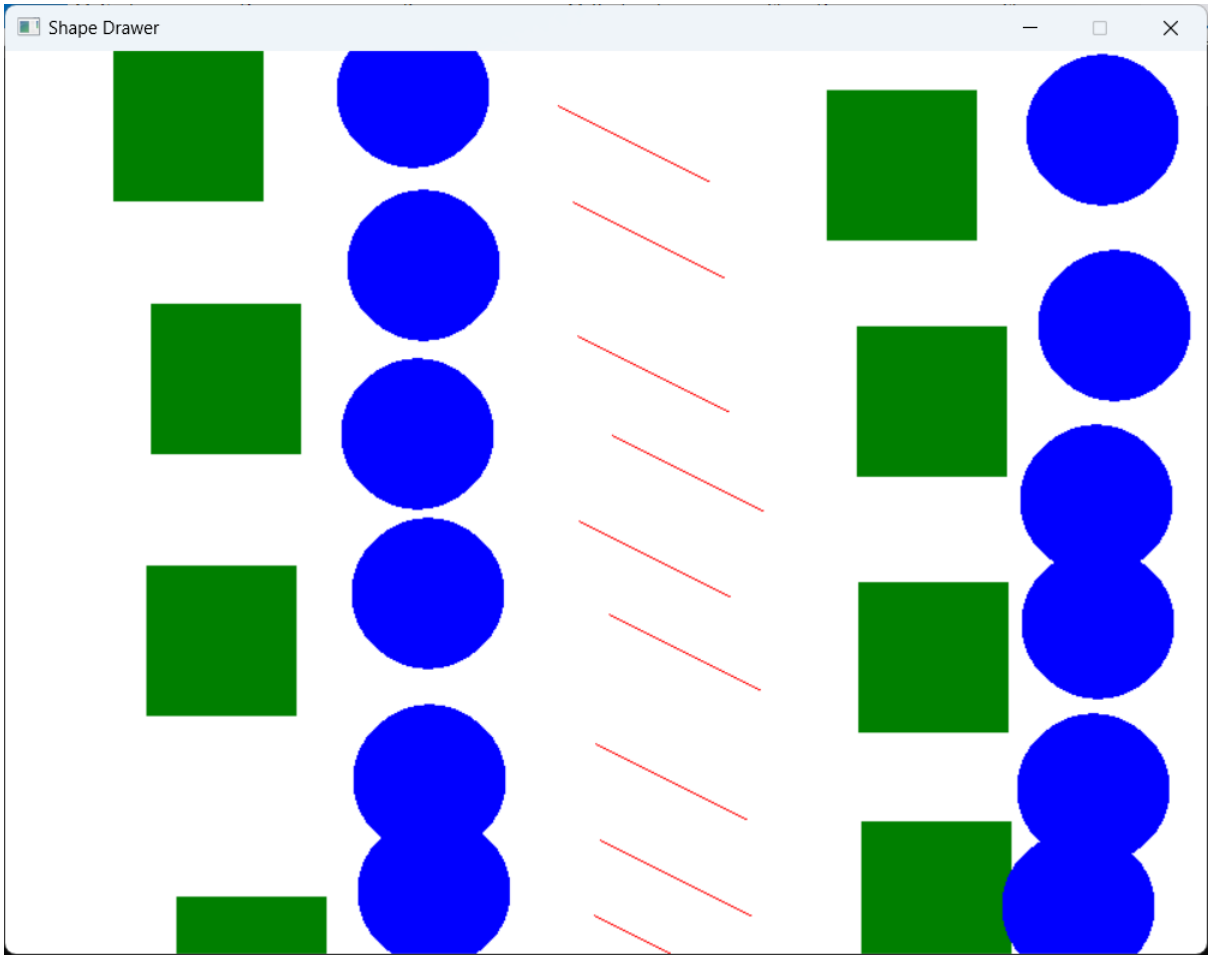
4.1 code and output



Shape.cs

```
using SplashKitSDK;
using System.IO;
namespace ShapeDrawer
{
    public abstract class Shape
    {
        private Color _color;
        private float _x, _y;
        private bool _selected;

        public virtual void SaveTo(StreamWriter writer)
        {
```

```
    writer.WriteLine(Color.ToString());
    writer.WriteLine(X);
    writer.WriteLine(Y);
}

public virtual void LoadFrom(StreamReader reader)
{
    Color = reader.ReadColor();
    X = reader.ReadInteger();
    Y = reader.ReadInteger();
}

// Property for Color
public Color Color
{
    get { return _color; }
    set { _color = value; }
}

// Property for X coordinate
public float X
{
    get { return _x; }
    set { _x = value; }
}

// Property for Y coordinate
public float Y
{
    get { return _y; }
    set { _y = value; }
```

```
}

// Property for Selected
public bool Selected
{
   get { return _selected; }
   set { _selected = value; }
}

// Default constructor that initializes with Color.Yellow
public Shape() : this(Color.Yellow)
{
}

// Constructor that accepts color as a parameter
public Shape(Color color)
{
   _x = 0.0f;
   _y = 0.0f;
   _color = color;
   _selected = false;
}

// Empty method bodies for Draw and DrawOutline
public virtual void Draw() { }

public virtual void DrawOutline() { }

// Virtual IsAt method returns false
public virtual bool IsAt(Point2D pt)
{
```

```
            return false;
        }
    }
}
```

MyRectangle.cs

```
using SplashKitSDK;

namespace ShapeDrawer
{
    public class MyRectangle : Shape
    {
        private int _width, _height;

        // Property for Width
        public int Width
        {
            get { return _width; }
            set { _width = value; }
        }

        // Property for Height
        public int Height
        {
            get { return _height; }
            set { _height = value; }
        }

        // Default constructor initializes with Color.Green, 0.0f for x and y, and 100 for width and height
        public MyRectangle() : this(Color.Green, 0.0f, 0.0f, 100, 100)
        {
```

```
}

public override void SaveTo(StreamWriter writer)
{
    base.SaveTo(writer);
    writer.WriteLine("Rectangle");
    writer.WriteLine(Width);
    writer.WriteLine(Height);

}

public override void LoadFrom(StreamReader reader)
{
    base.LoadFrom(reader);
