ggame Documentation

Release 0.1.0

Eric Dennison

Aug 07, 2019

Contents

The simple sprite and game platform for Brython Server and Runpython.

Ggame stands for a couple of things: "good game" (of course!) and also "git game" or "github game" because it is designed to operate with Runpython in concert with Github as its backend file store.

CHAPTER 1

User's Guide

This part of the documentation, which is mostly prose, begins with some background information about ggame, then presents several tutorials to illustrate typical uses.

1.1 Introduction

Ggame is **not** intended to be a full-featured gaming API, with every bell and whistle. It is designed primarily as a tool for teaching computer programming, recognizing that the ability to create engaging and interactive games is a powerful motivator for many progamming students. Accordingly, any functional or performance enhancements that *can* be reasonably implemented by the user are left as an exercise.

1.1.1 Functionality Goals

The ggame library is intended to be trivially easy to use. For example:

```
"""
Trivial example of a minimal ggame App with Sprite.
"""
from ggame import App, ImageAsset, Sprite

# Create a displayed object at 100,100 using an image asset
Sprite(ImageAsset("bunny.png"), (100, 100))
# Create the app, with a default stage
APP = App()
# Run the app
APP.run()
```

1.1.2 Extensions

Ggame is being extended for geometry exploration in a manner reminiscent of Geogebra, digital logic simulation, and with tools and classes to use with rocket and orbital simulations.

1.1.3 Overview

There are three major pieces in a ggame app: assets, sprites and the app itself.

Assets

Asset objects (i.e. `ImageAsset`, etc.) typically represent separate files that are provided by the "art department". These might be background images, user interface images, or images that represent objects in the game. In addition, `SoundAsset` are used to represent sound files (.wav or .mp3 format) that can be played in the game.

Ggame also extends the asset concept to include graphics that are generated dynamically at run-time, such as geometrical objects, e.g. rectangles, lines, etc.

Sprites

All of the visual aspects of the game are represented by instances of `Sprite` or subclasses of it.

App

Every ggame application must create a single instance of the `App` class (or a sub-class of it). Create an instance of the `App` class to draw a graphics canvas in your browser window. Execute the app's `run()` method to start refreshing and redrawing the visual assets on the screen.

Events

No game is complete without a player and players make events. Your code handles user input by registering to receive keyboard and mouse events using `listenKeyEvent()` and `listenMouseEvent()` methods of the `App` class.

1.1.4 Execution Environment

Ggame is designed to execute in a web browser using Brython, Pixi.js and Buzz. The easiest way to do this is by executing from runpython, with your source code stored at github. When you use ggame from within runpython, the Github ggame repository is automatically placed on the import search path.

1.1.5 Geometry

When referring to screen coordinates, note that the x-axis of the computer screen is *horizontal* with the zero position on the left hand side of the screen. The y-axis is *vertical* with the zero position at the **top** of the screen.

Increasing positive y-coordinates correspond to the downward direction on the computer screen. Note that this is **different** from the way you may have learned about x and y coordinates in math class!

CHAPTER 2

API Reference

2.1 ggame Core API

2.1.1 ggame Application Classes

App

class ggame.app.**App**(**args*)

The *App* class is a (typically subclassed) class that encapsulates handling of the display system, and processing user events. The *App* class also manages lists of all `Sprite` instances in the application.

When subclassing *App* you may elect to instantiate most of your sprite objects in the initialization section.

Processing that must occur on a per-frame basis may be included by overriding the *run()* method. This is also an appropriate location to call similar 'step' methods for your various customized sprite classes.

Once your application class has been instantiated, begin the frame drawing process by calling its *run()* method.

The *App* class is instantiated either by specifying the desired app window size in pixels, as two parameters::

```
myapp = App(640,480)
```

or by providing no size parameters at all::

```
myapp = App()
```

in which case, the full browser window size is used.

NOTE: Only **one** instance of an *App* class or subclass may be instantiated at a time.

spritelist = []

List of all sprites currently active in the application.

classmethod getSpritesbyClass(*sclass*)

Returns a list of all active sprites of a given class.

Parameters **sclass** (*class*) – A class name (e.g. 'Sprite') or subclass.

Returns A (potentially empty) list of sprite references.

classmethod listenKeyEvent (*eventtype*, *key*, *callback*)
Register to receive keyboard events.

