SVGAssets 2.0.1 (AmanithSVG for Unity)

Introduction

SVGAssets is a plugin for reading and rendering SVG files, supporting static features of SVG Full 1.1, plus some static features of SVG Tiny 1.2.

SVGAssets simplifies the management of image assets for all those projects that need to run across a wide range of devices with different resolutions: using the plugin, the developer keeps all image files in SVG format and render them on the target platform at runtime, at the desired resolution.

The API exposes methods for the manipulation of viewports, drawing surfaces and SVG documents: there are methods for creating drawing surfaces, load SVG documents, get their information and draw them on surfaces. The Document Object Model (DOM) is not exposed because the plugin is aimed to the rendering only.

SVGAssets consists of:

- a native Unity plugin that is Mazatech's AmanithSVG Lite SDK
- a low level C# layer, that uses the Platform Invoke mechanism: PInvoke allows .NET code to call functions that are implemented within the AmanithSVG C/C++ native .dll (Windows), .dylib (MacOS X), .so (Linux). It is important to understand that the main benefit of this approach is in the ability to call the same C/C++ code on all .NET platforms.
- a high level C# layer that abstracts some details of the low level API
- an even more higher level C# layer that consists of Unity specific (integration) scripts

Some headlines about the native plugin:

- Standalone: the library does not have external dependencies, it can be used just as is
- Cross platform: the API, and so the rendering, remains consistent across all supported platforms
- Really fast rendering and high antialiasing quality (analytical pixel coverage)
- Based on the robust AmanithVG SRE software rendering engine

Package content

SVGAssets package consists of the following `Assets` subfolders:

- `Anim`: animation files relative to the orc example scene
- `Atlases`: texture atlases and sprites generated by SVGAssets
- `Editor`: Unity editor scripts
- `Plugins`: AmanithSVG native plugins for all supported platforms
- `Resources`: general configuration file along with font resources
- `Scenes`: example scenes
- `Scripts`: scripts for the manipulation of SVG documents, viewports, drawing surfaces, sprites
- `SVGFiles`: all SVG files used by example scenes

SVG Features compatibility

AmanithSVG native plugin implements the rendering of static SVG files, supporting a limited set of SVG elements and attributes, as follow.

Index:

- Elements (complete list)
  - Document structure elements
  - Styling elements
  - Paths elements
  - Basic shapes elements
  - Text elements
  - Painting elements
  - Color elements
  - Gradients and patterns elements
  - Clipping, masking and compositing elements
  - Filter effects elements
  - Interactivity elements
  - Linking elements
  - Scripting elements
  - Animation elements
  - Fonts elements
  - Metadate elements
  - Extensibility elements
- Attributes
  - Animation event attributes
  - Animation attribute target attributes
  - Animation timing attributes
  - Animation value attributes
  - Animation addition attributes
  - Conditional processing attributes
  - Core attributes
  - Document event attributes
  - Filter primitive attributes
  - Graphical event attributes
  - Presentation attributes
  - Transfer function element attributes
  - Xlink attributes

Elements

Elements	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG Full	AmanithSVG Lite	Notes
<svg>	Yes	Yes	Yes	Yes	-
<g>	Yes	Yes	Yes	Yes	-
<defs>	Yes	Yes	Yes	Yes	-
<desc>	Yes	Yes	Yes	Yes	-
<title>	Yes	Yes	Yes	Yes	-
<symbol>	Yes	No	Yes	Yes	-
<use>	Yes	Yes	Yes	Yes	-
<image>	Yes	No	Partial	No	Embedded base64 encoded JPEG and PNG images (no 16bit formats), and through `svgtResourceSet` API.
<switch>	Yes	Yes	Partial	Partial	'requiredExtensions' attribute is not supported.
<style>	Yes	No	No	No	-
<path>	Yes	Yes	Yes	Yes	-
<rect>	Yes	Yes	Yes	Yes	-
<circle>	Yes	Yes	Yes	Yes	-
<ellipse>	Yes	Yes	Yes	Yes	-
<line>	Yes	Yes	Yes	Yes	-
<polyline>	Yes	Yes	Yes	Yes	-
<polygon>	Yes	Yes	Yes	Yes	-
<text>	Yes	Yes	Yes	Yes	-
<tspan>	Yes	Yes	Yes	Yes	-
<tref>	Yes	No	No	No	-
<textPath>	Yes	No	Yes	Yes	-
<altGlyph>	Yes	No	No	No	-
<altGlyphDef>	Yes	No	No	No	-
<altGlyphItem>	Yes	No	No	No	-
<glyphRef>	Yes	No	No	No	-
<textArea>	No	Yes	No	No	-
<tbreak>	No	Yes	No	No	-
<marker>	Yes	No	No	No	-
<solidColor>	No	Yes	Yes	Yes	-

Elements	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG Full	AmanithSVG Lite	Notes
<color-profile>	Yes	No	No	No	-
<linearGradient>	Yes	Yes	Yes	No	Solid color equal to the first stop is used instead, in AmanithSVG Lite.
<radialGradient>	Yes	Yes	Yes	No	Solid color equal to the first stop is used instead, in AmanithSVG Lite.
<stop>	Yes	Yes	Yes	No	Only the first stop is used in AmanithSVG Lite.
<pattern>	Yes	No	Yes	No	Solid white is used instead, in AmanithSVG Lite.
<clipPath>	Yes	No	Partial	Partial	<clipPath> elements containing a single child and a maximum of 31 concurrent active <clipPath> only are supported.
<mask>	Yes	No	Yes	No	-
<filter>	Yes	No	Yes	No	'filterRes' attribute is not supported.
<feDistantLight>	Yes	No	Yes	No	-
<fePointLight>	Yes	No	Yes	No	-
<feSpotLight>	Yes	No	Yes	No	-
<feBlend>	Yes	No	Yes	No	-
<feColorMatrix>	Yes	No	Yes	No	-
<feComponentTransfer>	Yes	No	No	No	-
<feFuncR>	Yes	No	No	No	-
<feFuncG>	Yes	No	No	No	-
<feFuncB>	Yes	No	No	No	-
<feFuncA>	Yes	No	No	No	-
<feComposite>	Yes	No	Yes	No	-
<feConvolveMatrix>	Yes	No	No	No	-
<feDiffuseLighting>	Yes	No	Yes	No	'kernelUnitLength' attribute is not supported.
<feDisplacementMap>	Yes	No	Yes	No	-
<feFlood>	Yes	No	Yes	No	-
<feGaussianBlur>	Yes	No	Yes	No	-
<feImage>	Yes	No	No	No	-
<feMerge>	Yes	No	Yes	No	-
<feMergeNode>	Yes	No	Yes	No	-

Elements	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG Full	AmanithSVG Lite	Notes
<feMorphology>	Yes	No	Yes	No	-
<feOffset>	Yes	No	Yes	No	-
<feSpecularLighting>	Yes	No	Partial	No	'kernelUnitLength' attribute is not supported.
<feTile>	Yes	No	No	No	-
<feTurbulence>	Yes	No	Yes	No	-
<cursor>	Yes	No	No	No	-
<a>	Yes	Yes	Yes	Yes	-
<view>	Yes	No	No	No	-
<script>	Yes	Yes	No	No	-
<animate>	Yes	Yes	No	No	-
<set>	Yes	Yes	No	No	-
<animateMotion>	Yes	Yes	No	No	-
<animateColor>	Yes	Yes	No	No	-
<animateTransform>	Yes	Yes	No	No	-
<mpath>	Yes	Yes	No	No	-
<font>	Yes	Yes	No	No	-
<glyph>	Yes	Yes	No	No	-
<missing-glyph>	Yes	Yes	No	No	-
<hkern>	Yes	Yes	No	No	-
<vkern>	Yes	Yes	No	No	-
<font-face>	Yes	Yes	No	No	-
<font-face-src>	Yes	Yes	No	No	-
<font-face-uri>	Yes	Yes	No	No	-
<font-face-format>	Yes	No	No	No	-
<font-face-name>	Yes	No	No	No	-
<metadata>	Yes	Yes	No	No	-
<foreignObject>	Yes	Yes	No	No	-

Attributes

Note for AmanithSVG Lite: although indicated as supported, attributes are intended to be truly supported only if the element to which they refer is itself supported.

Animation event attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
onbegin	Yes	No	No	-
onend	Yes	No	No	-
onrepeat	Yes	No	No	-
onload	Yes	No	No	-

Animation attribute target attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
attributeType	Yes	Yes	No	-
attributeName	Yes	Yes	No	-

Animation timing attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
begin	Yes	Yes	No	-
dur	Yes	Yes	No	-
end	Yes	Yes	No	-
min	Yes	Yes	No	-
max	Yes	Yes	No	-
restart	Yes	Yes	No	-
repeatCount	Yes	Yes	No	-
repeatDur	Yes	Yes	No	-
fill	Yes	Yes	No	-

Animation value attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
calcMode	Yes	Yes	No	-
values	Yes	Yes	No	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
keyTimes	Yes	Yes	No	-
keySplines	Yes	Yes	No	-
from	Yes	Yes	No	-
to	Yes	Yes	No	-
by	Yes	Yes	No	-

Animation addition attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
additive	Yes	Yes	No	-
accumulate	Yes	Yes	No	-

Conditional processing attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
requiredFeatures	Yes	Yes	Yes	-
requiredExtensions	Yes	Yes	No	-
requiredFormats	No	Yes	No	-
requiredFonts	No	Yes	No	-
systemLanguage	Yes	Yes	Yes	-

Core attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
id	Yes	Yes	Yes	-
xml:base	Yes	Yes	No	-
xml:lang	Yes	Yes	No	-
xml:space	Yes	Yes	Yes	-
xml:id	No	Yes	Yes	-

Document event attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
onunload	Yes	No	No	-
onabort	Yes	No	No	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
onerror	Yes	No	No	-
onresize	Yes	No	No	-
onscroll	Yes	No	No	-
onzoom	Yes	No	No	-

Filter primitive attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
x	Yes	No	No	-
y	Yes	No	No	-
width	Yes	No	No	-
height	Yes	No	No	-
result	Yes	No	Yes	-

Graphical event attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
onfocusin	Yes	No	No	-
onfocusout	Yes	No	No	-
onactivate	Yes	No	No	-
onclick	Yes	No	No	-
onmousedown	Yes	No	No	-
onmouseup	Yes	No	No	-
onmouseover	Yes	No	No	-
onmousemove	Yes	No	No	-
onmouseout	Yes	No	No	-
onload	Yes	No	No	-

Presentation attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
alignment-baseline	Yes	No	No	-
audio-level	No	Yes	No	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
baseline-shift	Yes	No	No	-
buffered-rendering	No	Yes	No	-
clip	Yes	No	No	-
clip-path	Yes	No	Yes	-
clip-rule	Yes	No	Yes	-
color	Yes	Yes	Yes	-
color-interpolation	Yes	No	Partial	supported on <mask> element only
color-interpolation-filters	Yes	No	Yes	-
color-profile	Yes	No	No	-
color-rendering	Yes	Yes	No	-
cursor	Yes	No	No	-
direction	Yes	Yes	No	-
display	Yes	Yes	Partial	inline, none, inherit; other values are treated as inline
display-align	No	Yes	No	-
dominant-baseline	Yes	No	No	-
enable-background	Yes	No	No	-
fill	Yes	Yes	Yes	-
fill-opacity	Yes	Yes	Yes	-
fill-rule	Yes	Yes	Yes	-
filter	Yes	No	Yes	-
flood-color	Yes	No	Yes	-
flood-opacity	Yes	No	Yes	-
font-family	Yes	Yes	Yes	-
font-size	Yes	Yes	Yes	-
font-size-adjust	Yes	No	No	-
font-stretch	Yes	No	Yes	-
font-style	Yes	Yes	Yes	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
font-variant	Yes	No	Yes	-
font-weight	Yes	Yes	Yes	-
glyph-orientation-horizontal	Yes	No	No	-
glyph-orientation-vertical	Yes	No	Yes	non-deprecated values only
image-rendering	Yes	Yes	No	-
kerning	Yes	No	No	-
letter-spacing	Yes	No	Yes	-
lighting-color	Yes	No	Yes	-
line-increment	No	Yes	No	-
marker-end	Yes	No	No	-
marker-mid	Yes	No	No	-
marker-start	Yes	No	No	-
mask	Yes	No	Yes	-
opacity	Yes	Yes	Yes	-
overflow	Yes	No	Partial	visible, hidden, on <svg> and <symbol> elements only
pointer-events	Yes	Yes	No	-
shape-rendering	Yes	Yes	No	-
solid-color	No	Yes	Yes	-
solid-opacity	No	Yes	Yes	-
stop-color	Yes	Yes	Yes	-
stop-opacity	Yes	Yes	Yes	-
stroke	Yes	Yes	Yes	-
stroke-dasharray	Yes	Yes	Yes	-
stroke-dashoffset	Yes	Yes	Yes	-
stroke-linecap	Yes	Yes	Yes	-
stroke-linejoin	Yes	Yes	Yes	-
stroke-miterlimit	Yes	Yes	Yes	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
stroke-opacity	Yes	Yes	Yes	-
stroke-width	Yes	Yes	Yes	-
text-align	No	Yes	No	-
text-anchor	Yes	Yes	Yes	-
text-decoration	Yes	No	No	-
text-rendering	Yes	Yes	No	-
unicode-bidi	Yes	Yes	No	-
vector-effect	No	Yes	Yes	-
viewport-fill	No	Yes	No	-
viewport-opacity	No	Yes	No	-
visibility	Yes	Yes	Yes	-
word-spacing	Yes	No	No	-
writing-mode	Yes	No	Yes	-

Transfer function element attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
type	Yes	No	No	-
tableValues	Yes	No	No	-
slope	Yes	No	No	-
intercept	Yes	No	No	-
amplitude	Yes	No	No	-
exponent	Yes	No	No	-
offset	Yes	No	No	-

Xlink attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
xlink:href	Yes	Yes	Yes	-
xlink:show	Yes	Yes	No	-
xlink:actuate	Yes	Yes	No	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
xlink:type	Yes	Yes	No	-
xlink:role	Yes	Yes	No	-
xlink:arcrole	Yes	Yes	No	-
xlink:title	Yes	Yes	No	-

Document structure elements

<svg> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Document event attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
x	Yes	No	Yes	-
y	Yes	No	Yes	-
width	Yes	Yes	Yes	-
height	Yes	Yes	Yes	-
viewBox	Yes	Yes	Yes	-
preserveAspectRatio	Yes	Yes	Yes	-
zoomAndPan	Yes	Yes	No	-
version	Yes	Yes	No	-
baseProfile	Yes	Yes	No	-
contentScriptType	Yes	Yes	No	-
contentStyleType	Yes	No	No	-
snapshotTime	No	Yes	No	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
playbackOrder	No	Yes	No	-
timelineBegin	No	Yes	No	-
focusable	No	Yes	No	-

<g> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-

<defs> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-

<desc> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
class	Yes	No	No	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
style	Yes	No	Yes	-

<title> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-

<symbol> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes		No	-
style	Yes		Yes	-
viewBox	Yes		Yes	-
preserveAspectRatio	Yes		Yes	-

<use> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
Xlink attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-
x	Yes	Yes	Yes	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
y	Yes	Yes	Yes	-
width	Yes	No	Yes	-
height	Yes	No	Yes	-
xlink:href	Yes	Yes	Yes	-

<image> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
Xlink attributes				-
class	Yes		No	-
style	Yes		Yes	-
externalResourcesRequired	Yes		No	-
preserveAspectRatio	Yes		Yes	-
transform	Yes		Yes	-
x	Yes		Yes	-
y	Yes		Yes	-
width	Yes		Yes	-
height	Yes		Yes	-
xlink:href	Yes		Partial	Embedded base64 encoded JPEG and PNG images (no 16bit formats), and through `svgtResourceSet` API.

<switch> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-

Styling elements

<style> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
type	Yes			-
media	Yes			-
title	Yes			-

Paths elements

<path> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
d	Yes	Yes	Yes	-
pathLength	Yes	Yes	No	-

Basic shapes elements

<rect> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-
x	Yes	Yes	Yes	-
y	Yes	Yes	Yes	-
width	Yes	Yes	Yes	-
height	Yes	Yes	Yes	-
rx	Yes	Yes	Yes	-
ry	Yes	Yes	Yes	-

<circle> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-
cx	Yes	Yes	Yes	-
cy	Yes	Yes	Yes	-
r	Yes	Yes	Yes	-

<ellipse> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-
cx	Yes	Yes	Yes	-
cy	Yes	Yes	Yes	-
rx	Yes	Yes	Yes	-
ry	Yes	Yes	Yes	-

<line> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
transform	Yes	Yes	Yes	-
x1	Yes	Yes	Yes	-
y1	Yes	Yes	Yes	-
x2	Yes	Yes	Yes	-
y2	Yes	Yes	Yes	-

<polyline> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-
points	Yes	Yes	Yes	-

<polygon> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-
points	Yes	Yes	Yes	-

Text elements

<text> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-
lengthAdjust	Yes	No	No	-
x	Yes	Yes	Yes	-
y	Yes	Yes	Yes	-
dx	Yes	No	Yes	-
dy	Yes	No	Yes	-
rotate	Yes	Yes	Yes	-
textLength	Yes	No	No	-
editable	No	Yes	No	-

<tspan> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
x	Yes	No	Yes	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
y	Yes	No	Yes	-
dx	Yes	No	Yes	-
dy	Yes	No	Yes	-
rotate	Yes	No	Yes	-
textLength	Yes	No	No	-
lengthAdjust	Yes	No	No	-

<tref> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
Xlink attributes				-
class	Yes			-
style	Yes			-
externalResourcesRequired	Yes			-
xlink:href	Yes			-

<textPath> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
Xlink attributes				-
class	Yes		No	-
style	Yes		Yes	-
externalResourcesRequired	Yes		No	-
xlink:href	Yes		Yes	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
startOffset	Yes		Yes	-
method	Yes		No	-
spacing	Yes		No	-

<altGlyph> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
Xlink attributes				-
class	Yes			-
style	Yes			-
externalResourcesRequired	Yes			-
x	Yes			-
y	Yes			-
dx	Yes			-
dy	Yes			-
glyphRef	Yes			-
format	Yes			-
rotate	Yes			-
xlink:href	Yes			-

<altGlyphDef> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-

<altGlyphItem> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-

<glyphRef> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Xlink attributes				-
class	Yes			-
style	Yes			-
x	Yes			-
y	Yes			-
dx	Yes			-
dy	Yes			-
glyphRef	Yes			-
format	Yes			-
xlink:href	Yes			-

<textArea> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
x		Yes		-
y		Yes		-
width		Yes		-
height		Yes		-
editable		Yes		-
format		Yes		-

<tbreak> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-

Painting elements

<marker> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
class	Yes			-
style	Yes			-
externalResourcesRequired	Yes			-
viewBox	Yes			-
preserveAspectRatio	Yes			-
refX	Yes			-
refY	Yes			-
markerUnits	Yes			-
markerWidth	Yes			-
markerHeight	Yes			-
orient	Yes			-

<solidColor> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
solid-color		Yes	Yes	-
solid-opacity		Yes	Yes	-

Color elements

<color-profile> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Xlink attributes				-
local	Yes			-
name	Yes			-
rendering-intent	Yes			-
xlink:href	Yes			-

Gradients and patterns elements

<linearGradient> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Xlink attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-
x1	Yes	Yes	Yes	-
y1	Yes	Yes	Yes	-
x2	Yes	Yes	Yes	-
y2	Yes	Yes	Yes	-
gradientUnits	Yes	Yes	Yes	-
gradientTransform	Yes	No	Yes	-
spreadMethod	Yes	No	Yes	-
xlink:href	Yes	No	Yes	-

<radialGradient> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Xlink attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-
cx	Yes	Yes	Yes	-
cy	Yes	Yes	Yes	-
r	Yes	Yes	Yes	-
fx	Yes	No	Yes	-
fy	Yes	No	Yes	-
gradientUnits	Yes	Yes	Yes	-
gradientTransform	Yes	No	Yes	-
spreadMethod	Yes	No	Yes	-
xlink:href	Yes	No	Yes	-

<stop> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
offset	Yes	Yes	Yes	-
stop-color	Yes	Yes	Yes	-
stop-opacity	Yes	Yes	Yes	-

<pattern> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Presentation attributes				-
Xlink attributes				-
class	Yes		No	-
style	Yes		Yes	-
externalResourcesRequired	Yes		No	-
viewBox	Yes		Yes	-
preserveAspectRatio	Yes		Yes	-
x	Yes		Yes	-
y	Yes		Yes	-
width	Yes		Yes	-
height	Yes		Yes	-
patternUnits	Yes		Yes	-
patternContentUnits	Yes		Yes	-
patternTransform	Yes		Yes	-
xlink:href	Yes		Yes	-

Clipping, masking and compositing elements

<clipPath> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Presentation attributes				-
class	Yes		No	-
style	Yes		Yes	-
externalResourcesRequired	Yes		No	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
transform	Yes		Yes	-
clipPathUnits	Yes		Yes	-

<mask> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Presentation attributes				-
class	Yes		No	-
style	Yes		Yes	-
externalResourcesRequired	Yes		No	-
x	Yes		Yes	-
y	Yes		Yes	-
width	Yes		Yes	-
height	Yes		Yes	-
maskUnits	Yes		Yes	-
maskContentUnits	Yes		Yes	-

Filter effects elements

<filter> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Xlink attributes				-
class	Yes		No	-
style	Yes		Yes	-
externalResourcesRequired	Yes		No	-
x	Yes		Yes	-
y	Yes		Yes	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
width	Yes		Yes	-
height	Yes		Yes	-
filterRes	Yes		No	-
filterUnits	Yes		Yes	-
primitiveUnits	Yes		Yes	-
xlink:href	Yes		Yes	-

<feDistantLight> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
azimuth	Yes		Yes	-
elevation	Yes		Yes	-

<fePointLight> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
x	Yes		Yes	-
y	Yes		Yes	-
z	Yes		Yes	-

<feSpotLight> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
x	Yes		Yes	-
y	Yes		Yes	-
z	Yes		Yes	-
pointsAtX	Yes		Yes	-
pointsAtY	Yes		Yes	-
pointsAtZ	Yes		Yes	-
specularExponent	Yes		Yes	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
limitingConeAngle	Yes		Yes	-

<feBlend> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Filter primitive attributes				-
class	Yes		No	-
style	Yes		Yes	-
in	Yes		Partial	SourceGraphic, SourceAlpha, <filter-primitive-reference> only
in2	Yes		Partial	SourceGraphic, SourceAlpha, <filter-primitive-reference> only
mode	Yes		Yes	-

<feColorMatrix> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Filter primitive attributes				-
class	Yes		No	-
style	Yes		Yes	-
in	Yes		Partial	SourceGraphic, SourceAlpha, <filter-primitive-reference> only
type	Yes		Yes	-
values	Yes		Yes	-

<feComponentTransfer> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Filter primitive attributes				-
class	Yes			-
style	Yes			-
in	Yes			-

<feFuncR> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Transfer function element attributes				-
type	Yes			-
tableValues	Yes			-
slope	Yes			-
intercept	Yes			-
amplitude	Yes			-
exponent	Yes			-
offset	Yes			-

<feFuncG> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Transfer function element attributes				-
type	Yes			-
tableValues	Yes			-
slope	Yes			-
intercept	Yes			-
amplitude	Yes			-
exponent	Yes			-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
offset	Yes			-

<feFuncB> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Transfer function element attributes				-
type	Yes			-
tableValues	Yes			-
slope	Yes			-
intercept	Yes			-
amplitude	Yes			-
exponent	Yes			-
offset	Yes			-

<feFuncA> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Transfer function element attributes				-
type	Yes			-
tableValues	Yes			-
slope	Yes			-
intercept	Yes			-
amplitude	Yes			-
exponent	Yes			-
offset	Yes			-

<feComposite> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Presentation attributes				-
Filter primitive attributes				-
class	Yes		No	-
style	Yes		Yes	-
in	Yes		Partial	SourceGraphic, SourceAlpha, <filter-primitive-reference> only
in2	Yes		Partial	SourceGraphic, SourceAlpha, <filter-primitive-reference> only
operator	Yes		Yes	-
k1	Yes		Yes	-
k2	Yes		Yes	-
k3	Yes		Yes	-
k4	Yes		Yes	-

<feConvolveMatrix> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Filter primitive attributes				-
class	Yes			-
style	Yes			-
in	Yes			-
order	Yes			-
kernelMatrix	Yes			-
divisor	Yes			-
bias	Yes			-
targetX	Yes			-
targetY	Yes			-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
edgeMode	Yes			-
kernelUnitLength	Yes			-
preserveAlpha	Yes			-

<feDiffuseLighting> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Filter primitive attributes				-
class	Yes		No	-
style	Yes		Yes	-
in	Yes		Partial	SourceGraphic, SourceAlpha, <filter-primitive-reference> only
surfaceScale	Yes		Yes	-
diffuseConstant	Yes		Yes	-
kernelUnitLength	Yes		No	-

<feDisplacementMap> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Filter primitive attributes				-
class	Yes		No	-
style	Yes		Yes	-
in	Yes		Partial	SourceGraphic, SourceAlpha, <filter-primitive-reference> only

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
in2	Yes		Partial	SourceGraphic, SourceAlpha, <filter-primitive-reference> only
scale	Yes		Yes	-
xChannelSelector	Yes		Yes	-
yChannelSelector	Yes		Yes	-

<feFlood> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Filter primitive attributes				-
class	Yes		No	-
style	Yes		Yes	-
flood-color	Yes		Yes	-
flood-opacity	Yes		Yes	-

<feGaussianBlur> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Filter primitive attributes				-
class	Yes		No	-
style	Yes		Yes	-
in	Yes		Partial	SourceGraphic, SourceAlpha, <filter-primitive-reference> only
stdDeviation	Yes		Yes	-

<feImage> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Filter primitive attributes				-
Xlink attributes				-
class	Yes			-
style	Yes			-
externalResourcesRequired	Yes			-
preserveAspectRatio	Yes			-
xlink:href	Yes			-

<feMerge> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Filter primitive attributes				-
class	Yes		No	-
style	Yes		Yes	-

<feMergeNode> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
in	Yes		Partial	SourceGraphic, SourceAlpha, <filter-primitive-reference> only

<feMorphology> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Presentation attributes				-
Filter primitive attributes				-
class	Yes		No	-
style	Yes		Yes	-
in	Yes		Partial	SourceGraphic, SourceAlpha, <filter-primitive-reference> only
operator	Yes		Yes	-
radius	Yes		Yes	-

<feOffset> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Filter primitive attributes				-
class	Yes		No	-
style	Yes		Yes	-
in	Yes		Partial	SourceGraphic, SourceAlpha, <filter-primitive-reference> only
dx	Yes		Yes	-
dy	Yes		Yes	-

<feSpecularLighting> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Filter primitive attributes				-
class	Yes		No	-
style	Yes		Yes	-
in	Yes		Partial	SourceGraphic, SourceAlpha, <filter-primitive-reference> only
surfaceScale	Yes		Yes	-
specularConstant	Yes		Yes	-
specularExponent	Yes		Yes	-
kernelUnitLength	Yes		No	-

<feTile> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Filter primitive attributes				-
class	Yes			-
style	Yes			-
in	Yes			-

<feTurbulence> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
Filter primitive attributes				-
class	Yes		No	-
style	Yes		Yes	-
baseFrequency	Yes		Yes	-
numOctaves	Yes		Yes	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
seed	Yes		Yes	-
stitchTiles	Yes		Yes	-
type	Yes		Yes	-

Interactivity elements

<cursor> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Xlink attributes				-
externalResourcesRequired	Yes			-
x	Yes			-
y	Yes			-
xlink:href	Yes			-

Linking elements

<a> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
Xlink attributes				-
class	Yes	No	No	-
style	Yes	No	Yes	-
externalResourcesRequired	Yes	No	No	-
transform	Yes	Yes	Yes	-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
xlink:href	Yes	Yes	No	-
xlink:show	Yes	Yes	No	-
xlink:actuate	Yes	Yes	No	-
target	Yes	Yes	No	-

<view> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
externalResourcesRequired	Yes	Yes		-
viewBox	Yes	Yes		-
preserveAspectRatio	Yes	Yes		-
zoomAndPan	Yes	Yes		-
viewTarget	Yes	Yes		-

Scripting elements

<script> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Xlink attributes				-
externalResourcesRequired	Yes	No		-
type	Yes	Yes		-
xlink:href	Yes	Yes		-

Animation elements

<animate> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Animation event attributes				-
Xlink attributes				-
Animation attribute target attributes				-
Animation timing attributes				-
Animation value attributes				-
Animation addition attributes				-
Presentation attributes				-
externalResourcesRequired	Yes	No		-

<set> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Animation event attributes				-
Xlink attributes				-
Animation attribute target attributes				-
Animation timing attributes				-
externalResourcesRequired	Yes	No		-
to	Yes	Yes		-

<animateMotion> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Animation event attributes				-
Xlink attributes				-
Animation timing attributes				-
Animation value attributes				-
Animation addition attributes				-
externalResourcesRequired	Yes	No		-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
path	Yes	Yes		-
keyPoints	Yes	Yes		-
rotate	Yes	Yes		-
origin	Yes	Yes		-

<animateColor> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Animation event attributes				-
Xlink attributes				-
Animation attribute target attributes				-
Animation timing attributes				-
Animation value attributes				-
Animation addition attributes				-
Presentation attributes				-
externalResourcesRequired	Yes	No		-

<animateTransform> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Animation event attributes				-
Xlink attributes				-
Animation attribute target attributes				-
Animation timing attributes				-
Animation value attributes				-
Animation addition attributes				-
externalResourcesRequired	Yes	No		-
type	Yes	Yes		-

<mpath> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Xlink attributes				-
externalResourcesRequired	Yes	No		-
xlink:href	Yes	Yes		-

Fonts elements

<font> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
class	Yes	No		-
style	Yes	No		-
externalResourcesRequired	Yes	No		-
horiz-origin-x	Yes	Yes		-
horiz-origin-y	Yes	No		-
horiz-adv-x	Yes	Yes		-
vert-origin-x	Yes	No		-
vert-origin-y	Yes	No		-
vert-adv-y	Yes	No		-

<glyph> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
class	Yes	No		-
style	Yes	No		-
d	Yes	Yes		-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
horiz-adv-x	Yes	Yes		-
vert-origin-x	Yes	No		-
vert-origin-y	Yes	No		-
vert-adv-y	Yes	No		-
unicode	Yes	Yes		-
glyph-name	Yes	Yes		-
orientation	Yes	No		-
arabic-form	Yes	Yes		-
lang	Yes	Yes		-

<missing-glyph> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Presentation attributes				-
class	Yes	No		-
style	Yes	No		-
d	Yes	Yes		-
horiz-adv-x	Yes	Yes		-
vert-origin-x	Yes	No		-
vert-origin-y	Yes	No		-
vert-adv-y	Yes	No		-

<hkern> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
u1	Yes	Yes		-
g1	Yes	Yes		-
u2	Yes	Yes		-
g2	Yes	Yes		-
k	Yes	Yes		-

<vkern> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
u1	Yes	Yes		-
g1	Yes	Yes		-
u2	Yes	Yes		-
g2	Yes	Yes		-
k	Yes	Yes		-

<font-face> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
font-family	Yes	Yes		-
font-style	Yes	Yes		-
font-variant	Yes	No		-
font-weight	Yes	Yes		-
font-stretch	Yes	No		-
font-size	Yes	No		-
unicode-range	Yes	Yes		-
units-per-em	Yes	Yes		-
panose-1	Yes	Yes		-
stemv	Yes	Yes		-
stemh	Yes	Yes		-
slope	Yes	Yes		-
cap-height	Yes	Yes		-
x-height	Yes	Yes		-
accent-height	Yes	Yes		-
ascent	Yes	Yes		-
descent	Yes	Yes		-
widths	Yes	Yes		-

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
bbox	Yes	Yes		-
ideographic	Yes	Yes		-
alphabetic	Yes	Yes		-
mathematical	Yes	Yes		-
hanging	Yes	Yes		-
v-ideographic	Yes	No		-
v-alphabetic	Yes	No		-
v-mathematical	Yes	No		-
v-hanging	Yes	No		-
underline-position	Yes	Yes		-
underline-thickness	Yes	Yes		-
strikethrough-position	Yes	Yes		-
strikethrough-thickness	Yes	Yes		-
overline-position	Yes	Yes		-
overline-thickness	Yes	Yes		-

<font-face-src> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-

<font-face-uri> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
Xlink attributes				-
xlink:href	Yes	Yes		-

<font-face-format> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
string	Yes			-

<font-face-name> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-
name	Yes			-

Metadate elements

<metadata> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Core attributes				-

Extensibility elements

<foreignObject> attributes

Attributes	SVG Full 1.1	SVG Tiny 1.2	AmanithSVG	Notes
Conditional processing attributes				-
Core attributes				-
Graphical event attributes				-
Presentation attributes				-
class	Yes	No		-
style	Yes	No		-
externalResourcesRequired	Yes	No		-
transform	Yes	Yes		-
x	Yes	Yes		-
y	Yes	Yes		-
width	Yes	Yes		-
height	Yes	Yes		-

C# layers

As discussed in the introduction, SVGAssets includes two different C# layers:

- a low level layer, that uses the Platform Invoke mechanism: PInvoke allows .NET code to call functions that are implemented within the AmanithSVG C/C++ native .dll (Windows), .dylib (MacOS X), .so (Linux). It is important to understand that the main benefit of this approach is in the ability to call the same C/C++ code on all .NET platforms. The low level API (i.e., the one implemented by the AmanithSVG native plugin) is exposed through the static class `AmanithSVG`.
- a high level layer that abstracts some details of the low level API

Here's a list of classes exposed by the high level C# layer:

- SVGError
- SVGAlign
- SVGMeetOrSlice
- SVGRenderingQuality
- SVGLogLevel
- SVGResourceType
- SVGResourceHint
- SVGColor
- SVGPoint
- SVGViewport
- SVGAspectRatio
- SVGDocument
- SVGSurface
- SVGPacker
- SVGAssetsConfig
- SVGResource
- SVGAssets

SVGError

The `SVGError` enumeration wraps the native SVGTErrorCode enum type, and it represents all possible error codes signaled by AmanithSVG functions.

