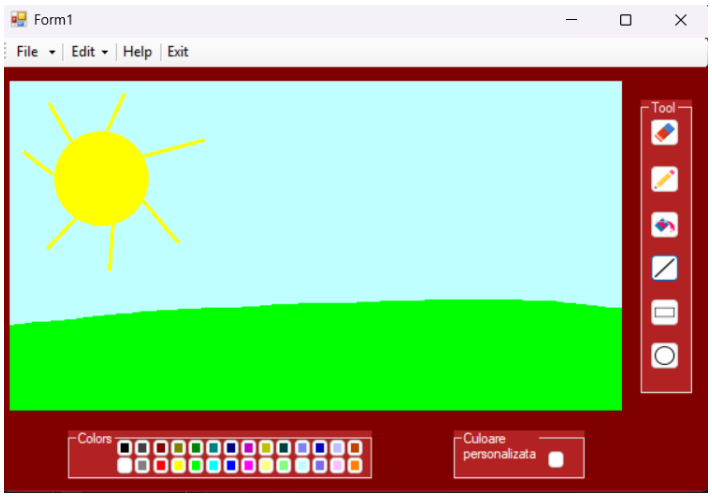
Diagrama de activitate:

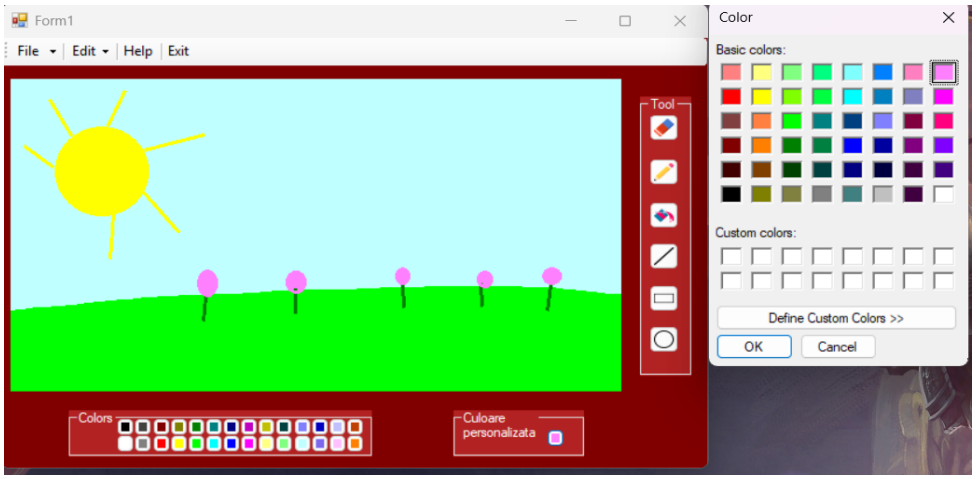


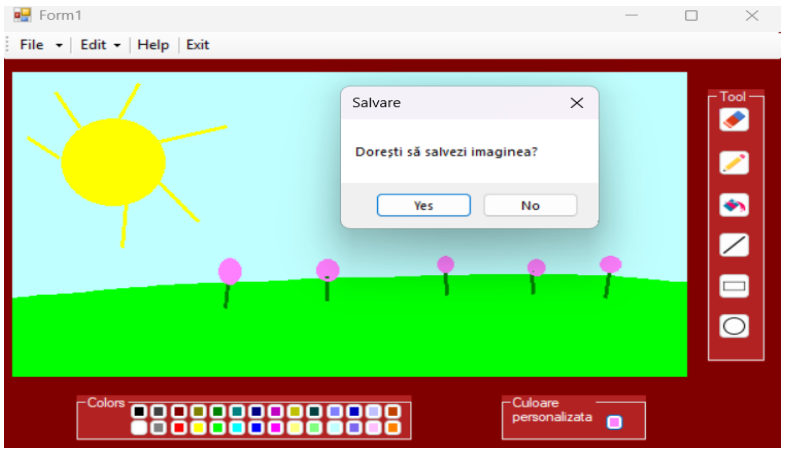
Diagramă de cazuri de utilizare

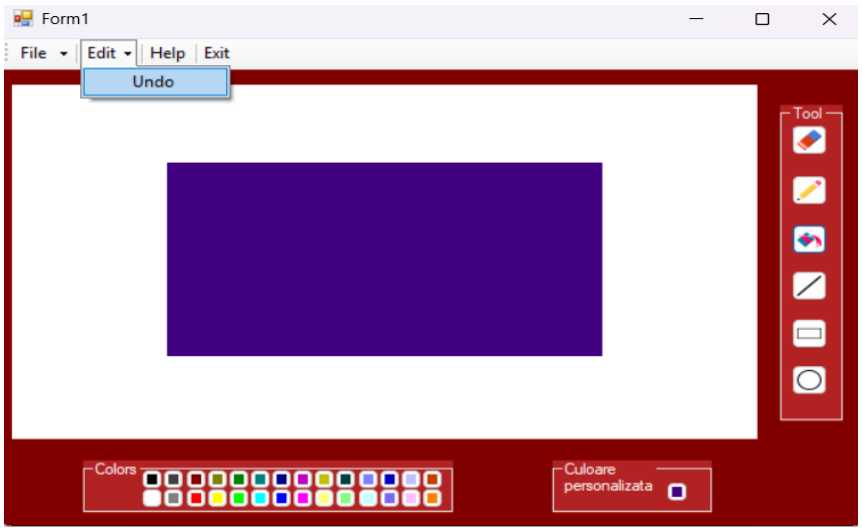


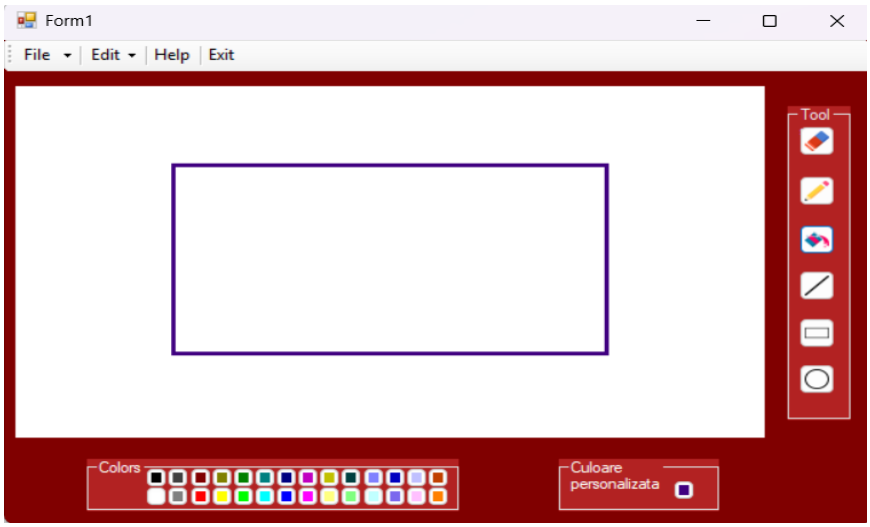
Cum arată programul în execuție:











Părți semnificative din codul sursă:

Caretaker.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
namespace Paint_IP_proiect
{
```

```
//Clasa pentru sablonul Originator-produce obiecte de tip Memento
class Originator
{


    public class Memento
    {
        private Bitmap _img;
        public Memento(Bitmap img)
        {
            _img = new Bitmap(img);
        }

        public Bitmap GetState()
        {
            return _img;
        }

    }

    private Bitmap _img;
    public Memento Save()
    {
        return new Memento(_img);
    }
    public Bitmap Image
    {
        get => _img;
        set=>_img= value;
    }
    public void Restore(Memento memento)
    {
        _img = new Bitmap(memento.GetState());
    }
}

//clasa pentru sablonul Memento, responsabila pentru salvat/istoric
class Caretaker
{
    private Stack<Originator.Memento> _history;
    private Originator _originator;

    public Originator Originator
    {
        get => _originator;
        set => _originator = value;
    }

    public Caretaker(Originator originator)
    {
        _originator = originator;
        _history = new Stack<Originator.Memento>();
    }
```

```
        public void Save()
        {
            _history.Push(_originator.Save());
        }


        public void Undo()
        {
            if (_history.Count > 0)
            {
                Originator.Memento m = _history.Pop();
                _originator.Restore(m);
            }
        }
    }

}
```

Form1.cs (funcții exemple)

```
  private void pictureBoxCanvas_MouseDown(object sender, MouseEventArgs e)