Parameters

- **eventtype** (*str*) – The type of key event to receive (value is one of: *'keydown'*, *'keyup'* or *'keypress'*).
- **key** (*str*) – Identify the keyboard key (e.g. *'space'*, *'left arrow'*, etc.) to receive events for.
- **callback** (*function*) – The function or method that will be called with the *KeyEvent* object when the event occurs.

Returns Nothing

See the source for *KeyEvent* for a list of key names to use with the *key* paramter.

classmethod listenMouseEvent (*eventtype*, *callback*)
Register to receive mouse events.

Parameters

- **eventtype** (*str*) – The type of mouse event to receive (value is one of: *'mousemove'*, *'mousedown'*, *'mouseup'*, *'click'*, *'dblclick'* or *'mousewheel'*).
- **callback** (*function*) – The function or method that will be called with the *ggame.event.MouseEvent* object when the event occurs.

Returns Nothing

classmethod unlistenKeyEvent (*eventtype*, *key*, *callback*)
Use this method to remove a registration to receive a particular keyboard event. Arguments must exactly match those used when registering for the event.

Parameters

- **eventtype** (*str*) – The type of key event to stop receiving (value is one of: *'keydown'*, *'keyup'* or *'keypress'*).
- **key** (*str*) – The keyboard key (e.g. *'space'*, *'left arrow'*, etc.) to stop receiving events for.
- **callback** (*function*) – The function or method that will no longer be called with the *KeyEvent* object when the event occurs.

Returns Nothing

See the source for *KeyEvent* for a list of key names to use with the *key* paramter.

classmethod unlistenMouseEvent (*eventtype*, *callback*)
Use this method to remove a registration to receive a particular mouse event. Arguments must exactly match those used when registering for the event.

Parameters

- **eventtype** (*str*) – The type of mouse event to stop receiving events for (value is one of: *'mousemove'*, *'mousedown'*, *'mouseup'*, *'click'*, *'dblclick'* or *'mousewheel'*).
- **callback** (*function*) – The function or method that will no longer be called with the *ggame.event.MouseEvent* object when the event occurs.

Returns Nothing

run(*userfunc=None*)

Calling the *run()* method begins the animation process whereby the *step()* method is called once per animation frame.

Parameters **userfunc** (*function*) – Any function or method which shall be called once per animation frame.

Returns Nothing

step()

The *step()* method is called once per animation frame. Override this method in your own subclass of *App* to perform periodic calculations, such as checking for sprite collisions, or calling 'step' functions in your own customized sprite classes.

The base class *step()* method is empty and is intended to be overriden.

Returns Nothing

Events

ggame events are objects that are created by the ggame system as a result of some user action (mouse, keyboard). If the ggame application has called *listenKeyEvent()* or *listenMouseEvent()* then an appropriate Event object is instantiated by ggame and returned to a user-provided handler function. The ggame user code should examine the attributes of the Event object to find out more information about the event that occurred.

class ggame.event.**MouseEvent**(*app*, *hwevent*)

A MouseEvent object encapsulates information about a user mouse action that is being reported by the system. This class is not instantiated by the ggame user.

wheeldelta = None

Integer representing up/down motion of the scroll wheel.

x = None

The window x-coordinate of the mouse pointer when the event occurred.

y = None

The window y-coordinate of the mouse pointer when the event occurred.

route(*evtlist*)

Execute all callbacks configured for this event.

class ggame.event.**KeyEvent**(*hwevent*)

A KeyEvent object encapsulates information regarding a user keyboard action that is being reported by the system. This class is not instantiated by the ggame user.

route(*evtlist*)

Execute all callbacks configured for this event.

keynum = None

The *keynum* attribute identifies a keycode (number).

key = None

The *key* attribute identifes the key in text form (e.g. 'back slash').

The list of key numbers and description strings follows:

```
8: 'backspace',
9: 'tab',
13: 'enter',
```

(continues on next page)

(continued from previous page)

```
16: 'shift',
17: 'ctrl',
18: 'alt',
19: 'pause/break',
20: 'caps lock',
27: 'escape',
32: 'space',
33: 'page up',
34: 'page down',
35: 'end',
36: 'home',
37: 'left arrow',
38: 'up arrow',
39: 'right arrow',
40: 'down arrow',
45: 'insert',
46: 'delete',
48: '0',
49: '1',
50: '2',
51: '3',
52: '4',
53: '5',
54: '6',
55: '7',
56: '8',
57: '9',
65: 'a',
66: 'b',
67: 'c',
68: 'd',
69: 'e',
70: 'f',
71: 'g',
72: 'h',
73: 'i',
74: 'j',
75: 'k',
76: 'l',
77: 'm',
78: 'n',
79: 'o',
80: 'p',
81: 'q',
82: 'r',
83: 's',
84: 't',
85: 'u',
86: 'v',
87: 'w',
88: 'x',
89: 'y',
90: 'z',
91: 'left window key',
92: 'right window key',
93: 'select key',
96: 'numpad 0',
```

