5.3 Drawing And Saving

This is a resubmission, I made sure to include all files in the PDF this time:

Program File

|  |
| --- |
| using System;  using System.Runtime.InteropServices;  using SplashKitSDK;  using System.IO;  namespace ShaperDrawer  {  public class Program  {  private enum ShapeKind  {  Rectangle,  Circle,  Line  }  public static void Main()  {  Window window = new Window("Shape Drawer", 800, 600);  Drawing myDrawing = new Drawing();  ShapeKind kindToAdd = ShapeKind.Circle;  do  {  SplashKit.ProcessEvents();  SplashKit.ClearScreen();  if (SplashKit.KeyTyped(KeyCode.RKey))  {  kindToAdd = ShapeKind.Rectangle;  }  if (SplashKit.KeyTyped(KeyCode.CKey))  {  kindToAdd = ShapeKind.Circle;  }  if (SplashKit.KeyTyped(KeyCode.LKey))  {  kindToAdd = ShapeKind.Line;  }  if (SplashKit.MouseClicked(MouseButton.LeftButton))  {  Shape newShape;  switch(kindToAdd)  {  case ShapeKind.Circle:  newShape = new MyCircle();  break;  case ShapeKind.Line:  newShape = new MyLine();  break;  default:  newShape = new MyRectangle();  break;  }  newShape.X = SplashKit.MouseX();  newShape.Y = SplashKit.MouseY();  myDrawing.AddShape(newShape);  }  if (SplashKit.KeyTyped(KeyCode.SpaceKey))  {  myDrawing.Background = SplashKit.RandomColor();  }  if (SplashKit.MouseClicked(MouseButton.RightButton))  {  myDrawing.SelectShapeAt(SplashKit.MousePosition());  }  if (SplashKit.KeyTyped(KeyCode.DeleteKey) || SplashKit.KeyTyped(KeyCode.BackspaceKey))  {  List<Shape> selectedShapes = myDrawing.SelectedShapes;  foreach (Shape s in selectedShapes)  {  myDrawing.RemoveShape(s);  }  }  if (SplashKit.KeyTyped(KeyCode.SKey))  {  string filePath = @"C:\Personal\Computer Science\Sem 2\OOP\OOP GIT\4.1\ShaperDrawer\TestDrawing.txt";  myDrawing.Save(filePath);  }  if (SplashKit.KeyTyped(KeyCode.OKey))  {  try  {  string filePath = @"C:\Personal\Computer Science\Sem 2\OOP\OOP GIT\4.1\ShaperDrawer\TestDrawing.txt";  myDrawing.Load(filePath);  }  catch (Exception e)  {  Console.Error.WriteLine("Error loading file: {0}", e.Message);  }  }  myDrawing.Draw();  SplashKit.RefreshScreen();  } while (!window.CloseRequested);    }  }  } |

Drawing File

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Xml;  using SplashKitSDK;  namespace ShaperDrawer  {  internal class Drawing  {  private readonly List<Shape> \_shapes; //list of shapes  private Color \_background;  public Drawing(Color background)  {  \_shapes = new List<Shape>();  \_background = background;  }  public Color Background  {  get { return \_background; }  set { \_background = value; }  }  public Drawing() : this(Color.White)  {    }  public int ShapeCount()  { return \_shapes.Count; }  public void AddShape(Shape s)  {  \_shapes.Add(s);  }  public bool RemoveShape(Shape s)  {  if (\_shapes.Contains(s))  {  \_shapes.Remove(s);  return true;  }  return false;    }  public void SelectShapeAt(Point2D pt)  {  foreach (Shape s in \_shapes)  {  if (!s.Selected)  {  s.Selected = s.IsAt(pt);  }  else  {  s.Selected = !s.IsAt(pt);  }  }  }  //create a list of shapes that are to be deleted  public List<Shape> SelectedShapes  {  get  {  List<Shape> shapes = new List<Shape>();  foreach (Shape s in \_shapes)  {  if (s.Selected == true)  { shapes.Add(s); }  }  return shapes;  }  }  public void Draw()  {  SplashKit.ClearScreen(\_background); //change color of background to \_background color  foreach (Shape s in \_shapes)  {  s.Draw();  }  }  public void Save(string filename)  {  StreamWriter writer = new StreamWriter(filename);  try  {  writer.WriteColor(Background);  writer.WriteLine(\_shapes.Count);  foreach (Shape s in \_shapes)  {  s.SaveTo(writer);  }  }  finally  {  writer.Close();  }  }  public void Load(string filename)  {  StreamReader reader = new StreamReader(filename);  try  {  Background = reader.readColor();  int count = reader.readInteger();  \_shapes.Clear();  for (int i = 0; i < count; i++)  {  string? kind = reader.ReadLine();  Shape s;  switch (kind)  {  case "Rectangle":  s = new MyRectangle();  break;  case "Circle":  s = new MyCircle();  break;  case "Line":  s = new MyLine();  break;  default:  throw new InvalidDataException("Unknown shape kind: " + kind);  }  s.LoadFrom(reader);  AddShape(s);  }  }  finally  {  reader.Close();  }  }  }  } |

Shape File

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using SplashKitSDK;  namespace ShaperDrawer  {  public abstract class Shape  {  public float \_x, \_y;    public SplashKitSDK.Color \_color;  public bool \_selected;  public Shape(SplashKitSDK.Color color)  {  \_x = 0.0f;  \_y = 0.0f;  \_color = color;    }  public Shape() : this(SplashKitSDK.Color.Yellow)  {  }  public float X  {  get { return \_x; }  set { \_x = value; }  }  public float Y  {  get { return \_y; }  set { \_y = value; }  }  public SplashKitSDK.Color Color  {  get { return \_color; }  set { \_color = value; }  }  public abstract void Draw();  // Check if mouse is within the shape  public abstract bool IsAt(Point2D pt);  public bool Selected  {  get { return \_selected; }  set { \_selected = value; }  }  public abstract void DrawOutline();  public virtual void SaveTo(StreamWriter writer)  {  writer.WriteColor(Color);  writer.WriteLine(X);  writer.WriteLine(Y);  }  public virtual void LoadFrom(StreamReader reader)  {  Color = reader.readColor();  X = reader.readInteger();  Y = reader.readInteger();  }  }  } |

