G-Raff Engine Documentation Document

Short version

By Marius Utheim

# Creating a new game

To run the game, call Giraffe.Run() with appropriate parameters. The project type should be a Windows Application, or a Console Application if you want to enable the console.

# Defining game Elements

If your class represents something that has a clear location in the room (such as player, enemies, trees, bullets etc.), then it should inherit GameObject. Otherwise, if it does not have a position in the room (such as GUI, Alarms, Tweening functions etc.) but still should draw something or do something each step, then it should inherit GameElement.

To make your class do something each step, override GameElement.OnStep().

To make your class draw something, override GameElement.OnDraw().

# Instantiating game elements

Calling the constructor of your class automatically places the instance in the room. For example, to create an instance of MyGameObject, just call new MyGameObject();

You don’t have to store the result in a variable. It will not be garbage collected until you have called GameElement.Destroy() on that instance.

# Reacting to keyboard presses

To do something each step while the player holds the key, implement IKeyListener.

To do something once when the player presses the key, implement IKeyPressListener.

To do something when the player releases the key, implement IKeyReleaseListener.

Use the static class Keyboard to check keyboard state outside of the aforementioned events, for example to check state during the Step event, or to see if multiple keys were held simultaneously.

# Reacting to mouse button clicks

To do something each step while holding the button, implement IGlobalMouseListener.

To do something when the player presses the button, implement IGlobalMousePressListener.

To do something when the player releases the button, implement IGlobalMouseReleaseListener.

If you want the instance to only perform the event if the cursor is hovering over the instance when it occurred, instead implement IMouseListener, IMousePressListener or IMouseReleaseListener. This only works if the instance has a collision mask (see below).

Use the static class Mouse to check keyboard state outside of the aforementioned events.

# Defining the collision mask of an object

There are two standard ways of defining the mask of an instance. By setting the Sprite of an instance, the Mask shape is by default set to MaskShape.SameAsSprite to indicate it will use same MaskShape that was defined by that Sprite. Optionally, one can create a custom MaskShape by calling a static method of the MaskShape class.

If you want the instance to have no collision mask, set the mask shape to MaskShape.None. Never set the mask shape to null.

The actual position of the mask in the room is based on the Transform of its owner.

# reacting to collisions between object

To make the instance react to collisions with other instances of the generic class TObject, implement ICollisionListener<TObject>.

# Loading a sprite

Instantiate a new Sprite to associate with the image file. This can be done at the beginning of the game and will not load the file into memory.

To load the file into memory, use Sprite.Load(), or set the preload parameter to true when instantiating Sprite to load automatically. This is a little slow, so don’t do it while there is a lot of action going on. Call Sprite.Unload() to unload it.

# drawing a sprite

Instances that inherit from GameObject automatically draw their Sprite in GameObject.OnDraw(), using the Transform of that instance.

When overriding GameObject.OnDraw(), call base.OnDraw() only if you want to draw the Sprite at the location of that instance.

Use Draw.Sprite() to draw sprites manually.

# Drawing Geometric shapes

Use static methods from Draw or Fill. For example, to draw a circle, call Draw.Circle(). If you want it to be filled, instead draw Fill.Circle().

# Loading a sound

Instantiate a new Sound to associate with the sound file. This can be done at the beginning of the game and will not load the file into memory immediately.

To load the sound into memory, use Sound.Load(), or set the preload argument to true in the constructor. This can be slow, especially for large sound files such as background music.

**NB!** Currently, only OGG files are supported, and the sound format must be 16-bit mono channel! Support for other formats will be added soon.

# Playing soundS

Call Sound.Play(). This returns a SoundInstance that can be used to control that particular instance, for example stopping it.

To make a sound loop automatically, set the loop argument to true.

To stop the sound, call SoundInstance.Stop(), or Sound.StopAll() to stop all instances of that sound.