(continues on next page)

(continued from previous page)

```
97: 'numpad 1',
98: 'numpad 2',
99: 'numpad 3',
100: 'numpad 4',
101: 'numpad 5',
102: 'numpad 6',
103: 'numpad 7',
104: 'numpad 8',
105: 'numpad 9',
106: 'multiply',
107: 'add',
109: 'subtract',
110: 'decimal point',
111: 'divide',
112: 'f1',
113: 'f2',
114: 'f3',
115: 'f4',
116: 'f5',
117: 'f6',
118: 'f7',
119: 'f8',
120: 'f9',
121: 'f10',
122: 'f11',
123: 'f12',
144: 'num lock',
145: 'scroll lock',
186: 'semicolon',
187: 'equal sign',
188: 'comma',
189: 'dash',
190: 'period',
191: 'forward slash',
192: 'grave accent',
219: 'open bracket',
220: 'back slash',
221: 'close bracket',
222: 'single quote'
```

2.1.2 ggame Assets

ggame assets and related objects (`Color` and `LineStyle`) are classes that encapsulate and represent displayable images. A single asset may be used in multiple sprites. Animated assets may be created from any image that includes multiple images within it (i.e. a sprite sheet).

Frame

class ggame.asset.**Frame**(*x*, *y*, *w*, *h*)

Frame is a utility class for expressing the idea of a rectangular region.

Initializing parameters describe the position of the upper-left corner of the frame, and the frame's width and height. The units are *typically* in pixels, though a frame can be generic.

Parameters

- **x** (*int*) – X-coordinate of frame upper-left corner
- **y** (*int*) – Y-coordinate of frame upper-left corner
- **w** (*int*) – Width of the frame
- **h** (*int*) – Height of the frame

x

X-coordinate of the upper left hand corner of this frame.

y

Y-coordinate of the upper left hand corner of this frame.

w

Width of the frame.

h

Height of the frame.

center

The *center* property computes a coordinate pair (tuple) for the center of the frame.

The *center* property, when set, redefines the *x* and *y* properties of the frame in order to make the center agree with the coordinates (tuple) assigned to it.

Color

class ggame.asset.**Color**(*color*, *alpha*)

The Color class is used to represent colors and/or colors with transparency.

Parameters

- **color** (*int*) – an integer in the conventional format (usually as a hexadecimal literal, e.g. 0xffbb33 that represents the three color components, red, green and blue)
- **alpha** (*float*) – a transparency value, or *alpha* as a floating-point number in the range of 0.0 to 1.0 where 0.0 represents completely transparent and 1.0 represents completely opaque.

Example:

```python
"""
Example of using Color class.
"""
from ggame import Color

RED = Color(0xFF0000, 1.0)
```

LineStyle

class ggame.asset.**LineStyle**(*width*, *color*)

The LineStyle class is used to represent line style when drawing geometrical objects such as rectangles, ellipses, etc.

Parameters

- **width** (*int*) – the *width* of the line in pixels
- **color** (Color) – the *color* as a valid Color instance.

Example:

```
"""
Example of using LineStyle class.
"""
from ggame import LineStyle, Color

LINE = LineStyle(3, Color(0x00FF00, 1.0))
```

This defines a 3-pixel wide green line.

Predefined Colors and Lines

`ggame.asset.`**`BLACK = BLACK`**
Default black color

`ggame.asset.`**`WHITE = WHITE`**
Default white color

`ggame.asset.`**`BLACKLINE = LineStyle(1, BLACK)`**
Default thin black line

`ggame.asset.`**`WHITELINE = LineStyle(1, WHITE)`**
Default thin white line

ImageAsset

`class` `ggame.asset.`**`ImageAsset`**(*url*, *frame=None*, *qty=1*, *direction='horizontal'*, *margin=0*)
The ImageAsset class connects ggame to a specific image **file**.

