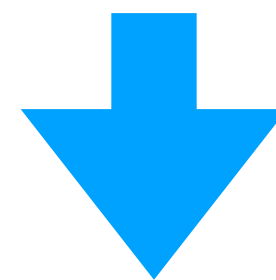


# Form Scene Graph

## A New UI Rendering Framework

Ronie Salgado  
RMoD, Inria, Lille, France

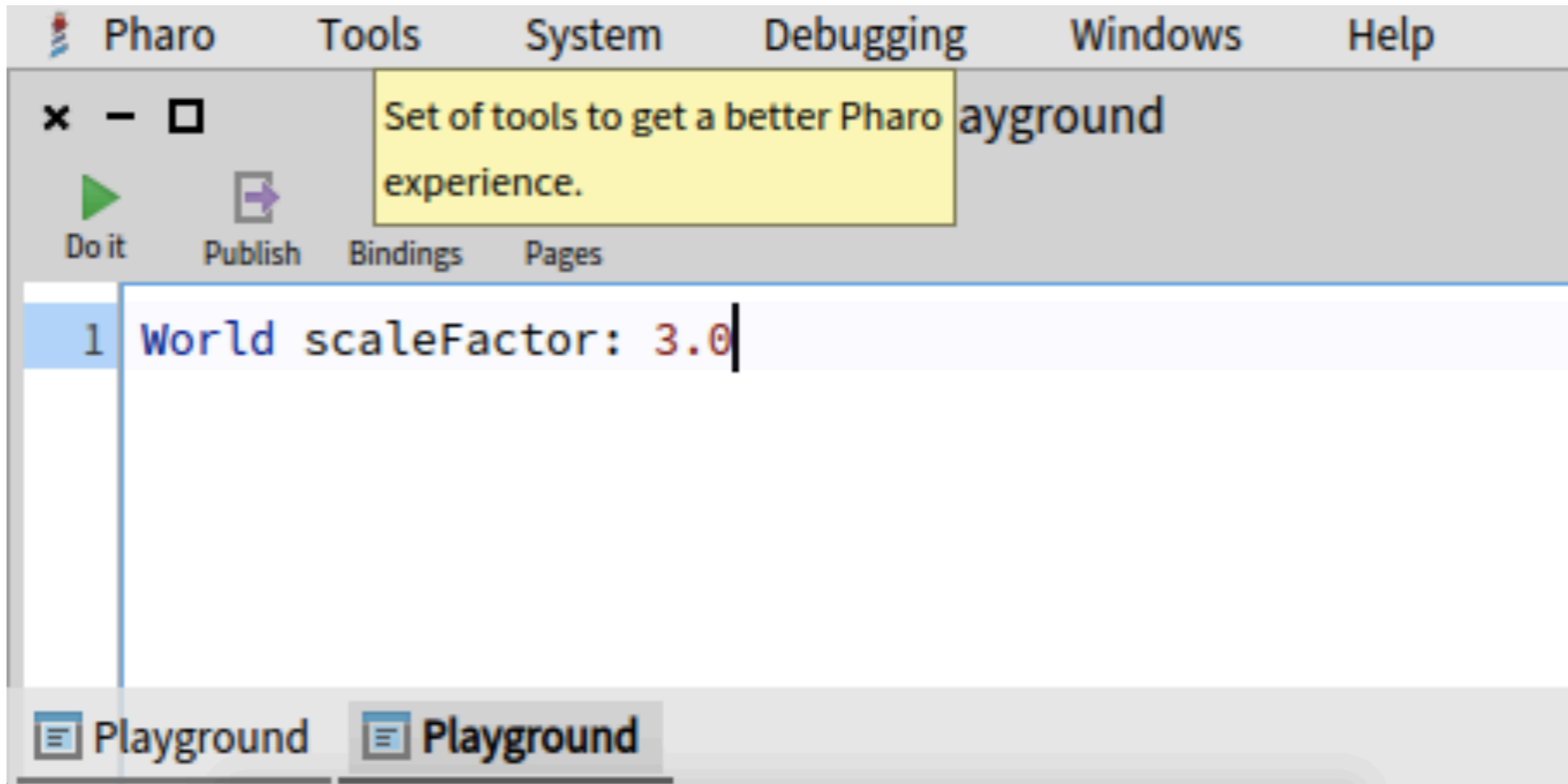
```
1 World scaleFactor: 5.0
```



```
1 World scaleFactor: 5.0
```