    Width = reader.ReadInteger();
    Height = reader.ReadInteger();

}
// Constructor that accepts color, x, y, width, and height
public MyRectangle(Color color, float x, float y, int width, int height) : base(color)
{
    X = x;
    Y = y;
    _width = width;
    _height = height;
}

// Override Draw method
public override void Draw()
{
    SplashKit.FillRectangle(Color, X, Y, _width, _height);
```

```
            if (Selected)
            {
                DrawOutline(); // Draw outline if selected
            }
        }

        // Override DrawOutline method
        public override void DrawOutline()
        {
            SplashKit.DrawRectangle(Color.Black, X - 2, Y - 2, _width + 4, _height + 4);
        }

        // Override IsAt method to check if a point is within the rectangle
        public override bool IsAt(Point2D pt)
        {
            return (pt.X >= X && pt.X <= X + _width && pt.Y >= Y && pt.Y <= Y + _height);
        }
    }
}
```

MyLine.cs

```
using SplashKitSDK;
using System;

namespace ShapeDrawer
{
    public class MyLine : Shape
    {
        private float _endX;
        private float _endY;
```

```
// Default constructor initializing color and coordinates
public MyLine() : this(Color.Red, 0, 0, 100, 100)
{
}

// Constructor with parameters for color and coordinates
public MyLine(Color color, float startX, float startY, float endX, float endY) : base(color)
{
   X = startX;
   Y = startY;
   _endX = endX;
   _endY = endY;
}

// Property for EndX
public float EndX
{
   get { return _endX; }
   set { _endX = value; }
}

// Property for EndY
public float EndY
{
   get { return _endY; }
   set { _endY = value; }
}

// Override Draw to actually draw the line
public override void Draw()
```

```
        {
            SplashKit.DrawLine(Color, X, Y, _endX, _endY);
        }

        // Override DrawOutline for the selection
        public override void DrawOutline()
        {
            if (Selected)
            {
                SplashKit.DrawLine(Color.Black, X - 5, Y - 5, _endX + 5, _endY + 5);
            }
        }

        public override bool IsAt(Point2D pt)
        {
            // Create a Line object from the start and end points
            Line line = SplashKit.LineFrom(X, Y, _endX, _endY);

            // Use the PointOnLine method, which expects a Line object
            return SplashKit.PointOnLine(pt, line);
        }

    }
}
```