Parameters

- **`url`** (*`str`*) – All ImageAsset instances must specify a file name or url where a jpg or png image is located.
- **`frame`** (`Frame`) – If the desired sprite image exists in only a smaller sub-section of the original image, then specify an area *within* the image using *frame* parameter, which must be a valid *`Frame`* instance.
- **`qty=0`** (*`int`*) – If the image file actually is a *collection* of images, such as a so-called *sprite sheet*, then the ImageAsset class supports defining a list of images, provided they exist in the original image as a **row** of evenly spaced images or a **column** of images. To specify this, provide the *qty* (quantity) of images in the row or column.
- **`direction='horizontal'`** (*`str`*) – For an image *sprite sheet*, specify whether the images are oriented in a *'vertical'* or *'horizontal'* arrangement.
- **`margin=0`** (*`int`*) – If there is a gap between successive images in an image *sprite sheet*, then specify the size of the gap (in pixels). When used in this way, the *frame* parameter must define the area of only the **first** image in the collection; all subsequent images in the list are assumed to be the same size, but separated by the *margin* value.

Example:

```
"""
Example of using ImageAsset class.
"""
from ggame import ImageAsset
```

(continues on next page)

(continued from previous page)

```
IMG = ImageAsset("bunny.png")
```

url = None
A string that represents the path or url of the original file.

append(*url*, *frame=None*, *qty=1*, *direction='horizontal'*, *margin=0*)
Append a texture asset from a new image file (or url). This method allows you to build a collection of images into an asset (such as you might need for an animated sprite), but without using a single sprite sheet image.

The parameters for the *append* method are identical to those supplied to the *ImageAsset* initialization method.

This method allows you to build up an asset that consists of multiple rows or columns of images in a sprite sheet or sheets.

destroy()
Destroy or deallocate any underlying graphics resources used by the asset. Call this method on any asset that is no longer being used.

gfx
gfx property represents the underlying system object used to represent this asset. If this asset is composed of multiple assets, then the **first** asset is referenced by *gfx*.

RectangleAsset

class ggame.asset.**RectangleAsset**(*width*, *height*, *line=LineStyle(1, BLACK)*, *fill=BLACK*)
The RectangleAsset is a "virtual" asset that is created on the fly without requiring creation of an image file.

Parameters

- **width** (*int*) – Rectangle width, in pixels
- **height** (*int*) – Rectangle height, in pixels
- **line=BLACKLINE** (LineStyle) – The color and width of the rectangle border
- **fill=BLACK** (Color) – The color of the rectangle body

gfx
The *gfx* property represents the underlying system object.

destroy()
Destroy or deallocate any underlying graphics resources used by the asset. Call this method on any asset that is no longer being used.

CircleAsset

class ggame.asset.**CircleAsset**(*radius*, *line=LineStyle(1, BLACK)*, *fill=BLACK*)
The *ggame.CircleAsset* is a "virtual" asset that is created on the fly without requiring creation of an image file.

Parameters

- **radius** (*int*) – Circle radius, in pixels
- **line=BLACKLINE** (LineStyle) – The color and width of the circle border
- **fill=BLACK** (Color) – The color of the circle body

gfx
The *gfx* property represents the underlying system object.

destroy()
Destroy or deallocate any underlying graphics resources used by the asset. Call this method on any asset that is no longer being used.

EllipseAsset

class ggame.asset.**EllipseAsset**(*halfw*, *halfh*, *line=LineStyle(1*, *BLACK)*, *fill=BLACK*)
The *ggame.EllipseAsset* is a "virtual" asset that is created on the fly without requiring creation of an image file.

Parameters

- **halfw** (*int*) – Ellipse semi-axis dimension in the horizontal direction (half the width of the ellipse), in pixels
- **halfh** (*int*) – Ellipse semi-axis dimension in the vertical direction (half the height of the ellipse), in pixels
- **line=BLACKLINE** (LineStyle) – The color and width of the ellipse border
- **fill=BLACK** (Color) – The color of the ellipse body

destroy()
Destroy or deallocate any underlying graphics resources used by the asset. Call this method on any asset that is no longer being used.

gfx
The *gfx* property represents the underlying system object.

PolygonAsset

class ggame.asset.**PolygonAsset**(*path*, *line=LineStyle(1*, *BLACK)*, *fill=BLACK*)
The PolygonAsset is a "virtual" asset that is created on the fly without requiring creation of an image file.

Note: you should not specificy absolute screen coordinates for this asset, since you will use the `Sprite` position to locate your polygon on the screen.