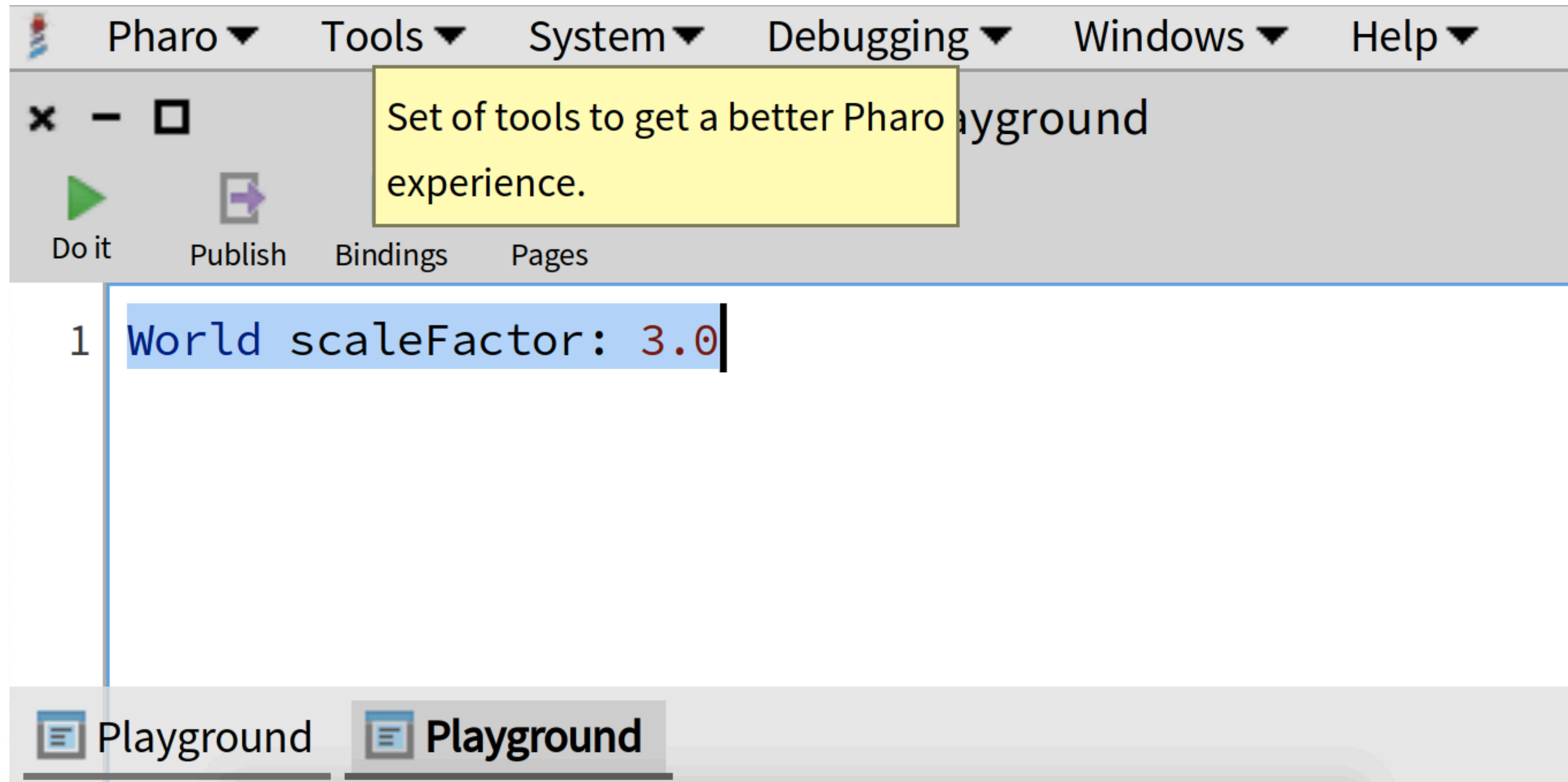
# UI Scaling

## Current Method - #fullDrawOn: & Canvas



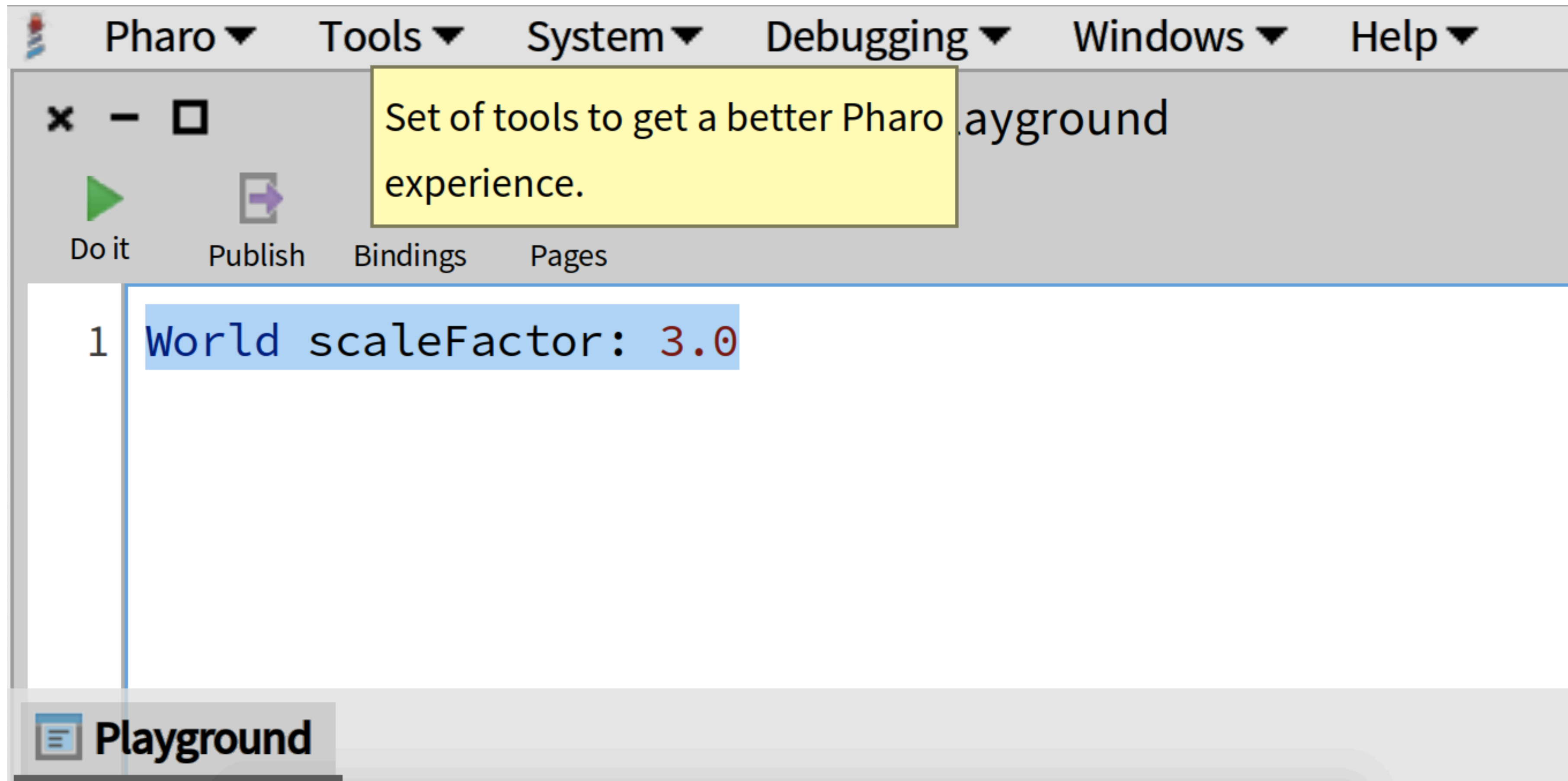
