G-Raff Engine Documentation Document

Short version

By Marius Utheim

# Creating a new Project

Create a new project as a Windows Application, or a Console Application if you want to enable the console. After adding a reference to GRaff.dll, run the game by calling Giraffe.Run() with appropriate parameters.

# 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

After instantiating an object that inherits from GameElement, call Instance.Create(GameElement) to register that instance in the instance list. This will call actions like OnStep automatically.

To remove an element from the instance list, call 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 react to mouse clicks, implement IGlobalMouseListener, IGlobalMousePressListener or 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

By default, the Mask shape is set to MaskShape.Automatic which indicates the object will use same MaskShape that was defined by the Sprite of that object. If the object has no sprite, it doesn’t have a collision mask either.

It is possible to create a custom MaskShape by calling a static method of the MaskShape class. This can be useful if your instance is visible, but has no sprite, e.g. if it draws vector graphics or has a particle effect.

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

Note that 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

When your game begins, you should instantiate a new Sprite instance for each image you want to use. This will just create a reference to that resource, it will not load the file to memory.

To load the file, call Sprite.Load() on the appropriate instance. It is also possible to call the static version of Sprite.Load() to create a new Sprite instance that is initially loaded. Keep in mind that loading sprites can be slow. Call Sprite.Unload() to unload sprites.

Sprites can be loaded in the background by calling Sprite.LoadAsync().

# 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 vector graphics

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

When your game begins, you should instantiate a new Sound instance for each image you want to use. This will just create a reference to that resource, it will not load the file to memory.

To load the file, call Sound.Load() on the appropriate instance. It is also possible to call the static version of Sound.Load() to create a new Sound instance that is initially loaded. Keep in mind that loading sprites can be slow. Call Sound.Unload() to unload sounds.

Sprites can be loaded in the background by calling Sound.LoadAsync().

# 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.