Parameters

- **path** (*list*) – A list of pixel-coordinate tuples. These coordinates should not be in absolute screen coordinates, but should be relative to the desired 'center' of the resulting `Sprite`. The final coordinate pair in the list must be the same as the first.
- **line=BLACKLINE** (LineStyle) – The color and width of the ellipse border
- **fill=BLACK** (Color) – The color of the ellipse body

Example:

```
"""
Example of using the PolygonAsset class.
"""

from ggame import PolygonAsset, LineStyle, Color, BLACK

POLY = PolygonAsset(
```

(continues on next page)

(continued from previous page)

```
    [(0, 0), (50, 50), (50, 100), (0, 0)], LineStyle(4, BLACK), Color(0x80FF00, 0.
↪8)
)
```

destroy()
Destroy or deallocate any underlying graphics resources used by the asset. Call this method on any asset that is no longer being used.

gfx
The *gfx* property represents the underlying system object.

LineAsset

class ggame.asset.**LineAsset**(*x*, *y*, *line=LineStyle(1, BLACK)*)
The LineAsset is a "virtual" asset that is created on the fly without requiring creation of an image file. A LineAsset instance represents a single line segment.

Note: you should not specificy absolute screen coordinates for this asset, since you will use the `Sprite` position to locate your line on the screen. The line segment will begin at pixel coordinates (0,0), and will end at the (x,y) coordinates given below.

As the LineAsset does not cover a region, only a *LineStyle* argument must be supplied (*line*) to specify the color.

Parameters

- **x** (*int*) – x-coordinate of the line endpoint, in pixel units
- **y** (*int*) – y-coordinate of the line endpoint, in pixel units
- **line=BLACKLINE** (`LineStyle`) – The color and width of the ellipse border

destroy()
Destroy or deallocate any underlying graphics resources used by the asset. Call this method on any asset that is no longer being used.

delta_x = None
This attribute represents the *x* parameter supplied during instantiation.

delta_y = None
This attribute represents the *y* parameter supplied during instantiation.

gfx
The *gfx* property represents the underlying system object.

TextAsset

class ggame.asset.**TextAsset**(*text*, ***kwargs*)
The TextAsset is a "virtual" asset that is created on the fly without requiring creation of an image file. A TextAsset instance represents a block of text, together with its styling (font, color, etc.).

Parameters

- **text** (*str*) – The text that should be displayed
- ****kwargs** – Optional formatting and style attributes (below)
- **style='20px Arial'** (*str*) – Text style, size and typeface. Example:

```
'italic 20pt Helvetica'
```

- **width=100** (*int*) – Width of the text block on screen, in pixels.
- **fill=BLACK** (Color) – *Color* instance to specify color and transparency of the text.
- **align='left'** (*str*) – Alignment style of the block. One of 'left', 'center', or 'right'.

Full example:

```
"""
Example of using TextAsset class.
"""
from ggame import TextAsset, Color

TA = TextAsset(
    "Sample Text", style="bold 40pt Arial", width=250, fill=Color(0x1122FF, 1.0)
)
```

destroy()
Destroy or deallocate any underlying graphics resources used by the asset. Call this method on any asset that is no longer being used.

gfx
gfx property represents the underlying system object used to represent this asset. If this asset is composed of multiple assets, then the **first** asset is referenced by *gfx*.

clone()
Create a duplicate asset with the current style settings.

Returns A text asset that is identical to the original, but with current styles.

Return type *TextAsset*

width
Width of the rendered text asset in pixels

height
Height of the rendered text asset in pixels

2.1.3 Sounds

Tools for loading and playing sound resources in ggame applications.

SoundAsset

class ggame.sound.**SoundAsset**(*url*)
Class representing a single sound asset (sound file, such as .mp3 or .wav).

Parameters url (*str*) – The URL or file name of the desired sound. Sound file formats may include *.wav* or *.mp3*, subject to browser compatibility.

Returns The asset instance

url = None
A string containing the url or name of the asset file.

Sound

class ggame.sound.**Sound**(*asset*)

The Sound class represents a sound, with methods for controlling when and how the sound is played in the application.

Parameters **asset** (SoundAsset) – A valid *SoundAsset* instance.

Returns the Sound instance

play()

Play the sound once.

loop()

Play the sound continuously, looping forever.

stop()

Stop playing the sound.

volume

The volume property is a number ranging from 0-100 that represents the volume or intensity of the sound when it is playing.