# UI Scaling

## Scene Graph - Athens Backend



# UI Scaling

## Scene Graph - SDL2 Backend



# How to Install

GitHub: <https://github.com/ronsaldo/form-scene-graph>

Metacello new

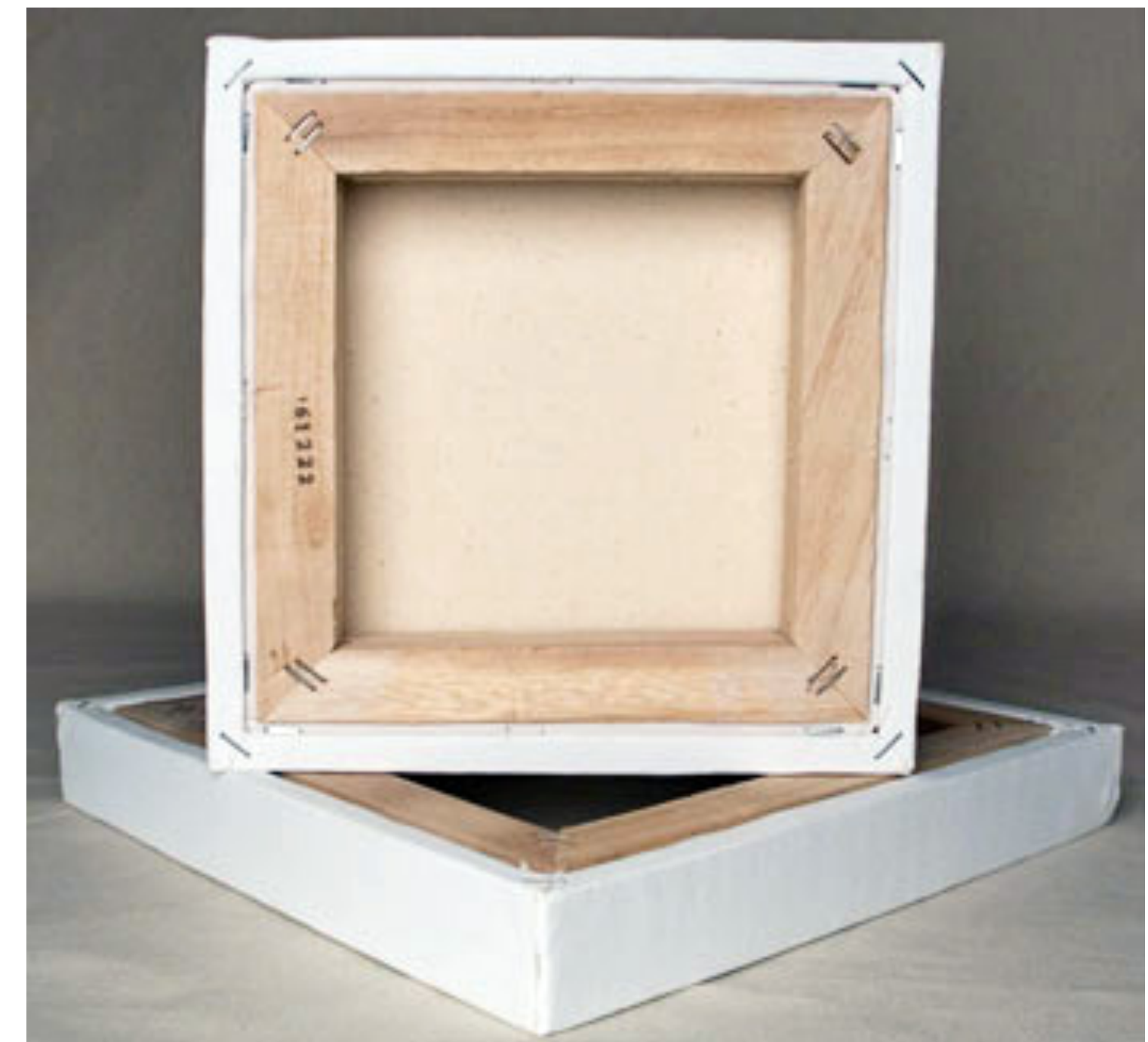
baseline: 'FormSceneGraph';

repository: 'github://ronsaldo/form-scene-graph';

load.