SVGAlign

The `SVGAlign` enumeration wraps the native SVGTAspectRatioAlign enum type, and it defines the alignment method to use in case the aspect ratio of the source (document) viewport doesn't match the aspect ratio of the destination (drawing surface) viewport.

SVGMeetOrSlice

The `SVGMeetOrSlice` enumeration wraps the native SVGTAspectRatioMeetOrSlice enum type, and it defines the scaling method to use in case the aspect ratio of the source (document) viewport doesn't match the aspect ratio of the destination (drawing surface) viewport.

SVGRenderingQuality

The `SVGRenderingQuality` enumeration wraps the native SVGTRenderingQuality enum type, and it defines the quality (and speed) of SVG drawings.

SVGLogLevel

The `SVGLogLevel` enumeration wraps the native SVGTLogLevel enum type, and it defines the possible logging levels (e.g. errors, warnings or just informational messages).

SVGResourceType

The `SVGResourceType` enumeration wraps the native SVGTResourceType enum type, and it defines the type of each external resource provided to AmanithSVG.

SVGResourceHint

The `SVGResourceHint` enumeration wraps the native SVGTResourceHint enum type, and it defines all the possible hints for tagging external resources provided to AmanithSVG.

SVGColor

`SVGColor` is a simple class that exposes four (float) color components (Red, Green, Blue, Alpha), within the `[0.0f; 1.0f]` range. The class is so simple that it does not require further explanation.

SVGPoint

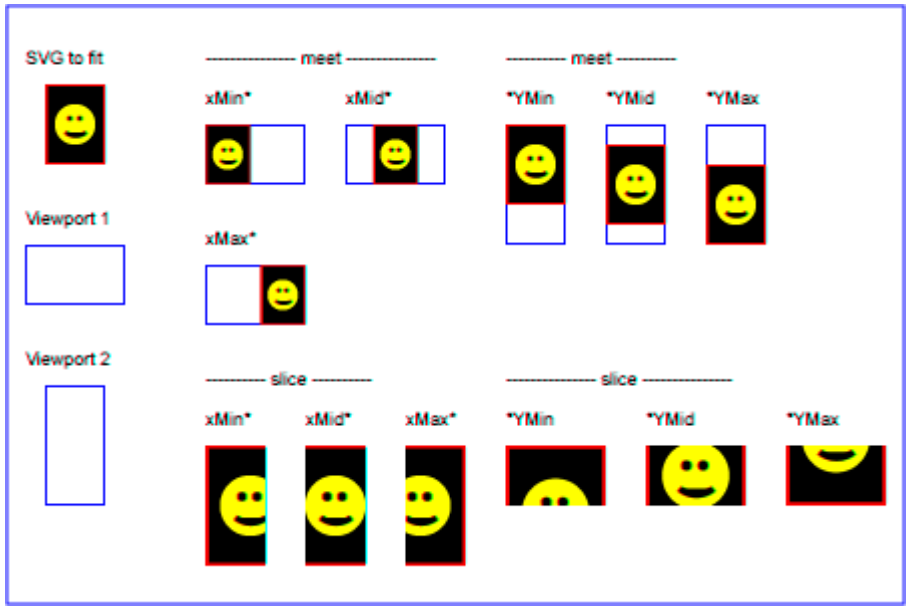
`SVGPoint` is a simple class representing a 2D point, the class exposes two (float) properties: abscissa (X) and ordinate (Y). The class is so simple that it does not require further explanation.

SVGViewport

A viewport represents a rectangular area, specified by its top/left corner, a width and an height. The positive x-axis points towards the right, the positive y-axis points down.
The class exposes four (float) properties: X, Y, Width, Height. The class is so simple that it does not require further explanation.

SVGAspectRatio

This class represents a couple of values, one taken from the `SVGAlign` enum type and the other taken from the `SVGMeetOrSlice` enum type.
Alignment indicates whether to force uniform scaling and, if so, the alignment method to use in case the aspect ratio of the source viewport doesn't match the aspect ratio of the destination viewport.



A brief visual example of possible alignments

SVGDocument

In order to render SVG contents, an optimized in-memory representation of SVG files must be created: a such representation is called an SVG document.
An SVG document can be created through `SVGAssets.CreateDocument` function, specifying the xml text.
The document will be parsed immediately, and the internal drawing tree will be created.
Once the document has been created, it can be drawn several times onto one (or more) drawing surface.
In order to draw a document, we must:

1. create a drawing surface using `SVGAssets.CreateSurface`
2. call surface `Draw` method, specifying the document to draw

Example:

```
string xml = File.ReadAllText("animals.svg");
SVGDocument doc = SVGAssets.CreateDocument(xml);
```

`SVGDocument` class exposes four main properties: `Handle`, `Width`, `Height`, `Viewport`, `AspectRatio`.

```
/*
   Document handle (read only); it's the handle created by the AmanithSVG native plugin.
*/
uint Handle;

/*
    SVG content itself optionally can provide information about the appropriate
    viewport region for the content via the 'width' and 'height' XML attributes
    on the outermost <svg> element.
    Use this (readonly) property to get the suggested viewport width, in pixels.

    It returns a negative number (i.e. an invalid width) in the following cases:
    - outermost element is not an <svg> element
```

```
    - outermost <svg> element doesn't have a 'width' attribute specified
    - outermost <svg> element has a negative 'width' attribute specified
      (e.g. width="-50")
    - outermost <svg> element has a 'width' attribute specified in relative measure
      units (i.e. em, ex, % percentage).
*/
float Width;

/*
    SVG content itself optionally can provide information about the appropriate
    viewport region for the content via the 'width' and 'height' XML attributes on
    the outermost <svg> element.
    Use this (readonly) property to get the suggested viewport height, in pixels.

    It returns a negative number (i.e. an invalid height) in the following cases:
    - outermost element is not an <svg> element
    - outermost <svg> element doesn't have a 'height' attribute specified
    - outermost <svg> element has a negative 'height' attribute specified
      (e.g. height="-30")
    - outermost <svg> element has a 'height' attribute specified in relative measure
      units (i.e. em, ex, % percentage).
*/
float Height;

/*
    The document (logical) viewport to map onto the destination (drawing surface)
    viewport. When an SVG document has been created through the SVGAssets.CreateDocument
    function, the initial value of its viewport is equal to the 'viewBox' attribute
    present in the outermost <svg> element.
*/
SVGViewport Viewport;

/*
    Viewport aspect ratio.
    The alignment parameter indicates whether to force uniform scaling and, if so,
    the alignment method to use in case the aspect ratio of the document viewport doesn't
    match the aspect ratio of the surface viewport.
*/
SVGAspectRatio AspectRatio;
```

SVGSurface

`SVGSurface` class represents a drawing surface.

A drawing surface is just a rectangular area made of pixels, where each pixel is represented internally by a 32bit unsigned integer.
A pixel is made of four 8-bit components: red, green, blue, alpha. Coordinate system is the same of SVG specifications: top/left pixel has coordinate (0, 0), with the positive x-axis pointing towards the right and the positive y-axis pointing down.

An `SVGSurface` can be created by using the `SVGAssets.CreateSurface` function, specifying its dimensions in pixels.

`SVGSurface` class exposes four main properties: `Handle`, `Width`, `Height`, `Viewport`.

```
/*
   Surface handle (read only); it's the handle created by the AmanithSVG native plugin.
*/
uint Handle;

/* Get current surface width, in pixels. NB: the property is readonly. */
uint Width;

/* Get current surface height, in pixels. NB: the property is readonly. */
uint Height;

/*
    The surface viewport (i.e. a drawing surface rectangular area), where to map the
    source document viewport.
    The combined use of surface and document viewport, induces a transformation matrix,
    that will be used to draw the whole SVG document. The induced matrix grants that
    the document viewport is mapped onto the surface viewport (respecting the specified
    alignment): all SVG content will be drawn accordingly.
*/
SVGViewport Viewport;
```

A drawing surface can be resized by calling the `Resize` method:

```
/*
    Resize the surface, specifying new dimensions in pixels.
    After resizing, the surface viewport will be reset to the whole surface.

    It returns SVGError.None if the operation was completed successfully, else
    an error code.
*/
SVGError Resize(uint newWidth, uint newHeight);
```

Once that an `SVGDocument` has been created, it can be rendered over a drawing surface using the following `SVGSurface` method:

```
/*
    Draw an SVG document, on this drawing surface.

    First the drawing surface is cleared if a valid (i.e. not null) clear color is
    provided. Then the specified document, if valid, is drawn.

    It returns SVGError.None if the operation was completed successfully, else
    an error code.
*/
SVGError Draw(SVGDocument document,
              SVGColor clearColor,
              SVGRenderingQuality renderingQuality);
```

Here's a complete example:

```
/* load an SVG file */
string xml = File.ReadAllText("animals.svg");
```

```
SVGDocument doc = SVGAssets.CreateDocument(xml);

/* create a 512x512 drawing surface */
SVGSurface srf = SVGAssets.CreateSurface(512, 512);

/* draw the SVG */
srf.Draw(doc, SVGColor.White, SVGRenderingQuality.Better);
< do something with surface pixels (e.g. upload pixels to a GPU texture) >

/* destroy surface and document */
doc.Dispose();
srf.Dispose();
```

Rendering result

By modifying the drawing surface viewport, it's possible to draw more than one SVG document on it. For example, suppose that we want to draw four different SVG documents within the same drawing surface (of course over 4 different non-overlapping sub-regions), the code would look as follow:

```
/* load SVG files */
SVGDocument doc1 = SVGAssets.CreateDocument(File.ReadAllText("car_coolant.svg"));
SVGDocument doc2 = SVGAssets.CreateDocument(File.ReadAllText("car_maintenance.svg"));
SVGDocument doc3 = SVGAssets.CreateDocument(File.ReadAllText("car_brake.svg"));
SVGDocument doc4 = SVGAssets.CreateDocument(File.ReadAllText("car_oil.svg"));

/* create a drawing surface */
SVGSurface srf = SVGAssets.CreateSurface(512, 512);
```

```
/* select an upper-left sub-region and draw the first SVG document */
srf.Viewport = new SVGViewport(0, 0, 256, 256);
srf.Draw(doc1, SVGColor.White, SVGRenderingQuality.Better);

/* select an upper-right sub-region and draw the second SVG document */
srf.Viewport = new SVGViewport(256, 0, 256, 256);
srf.Draw(doc2, null, SVGRenderingQuality.Better);

/* select the whole lower-left sub-region and draw the third SVG document */
srf.Viewport = new SVGViewport(0, 256, 256, 256);
srf.Draw(doc3, null, SVGRenderingQuality.Better);

/* select the whole lower-right sub-region and draw the fourth SVG document */
srf.Viewport = new SVGViewport(256, 256, 256, 256);
srf.Draw(doc4, null, SVGRenderingQuality.Better);

< do something with surface pixels (e.g. upload pixels to a GPU texture) >

/* destroy surface and documents */
doc1.Dispose();
doc2.Dispose();
doc3.Dispose();
doc4.Dispose();
srf.Dispose();
```

Usage of surface viewport

SVGPacker

Besides rendering single SVG documents over drawing surfaces, SVGAssets can pack the rendering of one or more SVG documents within one or more drawing surfaces, automatically. For each given SVG, you can choose to pack the whole document or instead to pack each first-level element separately.

The whole process consists of some steps that must be followed in this order:

- create an instance of `SVGPacker` class
- start the packing process using the `SVGPacker.Begin` method
- add one or more SVG document to the process, using the `SVGPacker.Add` method
- stop the packing process using the `SVGPacker.End` method
- for each atlas information (`SVGPackedBin`) returned by the `SVGPacker.End` method:
  - create a drawing surface with the same atlas dimensions, using the `SVGAssets.CreateSurface` function
  - draw the SVG documents/elements packed in the current atlas (`SVGPackedBin`), using the `SVGSurface.Draw` method

Here is a complete example code:

```
/* create an SVG document */
SVGDocument doc = SVGAssets.CreateDocument(File.ReadAllText("orc.svg"));

/* initialize the packing process: generate atlases with a maximum 512 pixels
   dimension and no additional scale */
SVGPacker packer = SVGAssets.CreatePacker(1.0f, 512, 1, false);

if (packer.Begin() == SVGError.None) {

    /* add the document to the packer, and get back the actual number of
       packed bounding boxes */
    uint[] info = packer.Add(doc, true, 1.0f);

    /* info[0] = number of collected bounding boxes
       info[1] = the actual number of packed bounding boxes */
    if ((info == null) || (info[1] != info[0])) {
        Console.WriteLine("Some SVG elements cannot be packed!");
        Console.WriteLine("Specified maximum texture dimensions do not allow to pack all
SVG elements.");
        /* close the packing process without doing anything */
        packer.End(false);
    }
    else {
        /* finalize the packing process and get back generated bins */
        SVGPackedBin[] bins = packer.End(true);

        for (int i = 0; i < bins.Length; ++i) {
```

```
            /* extract the bin (i.e. get atlas width, height and the number of packed
documents/elements) */
            SVGPackedBin bin = bins[i];
            /* create a drawing surface with the same atlas dimension */
            SVGSurface srf = SVGAssets.CreateSurface(bin.Width, bin.Height);
            /* draw all documents/elements over the drawing surface */
            srf.Draw(bin, SVGColor.Clear, SVGRenderingQuality.Better);
            /* if needed, process the drawing surface (e.g. upload its pixels on a GPU
texture) and then destroy it */
            < do something with surface pixels (e.g. upload pixels to a GPU texture) >
            /* destroy the surface */
            srf.Dispose();
        }
    }
}

/* destroy the document */
doc.Dispose();
```

If we run the example code using the `orc.svg` file, it will produce a single `452 x 108` bin, with the following packed rectangles:

	elemName	originalX	originalY	x	y	width	height	zOrder
rects[0]	head	5	9	0	0	133	105	9
rects[1]	body	26	80	133	0	93	97	7
rects[2]	sx_leg_down	75	191	226	0	48	61	2
rects[3]	dx_leg_down	41	191	274	0	38	63	4
rects[4]	sx_arm_up	95	102	312	0	38	54	5
rects[5]	dx_arm_up	19	98	350	0	38	54	6
rects[6]	dx_arm_down	19	141	388	0	32	68	8
rects[7]	sx_arm_down	99	139	420	0	32	68	0
rects[8]	dx_leg_up	43	147	312	54	38	54	3
rects[9]	sx_leg_up	70	146	350	54	38	54	1

Orc SVG

Orc atlas

It is important to note that the whole packing process, as well as the `SVGSurface.Draw(SVGPackedBin, SVGColor, SVGRenderingQuality)` method, is not affected by SVG documents viewports nor by drawing surfaces viewports. In other words, the viewport values set through `SVGDocument.Viewport` and `SVGSurface.Viewport` properties do not affect the behavior of packing processes and atlas generation.

SVGResource

This class represents an external (binary) resource, available for rendering SVG documents. Each resource is defined by a unique string identifier (typically the file name, without the path), a type (font or image), and a set of hints; all such properties must be specified in `SVGResource` constructor. `SVGResource` class cannot be instantiated directly, because it has an abstract `GetBytes` method that must be implemented according to the target platform (or development tool/environment): such method must return an in-memory representation of the underlying binary TTF / OTF / WOFF / JPEG / PNG file.

```
/* Constructor. */
abstract SVGResource(string id,
                     SVGResourceType type,
                     SVGResourceHint hints);

/* Get in-memory binary data for the resource. */
abstract byte[] GetBytes();

// Resource identifier.
```

```
string Id;
// Resource type.
SVGResourceType Type;
// Resource hints.
SVGResourceHint Hints;
```

Resources are provided to `SVGAssets` through an instance of a class derived from `SVGAssetsConfig`.

SVGAssetsConfig

This class provides configuration parameters, along with resources, to an `SVGAssets` instance.
Configuration parameters can be divided in two sub-sets:

- general user-defined parameters: curves quality, user-agent language, log level
- platform-specific parameters: device screen with/height in pixels, screen dpi

The class is `abstract` so it cannot be instantiated directly; it must be extended by implementing the mechanism to access resource files (defined by the `SVGResource` class).
In particular, each class that extends `SVGAssetsConfig` must implement the `ResourcesCount` and `GetResource` methods. See details:

```
/*
    Constructor, device screen properties must be supplied
    (with/height in pixels and dpi).
*/
abstract SVGAssetsConfig(uint screenWidth,
                         uint screenHeight,
                         float screenDpi);

/*
    Get the number of (external) resources provided by this configuration.
    It must be implemented by all derived classes.
*/
abstract int ResourcesCount();

/*
    Get a resource given an index.
    If the given index is less than zero or greater or equal to the value
    returned by ResourcesCount, a null resource is returned.

    It must be implemented by all derived classes.
*/
abstract SVGResource GetResource(int index);

/*
    Curves quality, used by AmanithSVG geometric kernel to approximate curves
    with straight line segments (flattening). Valid range is [1; 100], where 100
    represents the best quality.

    A zero or negative value means "keep the default one".
*/
float CurvesQuality;
```

```
/*
    User-agent language settings; a list of languages separated by
    semicolon (e.g. "en-US;en-GB;it;es"). It must not be empty.
*/
string Language;

/* Log level, if SVGLogLevel.None is specified, logging is disabled. */
SVGLogLevel LogLevel;

/* Log capacity, in characters; if zero is specified, logging is disabled. */
uint LogCapacity;
```

SVGAssets

`SVGAssets` is a static class through which SVGDocument, SVGSurface and SVGPacker instances can be created. In order to work properly, `SVGAssets` must be initialized in advance through its internal `Init` method:

```
/*
    Initialize AmanithSVG native plugin and load the given configuration.
    The provided settings include screen metrics, user-agent language and
    possible resources (fonts and images).

    It returns SVGError.None if the operation was completed successfully, else
    an error code.
*/
internal SVGError Init(SVGAssetsConfig config);
```

Here are the main methods:

```
/*
    Create a drawing surface, specifying its dimensions in pixels.

    Supplied dimensions should be positive numbers greater than zero, else
    a null instance will be returned.
*/
SVGSurface CreateSurface(uint width,
                         uint height);

/* Create and load an SVG document, specifying the whole XML string. */
SVGDocument CreateDocument(string xmlText);

/*
    Create an SVG packer, specifying a scale factor.

    Every collected SVG document/element will be packed into rectangular bins, whose
    dimensions won't exceed the specified 'maxTexturesDimension' in pixels.

    If true, 'pow2Textures' will force bins to have power-of-two dimensions.

    Each rectangle will be separated from the others by the specified 'border' in pixels.

    The specified 'scale' factor will be applied to all collected SVG documents/elements,
```

```
    in order to realize resolution-independent atlases.
*/
SVGPacker CreatePacker(float scale,
                       uint maxTexturesDimension,
                       uint border,
                       bool pow2Textures);

/* Get library version. */
string GetVersion();
```

SVGAssets can provide detailed information about error situations that can occur when parsing and rendering SVG content. For example SVGAssets can report if a closing tag did not match the opening one (or if some tag was not closed at all) when parsing an SVG document; or if a font needed for rendering has not been set. In case of error, it is possible to receive additional information on the conditions (which caused the error) in the form of text messages that SVGAssets can append to a given memory buffer.
Log level and the capacity (in characters) of the internal log buffer can be configured through the `SVGAssetsConfig` class, provided to `SVGAssets` initialization mechanism.

```
/* Get log content as a string. */
string LogGet();

/*
    Clear the log buffer set for the current thread.

    If specified by 'stopLogging' parameter, it is possible to stop SVGAssets from
    logging messages for the current thread. In this case, logging can be reactivated
    (for the calling thread) by calling this method with a 'false' parameter.
*/
SVGError LogClear(bool stopLogging);
```

From an application point of view, it could be useful to apppend some custom messages to the log buffer. One of the main purposes can be to insert debug messages, to better understand the interactions between the application and the AmanithSVG native plugin. In this way the application can keep track of all SVG-related activities, and the whole log buffer can be analyzed later (if needed). Here are the public methods, relative to logging, provided by `SVGAssets`:

```
/*
    Append an informational message to the log buffer
    set for the current thread.
*/
SVGError LogInfo(string message);

/*
    Append a warning message to the log buffer
    set for the current thread.
*/
SVGError LogWarning(string message);

/*
    Append an error message to the log buffer
    set for the current thread.
*/
```

```
SVGError LogError(string message);
```

AmanithSVG native plugins for Unity, requirements and limitations

AmanithSVG native plugins for Unity implement some specific methods to speed up the upload of pixels to textures as much as possible. These specific methods make direct use of native graphics APIs, which are different for each supported platform. In the detail, AmanithSVG native plugins for Unity support:

- Windows platform (x86, x86_64), Direct3D 11
- MacOS X platform (Apple Silicon, x64), Metal
- Linux platform (x86, x86_64), OpenGL
- iOS platform (Apple Silicon, x64), Metal
- Android platform (ARMv7, ARM64, x86, x86_64), OpenGL ES

Vulkan graphics API is not supported by AmanithSVG native plugins for Unity.

To make sure that SVGAssets can continue to be used even in the presence of unsupported native graphics APIs, several fallbacks have been implemented using standard (but slower) Unity calls (GetRawTextureData, GetUnsafeBufferPointerWithoutChecks, LoadRawTextureData).
The use of fast texture upload via native graphics APIs, or the fallbacks mechanism via Unity legacy functions, is controlled by the `Fast upload` flag of SVGTextureBehaviour, SVGBackgroundBehaviour, SVGAtlas classes.

Unity integration C# scripts

In addition to the C# layers already discussed, SVGAssets includes a set of Unity-specific classes that further simplify the use of various functionalities:

- SVGSurfaceUnity
- SVGResourceUnity
- SVGAssetsConfigUnity
- SVGAssetsUnity

Another set of scripts, finally, allows the integration of SVGAssets with Unity specific mechanisms (e.g. monobehaviour, sprites, texture atlas and so on):

- SVGTextureBehaviour
- SVGBackgroundBehaviour
- SVGCameraBehaviour
- SVGAtlas
- SVGSpriteLoaderBehaviour
- SVGUIAtlas
- SVGUISpriteLoaderBehaviour
- SVGCanvasBehaviour

It is mandatory to use `SVGAssetsUnity` class and not the basic `SVGAssets` one.

SVGSurfaceUnity

This class extends SVGSurface by providing methods to generate Unity textures out of surface pixels.

```
/*
    Create a 2D texture compatible with the drawing surface.

    NB: textures passed to Copy and CopyAndDestroy must be created
    through this function.
*/
Texture2D CreateCompatibleTexture(bool bilinearFilter,
                                  bool wrapRepeat,
                                  HideFlags hideFlags);


/*
    Copy drawing surface content into the specified Unity texture.
    NB: the given texture must have been created by the CreateCompatibleTexture
    method.

    It returns SVGError.None if the operation was completed successfully, else
    an error code.
*/
SVGError Copy(Texture2D texture);

/*
    Copy drawing surface content into the specified Unity texture, then destroy
    the native drawing surface. The SVGSurfaceUnity instance is not destroyed, but
    its native AmanithSVG counterpart it will. The result will be that every
    called method will fail silently.

    In order to ensure the maximum speed, the copy process will use native GPU
    platform-specific methods:

    - UpdateSubresource (Direct3D 11)
    - glTexSubImage2D (OpenGL and OpenGL ES)
    - replaceRegion (Metal)

    NB: the given texture must have been created by the CreateCompatibleTexture
    method.

    It returns SVGError.None if the operation was completed successfully, else
    an error code.
*/
SVGError CopyAndDestroy(Texture2D texture);
```

An `SVGSurfaceUnity` can be created by using the `SVGAssetsUnity.CreateSurface` function, specifying its dimensions in pixels.

SVGResourceUnity

This class extends SVGResource by providing the mandatory `GetBytes` method to get an in-memory representation of the underlying binary TTF / OTF / WOFF / JPEG / PNG file. The constructor takes a TextAsset and sets the given unique string identifier, the type (font or image) and hints. The access to the underlying binary data is implemented by using the `TextAsset.bytes` property.

SVGAssetsConfigUnity

This class extends SVGAssetsConfig by implementing an internal list of `SVGResourceUnity` resources. Items within this list can be added, moved and deleted. Most importantly this class implements the mandatory `ResourcesCount` and `GetResource` methods from `SVGAssetsConfig` class, in order to get the resources.

SVGAssetsUnity

SVGAssetsUnity is a static class through which SVGDocument, SVGSurfaceUnity and SVGPacker instances can be created. It extends SVGAssets class by implementing additional methods specific for Unity:

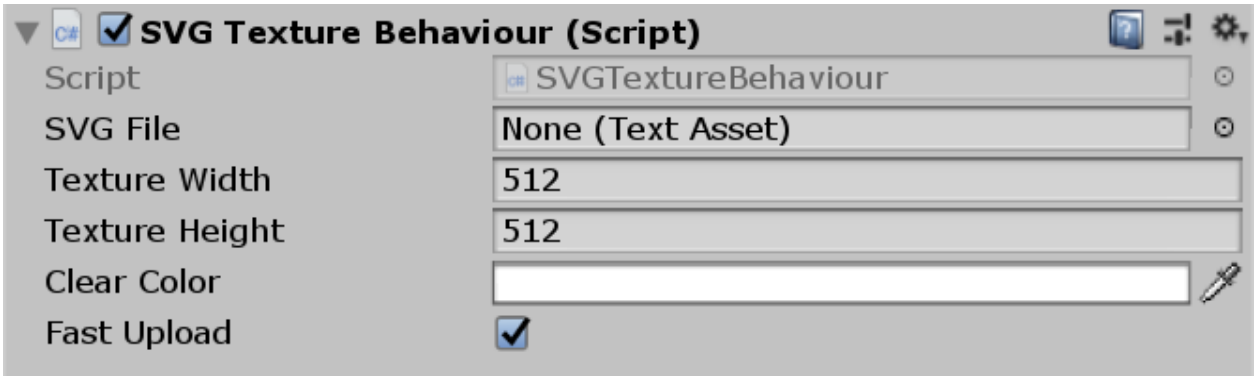
```
/* Get screen resolution width, in pixels. */
uint ScreenWidth;

/* Get screen resolution height, in pixels. */
uint ScreenHeight;

/* Get screen dpi. */
float ScreenDpi;

/*
    Create a drawing surface, specifying its dimensions in pixels.

    Supplied dimensions should be positive numbers greater than zero, else
    a null instance will be returned.
*/
SVGSurfaceUnity CreateSurface(uint width,
                              uint height);

/*
    Create and load an SVG document, specifying the whole XML string.
    If supplied XML string is null or empty, a null instance will be returned.
*/
SVGDocument CreateDocument(string xmlText);

/*
    Create an SVG packer, specifying a scale factor.

    Every collected SVG document/element will be packed into rectangular bins,
    whose dimensions won't exceed the specified 'maxTexturesDimension' in pixels.

    If true, 'pow2Textures' will force bins to have power-of-two dimensions.
    Each rectangle will be separated from the others by the specified 'border' in pixels.

    The specified 'scale' factor will be applied to all collected SVG documents/elements,
    in order to realize resolution-independent atlases.
```

```
*/
SVGPacker CreatePacker(float scale,
                       uint maxTexturesDimension,
                       uint border,
                       bool pow2Textures);

/*
    Create an SVG packer, specifying a scale factor.

    Every collected SVG document/element will be packed into rectangular bins.

    Each rectangle will be separated from the others by the specified 'border' in pixels.

    The specified 'scale' factor will be applied to all collected SVG documents/elements,
    in order to realize resolution-independent atlases.

    NB: maximum dimension of textures is auto-detected.
*/
SVGPacker CreatePacker(float scale,
                       uint border);

/*
    Create a sprite out of the given texture.
    The sprite will use the specified rectangular section of the
    texture, and it will have the provided pivot.
*/
Sprite CreateSprite(Texture2D texture,
                    Rect rect,
                    Vector2 pivot);
```

It is mandatory to use `SVGAssetsUnity` class and not the basic `SVGAssets` one.

SVGTextureBehaviour

This monobehaviour script performs a single SVG file rendering on a Texture2D.
It takes in input the SVG file (as a TextAsset), the texture dimensions and a clear color. Such color will be used to clear the surface before to start the rendering. The rendering is performed in the Start function, that assigns the generated texture to the `renderer.sharedMaterial.mainTexture` field of the associated GameObject. The `Fast upload` flag enables the native update of the texture instead to use the (slower) legacy LoadRawTextureData + Apply method.