2.1.4 Sprites

Sprite class for encapsulating all visible objects in ggame applications.

Sprite

class ggame.sprite.**Sprite**(*asset*, *pos=(0, 0)*, *edgedef=None*)

The Sprite class combines the idea of a visual/graphical asset, a position on the screen, and *behavior*. Although the Sprite can be used as-is, it is generally subclassed to give it some desired behavior.

When subclassing the Sprite class, you may customize the initialization code to use a specific asset. A 'step' or 'poll' method may be added for handling per-frame actions (e.g. checking for collisions). Step or poll functions are not automatically called by the *App* class, but you may subclass the *App* class in order to do this.

Furthermore, you may wish to define event callback methods in your customized sprite class. With customized creation, event handling, and periodic processing you can achieve fully autonomous behavior for your sprite objects.

Parameters

- **asset** (*asset*) – An existing graphical asset
- **pos** (*tuple(int, int)*) – The sprite position may be provided, which specifies the starting (x,y) coordinates of the sprite on the screen. By default, the position of a sprite defines the location of its upper-left hand corner. This behavior can be modified by customizing its *center*.
- **edgedef** (*asset*) – An edge definition asset may be provided, which specifies an asset that will be used to define the boundaries of the sprite for the purpose of collision detection. If no *edgedef* asset is given, the required asset is used, which will be a rectangular asset in the case of an image texture. This option is typically used to define a visible image outline for a texture-based sprite that has a transparent texture image background.

Returns Nothing. If the position is on screen the sprite will be displayed in the browser.

Example of use:

```
"""
Example of using Sprite class.
"""
from ggame.sprite import Sprite
from ggame.asset import ImageAsset, CircleAsset
from ggame.app import App

PLAYER = Sprite(ImageAsset("bunny.png"), (100, 100), CircleAsset(50))

App().run()
```

This creates a sprite using the 'player.png' image, positioned with its upper-left corner at coordinates (100,100) and with a 50 pixel radius circular collision border.

`setExtents`()
: update min/max x and y based on position, center, width, height

`firstImage`()
: Select and display the *first* image used by this sprite. This only does something useful if the asset is an `ImageAsset` defined with multiple images.

`lastImage`()
: Select and display the *last* image used by this sprite. This only does something useful if the asset is an `ImageAsset` defined with multiple images.

`nextImage`(*wrap=False*)
: Select and display the *next* image used by this sprite. If the current image is already the *last* image, then the image is not advanced.

  Parameters **`wrap`** (`boolean`) – If *True*, then calling `nextImage()` on the last image will cause the *first* image to be loaded.

  This only does something useful if the asset is an `ImageAsset` defined with multiple images.

`prevImage`(*wrap=False*)
: Select and display the *previous* image used by this sprite. If the current image is already the *first* image, then the image is not changed.

  Parameters **`wrap`** (`boolean`) – If *True*, then calling `prevImage()` on the first image will cause the *last* image to be loaded.

  This only does something useful if the asset is an `ImageAsset` defined with multiple images.

`setImage`(*index=0*)
: Select the image to display by giving its *index*.

  Parameters **`index`** (`int`) – An index to specify the image to display. A value of zero represents the *first* image in the asset.

  This is equivalent to setting the `index` property directly.

  This only does something useful if the asset is an `ImageAsset` defined with multiple images.

`index`
: This is an integer index into the list of images available for this sprite.

`width`
: This is an integer representing the display width of the sprite. Assigning a value to the width will scale the image horizontally.

height
This is an integer representing the display height of the sprite. Assigning a value to the height will scale the image vertically.

x
This represents the x-coordinate of the sprite on the screen. Assigning a value to this attribute will move the sprite horizontally.

y
This represents the y-coordinate of the sprite on the screen. Assigning a value to this attribute will move the sprite vertically.

position
Tuple indicates the position of the sprite on the screen.

fxcenter
This represents the horizontal position of the sprite "center", as a floating point number between 0.0 and 1.0. A value of 0.0 means that the x-coordinate of the sprite refers to its left hand edge. A value of 1.0 refers to its right hand edge. Any value in between may be specified. Values may be assigned to this attribute.

fycenter
This represents the vertical position of the sprite "center", as a floating point number between 0.0 and 1.0. A value of 0.0 means that the x-coordinate of the sprite refers to its top edge. A value of 1.0 refers to its bottom edge. Any value in between may be specified. Values may be assigned to this attribute.