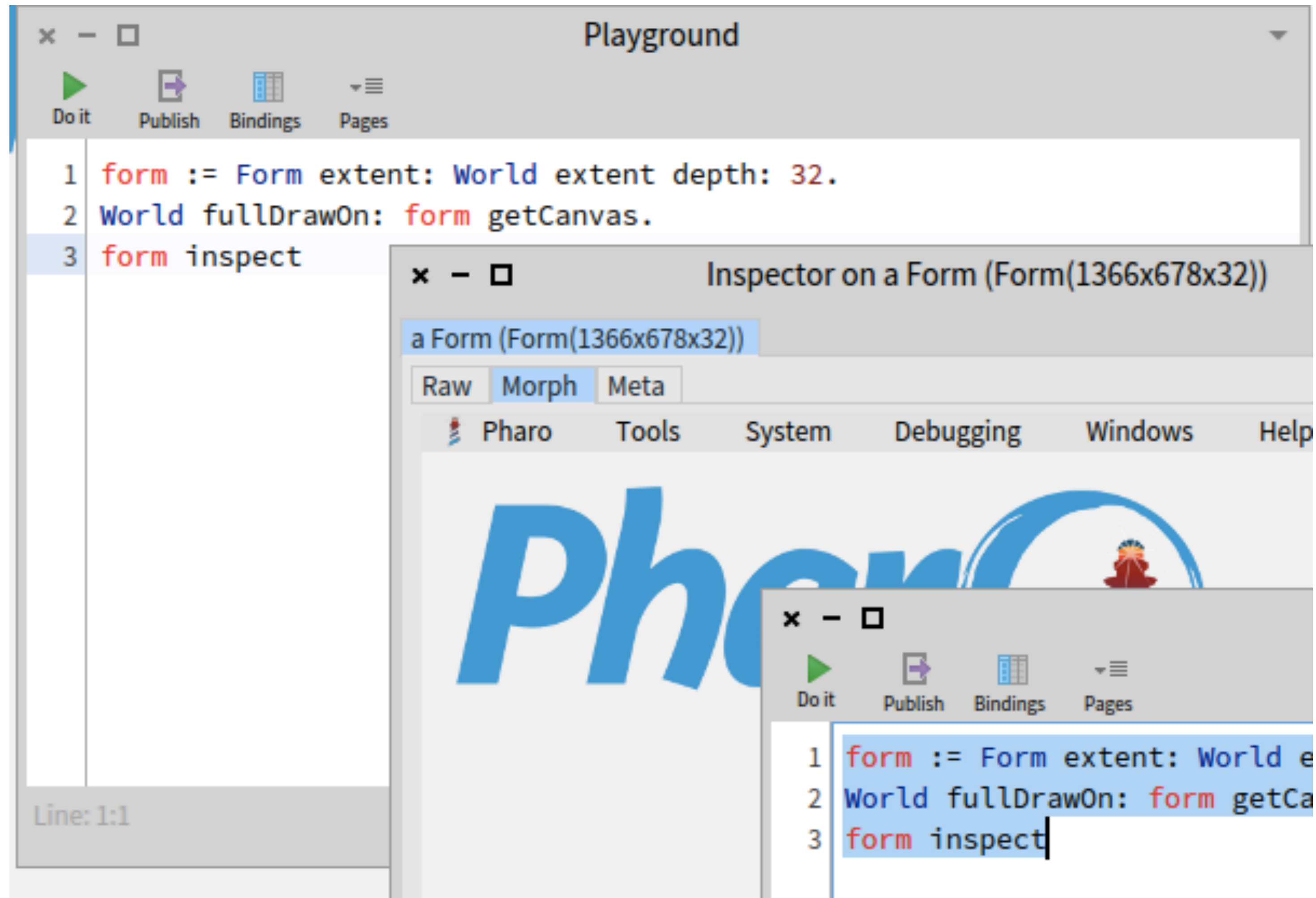
# The Current Method: Canvas Metaphor

- Start from Morph >> #fullDrawOn: aCanvas
- Pre-order traversal of Morphs.
- Sequential and immediate drawing on a Form Canvas.
- Requires immediate pixel access
  - Difficult to accelerate with GPU.
- Large API: Canvas selectors size => 89
- Non-scalable with arbitrary affine transforms.





# How to Draw Manually the world



# Morph Drawing Sequence

- fullDrawOn: aCanvas -> on: Error do: [.. drawRedScreenOfDeath ..]
  1. drawOn: aCanvas -> Draws the content of the morph itself
  2. drawSubmorphsOn: aCanvas -> Draw the children morphs on top.
  3. Additional effects (drawDropHighlightOn: and drawMouseDownHighlightOn:)
- In parallel we have fullDrawOnAthensCanvas:
- If we want more rendering mechanisms with this approach, all Morphs and subclasses must be extended, per-backend.



# What Other UI Frameworks are Moving To:

- Gtk 3 uses Cairo (Athens) for rendering. They complaint about its performance.
- Gtk 4 is introducing the Gtk Scene Graph (OpenGL and Vulkan).
- QT already uses a scene graph.
- 3D rendering frameworks are based on scene graphs.
- Spec 2 has a Gtk backend -> We need to integrate Morphic on Gtk.