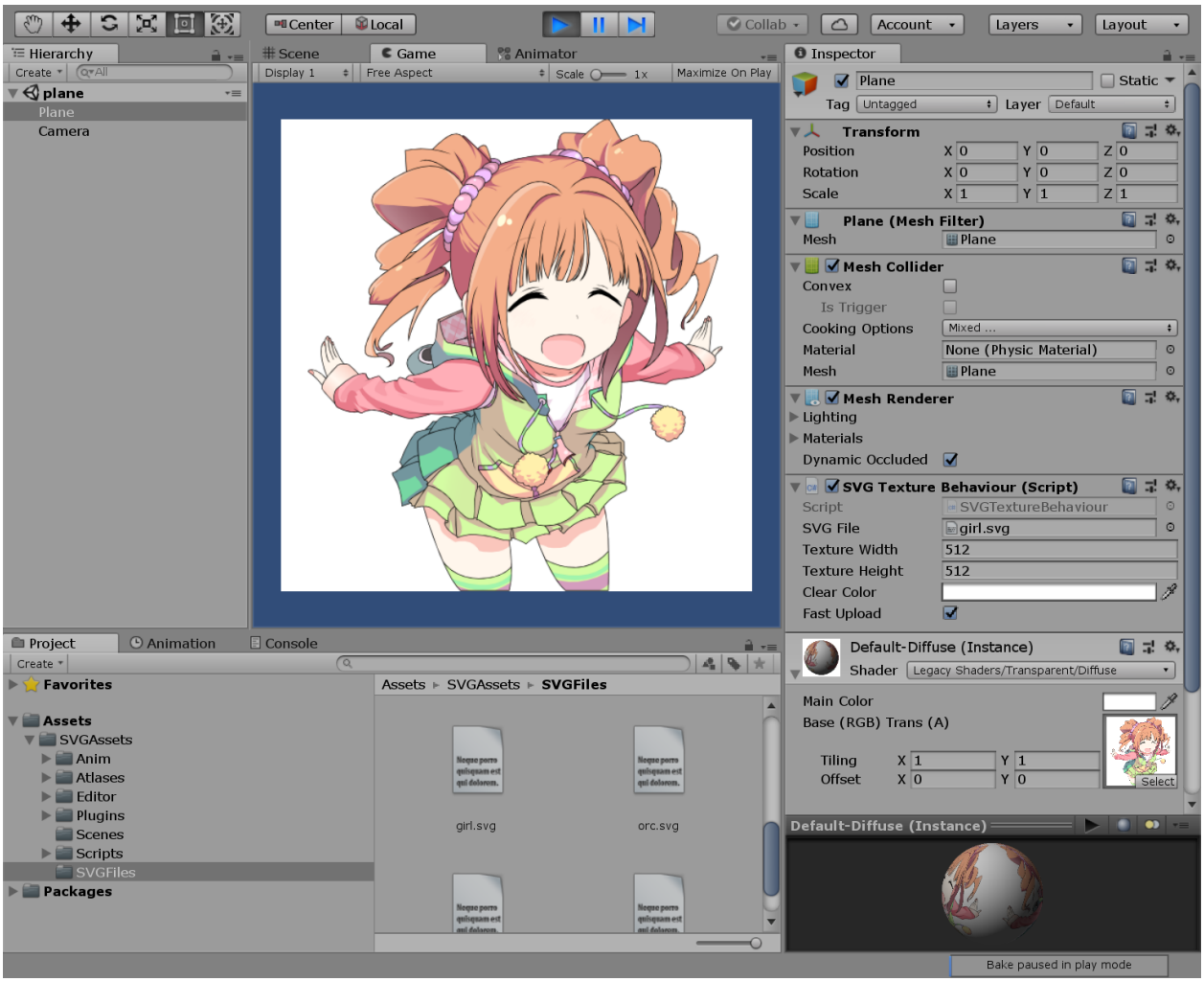


Editor mask for the SVGTextureBehaviour component

If the `Fast upload` checkbox is selected, AmanithSVG native plugins for Unity will update the Unity texture using the following methods:

- UpdateSubresource (Direct3D 11)
- glTexSubImage2D (OpenGL and OpenGL ES)
- replaceRegion (Metal)

You can see the usage of `SVGTextureBehaviour` script by opening the `plane.unity` scene.



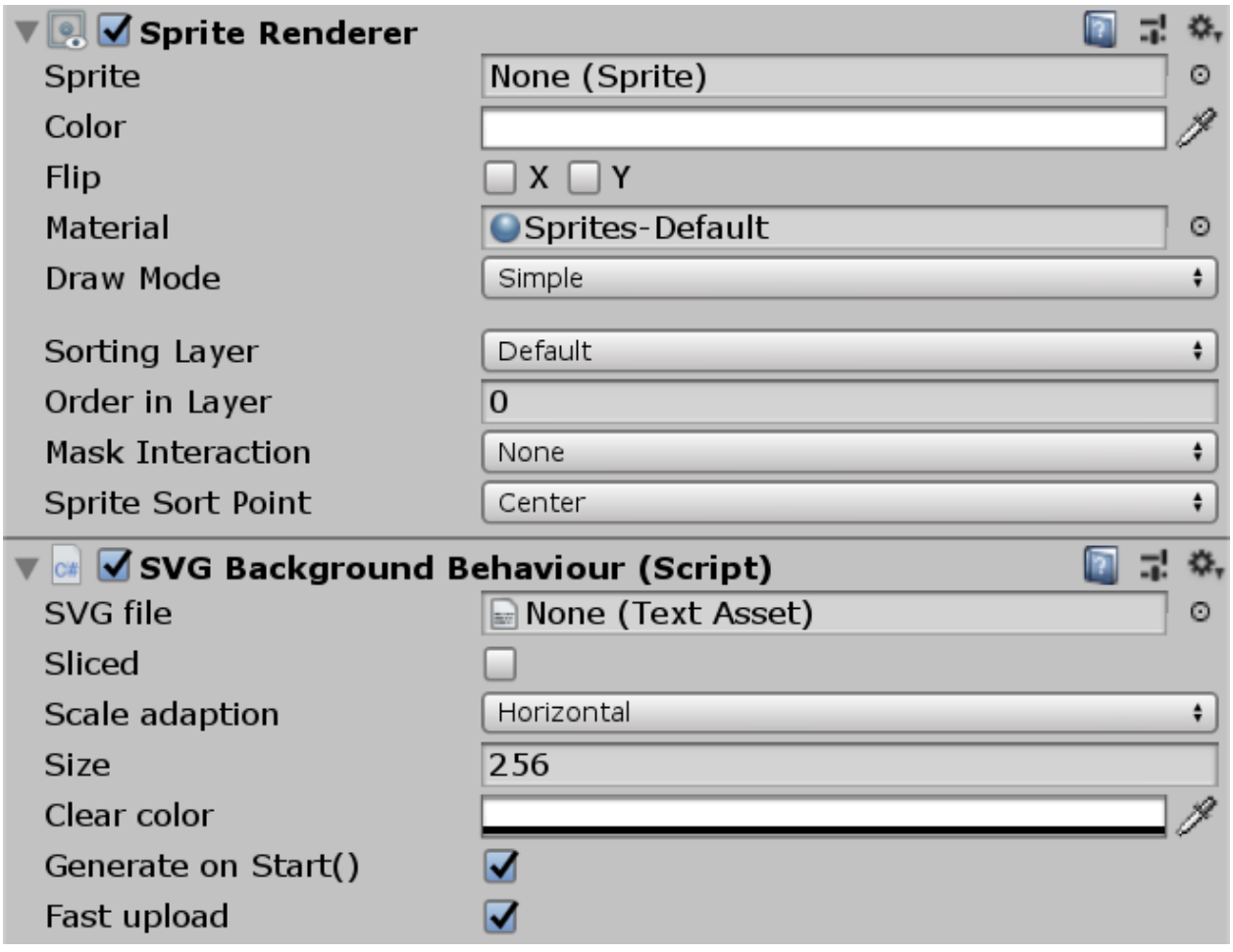
The "plane" scene

SVGBackgroundBehaviour

This monobehaviour script performs the rendering of a single SVG file on a `Texture2D`, and creates a new sprite out of it (covering the whole texture). The sprite is then assigned to the SpriteRenderer component. As for the SVGTextureBehaviour script, it is possible to specify a clear color and the `Fast upload` flag too.

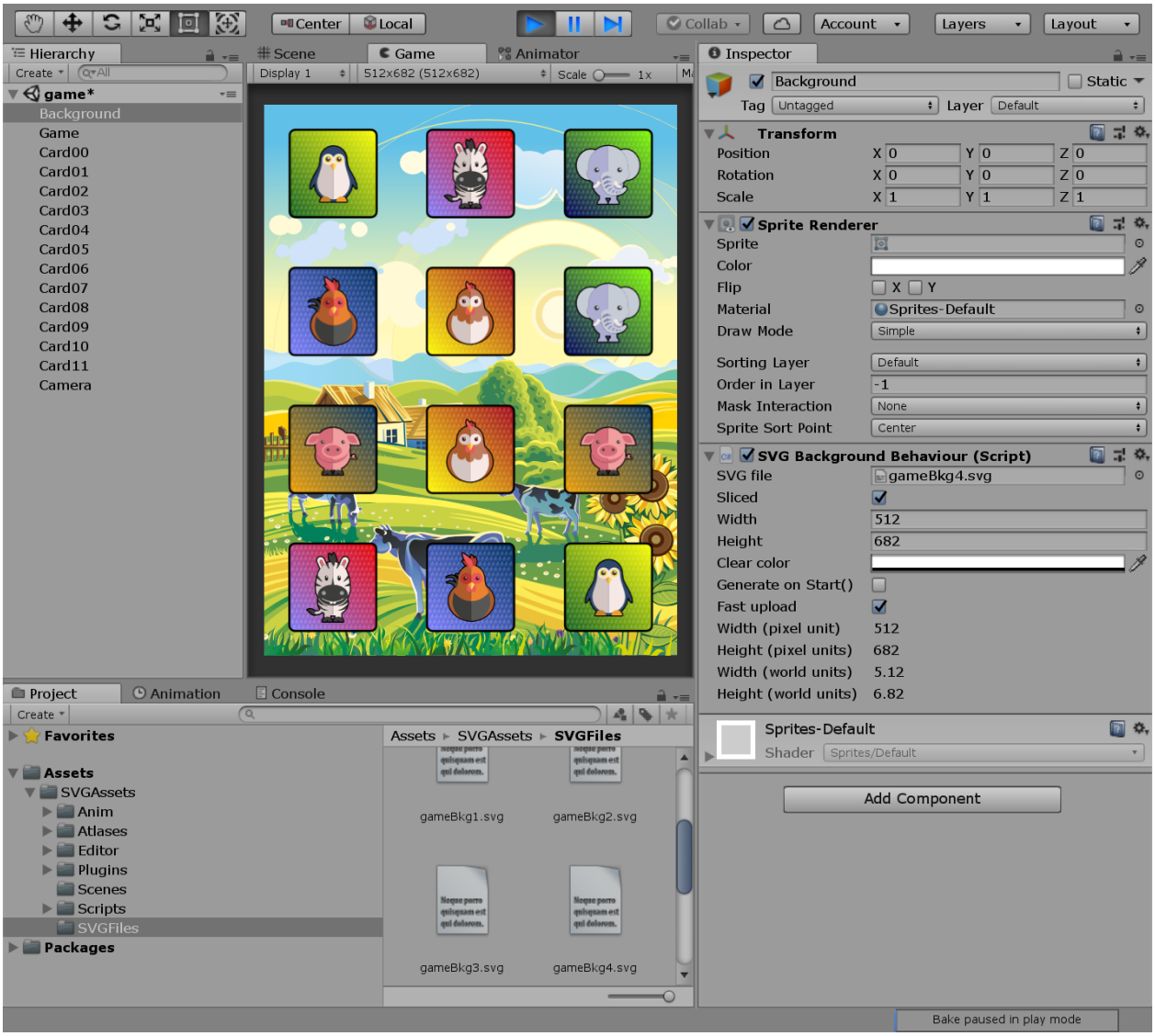
The script has two operational modes: sliced or not sliced (according to the relative checkbox).

When sliced, the script takes two additional parameters, `Width` and `Height`: at runtime, the generated texture/sprite will be sized at `Width` x `Height` exactly, in a way to preserve the SVG aspect ratio and at the same time to cover the largest dimension between `Width` and `Height` (SVG will be aligned to the center).



Editor mask for the SVGBackgroundBehaviour component

Sliced mode is useful to generate static backgrounds that must cover exactly the whole device screen; you can have a look at the `game.unity` scene for a such usage case.

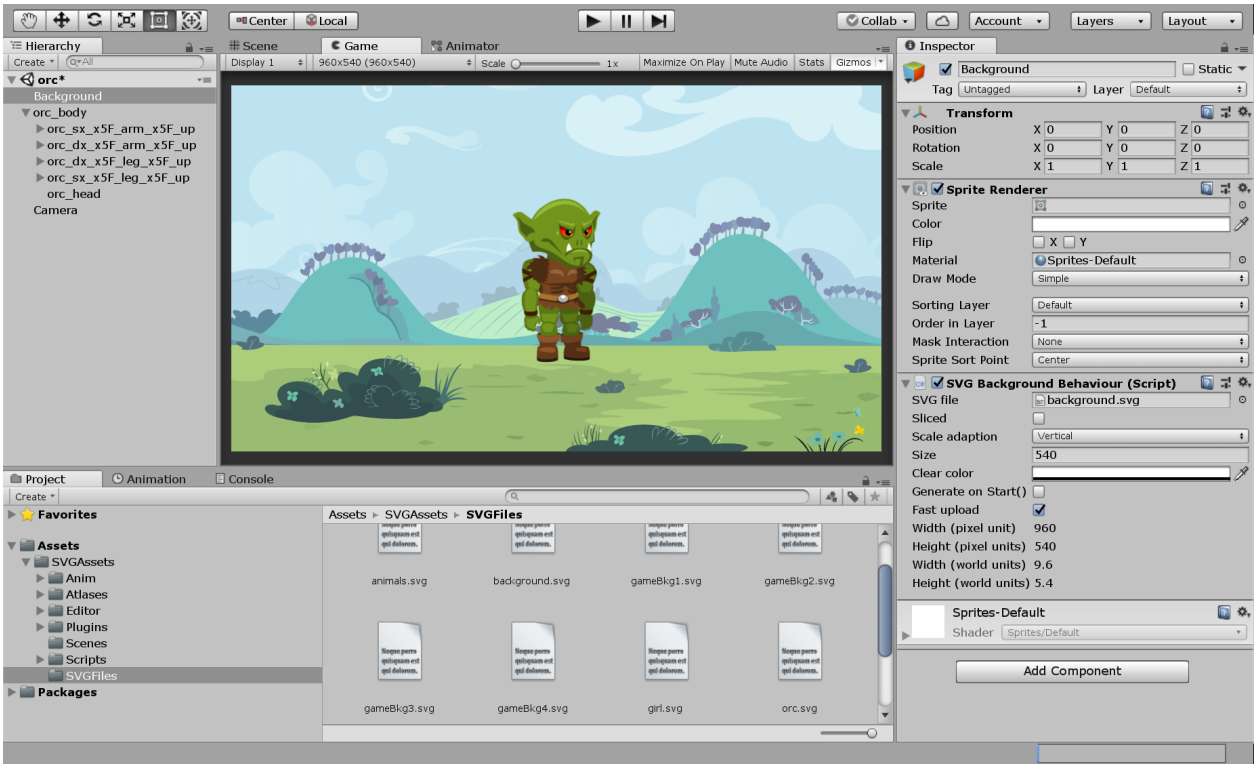


The "game" scene

In the non-sliced mode, the `SVGBackgroundBehaviour` script takes two different additional parameters, `Scale adaption` and `Size`. The first one could be `Horizontal` or `Vertical`, while the second one is expressed in pixels. The rendering is performed keeping the (horizontal or vertical) dimension equal to the specified `Size` value, and calculates the other texture dimension preserving the original SVG aspect ratio. So the final generated texture will have a size of:

- `Size` x (`Size` * `SVG aspect ratio`), if `Scale adaption == Horizontal`; or
- (`Size` * `SVG aspect ratio`) x `Size`, if `Scale adaption == Vertical`

The non-sliced mode is useful to generate static "scrollable" backgrounds (i.e. camera viewport smaller than the generated texture); you can have a look at the `orc.unity` scene for a such usage case.

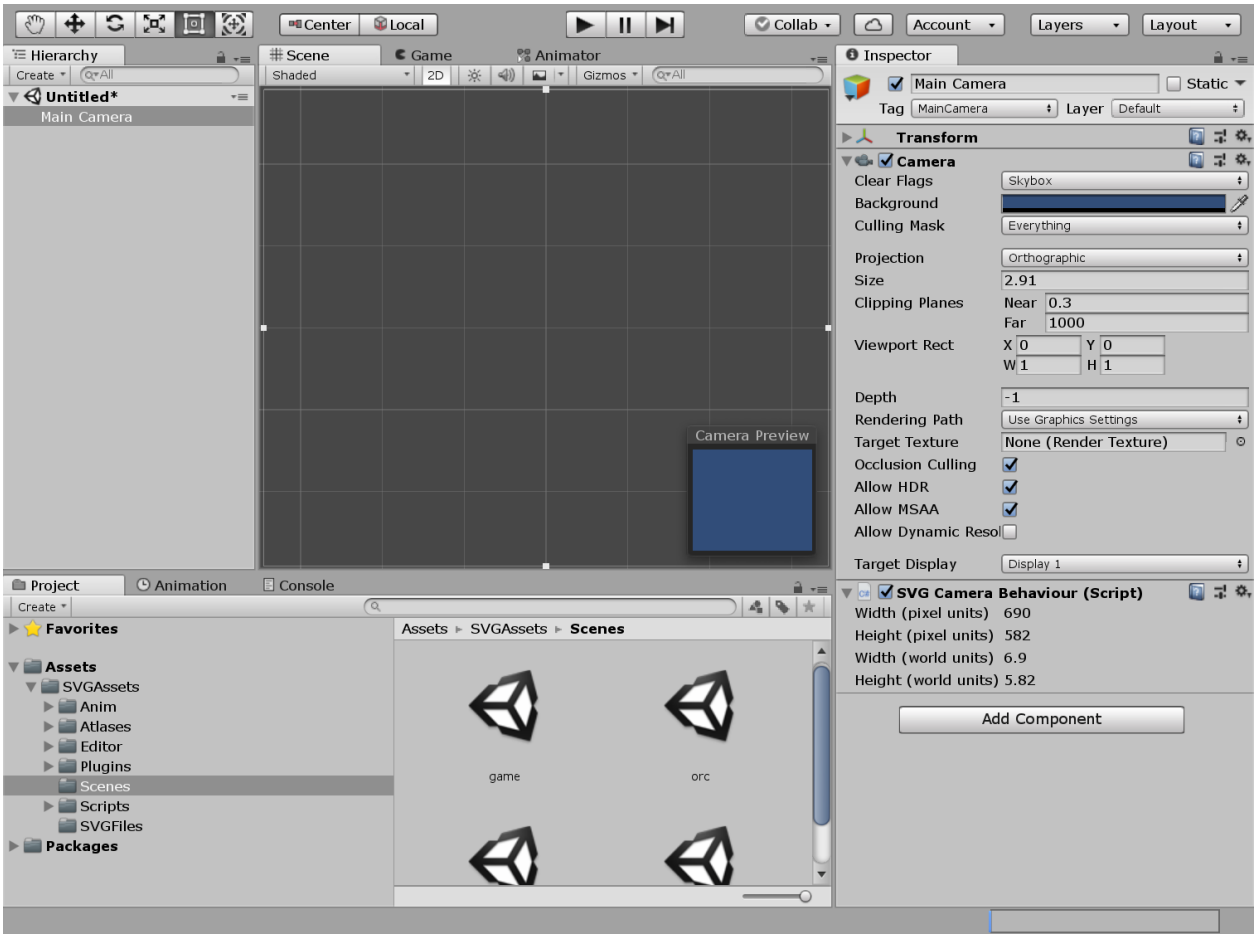


The "orc" scene

NB: in order to work properly, the `SVGBackgroundBehaviour` script must be assigned to a `GameObject` with a `SpriteRenderer` component already attached (e.g. you can create it by using the `GameObject` → `2D Object` → `Sprite` menu).

SVGCameraBehaviour

In order to work properly, SVGCameraBehaviour script must be attached to an orthographic Camera.
The script ensures that both camera aspect ratio and orthographic size match the current screen resolution.
In addition, the script is in charge of communicating a possible change of the screen resolution (e.g. when a mobile device is rotated or a window resized) to all `OnResize` listeners.

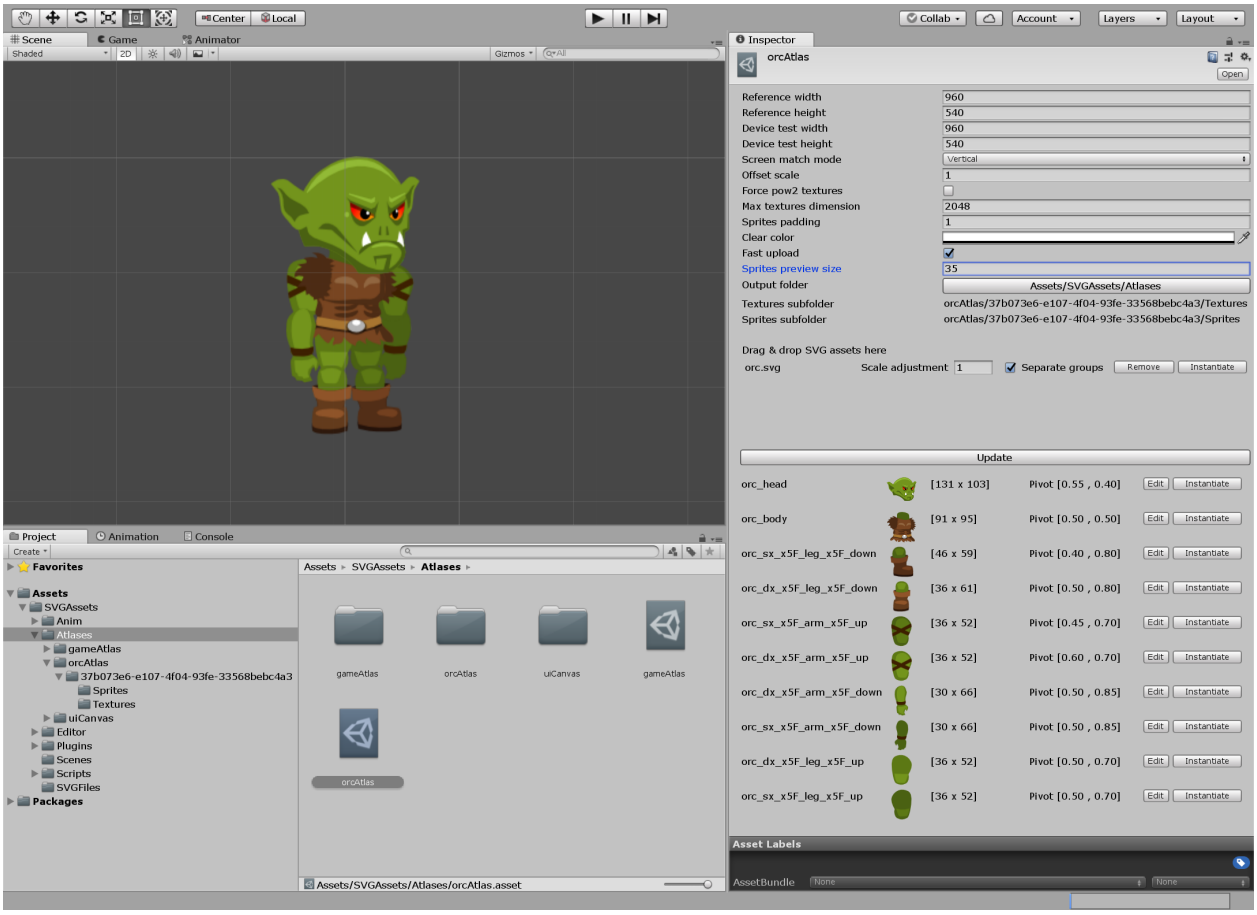


Editor mask for the SVGCameraBehaviour component

Both `orc.unity` scene and `game.unity` scene use the `SVGCameraBehaviour` script.

SVGAtlas

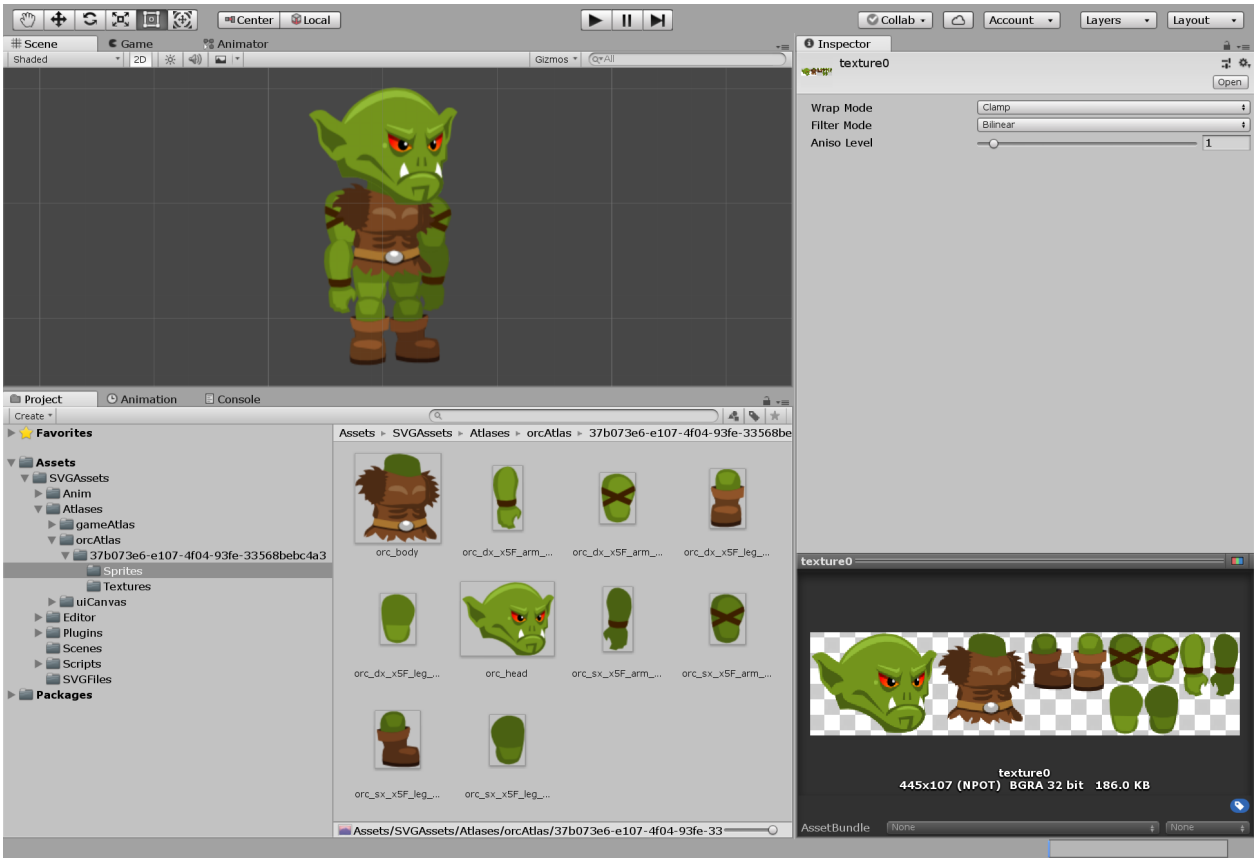
This ScriptableObject allows the generation of sprites from different SVG files, packing the generated sprites into one or more textures, according to the specified parameters.



Editor mask for the SVGAtlas scriptable object

Each SVG file can be drag&dropped in the relative section (the region labeled with "Drag & drop SVG assets here"). Dragged items can be moved within the list, changing their order: this will induce an automatic z-order (i.e. Order in Layer value of `SpriteRenderer` components) of generated sprites. In the detail, sprites will be ordered in a way such that the one on the top of the list will be placed behind all others.

For each SVG entry, it is possible to specify if the altas generator must render it as a whole single sprite ("Separate groups" option unchecked), or if first level groups (`<g>` tags) must be rendered as separate sprites ("Separate groups" option checked). This option will allow the animation of each group.



Example of generated textures and sprites

Being vector based, SVG can be freely scaled without visual quality loss. In order to give the user an effective way to control the scaling factor, `SVGAtlas` makes available the following parameters:

- Reference width, height: it's the resolution at which SVG files would be rendered at the native dimensions, specified within the SVG file itself (outermost `<svg>` element, attributes `width` and `height`). It is really important that all used SVG files contain such attributes, because if they are not available, AmanithSVG will render each sprite filling a whole texture.
- Device test (simulated) width, height: it's the resolution that will be simulated to generate and test sprites in the editor. These settings allow the user to test different device resolutions in advance.
- Screen match mode: it defines which device dimension will control the scaling factor. The match mode is defined by the `SVGScalerMatchMode` enum type (see implemenation in `Assets/SVGAssets/Scripts/SVGAssets/SVGAssets.cs`), and it's an extension of the Unity ScreenMatchMode.
- Offset scale: an additional scale factor that will be applied to all SVG files.

The general formula to calculate the scaling factor is:

`Scl` = (`Device dimension` / `Reference dimension`) * `OffsetScale`

where device dimension is the one chosen by the screen match mode parameter.

For example, lets have:

- an SVG file with `100 x 150` dimensions (i.e. `<svg width="100px" height="150px">`)
- a reference resolution set to `1024 x 768`

- screen match mode set to `Vertical`
- offset scale set to `1`

On a `640 x 480` device resolution, the sprite will be rendered at `62 x 94` pixels, because scaling factor is `0.625 (480 / 768)`.
On a `1536 x 2048` device resolution, the sprite will be rendered at `266 x 400` pixels, because scaling factor is `2.667 (2048 / 768)`.

For the complete formula, please have a look at the `ScaleFactorCalc` method of `SVGScaler` class (see implementation in `Assets/SVGAssets/Scripts/SVGAssets/SVGAssets.cs`).