center
This attribute represents the horizontal and vertical position of the sprite "center" as a tuple of floating point numbers. See the descriptions for *fxcenter* and *fycenter* for more details.

visible
This boolean attribute may be used to change the visibility of the sprite. Setting *~ggame.Sprite.visible* to *False* will prevent the sprite from rendering on the screen.

scale
This attribute may be used to change the size of the sprite ('scale' it) on the screen. Value may be a floating point number. A value of 1.0 means that the sprite image will keep its original size. A value of 2.0 would double it, etc.

rotation
This attribute may be used to change the rotation of the sprite on the screen. Value may be a floating point number. A value of 0.0 means no rotation. A value of 1.0 means a rotation of 1 radian in a counter-clockwise direction. One radian is 180/pi or approximately 57.3 degrees.

classmethod collidingCircleWithPoly(*circ*, *poly*)
Determine if a CircleAsset sprite overlaps with a PolygonAsset sprite. This method is called after determining that the two objects are overlapping in their overall extents.

Parameters

- **circ** (Sprite) – A CircleAsset-based sprite.
- **poly** (Sprite) – A PolygonAsset-based sprite.

Returns True if the sprites are overlapping, False otherwise.

Return type boolean

collidingPolyWithPoly(*obj*)
Determine if a pair of PolygonAsset-based sprites are overlapping. This method is called after determining that the two objects are overlapping in their overall extents. This should onlyb e called if *self* is a PolygonAsset-based sprite.

Parameters **`obj`** (`Sprite`) – A PolygonAsset-based sprite.

Returns True if slef overlaps with obj, False otherwise.

Return type boolean

`collidingWith`(*obj*)

Determine if this sprite is currently overlapping another sprite object.

Parameters **`obj`** (`Sprite`) – A reference to another Sprite object.

Return type boolean

Returns *True* if this the sprites are overlapping, *False* otherwise.

`collidingWithSprites`(*sclass=None*)

Determine if this sprite is colliding with any other sprites of a certain class.

Parameters **`sclass`** (*`class`*) – A class identifier that is either *`Sprite`* or a subclass of it that identifies the class of sprites to check for collisions. If *None* then all objects that are subclassed from the *`Sprite`* class are checked.

Return type list

Returns A (potentially empty) list of sprite objects of the given class that are overlapping with this sprite.

`static getImagePath`(*imagename*)

Determine a path to the ggame-provided image that will work for both online (runpython.org) and locally installed ggame libraries. Do not use this with user-provided images.

Parameters **`imagename`** (*`str`*) – The name of an image file found inside the ggame/images folder.

`destroy`()

Prevent the sprite from being displayed or checked in collision detection. Once this is called, the sprite can no longer be displayed or used. If you only want to prevent a sprite from being displayed, set the *`visible`* attribute to *False*.

2.2 ggMath Mathematics

These mathematics and geometry extensions subclass the *`App`* and *`Sprite`* classes to create a framework for building apps that mimic some of the functionality of online math tools like Geogebra.

This *`mathapp`* module implements base classes for *`Sprite`*-based classes defined in this module.

These extensions are very experimental and are not fully developed!

2.2.1 ggMath Application

MathApp

`class` `ggame.mathapp.`**`MathApp`**(*scale=200*)

MathApp is a subclass of the ggame *`App`* class. It incorporates the following extensions:

- Support for zooming the display using the mouse wheel
- Support for click-dragging the display using the mouse button
- **Automatic execution of step functions in all objects and sprites** sub-classed from _`MathDynamic`.

Parameters **scale** (*float*) – Optional parameter sets the initial scale of the display in units of pixels per logical unit. The default is 200.

Returns MathApp instance

view_position
Attribute is used to get or set the current logical coordinates at the center of the screen as a tuple of floats (x,y).

scale = 200

classmethod getSpritesbyClass(*sclass*)
Returns a list of all active sprites of a given class.

Parameters **sclass** (*class*) – A class name (e.g. 'Sprite') or subclass.

Returns A (potentially empty) list of sprite references.

classmethod listenKeyEvent(*eventtype*, *key*, *callback*)
Register to receive keyboard events.

Parameters

- **eventtype** (*str*) – The type of key event to receive (value is one of: *'keydown'*, *'keyup'* or *'keypress'*).
- **key** (*str*) – Identify the keyboard key (e.g. *'space'*, *'left arrow'*, etc.) to receive events for.
- **callback** (*function*) – The function or method that will be called with the *KeyEvent* object when the event occurs.