# Taking Inspiration on GSK

- A rendering tree.
- All shapes are rectangles.
- They are easy to draw individually.
- These shapes are composed.
- Selection of nodes based on CSS styling.
- Complex shapes are drawn statically with Cairo into textures. They are composed as rectangles.

```
typedef enum {  
    GSK_NOT_A_RENDER_NODE = 0,  
    GSK_CONTAINER_NODE,  
    GSK_CAIRO_NODE,  
    GSK_COLOR_NODE,  
    GSK_LINEAR_GRADIENT_NODE,  
    GSK_REPEATING_LINEAR_GRADIENT_NODE,  
    GSK_BORDER_NODE,  
    GSK_TEXTURE_NODE,  
    GSK_INSET_SHADOW_NODE,  
    GSK_OUTSET_SHADOW_NODE,  
    GSK_TRANSFORM_NODE,  
    GSK_OPACITY_NODE,  
    GSK_COLOR_MATRIX_NODE,  
    GSK_REPEAT_NODE,  
    GSK_CLIP_NODE,  
    GSK_ROUNDED_CLIP_NODE,  
    GSK_SHADOW_NODE,  
    GSK_BLEND_NODE,  
    GSK_CROSS_FADE_NODE,  
    GSK_TEXT_NODE,  
    GSK_BLUR_NODE,  
    GSK_OFFSET_NODE,  
    GSK_DEBUG_NODE  
} GskRenderNodeType;
```

# Form Scene Graph Design

- Copy the nodes from GSK in Pharo classes.
- Builder class for instancing nodes. Some Canvas APIs are emulated here, but they produce these same nodes.
- Renderer backends are implemented as visitors.
- Few nodes are needed for drawing most of Morphic.

# The Size of a Renderer Backend

FormSGAthensRenderer

- FormSGAthensRendererTest
- FormSGOSWindowGenericRender
- ▼ FormSGRenderer
  - FormSGAthensRenderer
  - FormSGCanvasRenderer
  - FormSGOSWindowGenericRe
- ▼ FormSGTextPreparationVisitor
  - FormSGOSWindowGenericRe

instance side

- rendering
- visiting
- overrides

render:on:  
render:withTransform:on:  
transformRectangle:  
▲ visitAthensNode:  
▲ visitBorderNode:  
▲ visitCachedSubScene:  
▲ visitClipNode:  
▲ visitColorNode:  
▲ visitContainerNode:  
▲ visitLinearGradientNode:  
▲ visitNullNode:  
▲ visitRadialGradientNode:  
▲ visitTextNode:  
▲ visitTextureNode:  
▲ visitTransformNode:  
▲ visitTranslationNode:

Filter...

# Rendering Morphic with the Scene Graph

- Like in the previous case, Morphic has to be extended:
- #buildFullSceneGraphWith:
  - #buildSceneGraphNodeWith: (Most morphs need to override this)
  - #buildClippedChildrenSceneGraphNodeWith:
- Once a Morph is adapted for building a scene graph, it can be rendered with all of the backends.
- The scene graph by definition is scalable (Hi-DPI support).



# Some Morph Adaptations

Morph >> `buildSceneGraphNodeWith: builder  
^ builder fillRectangle: self bounds fillStyle: self fillStyle borderStyle: self borderStyle`

FormSGBuilder - Extension for Morhic:

```
fillRectangle: aRectangle fillStyle: aFillStyle borderStyle: aBorderStyle  
(aBorderStyle isNil or: [aBorderStyle width < 0]) ifTrue: [  
    ^ self fillRectangle: aRectangle style: aFillStyle  
].  
  
^ (self fillRectangle: (aRectangle insetBy: aBorderStyle width) style: aFillStyle) ,  
  [(self borderRectangle: aRectangle style: aBorderStyle)
```

ImageMorph >> `buildSceneGraphNodeWith: builder  
^ (builder textureForm: image at: self innerBounds origin opaque: self isOpaque) ,  
 (builder borderRectangle: self bounds style: self borderStyle).`



# Performance

- *AbstractWorldRenderer startProfilingRenderingTime* enables displaying the rendering time in the top-level corner.
- **Athens** Backend has CPU rendering. It is comparable to the previous renderer. Performance degrades with resolution increments. (26 - 80 ms)
- **OSWindowGenericRenderer** Backend (**SDL2**) may use the GPU. Has better performance in general, but gradient nodes are difficult to implement. (3 and 40 ms)
- **OpenGL ES** backend is faster and better than the OSWindowGenericRenderer. Requires compiling and distributing **ANGLE** for OS X and Windows.
- **AbstractGPU** backend (**Vulkan, Metal, Direct3D12**) slightly faster than the OpenGL ES backend. It suffers of stability issues (crashes on window resize).

# Text Rendering

- This is a difficult process.
- Text rendering is the bottleneck.
- Subpixel antialiased font rendering ~~ Alpha Blending!
  - Each color channel is used as a separate alpha factor!
- Two Pass Algorithm:

1. Subtract the color mask from the background:

```
glBlendFunc(GL_ZERO, GL_ONE_MINUS_SRC_COLOR)
```

2. Add the color mask multiplied by the text color:

```
glBlendFunc(GL_SRC_COLOR, GL_ONE)
```

- One Pass rendering is possible with shaders and dual source blending.

