```
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 Camera {
		public Vector3 position {get; protected set;}
		public Vector3 angle {get; protected set;}
		public Camera(double distance, Vector3 angle) {
			// the camera always "points" to the origin
			this.angle = angle;
			position = Matrix3.IntrinsicZYXRotation(angle)*new Vector3(0,0,distance);

		}
		public Vector3 Transform(Vector3 position) {
			return Matrix3.ExtrinsicZYXRotation(this.angle)*(position - this.position);
		}
	}
	class SystemView : DrawingArea {

		public Camera camera {get; set;} = new Camera(50*AU,Vector3.zero);
		public double radius_multiplier {get; set;} = 1;
		public int line_max {get; set;} = 100;
		public double bounds_multiplier {get; set;} = 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;
			Redraw();
		}
		public void Redraw() {
			order = new int[sys.Count];
			for (int i = 0; i < sys.Count; i++) order[i] = i;
			max = 0;
			foreach (Body b in sys) {
				var p = Vector3.Magnitude(camera.Transform(b.position));
				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/bounds.x,AllocatedHeight/bounds.y);
            ctx.Scale(scale,scale);
            if (paths == null) {
                this.ClearPaths();
            }
            var origin = Program.Program.activesys.origin;
            order = order.OrderByDescending(x => Vector3.Magnitude(sys[x].position - camera.position)).ToArray();
            for (int i = 0; i < sys.Count; i++) {
                Body body = sys[order[i]];
                var r = radius_multiplier * body.radius;
                ctx.LineWidth = line_multiplier * radius_multiplier * body.radius;
                Vector3 lastPath = Vector3.zero;
                try {
                    lastPath = paths[order[i]][0];
                } catch (ArgumentOutOfRangeException) {};
                for (int j = -1; j < paths[order[i]].Count; j++) {

                    Vector3 true_position;
                    if (j == -1) true_position = body.position;
                    else true_position = paths[order[i]][j] + origin;
                    Vector3 pos;
                    pos = camera.Transform(true_position) - camera.Transform(origin);
                    var cl = body.color;
                    ctx.SetSourceRGB (cl.x,cl.y,cl.z);
                    if (j == -1) {
                        ctx.Arc(pos.x,pos.y,r,0,2*Math.PI);
                        ctx.Fill();
                    }
                    else if (j > 0) {
                        ctx.MoveTo(lastPath.x,lastPath.y);
                        ctx.LineTo(pos.x,pos.y);
                        ctx.Stroke();
                    } lastPath = pos;

                }
                paths[order[i]].Add(body.position - origin);
                if (paths[order[i]].Count > line_max + 1) {
          // if line_max has been reduced the paths must be removed
```

```
faster than they can be created
                    paths[order[i]].RemoveAt(0);
                                        paths[order[i]].RemoveAt(0);
                                }
                                else if (paths[order[i]].Count > line_max) {
                                        paths[order[i]].RemoveAt(0);
                                }
                        }
                        return true;
                }
        }
}
```