# Processing.Net Project Description

Kelson Ball

# Processing for DotNet

Processing has had a huge impact in my journey learning to code. Its intuitive approach to graphics is something I miss in the development environment I spend the most time in now, writing desktop applications in C#. Several months ago I set out to try and recreate processing in the dotnet environment and learn as much as I could about computer graphics along the way.

I created the current implementation of Processing.Net with the goal of maintaining the structure of the Processing API for 2D sketches while reimplementing its back end using dotnet types and conventions. The dotnet implementation is also designed with an eye toward being easily embeddable in existing WinForms and WPF applications. There is also a standalone build project that allows using the dotnet implementation of Processing by itself in the traditional sketch structure.

The C# implementation uses common dotnet conventions such as capitalizing public members and using C# properties rather than explicit getters and setters.

Processing Java Example

```
int y = 100;
void setup() {
  size(640, 360);
  stroke(255);
  frameRate(30);
}

void draw() {
  background(0);
  y = y - 1;
  if (y < 0) {
    y = height;
  }
  line(0, y, width, y);
}
```

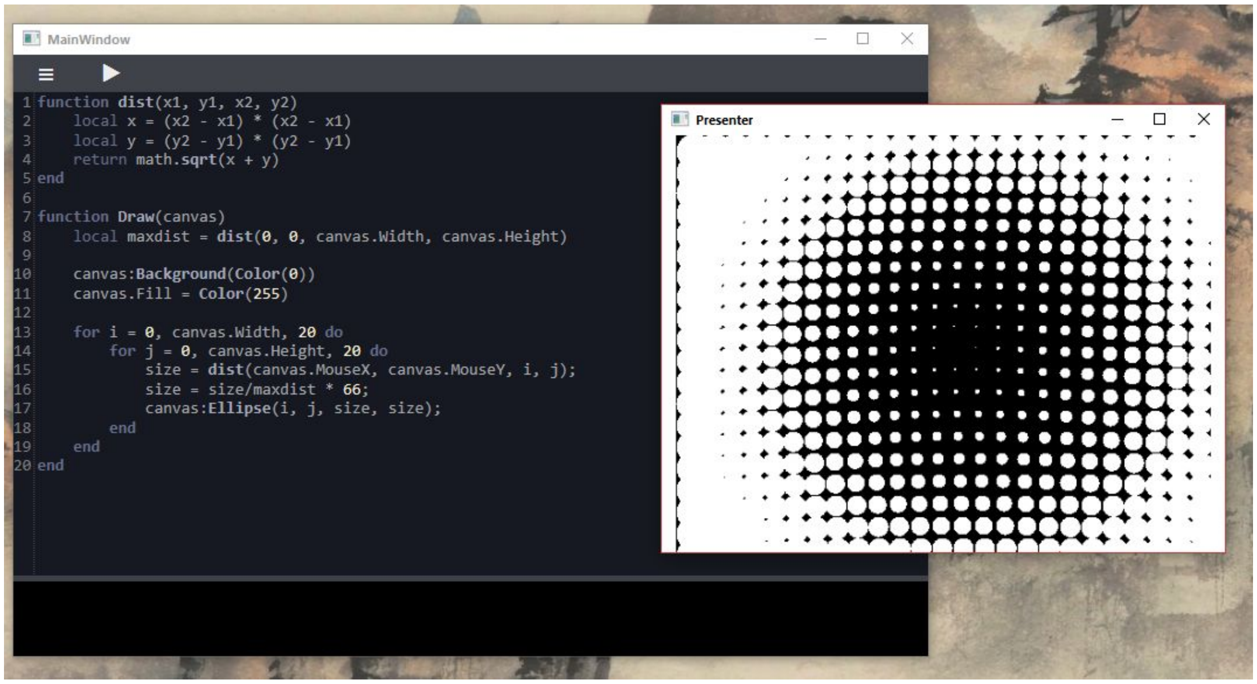
Processing C# Example

```
int y = 100;
void Setup()
{
    Size(640, 360);
    Stroke = Color.White;
    FrameRate(30);
}

Void Draw()
{
    Background(Color.Black);
    y = y - 1;
    if (y < 0)
    {
        y = Height;
    }
    Line(0, y, Width, y);
}
```

The C# implementation is also structured to be as independent as possible from the rendering mechanism. Currently the project uses the GDI+ software renderer included in the dotnet and Mono runtimes, allowing sketches to run on linux and mac computers with the Mono runtime installed. I am currently working on implementing the project's `IRenderer` interface with OpenGL to provide optional hardware accelerated and 3D rendering.

Having a dotnet implementation of Processing.Net also allows for using the api in a variety of scripting languages that can target the dotnet or Mono runtimes. Here is a screenshot of an example WPF application I am building that provides live fourier-transformed microphone data to a processing sketch that users can modify using the embeddable scripting language Lua. The sketch shown here is the Distance2D example from the processing website.