# Scaling Text Rendering

- In the Athens backend, we just leave Cairo perform this task.
- Each renderer has a `surfaceScaleFactor` property.
- The active transform is not used for computing the text font scale to prevent thrashing the glyph cache on a dynamic scale change.
- `scaledFont := font withSizeIncrementedBy: (font pointSize*surfaceScaleFactor) - font pointSize`
- If the current transform scale = `surfaceScaleFactor`, and there is no rotation, then attempt pixel perfect blitting to avoid blurring:
  - Compute the destination pixel and blit the glyph completely.
  - Otherwise, compute the transformed destination rectangle and blit with the scale.
- Compute the glyph destination position by using unscaled glyphs to preserve the metrics, but use the scaled font.

# Caching Sub-Scenes

- The CachingSubSceneNode is container that draws its child on a texture, which is then reused on subsequent redraws.
- Enabling the generation of this caching node on a Morph requires overriding the #shouldCacheSceneGraphSurface to return true.
- Pixel perfect should be attempted to avoid text blurring.
- Caching enabled for Windows, Menus, Rubric and tables.
- This caching textures typically end with premultiplied alpha, so an adequate blending mode is required here:
  - Premultiplied Alpha Over: `glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA)`
  - Composite Alpha Over: `glBlendFuncSeparate(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_ONE, GL_ONE_MINUS_SRC_ALPHA)`
- Subpixel antialiased text does not behave well on transparent backgrounds here.

# Caching Sub-Scene

PharoConsole\_2020.11.26\_13.50.16\_frame689.rdc - RenderDoc v1.10

File Window Tools Help

Timeline - Frame #689

EID: 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300 310

+ Depth-only Pass #1 + Colour Pass #1 (1 Targets) + Colour Pass #2 (1 Targets)

Usage for Swapchain Image 575: Reads (▲), Writes (▲), Read/Write (▲), Barriers (▲), and Clears (▲)

Event Browser

EID	Name	Duration (µs)
0	Frame #689	1909.0757
8-35	Capture Start	
42-278	Depth-only Pass #1	295.24882
281-313	Colour Pass #1 (1 Targets)	1005.57931
281-313	Colour Pass #2 (1 Targets)	608.24757
281	vkCmdBeginRenderPass(Clear)	
290	vkCmdDrawIndexed(6, 1)	290.74884
294	vkCmdDrawIndexed(6, 1)	13.66661
297	vkCmdDrawIndexed(6, 1)	28.49989
300	vkCmdDrawIndexed(6, 1)	105.41625
303	vkCmdDrawIndexed(6, 1)	143.41609
306	vkCmdDrawIndexed(6, 1)	15.41661
309	vkCmdDrawIndexed(6, 1)	11.08329
311	vkCmdEndRenderPass(Store)	
313	API Calls	
314	=> vkQueueSubmit(1)[0]: vkEndCommandBuff	
316	vkQueuePresentKHR( Swapchain Image 575)	

API Inspector

EID	Event
> 301	vkCmdBindPipeline
> 302	vkCmdBindDescriptorSets
> 303	vkCmdDrawIndexed

Texture Viewer

Channels: RGBA R G B A Subresource: Mip 0 - 1920x1001 Slice/Face: Range: 0.00 1.00

Cur Output 0 - Swapchain Image 575

Pharo

AGPUCommandList>>dispatchComputeIndirect:

AGPUBuffer ! AGPUBufferDescription ! AGPUBufferImageCopyRegion ! AGPUBindings ! AGPUColor4f ! AGPUCommandAllocator ! AGPUCommandList ! AGPUCommandQueue ! AGPUComponentsSwizzle ! AGPUComputePipelineBuild ! AGPUConstants ! AGPUDepthStencilValue ! AGPUDevice !

dispatchComputeIndirect:

addReference beginRenderPass:framebuffer:bu bufferMemoryBarrier:source\_staj close copyBuffer:source\_offset:dest\_bi copyBufferToTexture:texture:cop copyTextureToBuffer:buffer:copy dispatchCompute:group\_count: dispatchComputeIndirect: drawArrays:instance\_count:first drawArraysIndirect:drawcount: drawElements:instance\_count:fir drawElementsIndirect:drawcount endRenderPass