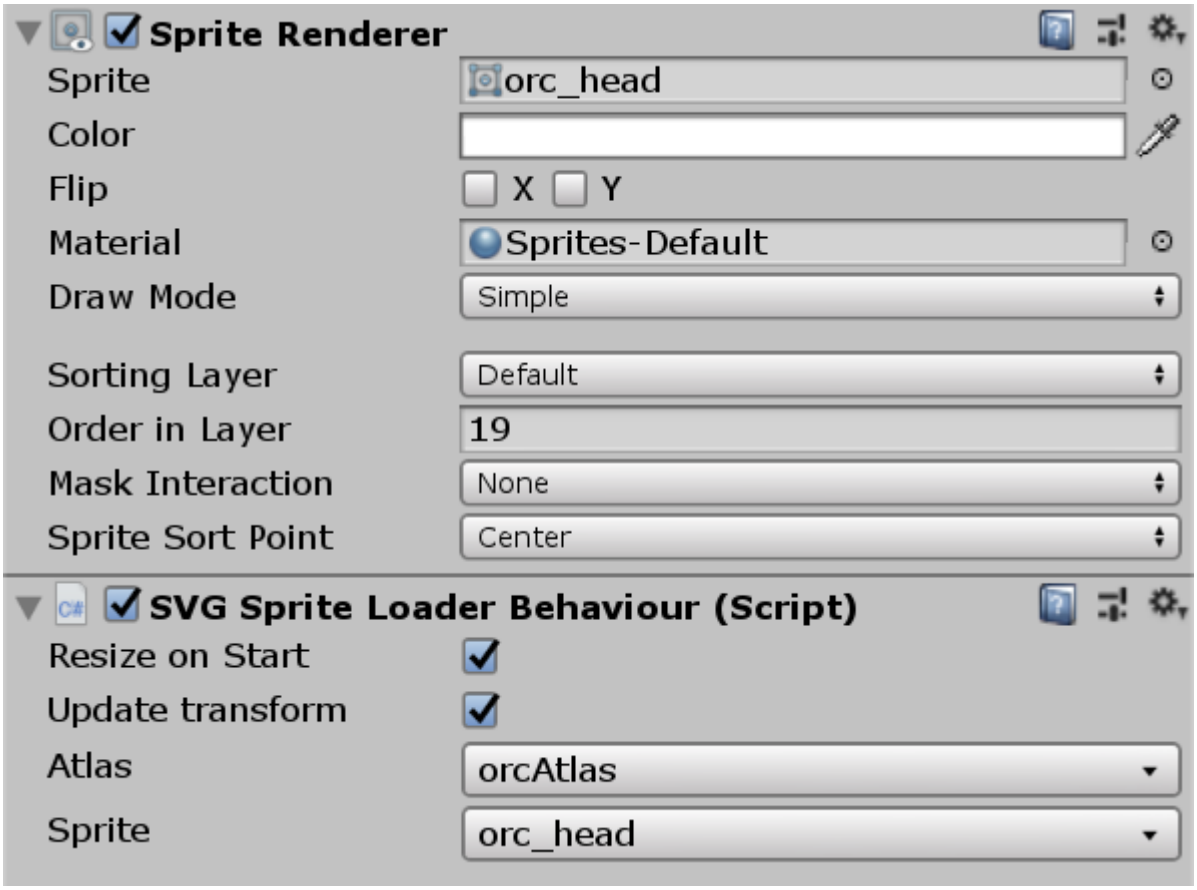
Two additional script parameters control some aspects relative to the generated textures:

- Force pow2 textures: if checked, this parameters will force the atlas generator to produce textures whose dimensions are power-of-two values.
- Max textures dimension: the maximum dimension allowed during the textures atlas generation. Please make sure that such value won't exceed the real device capability.

The last two script parameters control how many pixels will separate each sprite from any other (`Sprite padding`) and the color that is used to clear the surface before to start the rendering (`Clear color`). The `Fast upload` flag has the same meaning described for the SVGTextureBehaviour class.

SVGSpriteLoaderBehaviour

This monobehaviour script takes care to render the associated sprite at runtime on the device, according to its original SVGAtlas generator settings.



Editor mask for the SVGSpriteLoaderBehaviour component

Editable parameters consist in:

- `Resize on Start`: if checked (default value), the sprite will be regenerated at the Start of monobehaviour, else the user must update the sprite programmatically calling the public `UpdateSprite` function.
- `Update transform`: if checked (default value), at every monobehaviour LateUpdate call, the sprite local position will be fixed according to its current dimensions. This will ensure that animated child sprites will be in the correct position relative to their father (just the position is relevant, rotation and scale do not need to be fixed). If the sprite is moved programmatically by code (e.g. a root body part of a character), this option must be unchecked.

`Atlas` and `Sprite` properties refer to the `SVGAtlas` object that has generated this sprite.

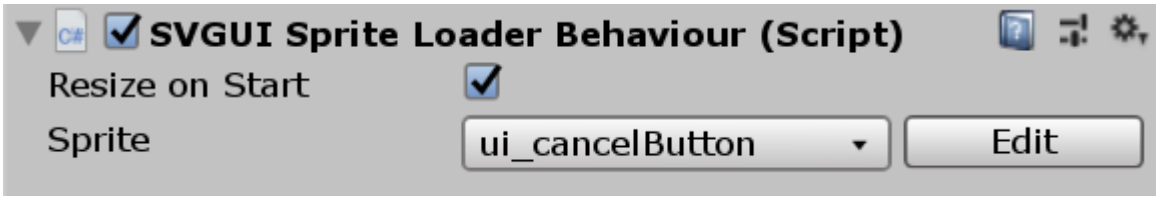
`SVGSpriteLoaderBehaviour` component is attached automatically to those sprites instantiated through the SVGAtlasEditor editor mask.

SVGUIAtlas

SVGUIAtlas is really similar to SVGAtlas, with the only exception that all parameters that determine the scale factor for SVG contents (e.g. reference resolution, screen match mode, and so on) are provided by a CanvasScaler component.
`SVGUIAtlas` objects cannot be instantiated as a standalone asset (as opposed to `SVGAtlas`), but they are strictly attached to SVGCanvasBehaviour components: each `SVGCanvasBehaviour` instance contains a single `SVGUIAtlas` instance.
Because `SVGCanvasBehaviour` components can be attached to Canvas only, it follows that `SVGUIAtlas` instances are in a 1-1 relation to canvas instances.

SVGUISpriteLoaderBehaviour

This monobehaviour script takes care to render the associated UI sprite at runtime on the device, according to its original SVGUIAtlas generator settings.



Editor mask for the SVGUISpriteLoaderBehaviour component

Editable parameters consist in:

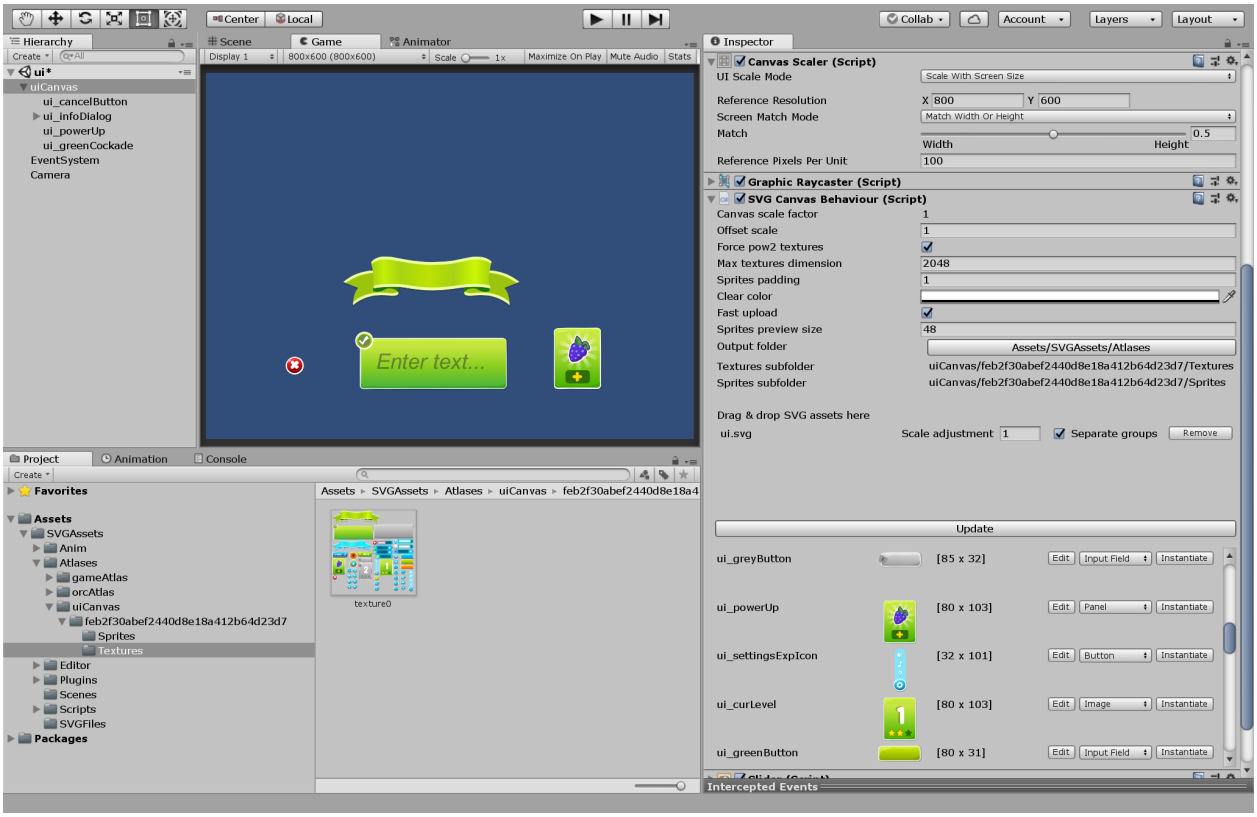
- `Resize on Start`: if checked (default value), the UI sprite will be regenerated at the Start of monobehaviour, else the user must update the sprite programmatically calling the public `UpdateSprite` function.
- `Sprite` property refers to the sprite generated by the relative `SVGUIAtlas` object.

The `Edit` button is a shortcut to the pivot and borders editor that would still be reachable by selecting the original SVGCanvasBehaviour component.

`SVGUISpriteLoaderBehaviour` component is attached automatically to those UI sprites instantiated through the SVGCanvasEditor editor mask.

SVGCanvasBehaviour

This monobehaviour script can be attached to Canvas components only, and it is in charge of storing an instance of SVGUIAtlas.
The main role of `SVGCanvasBehaviour` is to communicate the canvas scale factor to the maintained `SVGUIAtlas` instance.



Editor mask for the SVGCanvasBehaviour component

Editor scripts

To make sure that the use of SVGAssets is as simple and convenient as possible from the Unity editor, a set of editor scripts have been implemented:

- SVGAssetsConfigProvider
- SVGBackgroundEditor
- SVGCameraEditor
- SVGAtlasEditor
- SVGPivotEditor
- SVGSpriteLoaderEditor
- SVGAtlasSelector

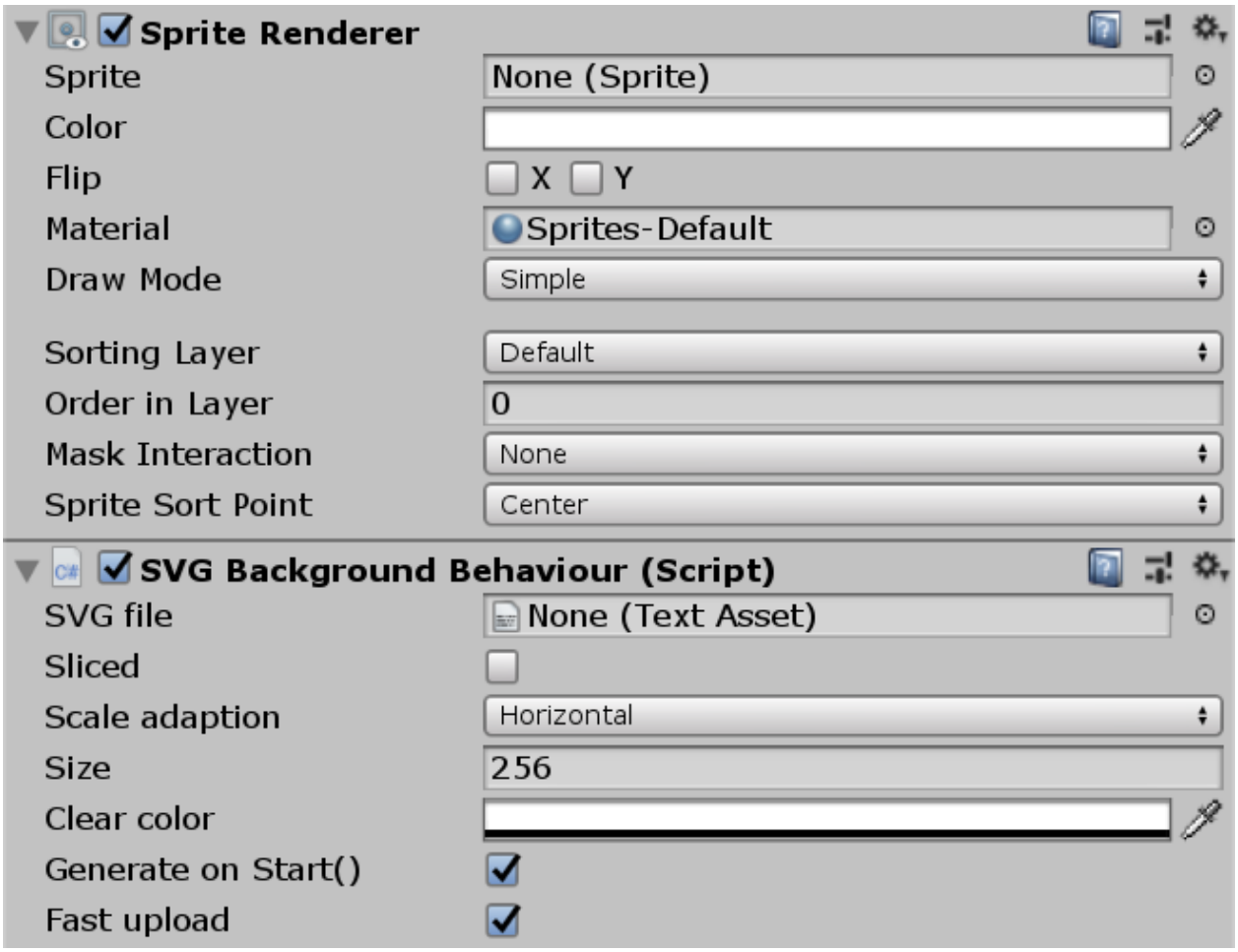
- SVGSpriteSelector
- SVGCanvasEditor
- SVGUISpriteLoaderEditor
- SVGRenamerImporter

SVGAssetsConfigProvider

This editor script implements a `SettingsProvider` for the editing of SVGAssetsConfigUnity. Is is possible to configure user-agent language, curves quality and log facilities. In addition it is possible to drag&drop resource files (fonts and images) and edit their parameters. The editing window is available throught the `Edit` → `Project Settings` → `SVGAssets` menu.

SVGBackgroundEditor

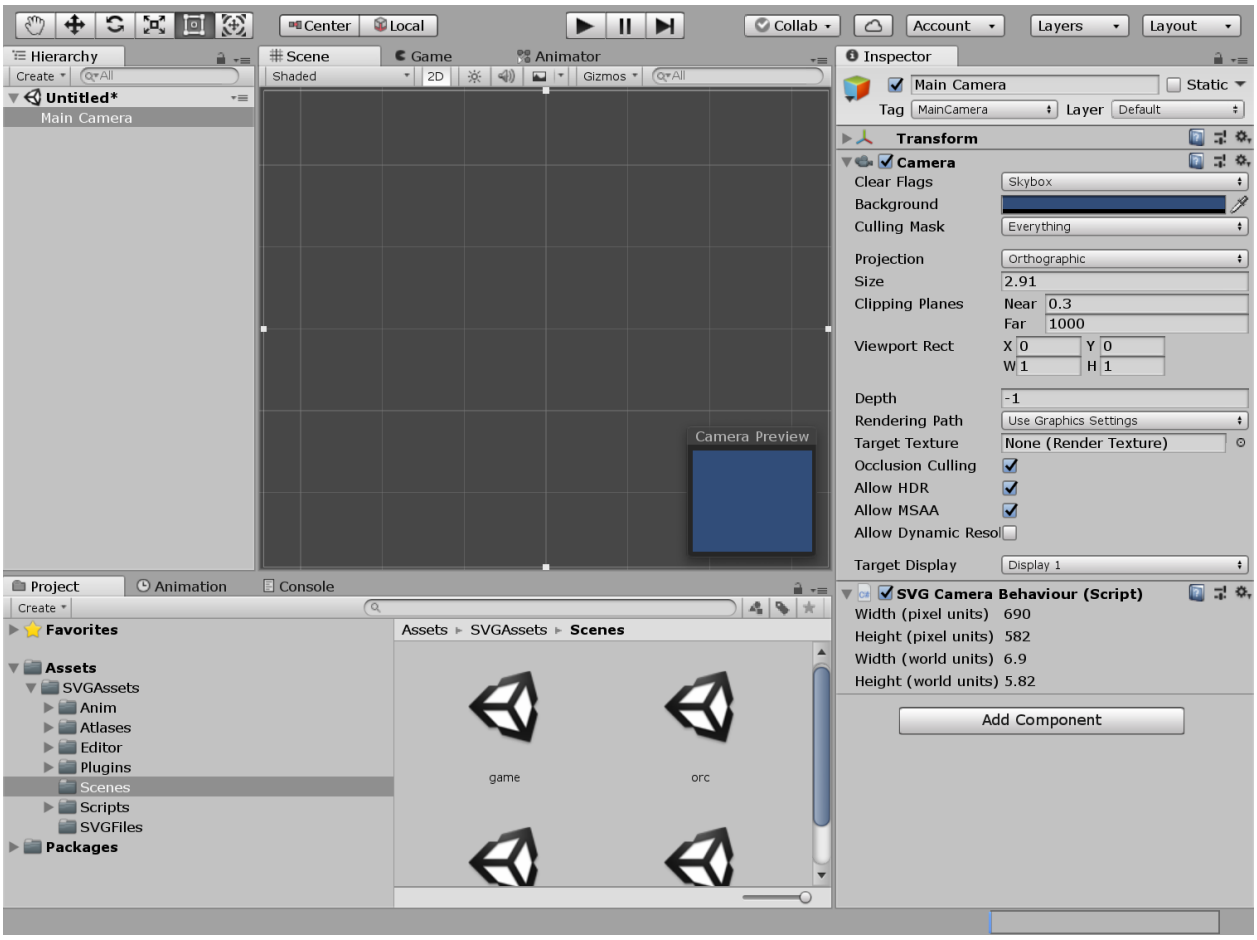
This editor script implements the inspector mask for the editing of SVGBackgroundBehaviour components. The layout allows to edit all `SVGBackgroundBehaviour` parameters, taking care to present only those needed ones according to the current operational mode (sliced or not-sliced).



Editor mask for the SVGBackgroundBehaviour component

SVGCameraEditor

This editor script implements the inspector mask for SVGCameraBehaviour components. The layout simply displays the camera viewport size, both in pixels and world units.



Editor mask for the SVGCameraBehaviour component

SVGAtlasEditor

This editor script implements the inspector mask for the editing of SVGAtlas assets. The base class `SVGBasicAtlasEditor` (from which `SVGAtlasEditor` is derived) also implements a `PostProcessBuild` static class that will take care of some aspects relative to the building phase. In particular, it ensures that during the building process all generated atlases won't be included in the final package (actually they are substituted by a dummy 1x1 texture), because they will be regenerated at runtime on the device. This will reduce the build size, showing the advantage of using SVG files instead of bitmaps.

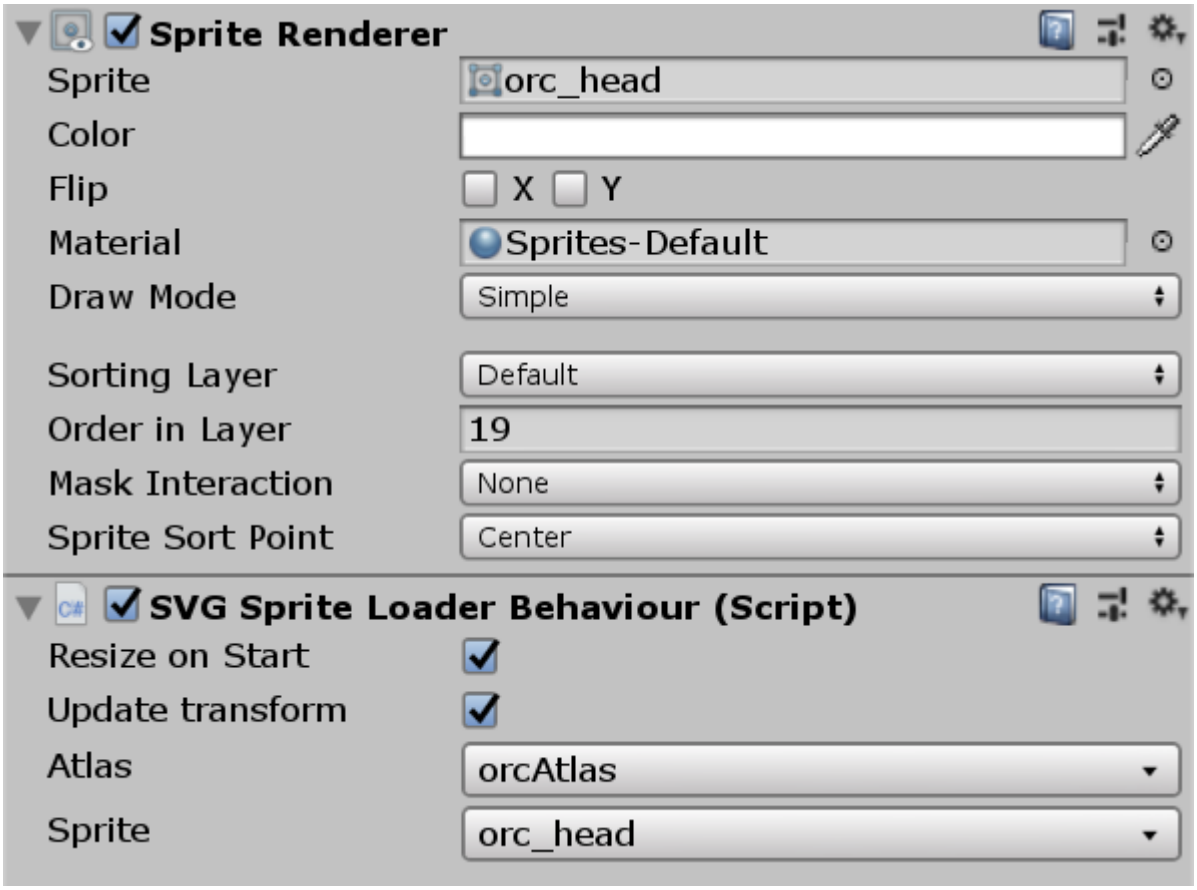
SVGPivotEditor

This editor script allows the user to edit the pivot point of generates sprites. The pivot can be moved by simply clicking the mouse on the desired position or by changing values by hand. It is desirable to setup pivot points before instantiating and/or animating sprites.

The pivot editor window

SVGSpriteLoaderEditor

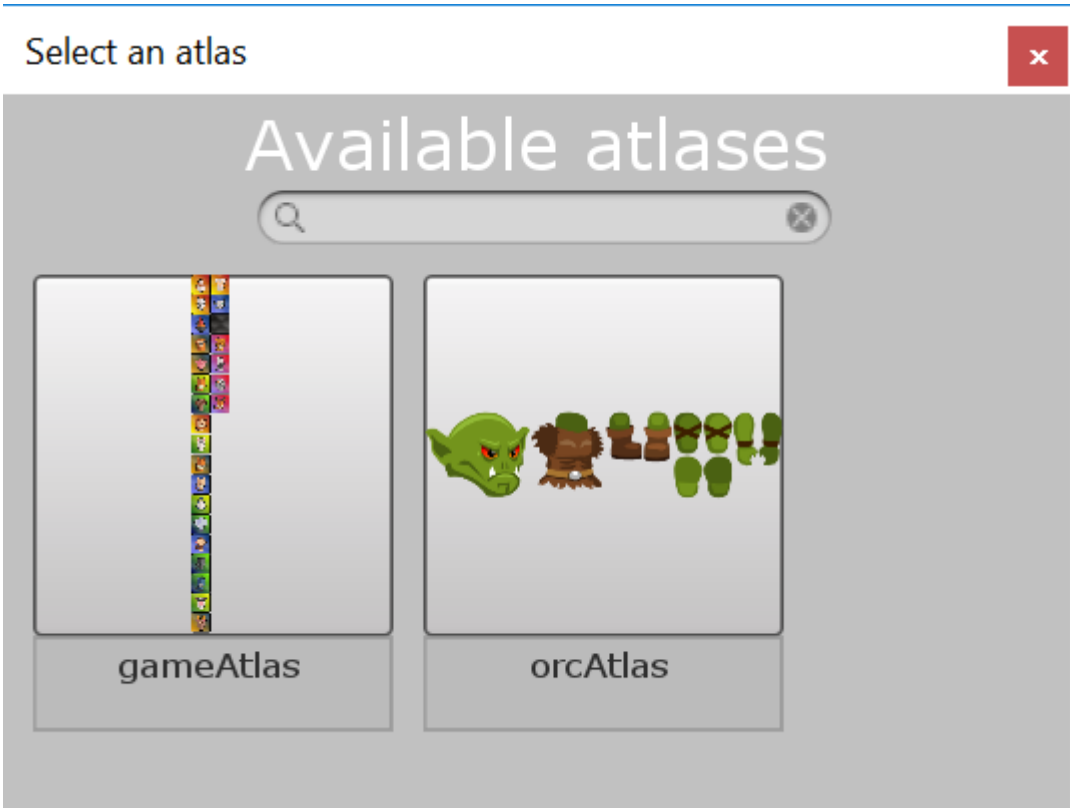
This editor script implements the inspector mask for the editing of SVGSpriteLoaderBehaviour components. The layout allows to edit (actually to check/uncheck) the `Resize on start` and `Update transform` properties. In addition the `Atlas` and `Sprite` buttons allow to show and/or select the sprite reference: in particular, such two buttons make use of SVGAtlasSelector and SVGSpriteSelector editor scripts.



Editor mask for the SVGSpriteLoaderBehaviour component

SVGAtlasSelector

This editor script implements a ScriptableWizard that allows the (visual) selection of `SVGAtlas` assets.



Select SVGAtlas objects wihtin the project

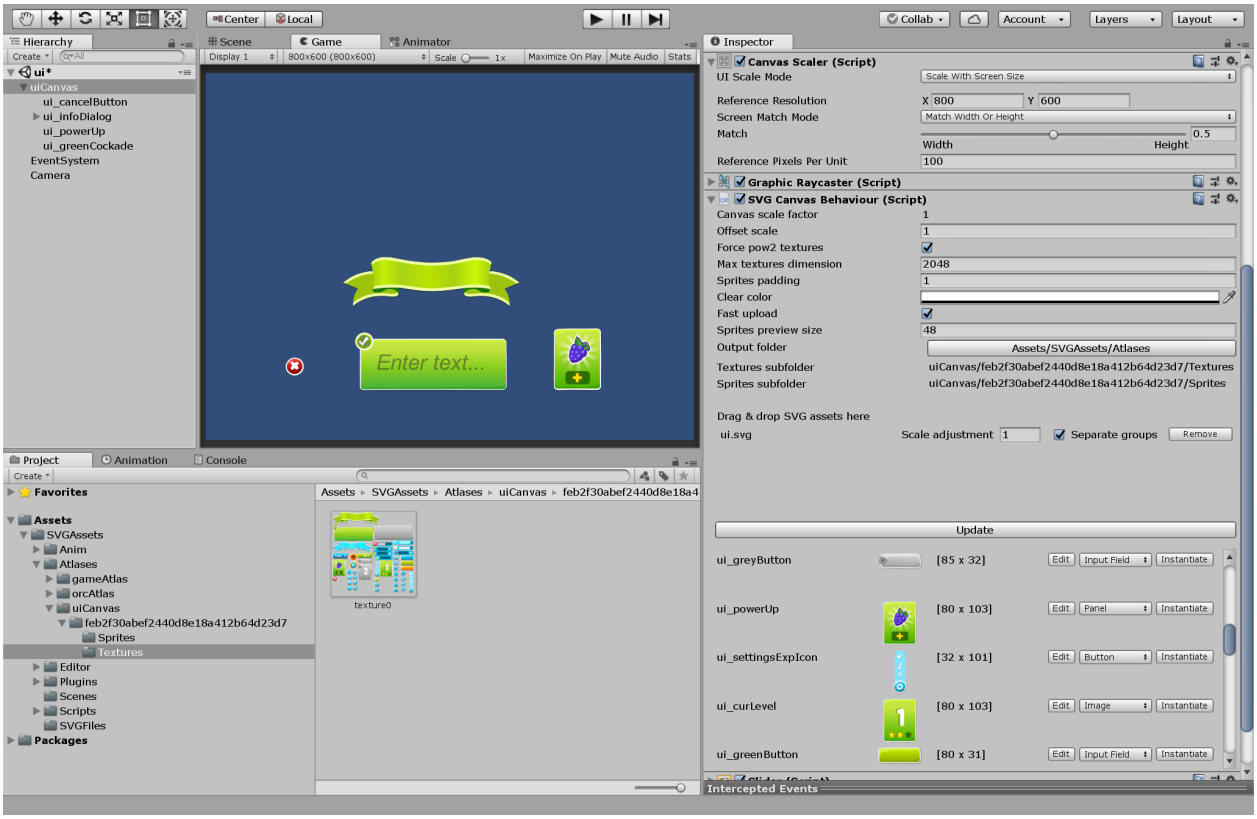
SVGSpriteSelector

This editor script implements a ScriptableWizard that allows the (visual) selection of a sprite asset generated by `SVGAtlas` objects.

Select a sprite belonging to an SVGAtlas object

SVGCanvasEditor

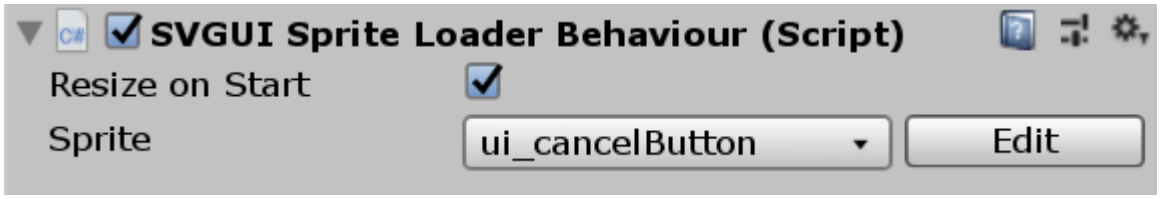
This editor script implements the inspector mask for the editing of SVGCanvasBehaviour components. In the detail the script displays the canvas scale factor and the underlying SVGUIAtlas object.
The base class `SVGBasicAtlasEditor` (from which `SVGCanvasEditor` is derived) also implements a `PostProcessBuild` static class that will take care of some aspects relative to the building phase. In particular, it ensures that during the building process all generated atlases won't be included in the final package (actually they are substituted by a dummy `1x1` texture), because they will be regenerated at runtime on the device. This will reduce the build size, showing the advantage of using SVG files instead of bitmaps.



Editor mask for the SVGCanvasBehaviour component

SVGUISpriteLoaderEditor

This editor script implements the inspector mask for the editing of SVGUISpriteLoaderBehaviour components. The layout allows to edit (actually to check/uncheck) the `Resize on Start` property. In addition the `Sprite` button allows to show and/or select the sprite reference: in particular, such button makes use of SVGSpriteSelector editor script.
The `Edit` button allows the modification of sprite borders and pivot.



Editor mask for the SVGUISpriteLoaderBehaviour component

It is desirable to setup pivot points before instantiating UI sprites.

SVGRenamerImporter

This script implements an asset post-processor (AssetPostprocessor class). Its task consists in changing the extension of new asset files from `.svg` to `.svg.txt`, so Unity can recognize those files as text assets. The same kind of task is also performed on `.ttf`, `.otf`, `.woff`, `.jpg` and `.png` files: in this case they will be renamed by adding a `.bytes` suffix.