Circle File

|  |
| --- |
| using SplashKitSDK;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ShaperDrawer  {  internal class MyCircle : Shape  {  private int \_radius;  public MyCircle(int radius, SplashKitSDK.Color color) : base(color)  {  \_radius = radius;  }  public int Radius  {  get { return \_radius; }  set { \_radius = value; }  }  public MyCircle() : this(50, SplashKitSDK.Color.Blue)  {  }  public override void Draw()  {  if (Selected)  {  DrawOutline();  }  SplashKit.FillCircle(\_color, \_x + \_radius, \_y + \_radius, \_radius);  }    public override bool IsAt(Point2D pt)  {  double centerX = \_x + \_radius;  double centerY = \_y + \_radius;  //Math.Sqrt calculates the square root of a number while  //Math.Pow calculates the power of a number  double distance = Math.Sqrt(Math.Pow(pt.X - centerX, 2) + Math.Pow(pt.Y - centerY, 2));  return distance <= \_radius;  }  public override void DrawOutline()  {  SplashKit.FillCircle(SplashKitSDK.Color.Black, \_x + \_radius, \_y + \_radius, \_radius + 2);  }  public override void SaveTo(StreamWriter writer)  {  writer.WriteLine("Circle");  base.SaveTo(writer);  writer.WriteLine(\_radius);  }  public override void LoadFrom(StreamReader reader)  {  base.LoadFrom(reader);  \_radius = reader.readInteger();  }  }  } |

Line File

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using SplashKitSDK;  namespace ShaperDrawer  {  internal class MyLine : Shape  {  public MyLine(SplashKitSDK.Color color) : base(color)  {    }  public MyLine() : this(SplashKitSDK.Color.Red)  {  }  public override void Draw()  {  if (Selected)  {  DrawOutline();  }  SplashKit.DrawLine(\_color, \_x, \_y, \_x + 100, \_y + 100);  }  public override bool IsAt(Point2D pt)  {  float endX = \_x + 100;  float endY = \_y + 100;    // Use the distance from point to line formula  double distance = Math.Abs((endY - \_y) \* pt.X + (\_x - endX) \* pt.Y + (endX \* \_y - endY \* \_x)) /  Math.Sqrt((endY - \_y) \* (endY - \_y) + (endX - \_x) \* (endX - \_x));  return distance <= 2;  }  public override void DrawOutline()  {  SplashKit.FillCircle(SplashKitSDK.Color.Black, \_x, \_y, 2);  SplashKit.FillCircle(SplashKitSDK.Color.Black, \_x + 100, \_y + 100, 2);  }  public override void SaveTo(StreamWriter writer)  {  writer.WriteLine("Line");  base.SaveTo(writer);  }  }  } |

Rectangle File

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Text;  using SplashKitSDK;  namespace ShaperDrawer  {  internal class MyRectangle : Shape  {  public int \_width, \_height;  public MyRectangle(SplashKitSDK.Color color, float x, float y, int width, int height) : base(color)  {  \_width = width;  \_height = height;  \_x = x;  \_y = y;  }  public MyRectangle() : this(SplashKitSDK.Color.Green, 0.0f, 0.0f, 100, 100)  {  }  public override bool IsAt(Point2D pt)  {  return (pt.X >= \_x && pt.X <= \_x + \_width && pt.Y >= \_y && pt.Y <= \_y + \_height);  }  public override void Draw()  {  if (Selected)  {  DrawOutline();  }  SplashKit.FillRectangle(\_color, \_x, \_y, \_width, \_height);  }  public override void DrawOutline()  {  SplashKit.FillRectangle(SplashKitSDK.Color.Black, \_x - 2, \_y - 2, \_width + 4, \_height + 4);  }  public override void SaveTo(StreamWriter writer)  {  writer.WriteLine("Rectangle");  base.SaveTo(writer);  writer.WriteLine(\_width);  writer.WriteLine(\_height);  }  public override void LoadFrom(StreamReader reader)  {  base.LoadFrom(reader);  \_width = reader.readInteger();  \_height = reader.readInteger();  }    }  } |

Extension Method File

|  |
| --- |
| using System;  using System.IO;  using SplashKitSDK;  namespace ShaperDrawer  {  public static class ExtensionMethods  {  public static int readInteger(this StreamReader reader)  {  return Convert.ToInt32(reader.ReadLine());  }  public static float readSingle(this StreamReader reader)  {  return Convert.ToSingle(reader.ReadLine());  }  public static Color readColor(this StreamReader reader)  {  return Color.RGBColor(reader.readSingle(), reader.readSingle(), reader.readSingle());  }  public static void WriteColor(this StreamWriter writer, Color clr)  {  writer.WriteLine("{0}\n{1}\n{2}", clr.R, clr.G, clr.B);  }  }  } |

TextDrawing File

|  |
| --- |
| 1  1  1  6  Circle  0  0  1  182  147  50  Rectangle  0  0.5  0  609  219  100  100  Line  1  0  0  294  457  Circle  0  0  1  474  142  50  Rectangle  0  0.5  0  90  342  100  100  Line  1  0  0  111  81 |

Current Directory Structure of my Shape Drawer File

A screenshot of a computer

Description automatically generated