  {
      _caretaker.Save();
      _isDrawing = true;
      _toolManager.MouseDown(e.Location);
      pictureBoxCanvas.Invalidate();
  }

  private void pictureBoxCanvas_MouseMove(object sender, MouseEventArgs e)
  {
      if (e.Button == MouseButtons.Left)
      {
          _toolManager.MouseMove(e.Location);
          pictureBoxCanvas.Invalidate();
      }
  }

  private void pictureBoxCanvas_MouseUp(object sender, MouseEventArgs e)
  {
      _toolManager.MouseUp(e.Location);
      _toolManager.ApplyToBitmap((Bitmap)pictureBoxCanvas.Image);


      _originator.Image = new Bitmap((Bitmap)pictureBoxCanvas.Image);
      _isDrawing = false;
      pictureBoxCanvas.Invalidate();


  }

  private void pictureBoxCanvas_Paint(object sender, PaintEventArgs e)
  {
      if (pictureBoxCanvas.Image != null)
      {
```

```
            e.Graphics.DrawImage(pictureBoxCanvas.Image, Point.Empty);
        }
        if (_isDrawing)
        {
            _toolManager.Draw(e.Graphics);
        }
    }

private void buttonCuloareVerde_Click(object sender, EventArgs e)
 {
     _currentColor = Color.Green;
     UpdateToolColor();
 }



   private void butonExit_Click(object sender, EventArgs e)
   {
       DialogResult result = MessageBox.Show("Dorești să salvezi imaginea?",
"Salvare", MessageBoxButtons.YesNo);

       if (result == DialogResult.Yes)
       {
           try
           {
               ImageIO.ImageIO.Save(pictureBoxCanvas.Image);
           }
           catch (ImageIOException)
           {
               //nimic, nu avem ce salva
           }

       }
       Application.Exit();
   }

   private void buttonBucketFill_Click(object sender, EventArgs e)
   {
       var fillTool = new FillTool(_currentColor);
       fillTool.SetTargetBitmap((Bitmap)pictureBoxCanvas.Image);
       _toolManager.SetTool(fillTool);
   }
```