FAQ

AmanithSVG native plugins for Android

AmanithSVG native plugins for Android must be placed in `Assets/SVGAssets/Plugins/Android/libs` folder. According to the target architecture, a different `libAmanithSVG.so` file is present in a different subdirectory. Be sure to follow this guide in order to import AmanithSVG native plugins for Android correctly; in the detail:

- select each `libAmanithSVG.so` file present in `Assets/SVGAssets/Plugins/Android/libs` subfolders and view it in the Inspector
- deselect `Any platform` option, if checked
- select `Include Platforms` --> `Android` option
- in the `Platform settings` section, set `CPU` to:
  - `ARMv7` for `armeabi-v7a/libAmanithSVG.so`
  - `ARM64` for `arm64-v8a/libAmanithSVG.so`
  - `X86` for `x86/libAmanithSVG.so`
  - `X86_64` for `x86_64/libAmanithSVG.so`

SVGAssets package already includes the plugins set correctly.

SVGAssets on iOS doesn't render the SVG(s) and XCode does not show any messages about it. How to fix it?

Please double check that the folder `Assets/SVGAssets/Plugins/iOS` of your project contains both the static library `libAmanithSVG.a` and `loader.m`

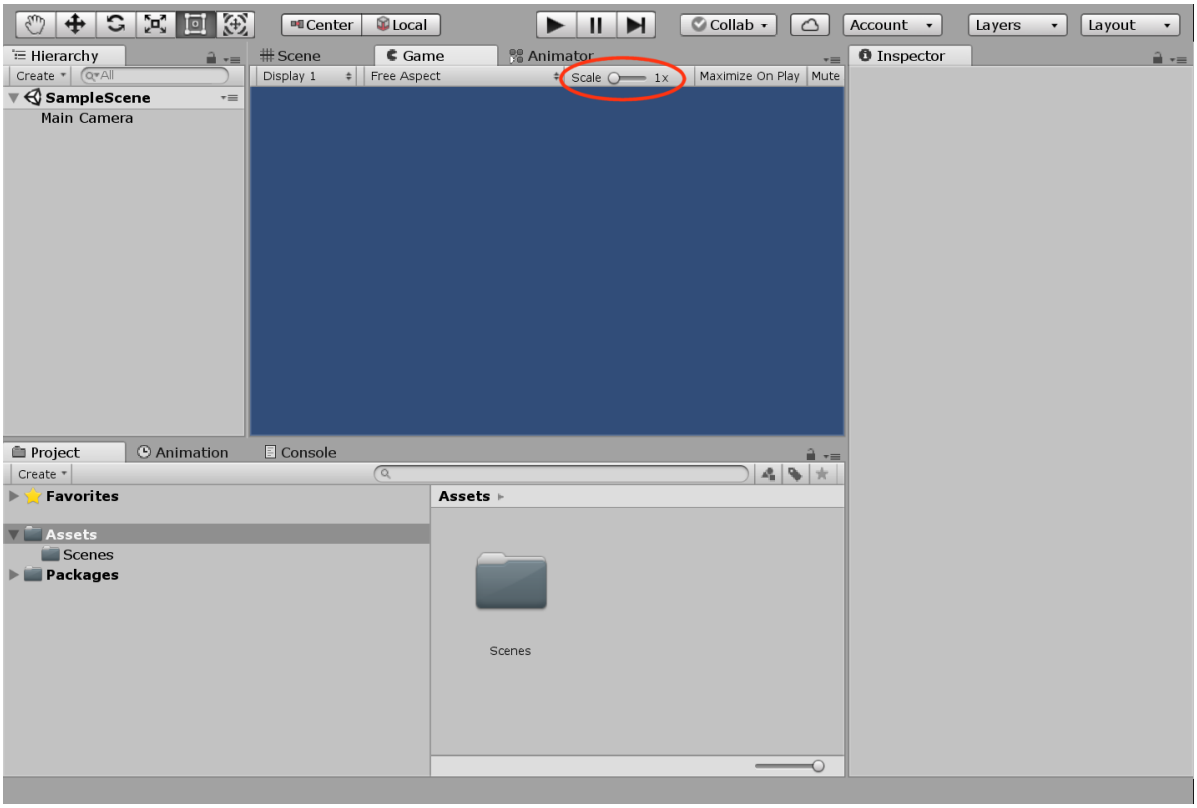
"<xyz> elements are not supported in AmanithSVG Lite version" messages

Because some features are not present in AmanithSVG Lite, you could see some warning messages about SVG unsupported elements in the Unity console window. In particular the following SVG elements are not supported in AmanithSVG Lite: <linearGradient>, <radialGradient>, <pattern>, <image>, <mask>, <filter>. Such elements are supported in AmanithSVG Full only, that is available for licensing on Mazatech's website. In order to use the Full version, it is not needed to change C# scripts, just substitute the native plugin.

Step-by-step tutorial: a minimal example

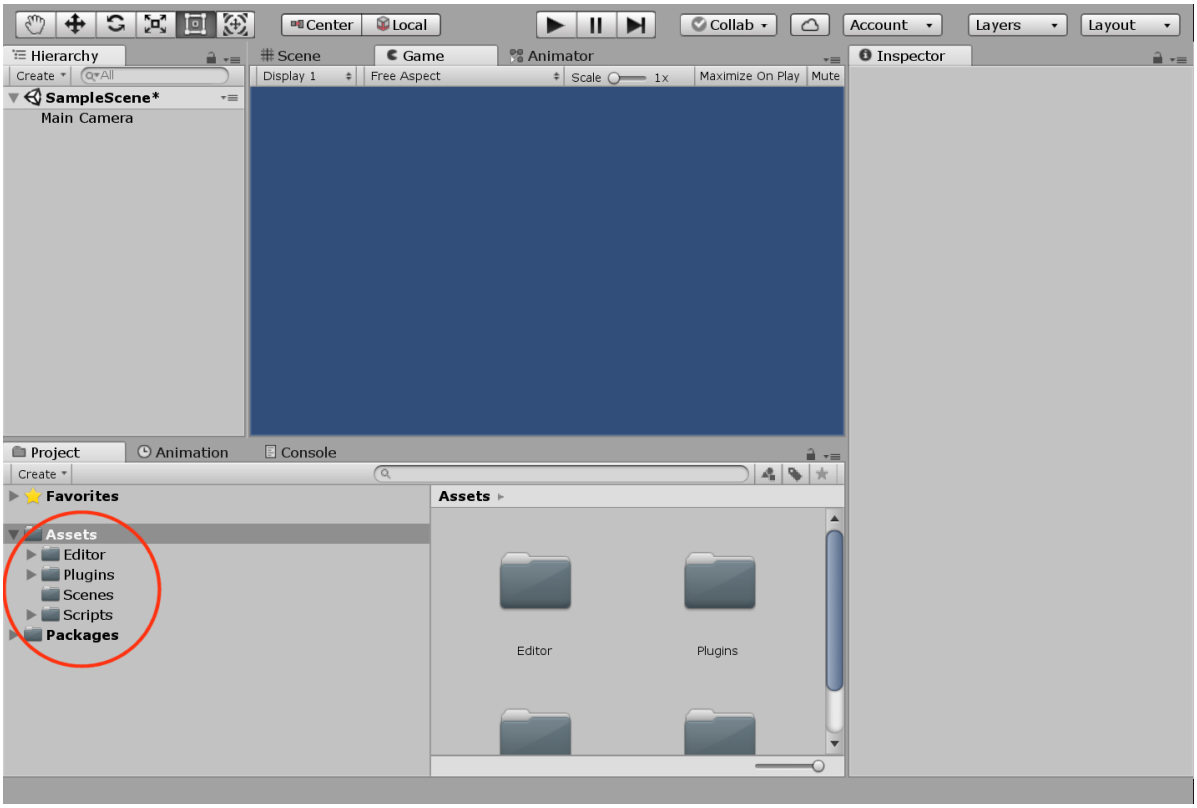
The example explained in this chapter is implemented in the `plane.unity` scene. The example shows a simple plane object with an attached texture, generated from an SVG file. The example has been realized following these steps:

- Create a new Unity project, SVGAssets can be used for 2D and 3D projects; in this case we select the 2D setup.
- Select the Game view, and make sure that "Scale" slider is completely on the left (i.e. `scale = 1x`); switch back to the Scene view.



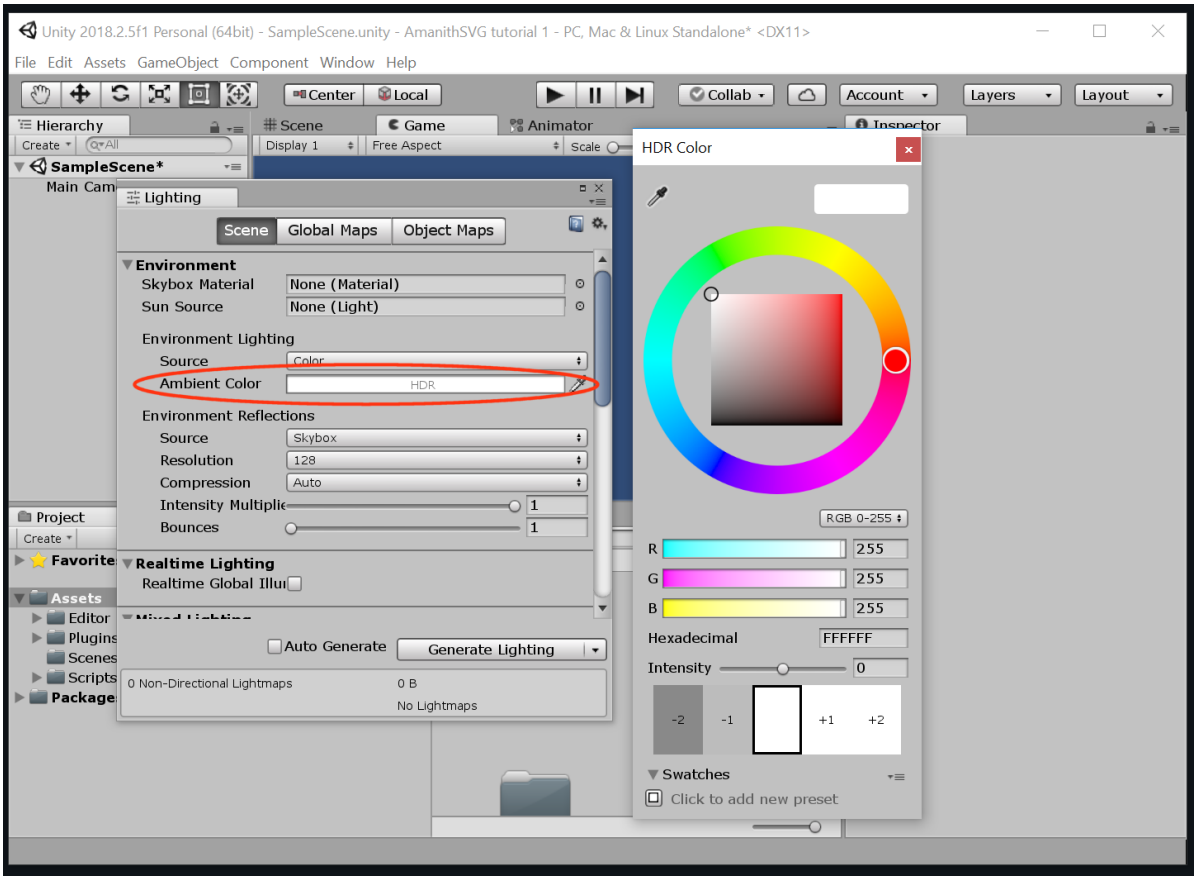
Set a unitary scene scale

- Because the project is a new one, it is required to copy SVGAssets folders (`Editor`, `Plugins`, `Resources`, `SVGFiles` and `Scripts`) inside the new project's `Assets/SVGAssets` folder; so that resources (e.g. config file, fonts, SVG files), native AmanithSVG plugins and its C# interface will be available for the project.



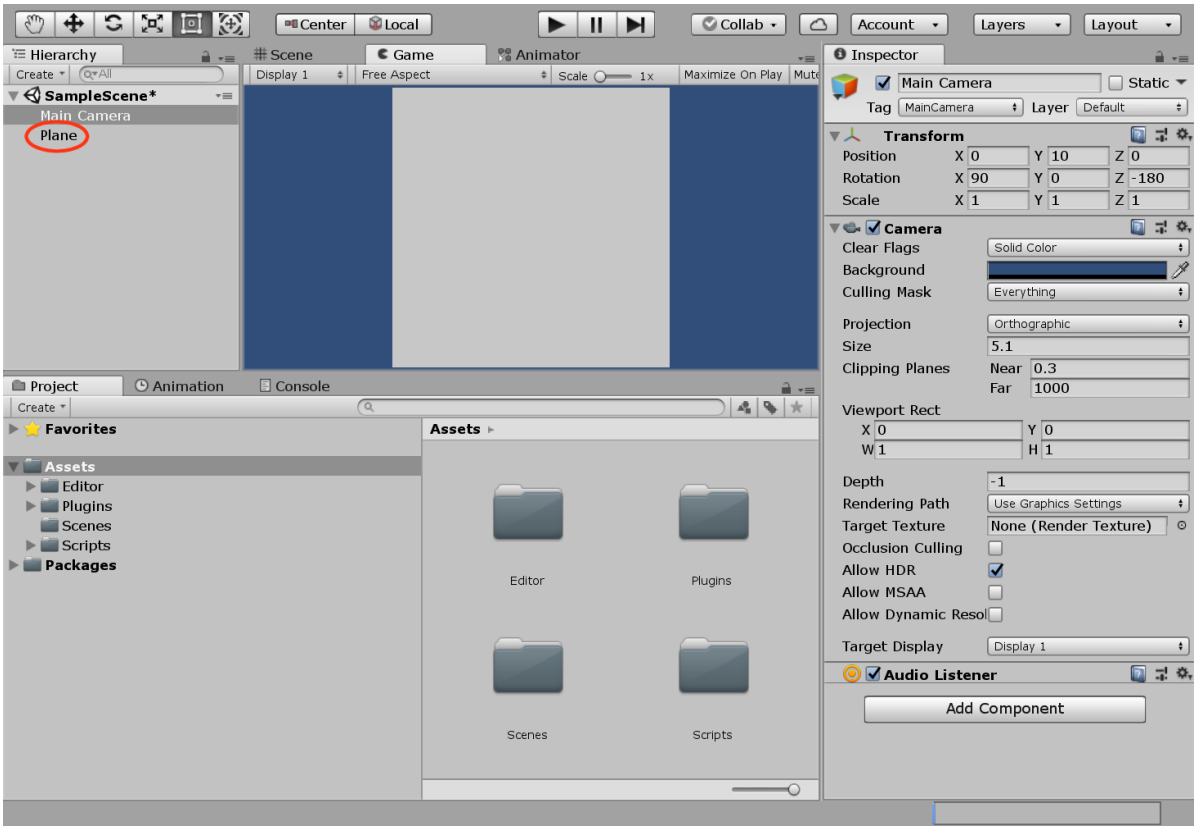
The minimal project layout after copying all needed files

- Set a full white `Ambient Color` light (menu: `Window` → `Rendering` → `Lighting` → `Environment` → `Environment Lighting`), in order to display SVG files with their original colors (in older Unity version the option was accessible through the `Window` → `Rendering` → `Lighting Settings` menu).



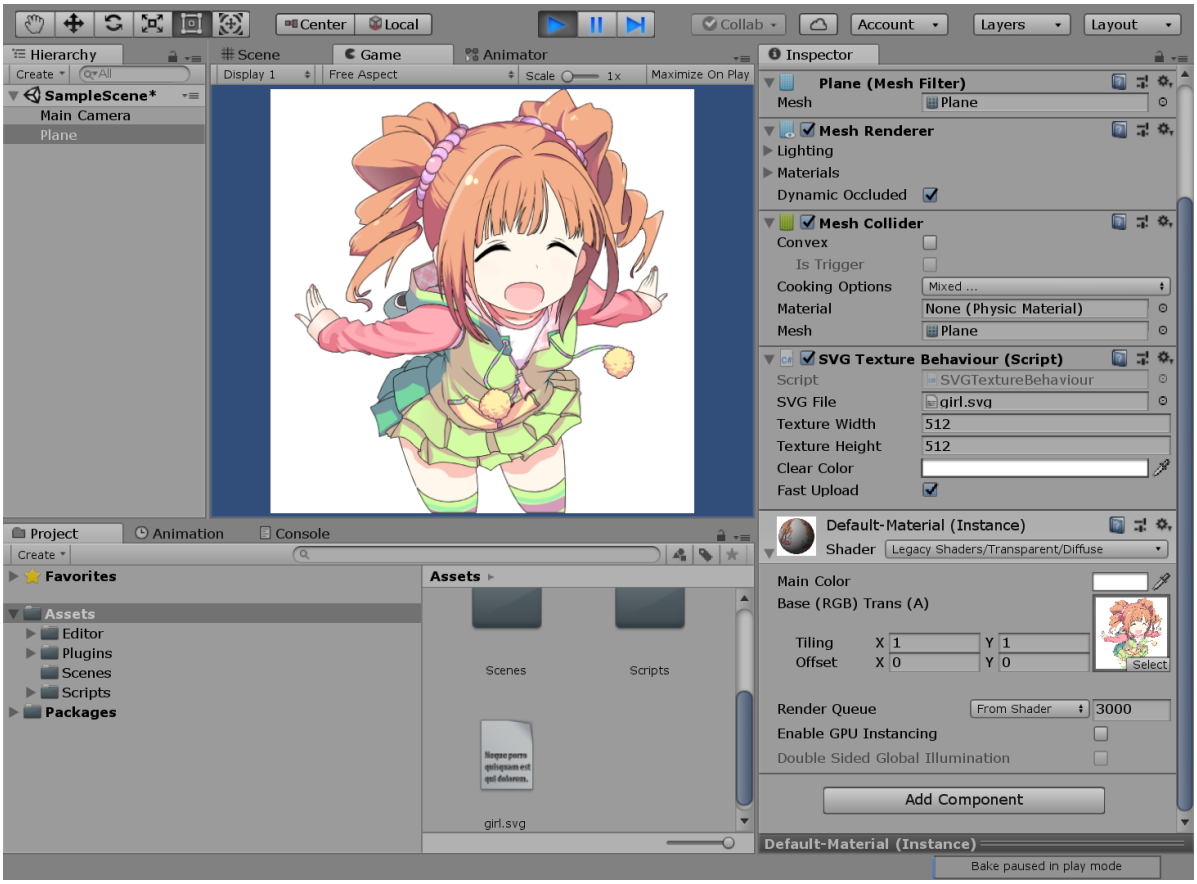
Ensure a pure white ambient light

- Create a plane object (menu: `GameObject` → `3D Object` → `Plane`) and make sure that the main camera is pointing it. To do so, place the plane object at `(0, 0, 0)` and the camera object at `(0, 10, 0)` with a rotation of `(90, 0, -180)`.



The plane gameobject

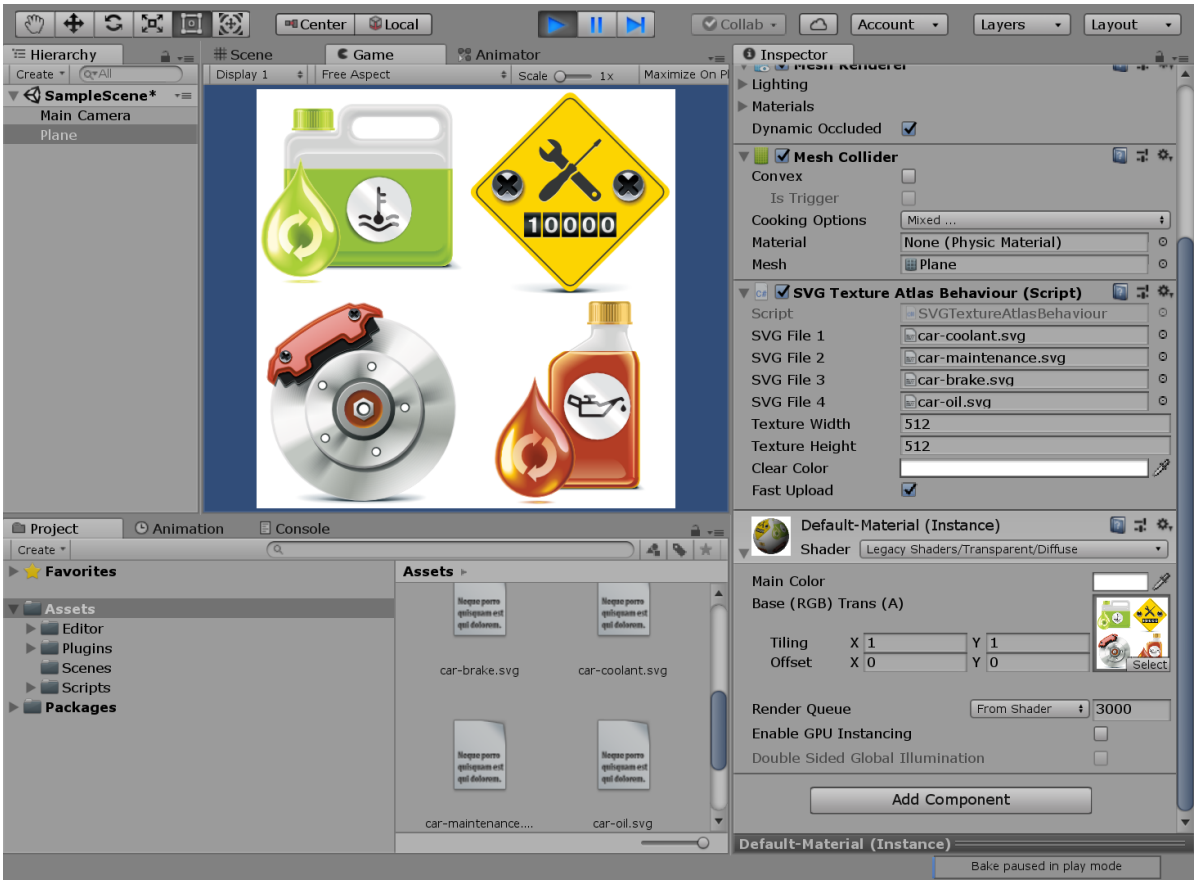
- Select the plane object and attach the "SVG Texture Behaviour" script (menu `Component` → `Add`, then `Scripts subsection`). This script is really simple, in the detail the script:
  - Creates a drawing surface with the same dimensions
  - Creates and load the SVG document
  - Renders the SVG document to the drawing surface
  - Creates the texture with the specified dimensions
  - Copies the drawing surface pixels to the texture
  - Destroys SVG document and drawing surface
  - Returns the created texture
- Find the `girl.svg` file (located in `Assets/SVGAssets/SVGFiles`) and drag&drop it from the Project window to the `SVG File` property field of the script. Let other properties unchanged to their default values (`512 x 512` pixels texture dimensions, white background clear color).
- Click play and see the result.



This is the result after pressing Play

SVGAssets also includes another script called `SVGTextureAtlasBehaviour`; in order to test it, stop the current execution and perform the following steps:

- Select the plane and remove the attached `SVGTextureBehaviour` component
- Add the `SVGTextureAtlasBehaviour` script component: this script allows you to specify four different SVG files to be rendered on a single texture
- Drag&drop four SVG files from the Project window to `SVG File` `1,2,3,4` property fields of the script.
- As before, click play and see the result in the Game window.



Four SVG files rendered on a single texture

Credits

Thanks to "Ror362" for its great girl svg. Original source: https://www.deviantart.com/ror362/art/Yayoi-iM-S-397886689

Step-by-step tutorial: an animated character example

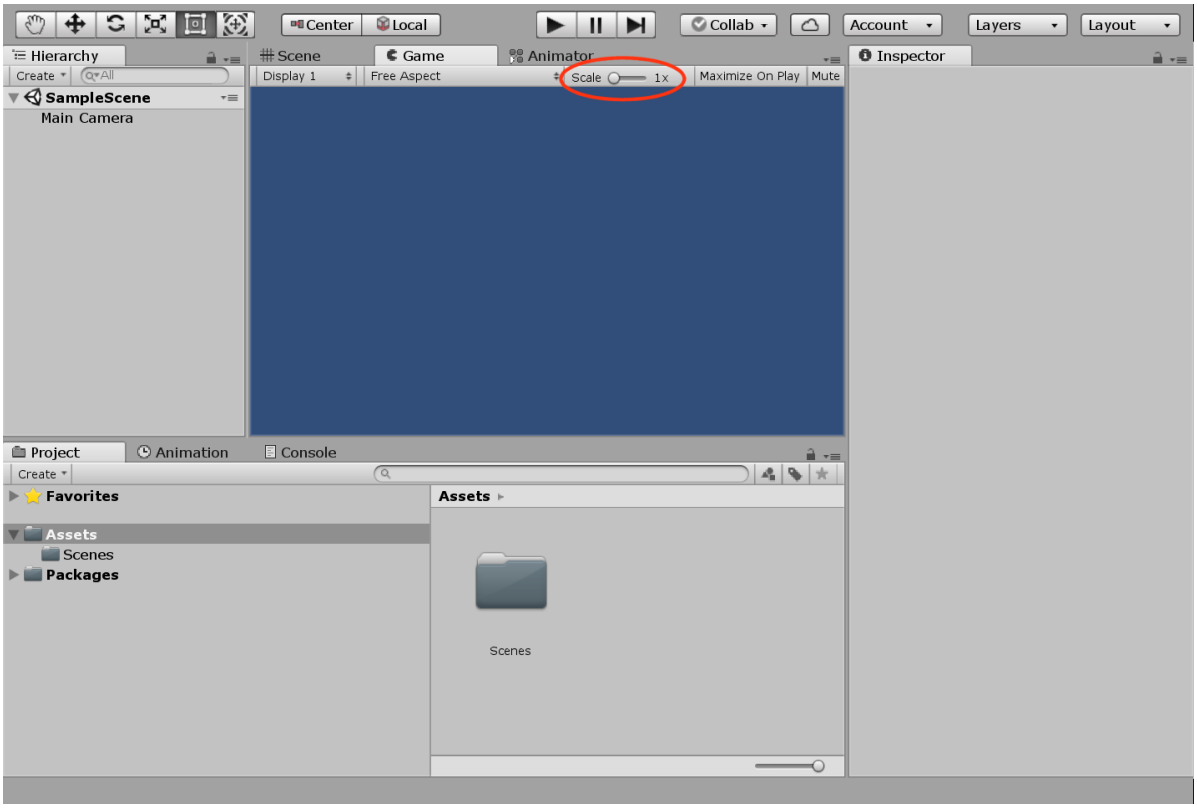
The example explained in this chapter is implemented in the `orc.unity` scene. The example shows a simple 2D character (the orc), walking on a (potentially scrollable) background. Both orc and background are native SVG files.

The orc asset is an SVG file where each body part is a separate first level group (`<g>` tag), each group has a well defined name (id attribute). This setup will allow to render each body part as a separate animable sprite. SVGAssets atlas generator will perform an optimized sprite packing routine in the editor, and later on the target device (at runtime).

Adobe Illustrator allows to show and edit groups

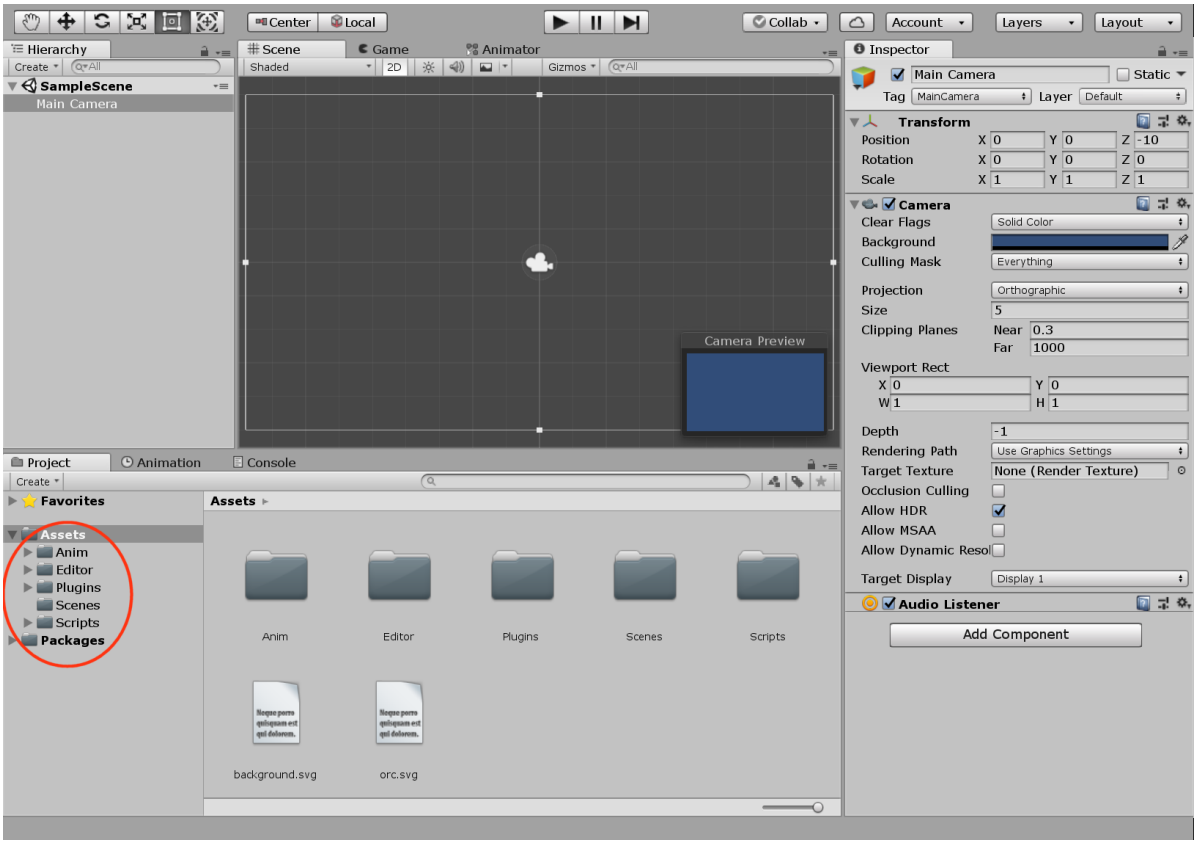
Now we can proceed in the following way:

- Create a new Unity project, choosing a 2D setup
- Select the Game view, and make sure that "Scale" slider is completely on the left (i.e. `scale = 1x`); switch back to the Scene view.