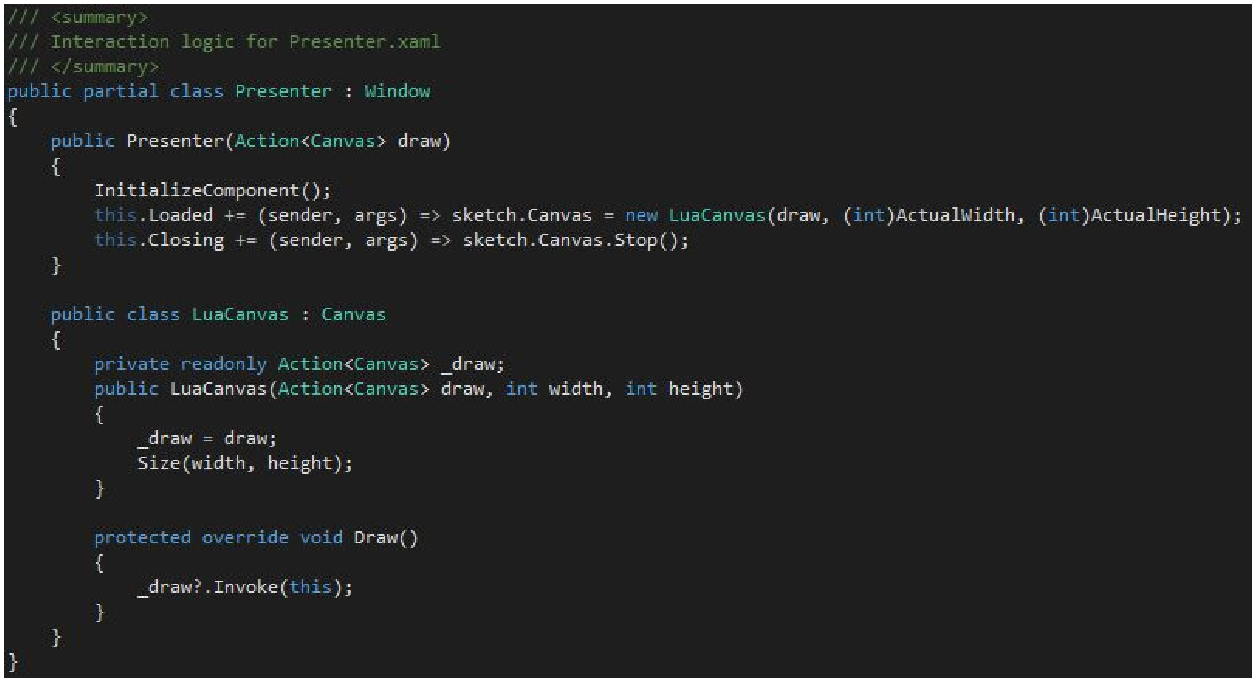
```lua
function dist(x1, y1, x2, y2)
    local x = (x2 - x1) * (x2 - x1)
    local y = (y2 - y1) * (y2 - y1)
    return math.sqrt(x + y)
end

function Draw(canvas)
    local maxdist = dist(0, 0, canvas.Width, canvas.Height)

    canvas:Background(Color(0))
    canvas.Fill = Color(255)

    for i = 0, canvas.Width, 20 do
        for j = 0, canvas.Height, 20 do
            size = dist(canvas.MouseX, canvas.MouseY, i, j);
            size = size/maxdist * 66;
            canvas:Ellipse(i, j, size, size);
        end
    end
end
```

The C# code necessary to allow that to happen consists of extending the Canvas class and passing the canvas to whatever external function is to be called on the draw loop. In the case of the lua embedded application it looks like this:



```
/// <summary>
/// Interaction logic for Presenter.xaml
/// </summary>
public partial class Presenter : Window
{
    public Presenter(Action<Canvas> draw)
    {
        InitializeComponent();
        this.Loaded += (sender, args) => sketch.Canvas = new LuaCanvas(draw, (int)ActualWidth, (int)ActualHeight);
        this.Closing += (sender, args) => sketch.Canvas.Stop();
    }

    public class LuaCanvas : Canvas
    {
        private readonly Action<Canvas> _draw;
        public LuaCanvas(Action<Canvas> draw, int width, int height)
        {
            _draw = draw;
            Size(width, height);
        }

        protected override void Draw()
        {
            _draw?.Invoke(this);
        }
    }
}
```

An additional consequence of Processing running on the Mono runtime is that sketches could be made embeddable in Xamarin Forms apps, allowing processing sketches to be a part of mobile applications running on Android, iOS, and Windows devices.

A more complete example of a sketch written in C# is available on github here.