FS 0[0] activeTexture

Pixel Context

Swapchain Image 575 - 1920x1001 1 mips - B8G8R8A8\_UNORM Hover - 0, 0 (0.0000, 0.0000) - Right click to pick a pixel

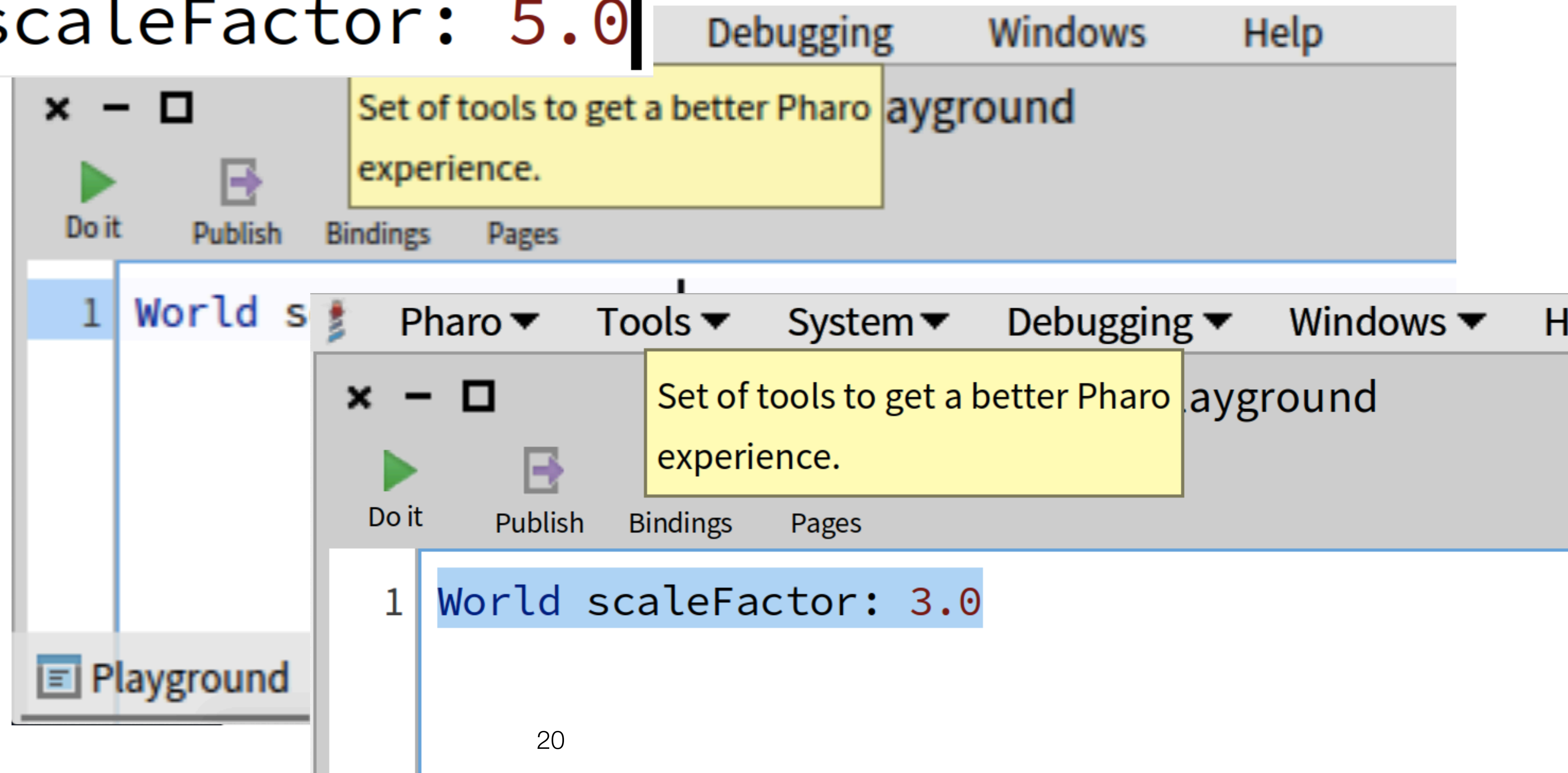
Replay Context: Local PharoConsole\_2020.11.26\_13.50.16\_frame689.rdc loaded. Capture has 1 issues. 1 Unread.



```
1 World scaleFactor: 5.0
```

# Questions?

```
1 World scaleFactor: 5.0
```



```
1 World scaleFactor: 3.0
```