MyCircle.cs

```
using SplashKitSDK;

namespace ShapeDrawer
{
    public class MyCircle : Shape
    {
```

```
private int _radius;

// Property for Radius
public int Radius
{
    get { return _radius; }
    set { _radius = value; }
}

public override void LoadFrom(StreamReader reader)
{
    base.LoadFrom(reader);
    Radius = reader.ReadInteger();
}



// Default constructor initializes with Color.Blue and Radius 50
public MyCircle() : this(Color.Blue, 50)
{
}

// Constructor that accepts color and radius
public MyCircle(Color color, int radius) : base(color)
{
    _radius = radius;
}

// Override Draw method
public override void Draw()
{
    SplashKit.FillCircle(Color, X, Y, _radius);
```

```
            if (Selected)
            {
                DrawOutline(); // Draw outline if selected
            }
        }

        // Override DrawOutline method
        public override void DrawOutline()
        {
            SplashKit.DrawCircle(Color.Black, X, Y, _radius + 2);
        }

        public override void SaveTo(StreamWriter writer)
        {
            base.SaveTo(writer);
            writer.WriteLine("Circle");
            writer.WriteLine(Radius);

        }

        // Override IsAt method to check if a point is within the circle
        public override bool IsAt(Point2D pt)
        {
            return SplashKit.PointInCircle(pt, SplashKit.CircleAt(X, Y, _radius));
        }
    }

}
```

Program.cs

```
using System;
using SplashKitSDK;

namespace ShapeDrawer
{
    public class Program
    {
        private enum ShapeKind
        {
            Rectangle,
            Circle,
            Line
        }

        public static void Main()
        {
            // Default shape to add is Circle
            ShapeKind kindToAdd = ShapeKind.Circle;

            Drawing myDrawing = new Drawing(); // Create a Drawing object using the default constructor
            Window window = new Window("Shape Drawer", 800, 600);

            // Set up the file path for saving/loading
            string desktopPath = "C:\\Users\\joshu\\OneDrive\\Desktop\\COS20007
OOP\\ShapeDrawer";
            string filePath = System.IO.Path.Combine(desktopPath, "TestDrawing.txt");

            do
            {
                SplashKit.ProcessEvents();
```

```
window.Clear(Color.White); // Clear with a temporary white color

myDrawing.Draw(); // Draw all shapes

if (SplashKit.MouseClicked(MouseButton.LeftButton))
{
    // Create a new shape based on kindToAdd
    Shape newShape = null;  // Initialize as null to handle lines differently

    if (kindToAdd == ShapeKind.Rectangle)
    {
        // Create a new rectangle
        newShape = new MyRectangle();
    }
    else if (kindToAdd == ShapeKind.Circle)
    {
        // Create a new circle
        newShape = new MyCircle();
    }
    else if (kindToAdd == ShapeKind.Line)
    {
        // Create a new line
        float startX = SplashKit.MouseX();
        float startY = SplashKit.MouseY();
        // Placeholder end points for the line, can be adjusted based on user input
        newShape = new MyLine(Color.Red, startX, startY, startX + 100, startY + 50);
    }

    if (newShape != null)
    {
        // Set the position of the new shape for non-line shapes
```

```
        if (kindToAdd != ShapeKind.Line)
        {
            newShape.X = SplashKit.MouseX();
            newShape.Y = SplashKit.MouseY();
        }

        myDrawing.AddShape(newShape); // Add the new shape to the drawing
    }
}

// Change shape to Rectangle if 'R' key is pressed
if (SplashKit.KeyTyped(KeyCode.RKey))
{
    kindToAdd = ShapeKind.Rectangle;
}

// Save the drawing if 'S' key is pressed
if (SplashKit.KeyTyped(KeyCode.SKey))
{
    // Call the Save method of Drawing class
    myDrawing.Save(filePath);
}

// Load the drawing if 'O' key is pressed
if (SplashKit.KeyTyped(KeyCode.OKey))
{
    myDrawing.Load(filePath);  // Load the drawing from the file
}

// Change shape to Circle if 'C' key is pressed
if (SplashKit.KeyTyped(KeyCode.CKey))
```

```
        {
            kindToAdd = ShapeKind.Circle;
        }

        if (SplashKit.KeyTyped(KeyCode.LKey))
        {
            kindToAdd = ShapeKind.Line;
        }

        // Change the background color to a random color when the space key is pressed
        if (SplashKit.KeyTyped(KeyCode.SpaceKey))
        {
            myDrawing.Background = SplashKit.RandomColor();
        }

        // Select shapes at the current mouse pointer position when the right mouse button is
clicked
        if (SplashKit.MouseClicked(MouseButton.RightButton))
        {
            myDrawing.SelectShapesAt(SplashKit.MousePosition());
        }

        // Remove selected shapes if the delete or backspace key is pressed
        if (SplashKit.KeyTyped(KeyCode.DeleteKey) || SplashKit.KeyTyped(KeyCode.BackspaceKey))
        {
            foreach (var shape in myDrawing.SelectedShapes)
            {
                myDrawing.RemoveShape(shape);
            }
        }
```

```
            window.Refresh(60);
        } while (!window.CloseRequested);

        window.Close();
    }
  }
}
```