```
using System;
using System.Collections.Generic;
using System.Linq;
using Gtk;
using Cairo;
using Structures;
using System.Threading;
using System.Threading.Tasks;
using static Program.Constants;
namespace Graphics {
        class SystemView : DrawingArea {
                public Camera camera {get; set;} //= new Camera(1,Vector3.zero);
                public double radius_multiplier {get; set;} = 1;
                public int line_max {get; set;} = 100;
                public double bounds_multiplier {get; set;} = 1;//0.25;
                protected PlanetarySystem sys;
                protected readonly double LINE_MULTIPLIER = 0.8;
                protected bool playing = false;
                protected List<Vector3>[] paths;
                protected int[] order;
                protected double max = 0;
                public SystemView(PlanetarySystem sys) {
                        this.sys = sys;
                        this.camera = new Camera(sys.Max(b => Vector3.Magnitude(b.position - sys.origin)),Vector3.zero);
                        SetMax();
                }
                public void SetMax() {
                        order = new int[sys.Count];
                        for (int i = 0; i < sys.Count; i++) order[i] = i;
                        max = 0;
                        foreach (Body b in sys) {
                                var v = camera.TransformProjection(camera.Transform(b.position - sys.origin));
                                var p = Vector3.Magnitude(new Vector3(v.x,v.y,0));
                                if (p > max) {
                                        max = p;
                                }
                        }

                }
                public void ClearPaths() {
                        this.paths = new List<Vector3>[sys.Count];
                        for (int i = 0; i < sys.Count; i++) {
                                this.paths[i] = new List<Vector3>();
                        }
                }
                public void Play(int interval) {
                        playing = true;
                        while (playing) {
                                this.QueueDraw();
                                Thread.Sleep(interval);
                        }
                }
                public void PlayAsync(int interval) {
                        Task.Run(() => Play(interval));
                }
                public void Stop() {
                        playing = false;
                }
                protected override bool OnDrawn (Cairo.Context ctx) {
                        // color the screen black
                        ctx.SetSourceRGB(0,0,0);
                        ctx.Paint();
```

```
			// Normally (0,0) is in the corner, but we want it in the middle, so we must translate:
			ctx.Translate(AllocatedWidth/2,AllocatedHeight/2);
			var bounds = bounds_multiplier * max * new Vector3(1,1,1);
			// we care about the limiting factor, since most orbits will be bounded roughly by a square
			// but screens are rectangular
			var scale = Math.Min((AllocatedWidth/2)*bounds.x,(AllocatedHeight/2)/bounds.y);
			ctx.Scale(scale,scale);

			if (paths == null) {
				this.ClearPaths();
			}
			order = order.OrderByDescending(x => Vector3.Magnitude(sys[x].position - camera.position)).ToArray();
			for (int i = 0; i < sys.Count; i++) {
				var body = sys[order[i]];
				var cl = body.color;
				ctx.SetSourceRGB (cl.x,cl.y,cl.z);

				var T = camera.Transform(body.position - sys.origin);//camera.position);// - camera.Transform(sys.origin);

				var r = radius_multiplier * camera.TransformProjectionRadius(T,body.radius);//body.radius;
				var pos = camera.TransformProjection(T);
				ctx.Arc(pos.x,pos.y,r,0,2*Math.PI);
				ctx.Fill();
				Vector3 lastPath;
				try {
					lastPath = camera.TransformProjection(camera.Transform(paths[order[i]][0]));
				} catch (ArgumentOutOfRangeException) {
					lastPath = Vector3.zero;
				}
				ctx.LineWidth = Math.Min(LINE_MULTIPLIER * radius_multiplier * body.radius, LINE_MULTIPLIER*r);
				foreach (Vector3 p in paths[order[i]]) {
					pos = camera.TransformProjection(camera.Transform(p));
					ctx.MoveTo(lastPath.x,lastPath.y);
					ctx.LineTo(pos.x,pos.y);
					ctx.Stroke();
					lastPath = pos;
				}
				paths[order[i]].Add(body.position - sys.origin);
				if (paths[order[i]].Count > line_max) paths[order[i]] = paths[order[i]].TakeLast(line_max).ToList();
			}
			return true;
		}
	}
}
```