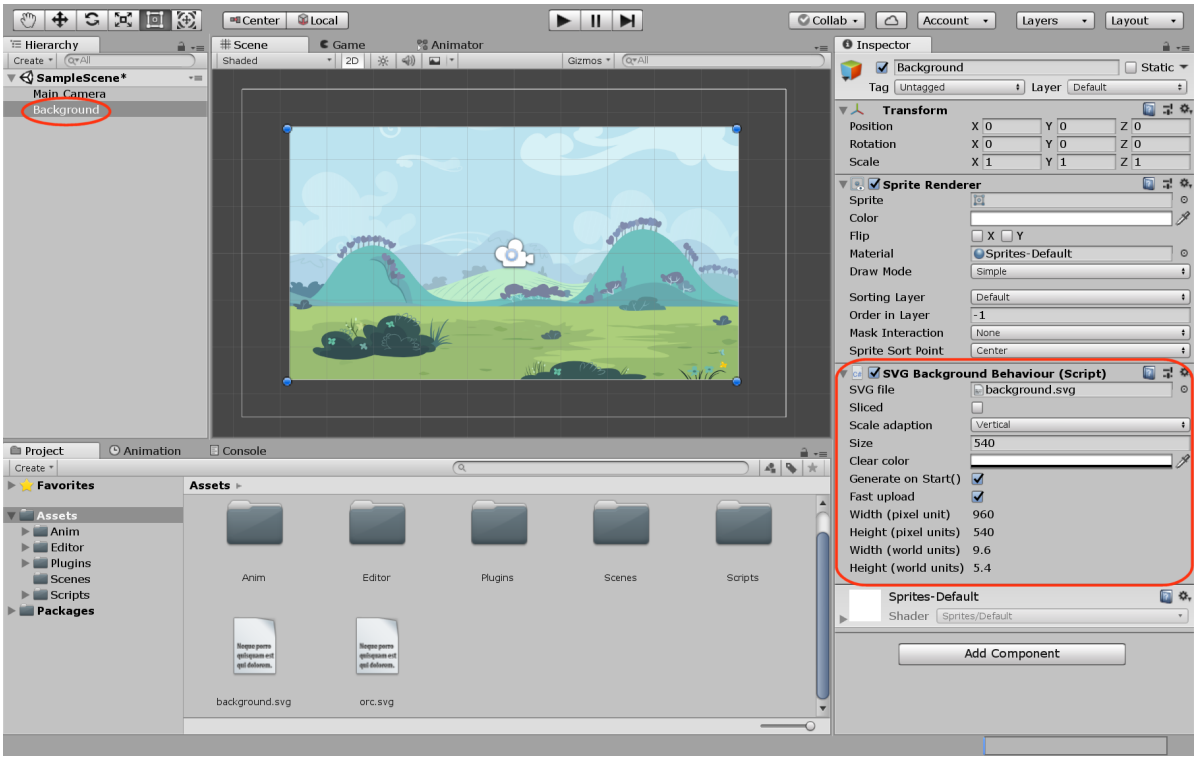
Set a unitary scene scale

- Because the project is a new one, it is required to copy SVGAssets folders (`Anim`, `Editor`, `Plugins`, `Resources`, `SVGFiles` and `Scripts`) inside the new project's `Assets/SVGAssets` folder; so that resources (e.g. config file, animations, fonts, SVG files), native AmanithSVG plugins and its C# interface will be available for the project.



The Orc project layout after copying all needed files

- The new scene should have been already populated with an orthographic camera; make sure that camera is positioned at `(0, 0, -10)`
- Create the sprite that we will use as background (menu `GameObject` → `2D Object` → `Sprite`). We rename the created object as `Background`, we position it at `(0, 0, 0)` and we set the `Order in Layer` value of the `SpriteRenderer` component to `-1`, so the background sprite will always be behind all other sprites. Now we can attach a `SVGBackgroundBehaviour` script to it (menu `Component` → `Add`, then `Scripts` subsection), then set its fields:
  - find the `background.svg` file (located in `Assets/SVGAssets/SVGFiles`) and drag&drop it from the Project window to the `SVG file` property
  - ensure that `Sliced` checkbox is unchecked
  - set scale adaption to `Vertical`, because we want the background to cover the whole device screen height, making the horizontal direction (potentially) scrollable
  - set the `Size` property to `540`, that is the original `background.svg` vertical size (i.e. `height` attribute)



The background SVG placed on the scene

- As you can see, the background still does not completely cover (vertically) the whole screen, but this is something that we're going to solve soon: first we must however create our orc character.
- Create an `SVGAtlas` atlas generator (menu `Assets` → `SVGAssets` → `Create SVG sprites atlas`), that will be used to create and pack all the sprites relative to the orc body parts. We rename the created asset in `orcAtlas` just to avoid confusion. Then we can proceed with its settings:
  - find the `orc.svg` file (located in `Assets/SVGAssets/SVGFiles`) and drag&drop it from the Project window to the region labeled with `Drag & drop SVG assets here`, then check the `Separate groups` option
  - set reference width and height to `960 x 540` (the dimensions of the background): at design time, we want to see if the orc character size is "compatible" with the background
  - set device test width and height to `960 x 540`, so in the editor we can simulate a device with a resolution equal to the background; setting a device test resolution equal to the reference resolution has the effect to generate all the orc sprites at their original dimension, as specified in the orc.svg file
  - Set scale adaption to vertical, for the same reason that we discussed for the background
  - Click the `Update` button
- At this point, the texture atlas and the sprites assets relative to orc character parts have been generated, and made available in the editor. Please note that each sprite starts with a pivot point equal to `[0.5, 0.5]`, that means "the center of sprite".

The generated orc sprite assets

- Now we can instantiate the whole orc (i.e. all its parts) by clicking the `Instantiate` button present in the row where the orc.svg file has been dropped. In this way, previously generated sprites assets will be instantiated in the scene.

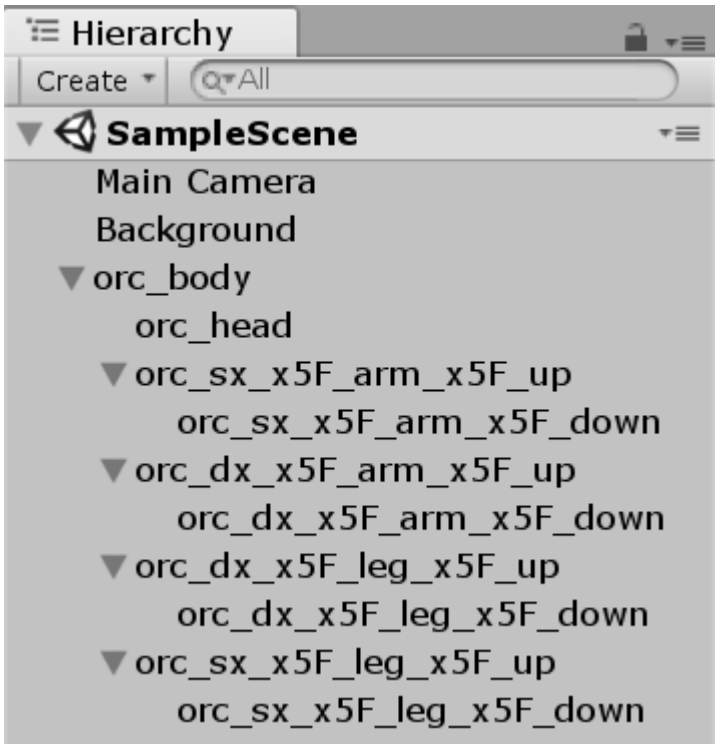
Orc has been instantiated with just one click

Please note that each instantiated sprite:

- has an attached SVGSpriteLoaderBehaviour script component, that will take care to regenerate the sprite on the device, at runtime, with the correct scale
- has the `Order in Layer` value of the `SpriteRenderer` component automatically set, in order to respect the correct z-order induced by the sequence in which groups appear within the orc.svg file

In order to animate each body part like a real character, we must setup:

- the correct hierarchy of all instantiated sprites (in the `Hierarchy` section on the left, simply drag&drop each child object under its father): we want the body part to be the root of our hierarchy



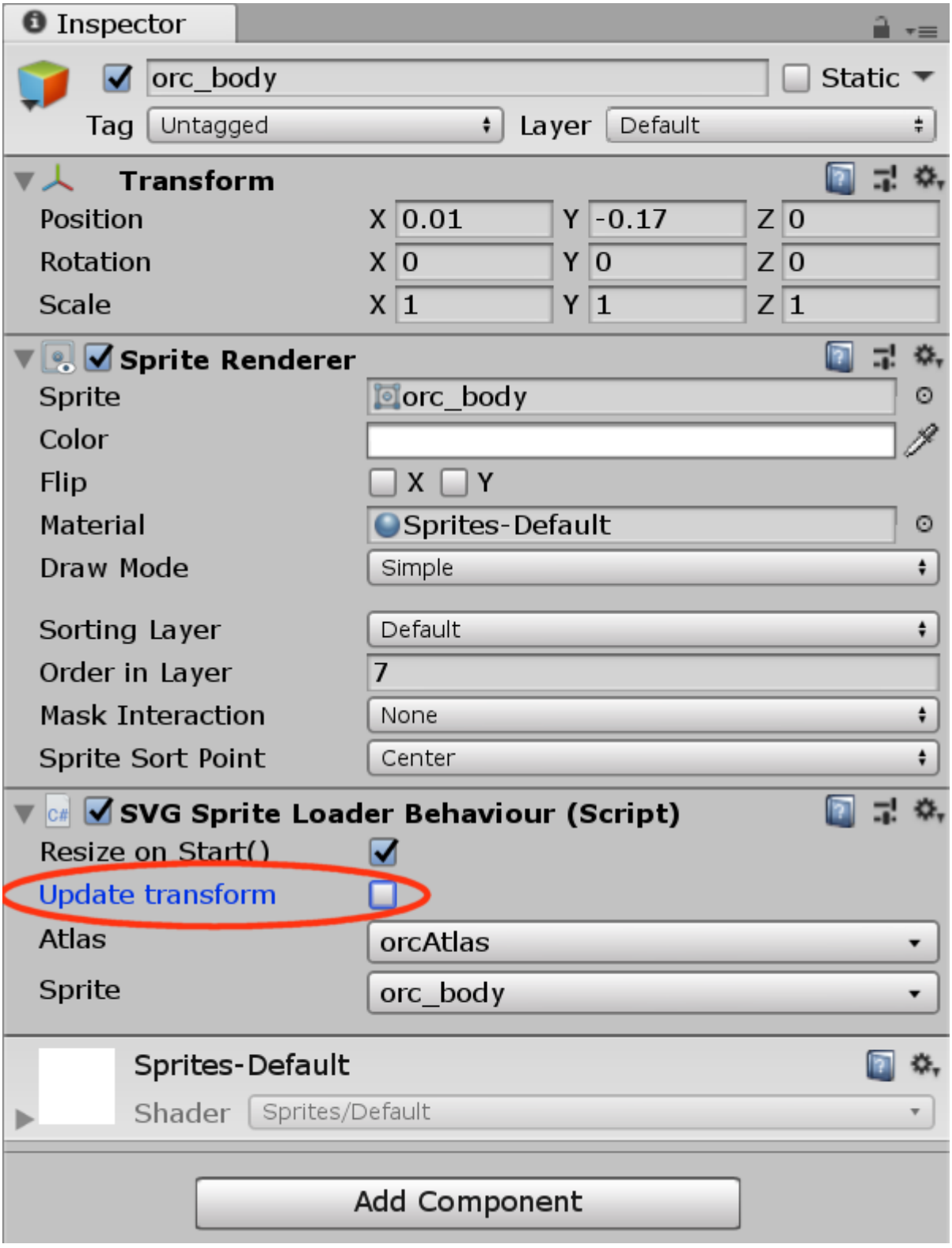
Orc hierarchy, body is the root

- the pivot points of each sprite: to do this, use the pivot editor made available by the `orcAtlas` object

The pivot editor window

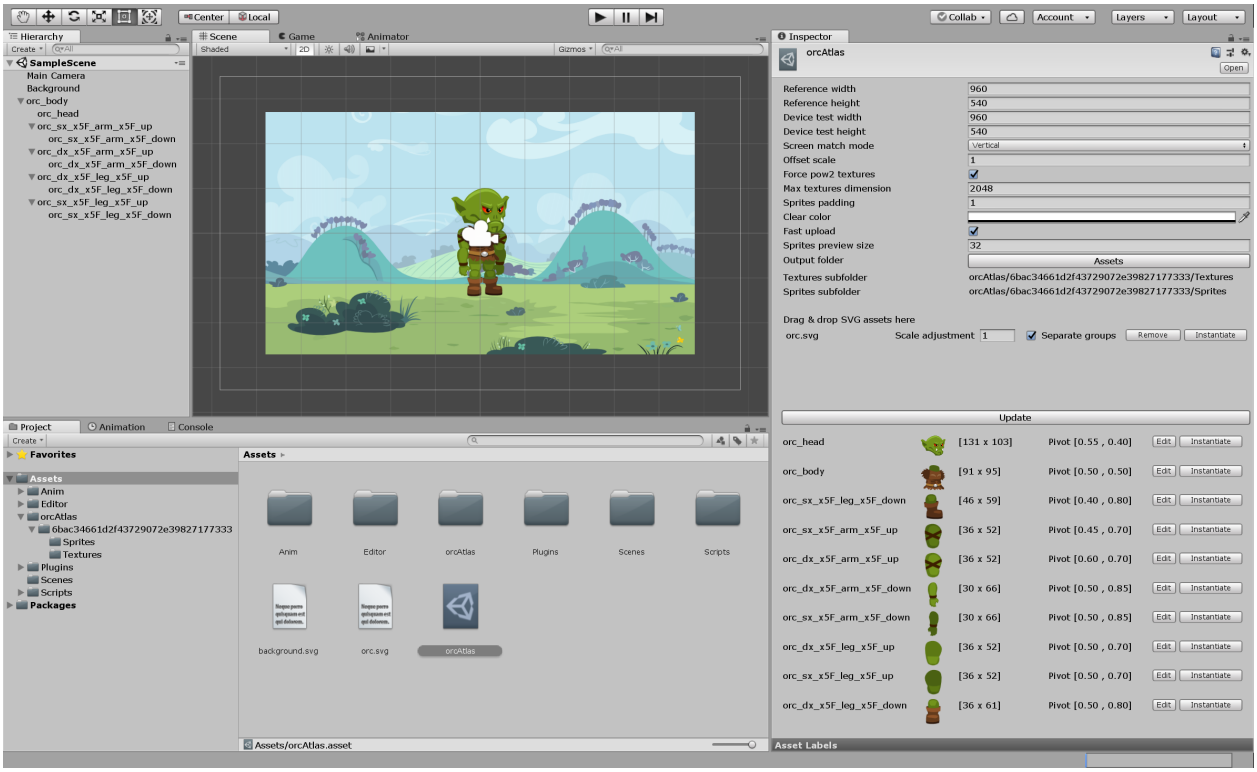
Sprite name	Pivot (x)	Pivot (y)
`orc_head`	`0.55`	`0.40`
`orc_body`	`0.50`	`0.50`
`orc_dx_x5F_arm_x5F_down`	`0.50`	`0.80`
`orc_dx_x5F_arm_x5F_up`	`0.60`	`0.70`
`orc_dx_x5F_leg_x5F_down`	`0.50`	`0.80`
`orc_dx_x5F_leg_x5F_up`	`0.50`	`0.70`
`orc_sx_x5F_arm_x5F_down`	`0.50`	`0.85`
`orc_sx_x5F_arm_x5F_up`	`0.45`	`0.70`
`orc_sx_x5F_leg_x5F_down`	`0.40`	`0.80`
`orc_sx_x5F_leg_x5F_up`	`0.50`	`0.70`

Because we are going to move the orc (actually the body part, and consequently the whole children hierarchy) programmatically by code, we must select the `orc_body` object and uncheck the `Update transform` option of the `SVGSpriteLoaderBehaviour` component.



"Update transform" must be unchecked for the body part

Now the scene is almost complete, so we can save it.



The scene is now complete

If we click the Play button, we see that:

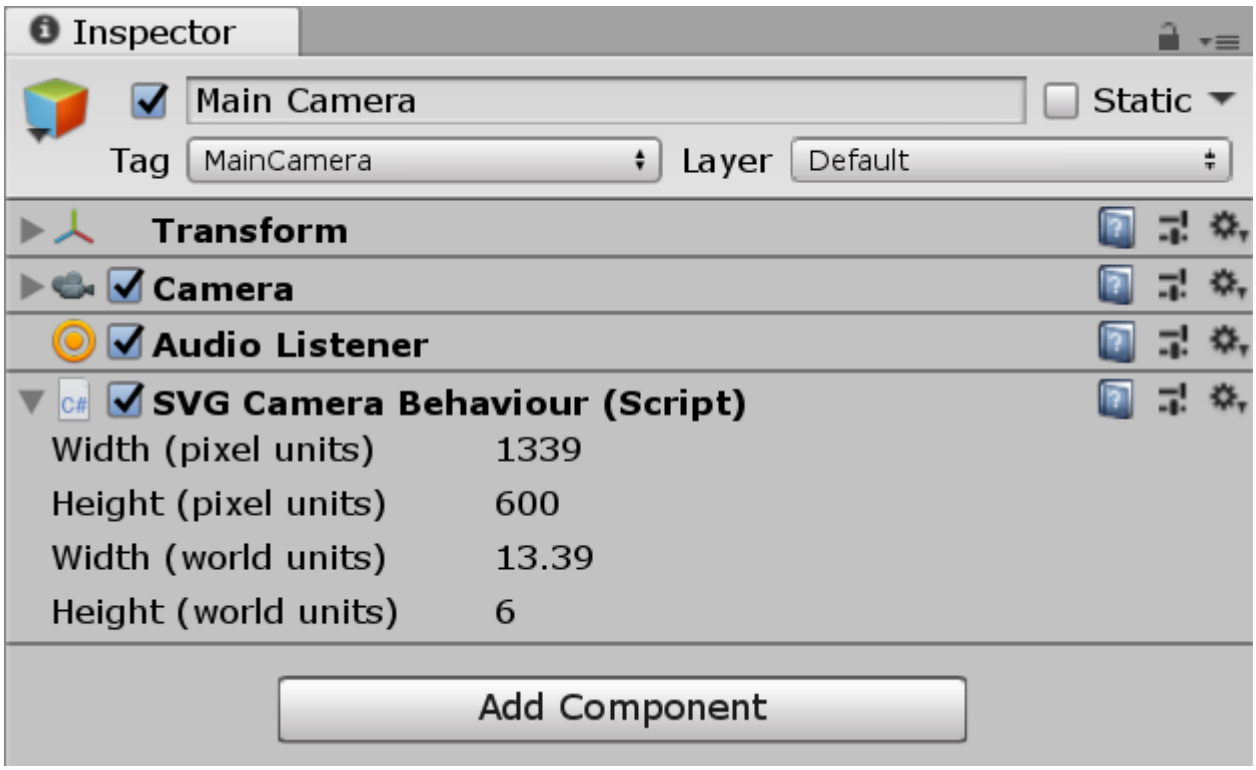
- orc sprites are resized according to the `Game` view, but the character position is not consistent respect to the background
- background sprite is not resized
- the main camera viewing volume is not covering the exact height of the background

Playing the scene

The reason is because at design time, in the editor, all sprites and their positions are valid for a (test) device resolution equal to `960 x 540`. So, we want to:

- make the camera viewing volume (actually the vertical size) to cover the whole screen height: this means to set the `Camera.orthographicSize` properly
- send a resize event to the background sprite: this means to set its `SVGBackgroundBehaviour.Size` property
- send a resize event to the orc character: this means to call the orc body sprite (actually its `SVGSpriteLoaderBehaviour` component) `UpdateSprite` function, specifying to update children too

In order to accomplish these tasks, we add a SVGCameraBehaviour component to the main camera (menu `Component` → `Add`, then `Scripts` subsection): this script, at each monobehaviour `Update` call, checks for a screen resolution change and, if this event occurs, first it sets the `Camera.orthographicSize` properly, then it informs all its listeners through the `OnResize` event.



The SVGCameraBehaviour script detects if screen resolution has changed

At this point, we add a script to the `orc_body` sprite (lets call it `OrcCharacterBehaviour`), that has the following input fields:

- the scene camera, so we can implement and register the `OnResize` event handler
- the background sprite, so we can set its `Size` property within the `OnResize` event handler

Because we are going to resize (i.e. generate at runtime) both background and orc sprites by C# code, before to implement the `OrcCharacterBehaviour`, we must:

- select the background sprite object and uncheck the `Generate on Start()` option
- select all the sprites that form the orc and uncheck the `Resize on Start()` option

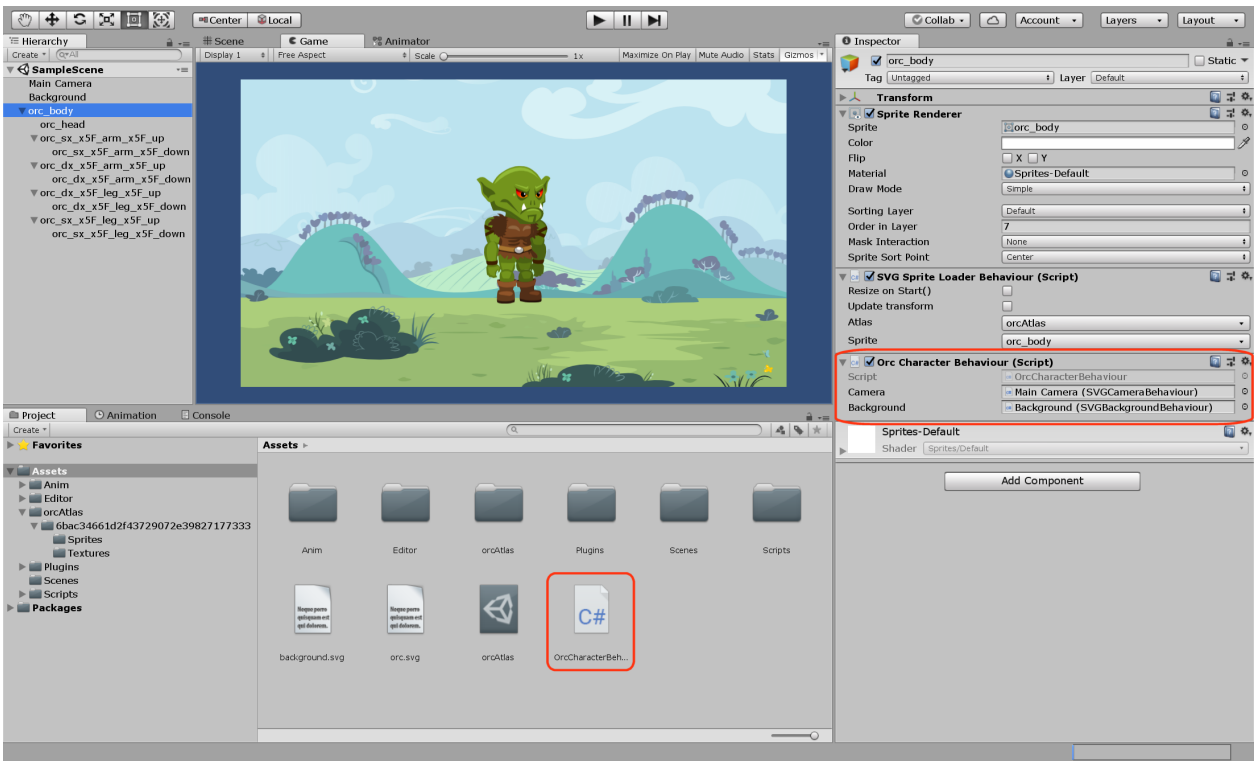
Then we can select the `orc_body` sprite and implement the `OrcCharacterBehaviour` script as follow (menu `Component` → `Add` → `New script`):

```
using UnityEngine;

public class OrcCharacterBehaviour : MonoBehaviour {

    private float BackgroundWalkingLine()
    {
        // walking line is located at ~12% of the background half height, in world
coordinates
        return (Background != null) ? (-Background.WorldHeight * 0.5f) * 0.12f : 0.0f;
    }

    private void ResetOrcPos()
    {
        // move the orc at the walking line
        transform.position = new Vector3(0, BackgroundWalkingLine(), 0);
    }

    private void ResizeBackground(int newScreenWidth, int newScreenHeight)
    {
        // we want to cover the whole screen
        Pair<SVGBackgroundScaleType, int> scaleData =
Background.CoverFullScreen(newScreenWidth, newScreenHeight);
        Background.ScaleAdaption = scaleData.First;
        Background.Size = scaleData.Second;
        Background.UpdateBackground(false);
    }

    private void ResizeOrcCharacter(int backgroundWidth, int backgroundHeight)
    {
        // get the orc (body) sprite loader
        SVGSpriteLoaderBehaviour spriteLoader =
gameObject.GetComponent<SVGSpriteLoaderBehaviour>();
        // update/regenerate all orc sprites; NB: we want to size the orc according to
        // the background sprite (actually the background height)
        spriteLoader.UpdateSprite(true, backgroundWidth, backgroundHeight);
    }

    private void OnResize(int newScreenWidth, int newScreenHeight)
    {
        // render the background so that it covers the whole screen
        ResizeBackground(newScreenWidth, newScreenHeight);
        // update/regenerate all orc sprites according to the background dimensions
        ResizeOrcCharacter((int)Background.PixelWidth, (int)Background.PixelHeight);
        // move the orc at the world origin
        ResetOrcPos();
    }

    // Use this for initialization
```

```
    void Start()
    {
        // register ourself for receiving resize events
        Camera.OnResize += OnResize;
        // now fire a resize event by hand
        Camera.Resize(true);
    }

    // the scene camera
    public SVGCameraBehaviour Camera;
    // the background gameobject
    public SVGBackgroundBehaviour Background;
}
```

Before to play the scene, we must set `OrcCharacterBehaviour` properties; so we must drag&drop:

- The `Main Camera` gameobject to the `Camera` property
- The `Background` gameobject to the `Background` property



The OrcCharacterBehaviour script, attached to the body sprite

Now if we click play, we see that the camera viewing volume is covering the whole screen, the background sprite is resized according to the screen dimensions and the orc character has the same vertical proportion respect to the background! The next step is to move the orc along the horizontal axis, and enable the camera to follow it and scroll the background. To accomplish these tasks, the `OrcCharacterBehaviour` script must implement a function that derives the camera position according to the orc position:

```
private Vector3 CameraPosCalc(Vector3 orcPos)
{
    // set the camera according to the orc position
```

```
    Vector3 cameraPos = new Vector3(orcPos.x, 0, -10);
    float cameraWorldLeft = cameraPos.x - (Camera.WorldWidth / 2);
    float cameraWorldRight = cameraPos.x + (Camera.WorldWidth / 2);
    // make sure the camera won't go outside the background
    if (cameraWorldLeft < Background.WorldLeft)
    {
        cameraPos.x += Background.WorldLeft - cameraWorldLeft;
    }
    else
    if (cameraWorldRight > this.Background.WorldRight)
    {
        cameraPos.x -= cameraWorldRight - Background.WorldRight;
    }
    return cameraPos;
}

private void CameraPosAssign(Vector3 orcPos)
{
    // set the camera according to the orc position
    Camera.transform.position = (Camera.PixelWidth > Background.PixelWidth) ? new
Vector3(0, 0, -10) : CameraPosCalc(orcPos);
}
```

and two functions to move the orc left or right, changing its body world position (the y coordinate will be fixed to the walking line); such functions will be called in the LateUpdate routine, if the user has pressed the mouse button:

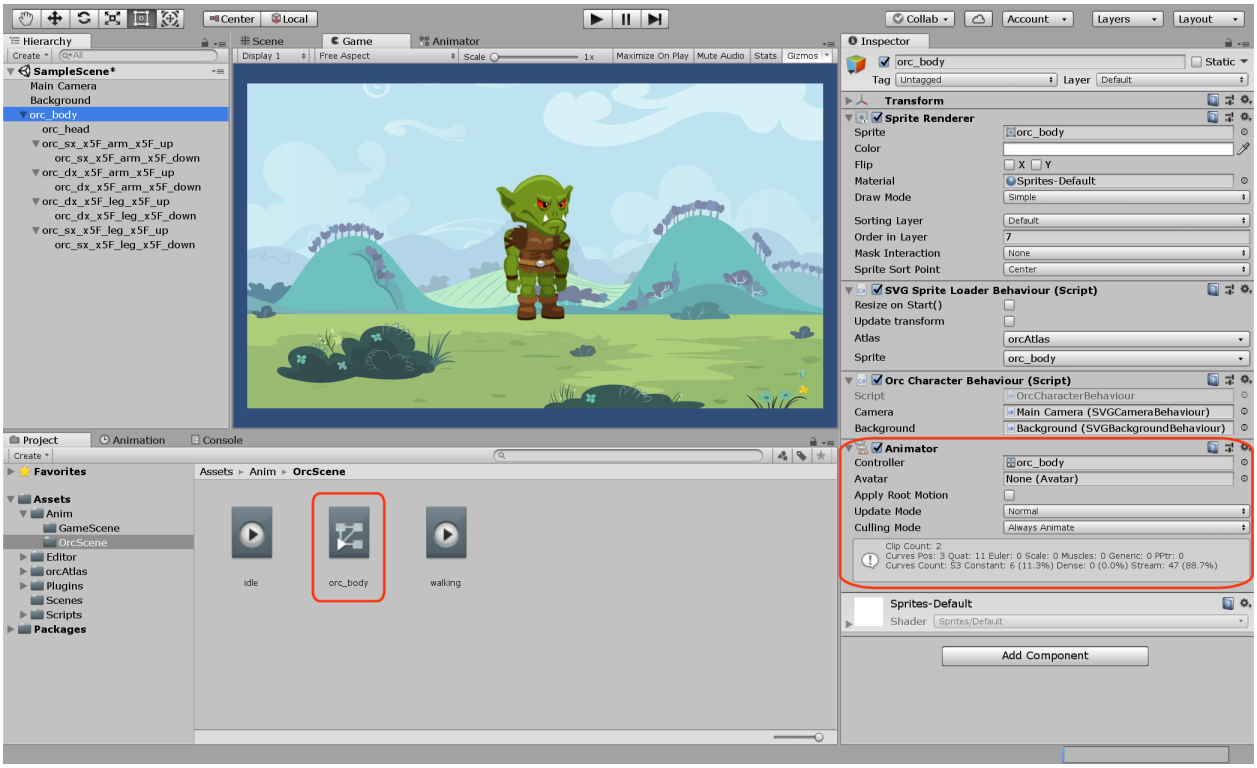
```
private void Move(Vector3 delta)
{
    // move the orc
    Vector3 orcPos = transform.position + delta;
    // get the orc (body) sprite loader
    SpriteRenderer spriteRenderer = gameObject.GetComponent<SpriteRenderer>();
    float orcBodyWidth = spriteRenderer.sprite.bounds.size.x;
    // orc body pivot is located at 50% of the whole orc sprite, so we can calculate
bounds easily
    float orcWorldLeft = orcPos.x - (orcBodyWidth / 2);
    float orcWorldRight = orcPos.x + (orcBodyWidth / 2);
    // make sure the orc won't go outside the background
    if (orcWorldLeft < Background.WorldLeft)
    {
        orcPos.x += (Background.WorldLeft - orcWorldLeft);
    }
    else
    if (orcWorldRight > Background.WorldRight)
    {
        orcPos.x -= (orcWorldRight - Background.WorldRight);
    }
    // update the orc position
    transform.position = orcPos;
    // set the camera according to the orc position
    this.CameraPosAssign(orcPos);
}
```

```
private void MoveLeft()
{
    // flip the orc horizontally
    this.transform.localScale = new Vector3(-1, this.transform.localScale.y,
this.transform.localScale.z);
    this.Move(new Vector3(-WALKING_SPEED, 0, 0));
}

private void MoveRight()
{
    this.transform.localScale = new Vector3(1, this.transform.localScale.y,
this.transform.localScale.z);
    this.Move(new Vector3(WALKING_SPEED, 0, 0));
}

void LateUpdate()
{
    if (Input.GetButton("Fire1"))
    {
        Vector3 worldMousePos = Camera.GetComponent<Camera>
().ScreenToWorldPoint(Input.mousePosition);
        if (worldMousePos.x > transform.position.x)
        {
            this.MoveRight();
        }
        else
        if (worldMousePos.x < transform.position.x)
        {
            this.MoveLeft();
        }
    }
}

// the walking speed, in world coordinates
private const float WALKING_SPEED = 0.04f;
```

As you can see, all the code (orc and camera movement) is based on world coordinates. The source code of the final script can be found in `Assets/SVGAssets/Scripts/OrcScene/OrcBehaviour.cs` file.

In order to complete the example, we must animate the orc; we can use the "idle" and "walking" animations that we have already prepared in the `Assets/SVGAssets/Anim/OrcScene` folder:

- select the `orc_body` gameobject
- add an `Animator` component (menu `Component` → `Miscellaneous` → `Animator`)
- drag&drop the `orc_body` controller from the `Assets/SVGAssets/Anim/OrcScene` to the Animator's `Controller` field



Animation attached to the orc body

Save the scene and click play and see the animated orc, starting with the "idle" animation; by pressing the mouse (or tapping the device screen), the orc will walk towards right or left.

Now you can have fun experimenting with it!

Credits

- Thanks to Chris Hildenbrand for its awesome orc svg. Original source: http://2dgameartforprogrammers.blogspot.com
- Thanks to Casper Tang Veje ("adcoon") for its beautiful background svg. Original source: https://www.deviantart.com/adcoon

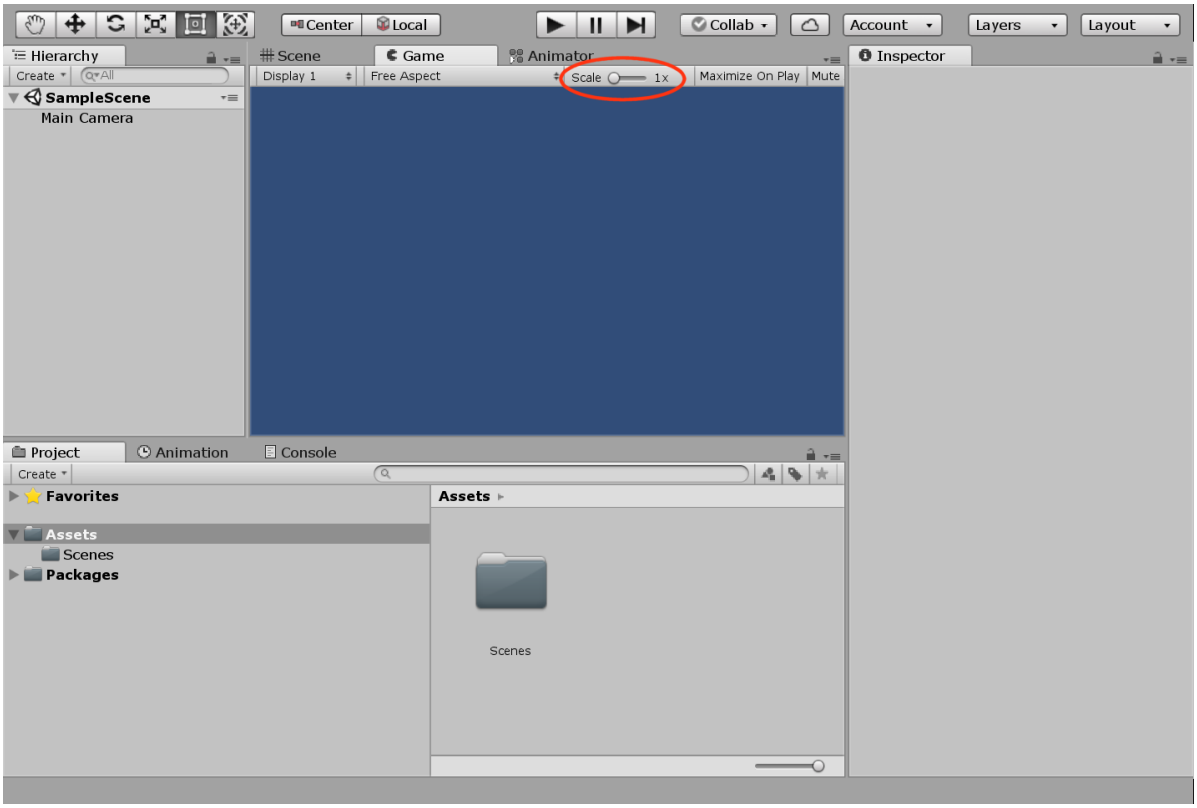
Step-by-step tutorial: a game example

In this tutorial, we will create a simple prototype of a memory-like game, using Unity and SVGAssets. The example explained in this chapter is implemented in the `game.unity` scene.

The setup

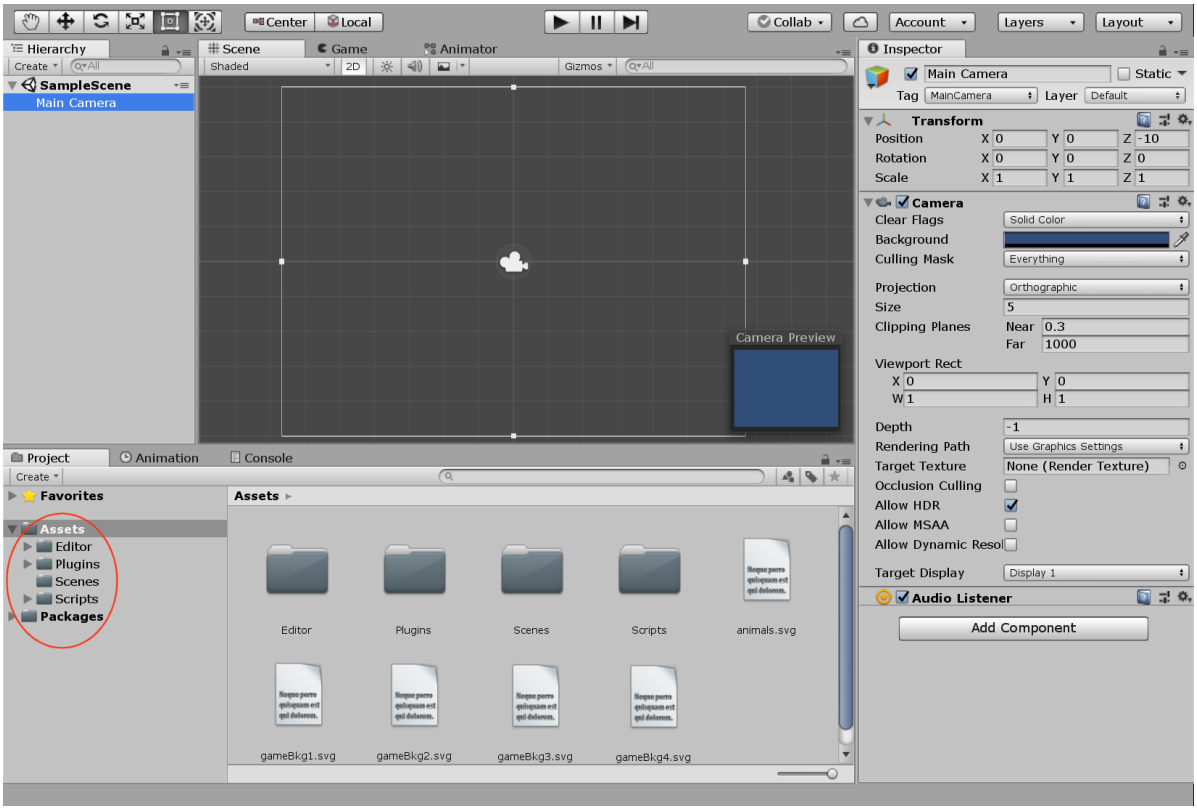
First, lets create the project structure:

- Create a new Unity project, choosing a 2D setup
- Select the Game view, and make sure that "Scale" slider is completely on the left (i.e. `scale = 1x`); switch back to the Scene view.



Set a unitary scene scale

- Because the project is a new one, it is required to copy SVGAssets folders (`Anim`, `Editor`, `Plugins`, `Resources`, `SVGFiles` and `Scripts`) inside the new project's `Assets/SVGAssets` folder; so that resources (e.g. config file, animations, fonts, SVG files), native AmanithSVG plugins and its C# interface will be available for the project.
- The new scene should have been already populated with an orthographic camera; make sure that camera is positioned at `(0, 0, -10)`



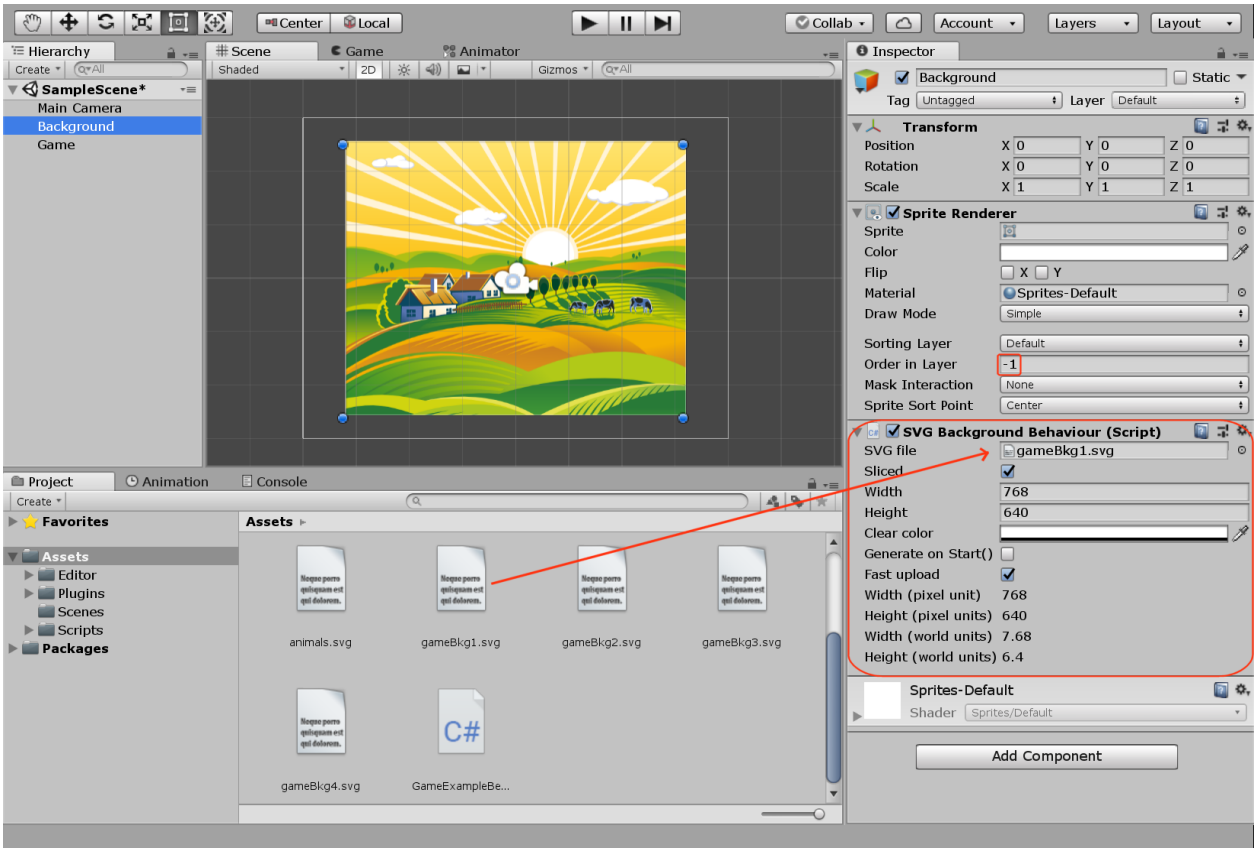
The Game project layout after copying all needed files

The game background

Like we did in the previous tutorial, in order to setup a background that covers the whole device screen, we proceed with the following steps:

- we add a SVGCameraBehaviour component to the main camera (menu `Component` → `Add`, then `Scripts` subsection): this script, at each monobehaviour `Update` call, checks for a screen resolution change and, if this event occurs, first it sets the `Camera.orthographicSize` properly, then it informs all its listeners through the `OnResize` event.
- we create the sprite that we will use as background (menu `GameObject` → `2D Object` → `Sprite`). We rename the created object as `Background`, we position it at `(0, 0, 0)` and we set the `Order in Layer` value of the `SpriteRenderer` component to `-1`, so the background sprite will always be behind all other sprites. Now we attach a `SVGBackgroundBehaviour` script to it (menu `Component` → `Add`, then `Scripts` subsection), then set its fields:
  - drag&drop one of the four backgrounds (e.g. `Assets/SVGAssets/SVGFiles/gameBkg1.svg`) file to the `SVG file` property
  - check the `Sliced` checkbox, because we don't want to generate a potentially scrollable background (see the relative documentation)
  - uncheck the `Generate on Start()` checkbox (we will perform the generation at runtime, when the camera `OnResize` event will inform us about a screen resolution change)
  - set `Width` and `Height` values to something valid (please set `768 x 640`, the reason will be clear in the next paragraph), just to have something visible in the editor

- we create an "empty" `GameObject` (menu `GameObject` → `Create Empty`) that will represent our main entry point (i.e. the game state and logic), we rename the created object as `Game`, then we add a script to it (menu `Component` → `Add` → `New script`); lets call it `GameExampleBehaviour` (the final full implementation can be seen in `Assets/SVGAssets/Scripts/GameScene/GameBehaviour.cs` file).



The basic template of our project

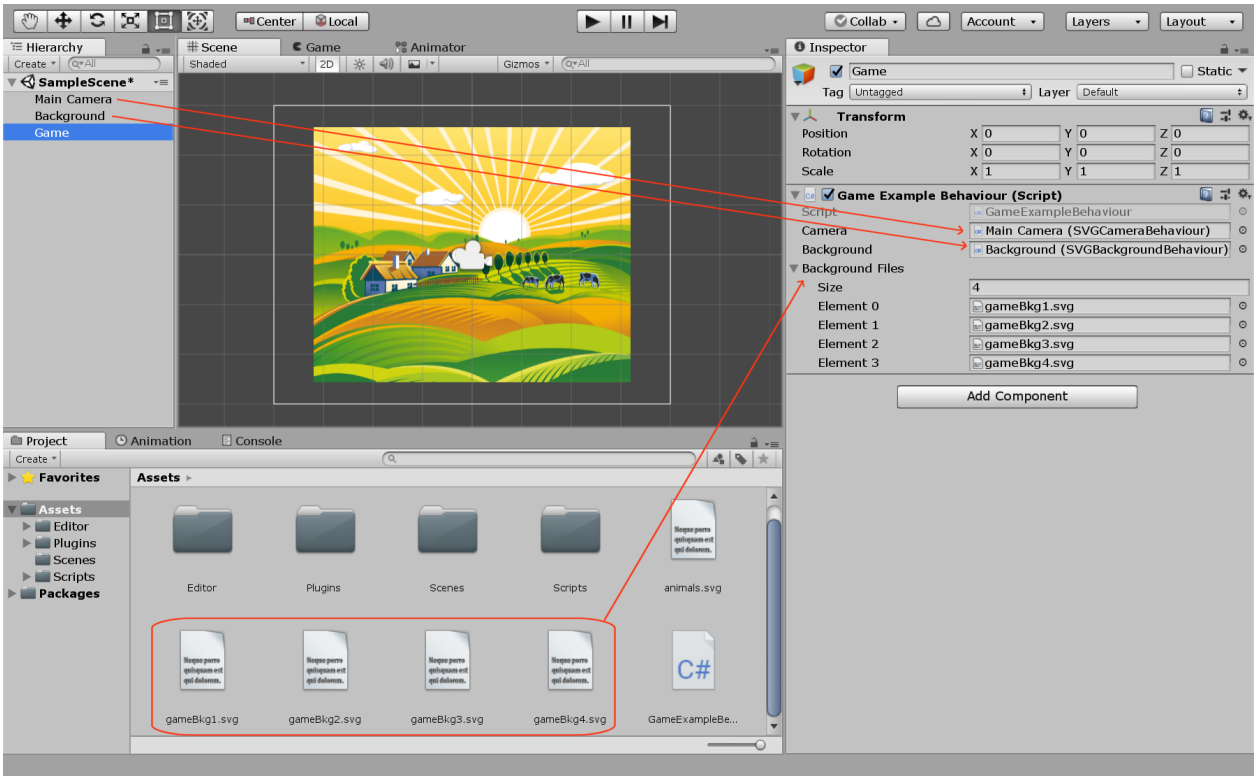
Here is `GameExampleBehaviour` basic C# layout (the code is self-explanatory):

```
using System;
using UnityEngine;

public class GameExampleBehaviour : MonoBehaviour {

    private void ResizeBackground(int newScreenWidth, int newScreenHeight)
    {
        // we want to cover the whole screen
        Background.SlicedWidth = newScreenWidth;
        Background.SlicedHeight = newScreenHeight;

        // generate the background texture at the desired resolution
        Background.UpdateBackground(true);
    }

    private void OnResize(int newScreenWidth, int newScreenHeight)
    {
        // resize the background
        ResizeBackground(newScreenWidth, newScreenHeight);
```

```
    }

    private void StartNewGame()
    {
        // destroy current background texture
        Background.DestroyAll(true);

        // assign a new SVG file
        int idx = (backgroundIndex % BackgroundFiles.Length);
        Background.SVGFile = BackgroundFiles[idx];

        // advance for the next background SVG
        backgroundIndex++;
    }

    // Use this for initialization
    void Start()
    {
        // start with the first background
        backgroundIndex = 0;

        // start a new game
        StartNewGame();

        // register ourself for receiving resize events
        Camera.OnResize += OnResize;
        // now fire a resize event by hand
        Camera.Resize(true);
    }

    // the main camera, used to intercept screen resize events
    public SVGCameraBehaviour Camera;
    // the game background
    public SVGBackgroundBehaviour Background;
    // array of usable SVG backgrounds
    public TextAsset[] BackgroundFiles;

    // the current background (i.e. the index within the BackgroundFiles array)
    [NonSerialized]
    private int backgroundIndex;
}
```

The last thing left to do is to assign the public fields, by drag&drop:

- the camera gameobject to the `Camera` field
- the background gameobject to the `Background` field
- the four background SVG (`gameBkg1.svg`, `gameBkg2.svg`, `gameBkg3.svg`, `gameBkg4.svg`) to the `BackgroundFiles` field



The game starts to take form: camera and the background sprite...done!

Now if we click play, we see that the camera viewing volume is covering the whole screen and the background sprite is resized according to the screen dimensions.

The cards

The characters hidden behind the cards are cute animals and we model a single card instance as a set of basic attributes (see full implementation in `Assets/SVGAssets/Scripts/GameScene/GameCardBehaviour.cs` file). Lets start by creating a new C# script (menu `Assets` → `Create` → `C# Script`) and call it `GameExampleCardBehaviour`:

```
using System;
using UnityEngine;
#if UNITY_EDITOR
    using UnityEditor;
#endif

public enum GameExampleCardType
{
    Undefined  =  -2,
    BackSide   =  -1,
    Panda      =   0,
    Monkey     =   1,
    Orangutan  =   2,
    Panther    =   3,
    Puma       =   4,
    Leopard    =   5,
    Lion       =   6,
    Cougar     =   7,
```

```
    Tiger      =  8,
    Elephant   =  9,
    Penguin    = 10,
    Zebra      = 11,
    Hen        = 12,
    Rooster    = 13,
    Pig        = 14,
    Dog        = 15,
    Rabbit     = 16,
    Owl        = 17,
    Sheep      = 18,
    Cat        = 19,
    Deer       = 20,
    Donkey     = 21,
    Cow        = 22,
    Fox        = 23
};

[ExecuteInEditMode]
public class GameExampleCardBehaviour : MonoBehaviour {

    // given an animal type, it returns the relative sprite name
    public static string AnimalSpriteName(GameExampleCardType animalType)
    {
        switch (animalType)
        {
            case GameExampleCardType.BackSide:
                return("animals_back");
            case GameExampleCardType.Panda:
                return("animals_Panda");
            case GameExampleCardType.Monkey:
                return("animals_Monkey");
            case GameExampleCardType.Orangutan:
                return("animals_Orangutan");
            case GameExampleCardType.Panther:
                return("animals_Panther");
            case GameExampleCardType.Puma:
                return("animals_Puma");
            case GameExampleCardType.Leopard:
                return("animals_Leopard");
            case GameExampleCardType.Lion:
                return("animals_Lion");
            case GameExampleCardType.Cougar:
                return("animals_Cougar");
            case GameExampleCardType.Tiger:
                return("animals_Tiger");
            case GameExampleCardType.Elephant:
                return("animals_Elephant");
            case GameExampleCardType.Penguin:
                return("animals_Penguin");
            case GameExampleCardType.Zebra:
                return("animals_Zebra");
            case GameExampleCardType.Hen:
                return("animals_Hen");
            case GameExampleCardType.Rooster:
```

```
                return("animals_Rooster");
            case GameExampleCardType.Pig:
                return("animals_Pig");
            case GameExampleCardType.Dog:
                return("animals_Dog");
            case GameExampleCardType.Rabbit:
                return("animals_Rabbit");
            case GameExampleCardType.Owl:
                return("animals_Owl");
            case GameExampleCardType.Sheep:
                return("animals_Sheep");
            case GameExampleCardType.Cat:
                return("animals_Cat");
            case GameExampleCardType.Deer:
                return("animals_Deer");
            case GameExampleCardType.Donkey:
                return("animals_Donkey");
            case GameExampleCardType.Cow:
                return("animals_Cow");
            case GameExampleCardType.Fox:
                return("animals_Fox");
            default:
                return("");
        }
    }

    // Number of total animal types
    public static int AnimalsCount() {

        return (GameExampleCardType.Fox - GameExampleCardType.Panda) + 1;
    }

    // Select a random animal
    public static GameExampleCardType RandomAnimal() {

        int v = (int)(UnityEngine.Random.value * (float)AnimalsCount())
              + (int)GameExampleCardType.Panda;

        return (GameExampleCardType)v;
    }

    // Get the next animal in the list
    public static GameExampleCardType NextAnimal(GameExampleCardType current) {

        int next = (((int)current + 1) % AnimalsCount()) + (int)CardType.Panda;
        return (GameExampleCardType)next;
    }

#if UNITY_EDITOR
    // Reset is called when the user hits the Reset button in the Inspector's
    // context menu or when adding the component the first time.
    // This function is only called in editor mode. Reset is most commonly used
    // to give good default values in the inspector.
    void Reset()
    {
```

```
        Active = true;
        BackSide = true;
        AnimalType = GameExampleCardType.Undefined;
        Game = null;
    }
#endif

    // true if card is active (i.e. still part of the current game), else false
    public bool Active;
    // true if card is back side, false if the card is turned
    // (i.e. we can see the animal character)
    public bool BackSide;
    // the animal character associated with this card
    public GameExampleCardType AnimalType;
    // a link to the game main script
    public GameExampleBehaviour Game;
}
```

Animals vector graphics are defined within a single SVG file `animals.svg`, where each animal is a single first-level group (a `<g>` element):

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"https://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1"
     id="Animals"
     xmlns="https://www.w3.org/2000/svg"
     xmlns:xlink="https://www.w3.org/1999/xlink"
     width="768px" height="640px"
     viewBox="0 0 3072 2560"
     xml:space="preserve">
    <g id="Panda">...</g>
    <g id="Monkey">...</g>
    <g id="Orangutan">...</g>
    <g id="Panther">...</g>
    <g id="Puma">...</g>
    <g id="Leopard">...</g>
    <g id="Lion">...</g>
    <g id="Cougar">...</g>
    <g id="Tiger">...</g>
    <g id="Elephant">...</g>
    <g id="Penguin">...</g>
    <g id="Zebra">...</g>
    <g id="Hen">...</g>
    <g id="Rooster">...</g>
    <g id="Pig">...</g>
    <g id="Dog">...</g>
    <g id="Rabbit">...</g>
    <g id="Owl">...</g>
    <g id="Sheep">...</g>
    <g id="Cat">...</g>
    <g id="Deer">...</g>
    <g id="Donkey">...</g>
    <g id="Cow">...</g>
```

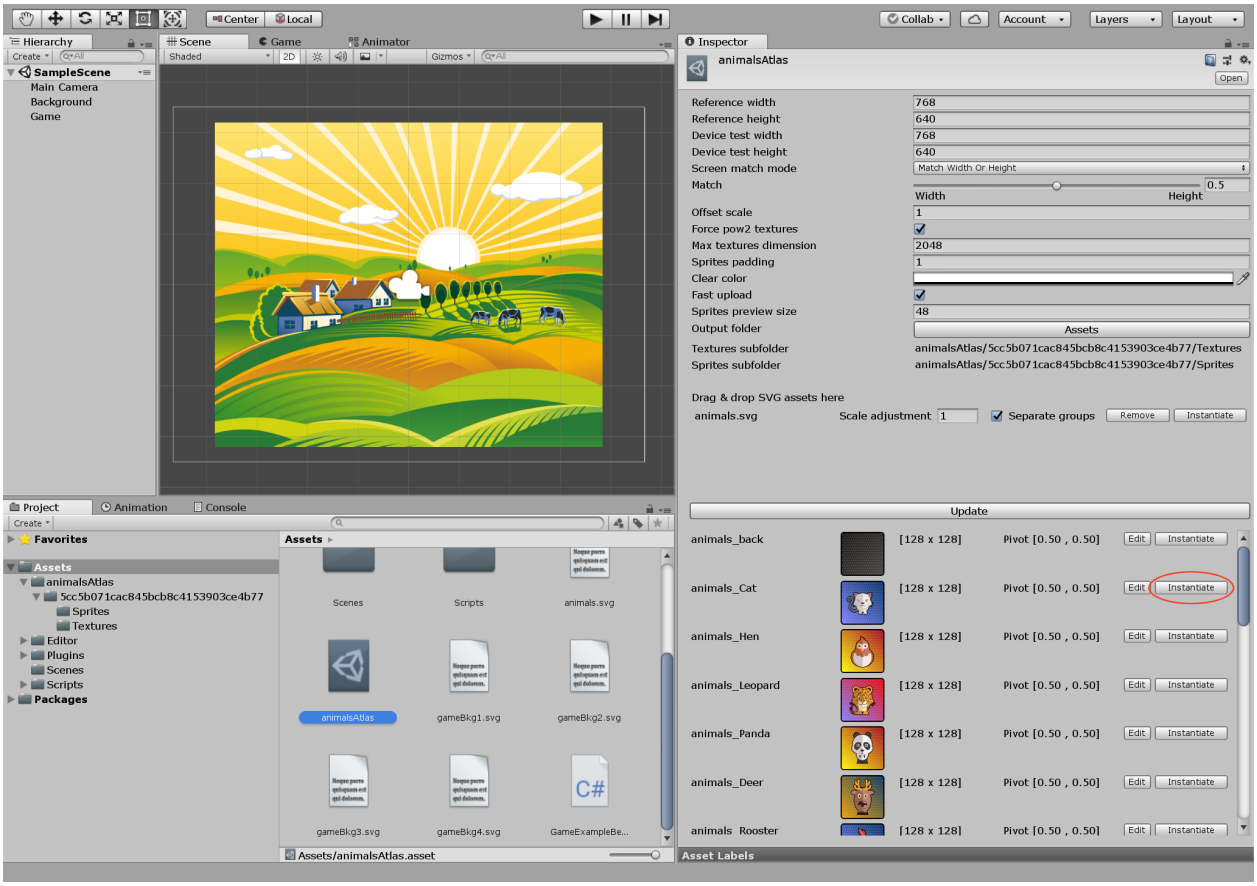
```
    <g id="Fox">...</g>
    <g id="back">...</g>
</svg>
```

animals.svg

As you can see from the `animals.svg` header, it has been designed for a `768 x 640` resolution (the same that we have chosen, not by chance, for the background).
Now we want to implement a function that generates animals sprites from the given SVG file, taking care of the current screen resolution. This is easy with SVGAssets, we make use of SVGAtlas class. We create an atlas generator (menu `Assets` → `SVGAssets` → `Create SVG sprites atlas`), that will be used to create and pack all the sprites relative to the animals. We rename the created asset as 'animalsAtlas' just to avoid confusion. Then we can proceed with its settings:

- drag&drop the `animals.svg` file to the region labeled with `Drag & drop SVG assets` here and check the `Separate groups` option
- set reference width and height to `768 x 640` (the dimensions of the background): so the animals sprites size will be "compatible" with the background
- set device test width and height to `768 x 640`, so in the editor we can simulate a device with a resolution equal to the background
- leave the `Screen match mode` to the defalt `Match Width Or Height`
- leave the `Match` slider to the default `0.5` value

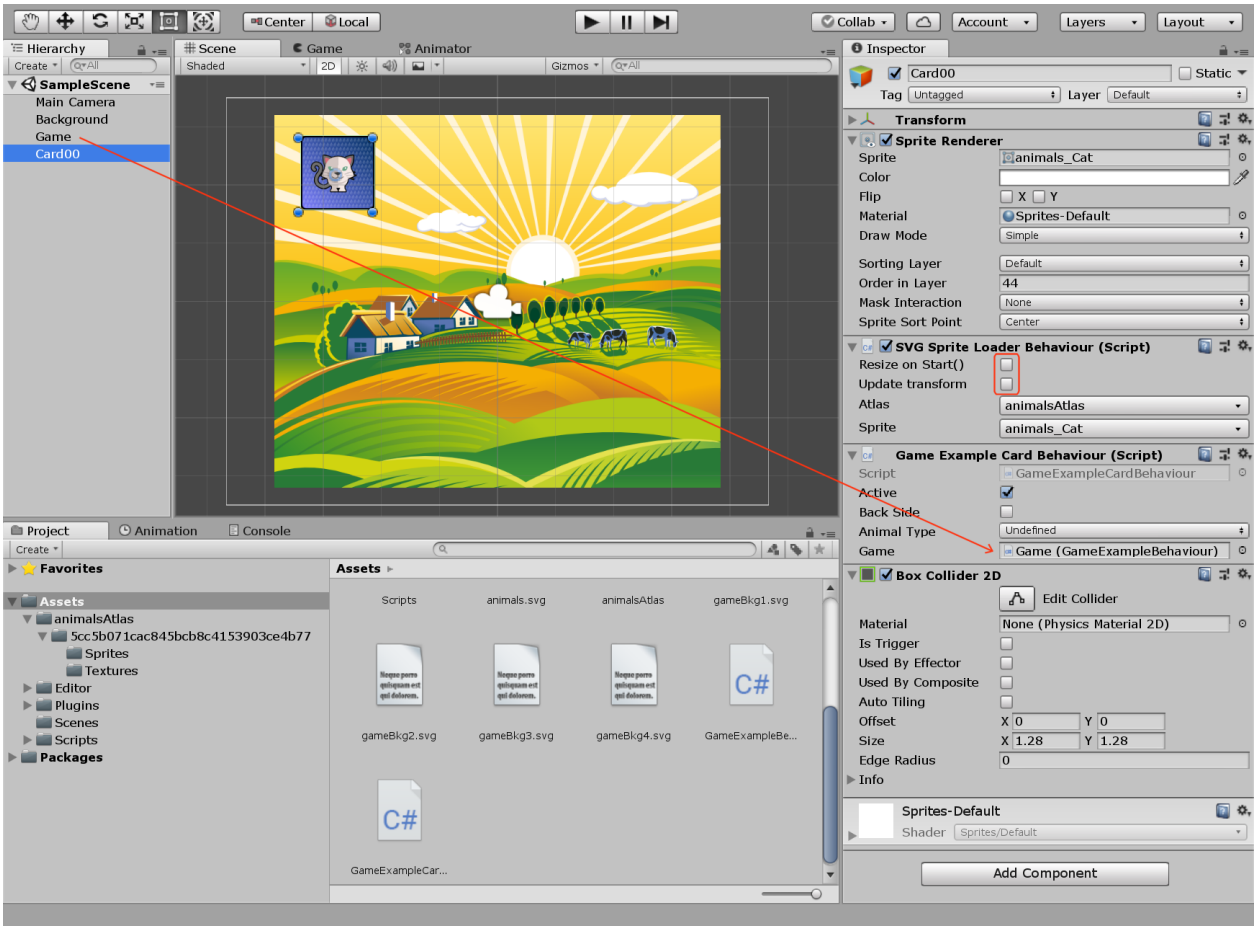
Click the Update button and you'll see the generated animals sprites. Note that at the `768 x 640 reference resolution`, each animal sprite will have a dimension of `128 x 128`: it will always guarantee that we can easily place them on a `4 x 3` grid (landscape layout) or on a `3 x 4` grid (portrait layout), regardless of screen resolution.



Animals sprites have been generated from animals.svg

Now we instantiate a single animal sprite, and we model a game card on it:

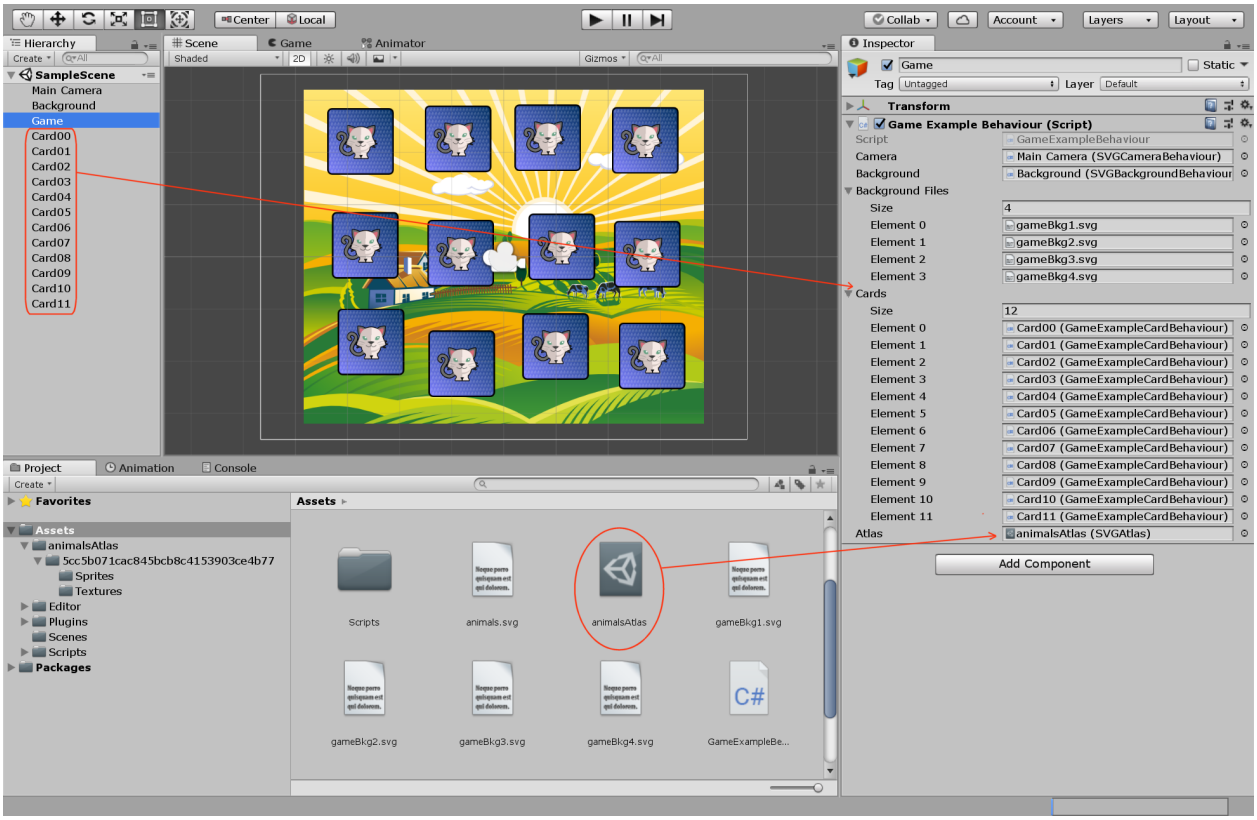
- choose an animal (for example the cat) and click on the relative `Instantiate` button
- rename the created gameobject to `Card00`
- uncheck both `Resize on Start()` and `Update transform` checkboxes (because we want to resize and position the sprite programmatically, by C# code at runtime)
- add a `GameExampleCardBehaviour` component (menu `Component` → `Add`, then `Scripts` subsection)
- drag&drop the `Game` gameobject to the `Game` property (so the card can access to its game)
- add a `Box Collider 2D` component (menu `Component` → `Physics 2D` → `Box Collider 2D`), because we want to intercept mouse/touch events



We have instantiated our first card!

The game will include a set of 12 cards, so all we have to do is to clone 11 times the newly created card gameobject (it doesn't matter where the clones are placed in the editor); in addition we will link the `GameExampleBehaviour` component (that is our main entry point) to the 12 cards by defining an internal array of `GameExampleCardBehaviour`:

```
// array of cards
public GameExampleCardBehaviour[] Cards;
// the atlas used to generate animals sprite
public SVGAtlas Atlas;
```



Now game has supervision on backgrounds SVG, card objects and animals sprites

The next step is to generate animals sprites at runtime, at each camera `OnResize` event; on the same occasion we have to rearrange the cards on the screen to take account of the new resolution. We split such two functionalities in different functions, within the `GameExampleBehaviour` component:

```
public Sprite UpdateCardSprite(GameExampleCardBehaviour card)
{
    GameExampleCardType cardType = card.BackSide ? GameExampleCardType.BackSide
                                                 : card.AnimalType;
    // get the sprite, given its name
    string name = GameExampleCardBehaviour.AnimalSpriteName(cardType);
    SVGRuntimeSprite data = Atlas.GetSpriteByName(name);

    // keep updated the SVGSpriteLoaderBehaviour component too
    SVGSpriteLoaderBehaviour loader =
        card.gameObject.GetComponent<SVGSpriteLoaderBehaviour>();

    card.gameObject.GetComponent<SpriteRenderer>().sprite = data.Sprite;
    loader.SpriteReference = data.SpriteReference;

    return data.Sprite;
}

private void UpdateCardsSprites()
{
    // assign the new sprites and update colliders
    for (int i = 0; i < Cards.Length; ++i)
    {
```

```
        Sprite sprite = UpdateCardSprite(Cards[i]);
        Cards[i].GetComponent<BoxCollider2D>().size = sprite.bounds.size;
    }
}

private void ResizeCards(int newScreenWidth,
                         int newScreenHeight)
{
    // update card sprites according to the current screen resolution
    if (Atlas.UpdateRuntimeSprites(newScreenWidth, newScreenHeight,
                                   out float scale))
    {
        // assign the new sprites and update colliders
        UpdateCardsSprites();
    }
}

private void PlaceCards(int screenWidth,
                        int screenHeight)
{
    int[] cardsIndexes;
    int slotsPerRow, slotsPerColumn;
    string name =
GameExampleCardBehaviour.AnimalSpriteName(GameExampleCardType.BackSide);
    SVGRuntimeSprite data = Atlas.GetSpriteByName(name);
    float cardWidth = data.Sprite.bounds.size.x;
    float cardHeight = data.Sprite.bounds.size.y;
    float worldWidth = Camera.WorldWidth;
    float worldHeight = Camera.WorldHeight;

    // check actual orientation on iOS and Android devices
    if ((Application.platform == RuntimePlatform.Android) ||
        (Application.platform == RuntimePlatform.IPhonePlayer))
    {
        // portrait orientation
        if (screenWidth <= screenHeight)
        {
            // number of card slots in each dimension
            slotsPerRow = 3;
            slotsPerColumn = 4;
            cardsIndexes = CARDS_INDEXES_NATIVE_PORTRAIT;
        }
        else
        {
            // get current (landscape) orientation
            ScreenOrientation orientation = SVGAssetsUnity.ScreenOrientation;
            // number of card slots in each dimension
            slotsPerRow = 4;
            slotsPerColumn = 3;
            invertCoord = true;
            cardsIndexes = (orientation == ScreenOrientation.LandscapeRight) ?
CARDS_INDEXES_LANDSCAPE_ROT90
                                                                             :
CARDS_INDEXES_LANDSCAPE_ROT270;
        }
```

```
    }
    else
    {
        // Desktop, detect orientation according to screen dimensions
        slotsPerRow = (screenWidth <= screenHeight) ? 3 : 4;
        slotsPerColumn = (screenWidth <= screenHeight) ? 4 : 3;
        cardsIndexes = (screenWidth <= screenHeight) ? CARDS_INDEXES_NATIVE_PORTRAIT
                                                     : CARDS_INDEXES_NATIVE_LANDSCAPE;
    }

    // 5% border
    float ofsX = worldWidth * 0.05f;
    float ofsY = worldHeight * 0.05f;
    float horizSeparator = ((worldWidth - (slotsPerRow * cardWidth) - (2.0f * ofsX))
                        / (slotsPerRow - 1));
    float vertSeparator = ((worldHeight - (slotsPerColumn * cardHeight) - (2.0f * ofsY))
                       / (slotsPerColumn - 1));
    int cardIdx = 0;

    for (int y = 0; y < slotsPerColumn; ++y) {
        for (int x = 0; x < slotsPerRow; ++x) {
            float posX = ofsX + (x * (cardWidth + horizSeparator))
                      - (worldWidth * 0.5f) + (cardWidth * 0.5f);
            float posY = ofsY + (y * (cardHeight + vertSeparator))
                      - (worldHeight * 0.5f) + (cardHeight * 0.5f);
            // invert coordinates, if needed
            Cards[cardsIndexes[cardIdx]].transform.position = invertCoord ? new Vector3(-
posX, -posY)
                                                                          : new
Vector3(posX, posY);
            cardIdx++;
        }
    }
}

private void OnResize(int newScreenWidth,
                      int newScreenHeight)
{
    // resize the background
    ResizeBackground(newScreenWidth, newScreenHeight);
    // resize animals sprites
    ResizeCards(newScreenWidth, newScreenHeight);
    // rearrange cards on the screen
    PlaceCards(newScreenWidth, newScreenHeight);
}

// cards arrangement when native device orientation is portrait
private static readonly int[] CARDS_INDEXES_NATIVE_PORTRAIT = {
    0,  1,  2,
    3,  4,  5,
    6,  7,  8,
    9, 10, 11
};
// landscape orientation, clockwise from the portrait orientation
private static readonly int[] CARDS_INDEXES_LANDSCAPE_ROT90 = {
```

```
    9, 6, 3, 0,
    10, 7, 4, 1,
    11, 8, 5, 2
};
// landscape orientation, counter-clockwise from the portrait orientation
private static readonly int[] CARDS_INDEXES_LANDSCAPE_ROT270 = {
    2, 5, 8, 11,
    1, 4, 7, 10,
    0, 3, 6,  9
};

// cards arrangement when native device orientation is landscape
private static readonly int[] CARDS_INDEXES_NATIVE_LANDSCAPE = {
    0,  1,  2, 3,
    4,  5,  6, 7,
    8,  9, 10, 11
};
```

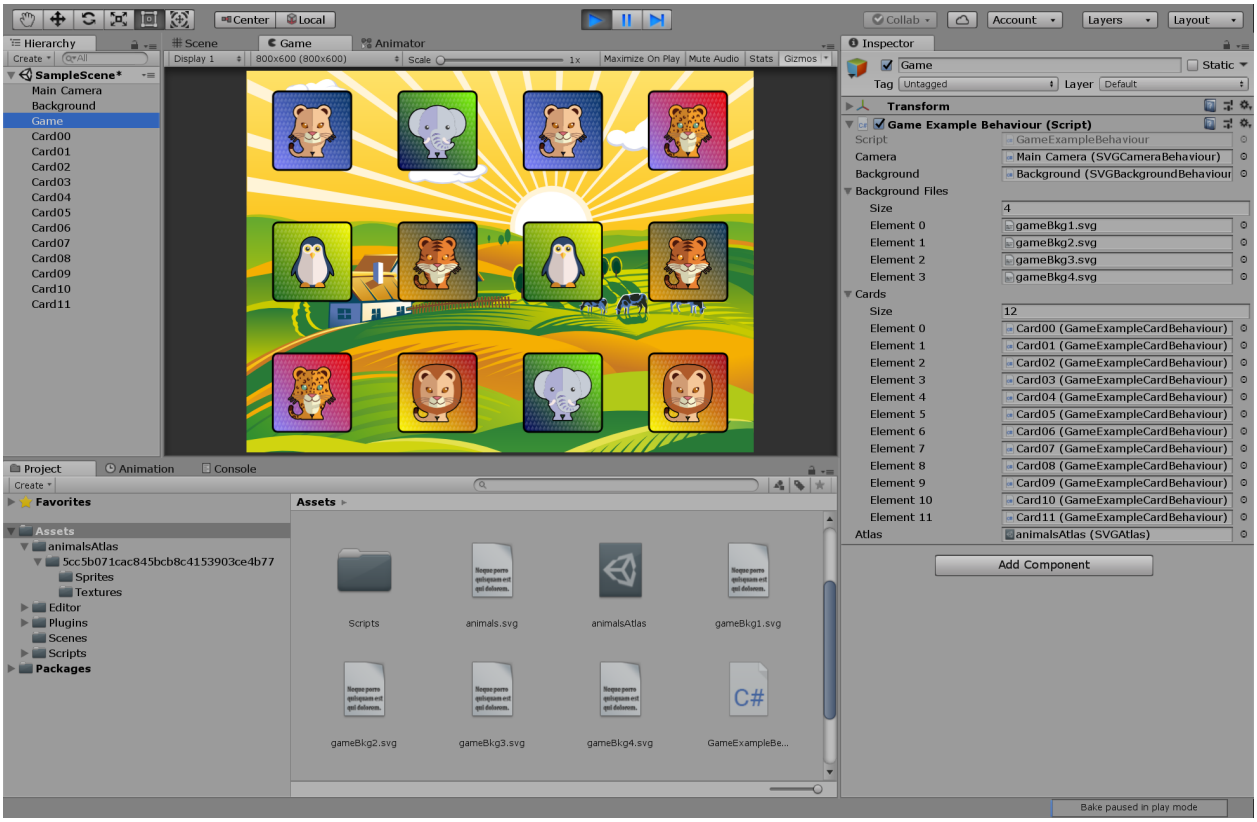
The code is really simple, here are some notes:

- when sprites are regenerated at runtime, due to a resolution change, the relative box colliders are update too (`UpdateCardsSprites` function)
- if the screen has a landscape layout, the cards are arranged on 3 rows and 4 columns; on portrait layouts, instead, they are arranged on 4 rows and 3 columns (`PlaceCards` function)

The `StartNewGame` function has been modified in order to generate six pairs of animals (12 cards in total), randomly chosen among the 25 available:

```
public void ShowCard(GameExampleCardBehaviour card)
{
    card.Active = true;
    // enable renderer
    card.gameObject.GetComponent<SpriteRenderer>().enabled = true;
    // enable collider
    card.GetComponent<BoxCollider2D>().enabled = true;
}

private void Shuffle(GameExampleCardType[] array)
{
    System.Random rnd = new System.Random(System.Environment.TickCount);
    int n = array.Length;
    // Knuth shuffle
    while (n > 1)
    {
        n--;
        int i = rnd.Next(n + 1);
        GameExampleCardType temp = array[i];
        array[i] = array[n];
        array[n] = temp;
    }
}

public void StartNewGame()
{
```

```
    GameExampleCardType[] animalCouples = new GameExampleCardType[Cards.Length];
    // start with a random animal
    GameExampleCardType currentAnimal = GameExampleCardBehaviour.RandomAnimal();

    // generate animal couples
    for (int i = 0; i < (Cards.Length / 2); ++i)
    {
        animalCouples[i * 2] = currentAnimal;
        animalCouples[(i * 2) + 1] = currentAnimal;
        currentAnimal = GameExampleCardBehaviour.NextAnimal(currentAnimal);
    }

    // shuffle couples
    Shuffle(animalCouples);

    // assign cards
    for (int i = 0; i < Cards.Length; ++i)
    {
        Cards[i].AnimalType = animalCouples[i];
        ShowCard(Cards[i]);
    }

    // destroy current background texture
    Background.DestroyAll(true);
    // assign a new SVG file
    int idx = (backgroundIndex % BackgroundFiles.Length);
    Background.SVGFile = BackgroundFiles[idx];
    // advance for the next background SVG
    backgroundIndex++;
}
```

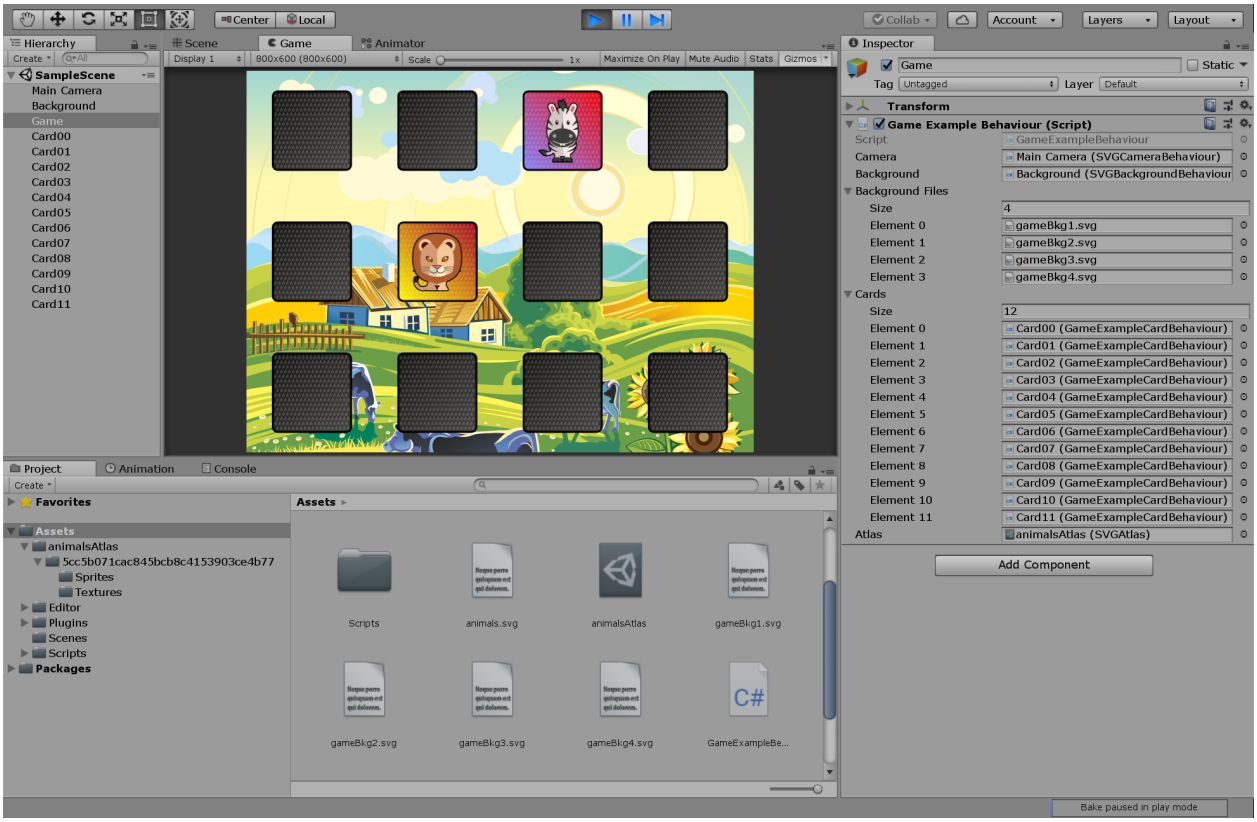


All the graphics are perfectly sized and respect the screen layout

The last detail that is missing is how we detect if a card is selected by mouse/touch actions. This is really simple, because we have already configured box colliders on cards: we just need to write the `OnMouseDown` event handler within the `GameExampleCardBehaviour` script.

```
void OnMouseDown()
{
    // forward the selection event to the Game
    Game.SelectCard(this);
}
```

The remaining code (not reported here because really trivial) simply deals with the gameplay: all the cards must start as backside and if the two selected cards match, they are made inactive (`GameExampleCardBehaviour.Active = false`) and removed from the game, otherwise they are covered again. The game ends when all the cards are inactive (i.e. all pairs of animals have been discovered).



A new game and we immediately chose a pair that does not match!

The "You win!" message

When the player has guessed all pairs, a "You win!"-like message will be displayed. This will be implemented using a simple SVG that uses a rotated `<text>` element:

```
<svg version="1.1" width="512" height="512" viewBox="0 0 512 512">
    <!-- 45 degree rotation around the center, then a
         translation to vertically center the baseline -->
    <g transform="rotate(-45 256 256) translate(0 42)"
       text-anchor="middle"
       font-family="Acme" font-size="120"
       fill="white" stroke="black" stroke-width="4">
        <!-- make the text centered at (256, 256) -->
        <switch>
            <!-- German language -->
            <text x="256" y="256" systemLanguage="de">Du gewinnst!</text>
            <!-- English language -->
            <text x="256" y="256" systemLanguage="en">You win!</text>
            <!-- Spanish language -->
            <text x="256" y="256" systemLanguage="es">Tú ganas!</text>
            <!-- French language -->
            <text x="256" y="256" systemLanguage="fr">Vous gagnez!</text>
            <!-- Italian language -->
            <text x="256" y="256" systemLanguage="it">Hai vinto!</text>
            <!-- the fallback element with no systemLanguage attribute
                 if none of them match -->
            <text x="256" y="256">You win!</text>
```

```
        </switch>
    </g>
</svg>
```

The SVG used to display a congratulation message to the player

In order to display the message, an object must be added to the scene (menu `GameObject` → `Create Empty`), its default location `(0, 0, 0)` will be just fine, because it corresponds to the center of the camera frame. Then a dedicated `GameCongratsBehaviour` script must be attached to it (menu `Component` → `Add`, then `Scripts` subsection): as soon as the script is attached to the object, it checks the presence of a `SpriteRenderer` component and creates it if not present (the default `Order in Layer` property value will be `0`, that is just fine because it means that the congratulation object/sprite will be displayed upon the background).
The script takes the congratulation SVG file as input, loads it when needed and destroys it at the `OnDestroy` event. In order to generate the texture according to screen resolution, a public `Resize` method has been exposed.

```
void OnDestroy()
{
    if (document != null)
    {
        // release SVG document
```

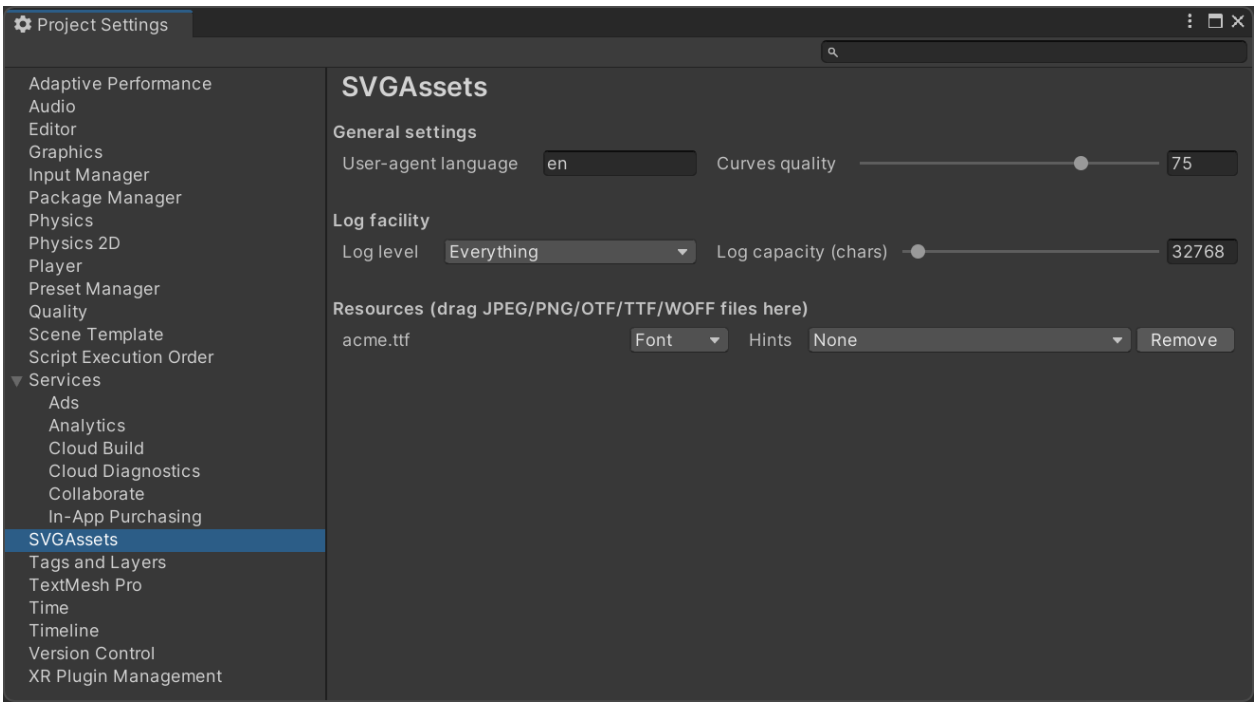
```
        document.Dispose();
    }
}

public void Resize(int newScreenWidth,
                   int newScreenHeight)
{
    // load the SVG document, if needed
    if ((document == null) && (SVGFile != null))
    {
        document = SVGAssetsUnity.CreateDocument(SVGFile.text);
    }

    if (document != null)
    {
        // congratulation SVG is squared by design, we choose to generate a texture with
        // a size equal to 3/5 of the smallest screen dimension; e.g. on a 1920 x 1080
        // device screen, texture will have a size of (1080 * 3) / 5 = 648 pixels
        uint size = (uint)(Math.Min(newScreenWidth, newScreenHeight) * 3) / 5;
        // create a drawing surface
        SVGSurfaceUnity surface = SVGAssetsUnity.CreateSurface(size, size);
        // draw SVG and generate the relative texture
        Texture2D texture = surface.DrawTexture(document, SVGColor.Clear);
        // create the sprite out of the texture
        SpriteRenderer renderer = gameObject.GetComponent<SpriteRenderer>();
        renderer.sprite = SVGAssetsUnity.CreateSprite(texture, new Vector2(0.5f, 0.5f));
        // destroy the temporary drawing surface
        surface.Dispose();
    }
}

// SVG to be displayed when the player wins.
public TextAsset SVGFile = null;
// The loaded SVG document.
private SVGDocument document = null;
```

The project settings

As seen in the previous chapter, the SVG file used to generate the "You win!" texture message is made of `<text>` elements; all such elements inherit the use of a Acme font family. In addition a `<switch>` element is used to provide an ability to specify alternate viewing depending on the capabilities of a given user agent or the user's language. In the detail not all `<text>` elements will be rendered, but only the one that matches the user's language.
In order to provide SVGAssets fonts and language settings, SVGAssetsConfigUnity class must be used. Such class is interfaced to Unity editor through the `SVGAssetsConfigUnityScriptable` scriptable object (see full implementation in `Assets/SVGAssets/Scripts/SVGAssets/SVGAssetsConfigUnityScriptable.cs`): it simply ensures that configuration parameters are saved/synchronized to an external resource file (`SVGAssetsConfigUnity.asset`, located in `Assets/SVGAssets/Resources` directory).

By opening the `Edit` → `Project Settings` menu, then `SVGAssets` subsection, it is possible to edit the basic parameters (e.g. user-agent language, curves quality) and provide resources (OTF/TTF/WOFF fonts and JPEG/PNG images) to SVGAssets. In this game example we must drag&drop the Acme font (from `Assets/SVGAssets/Resources` directory), since it is needed by the congratulation message SVG. In this

way, when SVGAssets will be initialized at runtime, it will find and use the font file to render `<text>` elements.



Editor panel for SVGAssets resources and configuration parameters

The Acme font

The splash screen

The implementation of an initial splash screen, showing a "Powered by AmanithSVG" logo, is really simple. It is just a matter to load the relative SVG document, and generate a texture. Because the logo must cover the whole screen, while maintaining its original aspect ratio (i.e. the aspect ratio induced by the `viewBox` attribute), we must calculate the smallest scale factor that "fits" the logo within the screen.
The code has been incapsulated within a `GameSplashScreenBehaviour` class, which is very similar to the `GameCongratsBehaviour` script, the only thing that changes is the method of calculating the texture size:

```
void OnDestroy()
{
    if (_document != null)
    {
        // release SVG document
        document.Dispose();
    }
}

public void Resize(int newScreenWidth,
                   int newScreenHeight)
{
```

```
    // load the SVG document, if needed
    if ((_document == null) && (SVGFile != null))
    {
        _document = SVGAssetsUnity.CreateDocument(SVGFile.text);
    }

    if (_document != null)
    {
        // get 'viewBox' dimensions
        float viewW = _document.Viewport.Width;
        float viewH = _document.Viewport.Height;
        // keep the SVG aspect ratio
        float sx = newScreenWidth / viewW;
        float sy = newScreenHeight / viewH;
        // keep 2% border on each side
        float scaleMin = Math.Min(sx, sy) * 0.96f;
        // and at the same time we fit the screen
        uint texW = (uint)Math.Round(viewW * scaleMin);
        uint texH = (uint)Math.Round(viewH * scaleMin);

        // create a drawing surface
        SVGSurfaceUnity surface = SVGAssetsUnity.CreateSurface(texW, texH);
        // draw SVG and generate the relative texture
        Texture2D texture = surface.DrawTexture(document, SVGColor.Clear);
        // create the sprite out of the texture
        SpriteRenderer renderer = gameObject.GetComponent<SpriteRenderer>();
        renderer.sprite = SVGAssetsUnity.CreateSprite(texture, new Vector2(0.5f, 0.5f));
        // destroy the temporary drawing surface
        surface.Dispose();
    }
}

// The "Powered by AmanithSVG" logo.
public TextAsset SVGFile = null;
// The loaded SVG document.
private SVGDocument document = null;
```

Powered by AmanithSVG logo (for dark backgrounds)

Powered by AmanithSVG logo (for light backgrounds)

As it can be seen, card objects have also an Animation component attached, used to perform a simple rotation when shuffling or when two selected cards match.

Now you can have fun experimenting with it!

Credits

- Sunny countryside backgrounds (gameBkg1.svg, gameBkg2.svg, gameBkg3.svg, gameBkg4.svg) have been downloaded from: http://xoo.me/it/template/details/6960-4-sunny-countryside-vector-scenes
- Thanks to Juan Pablo del Peral (juan@huertatipografica.com.ar) for the Acme font, that is covered by the SIL Open Font License, Version 1.1