Returns Nothing

See the source for *KeyEvent* for a list of key names to use with the *key* paramter.

classmethod listenMouseEvent(*eventtype*, *callback*)
Register to receive mouse events.

Parameters

- **eventtype** (*str*) – The type of mouse event to receive (value is one of: *'mousemove'*, *'mousedown'*, *'mouseup'*, *'click'*, *'dblclick'* or *'mousewheel'*).
- **callback** (*function*) – The function or method that will be called with the *ggame.event.MouseEvent* object when the event occurs.

Returns Nothing

classmethod unlistenKeyEvent(*eventtype*, *key*, *callback*)
Use this method to remove a registration to receive a particular keyboard event. Arguments must exactly match those used when registering for the event.

Parameters

- **eventtype** (*str*) – The type of key event to stop receiving (value is one of: *'keydown'*, *'keyup'* or *'keypress'*).
- **key** (*str*) – The keyboard key (e.g. *'space'*, *'left arrow'*, etc.) to stop receiving events for.
- **callback** (*function*) – The function or method that will no longer be called with the *KeyEvent* object when the event occurs.

Returns Nothing

See the source for `KeyEvent` for a list of key names to use with the *key* paramter.

classmethod unlistenMouseEvent(*eventtype*, *callback*)

Use this method to remove a registration to receive a particular mouse event. Arguments must exactly match those used when registering for the event.

Parameters

- **eventtype** (*str*) – The type of mouse event to stop receiving events for (value is one of: *'mousemove'*, *'mousedown'*, *'mouseup'*, *'click'*, *'dblclick'* or *'mousewheel'*).
- **callback** (*function*) – The function or method that will no longer be called with the `ggame.event.MouseEvent` object when the event occurs.

Returns Nothing

classmethod logicalToPhysical(*lp*)

Transform screen coordinates from logical to physical space. Output depends on the current 'zoom' and 'pan' of the screen.

Parameters **lp** (*tuple(float, float)*) – Logical screen coordinates (x, y)

Return type tuple(float,float)

Returns Physical screen coordinates (x, y)

classmethod physicalToLogical(*pp*)

Transform screen coordinates from physical to logical space. Output depends on the current 'zoom' and 'pan' of the screen.

Parameters **lp** (*tuple(float, float)*) – Physical screen coordinates (x, y)

Return type tuple(float,float)

Returns Logical screen coordinates (x, y)

classmethod translateLogicalToPhysical(*pp*)

Transform screen translation from logical to physical space. Output only depends on the current 'zoom' of the screen.

Parameters **lp** (*tuple(float, float)*) – Logical screen translation pair (delta x, delta y)

Return type tuple(float,float)

Returns Physical screen translation ordered pair (delta x, delta y)

classmethod translatePhysicalToLogical(*pp*)

Transform screen translation from physical to logical space. Output only depends on the current 'zoom' of the screen.

Parameters **lp** (*tuple(float, float)*) – Physical screen translation pair (delta x, delta y)

Return type tuple(float,float)

Returns Logical screen translation ordered pair (delta x, delta y)

classmethod distance(*pos1*, *pos2*)

Utility for calculating the distance between any two points.

Parameters

- **pos1** (*tuple(float, float)*) – The first point
- **pos2** (*tuple(float, float)*) – The second point

Return type float

Returns The distance between the two points (using Pythagoras)

classmethod addViewNotification(*handler*)
Register a function or method to be called in the event the view position or zoom changes.

Parameters **handler** (*function*) – The function or method to be called

Returns Nothing

classmethod removeViewNotification(*handler*)
Remove a function or method from the list of functions to be called in the event of a view position or zoom change.

Parameters **handler** (*function*) – The function or method to be removed

Returns Nothing

run(*userfunc=None*)
Calling the `run()` method begins the animation process whereby the `step()` method is called once per animation frame.

Parameters **userfunc** (*function*) – Any function or method which shall be called once per animation frame.

Returns Nothing

ggMath Base Class for Visual Objects

class ggame.mathapp.**_MathVisual**(*asset*, **args*, ***kwargs*)
Abstract Base Class for all visual, potentially dynamic objects.

Parameters

- **asset** (*Asset*) – A valid ggame asset object.
- **args** (*list*) – A list of required positional or non-positional arguments as named in the _posinputsdef and _nonposinputsdef lists overridden by child classes.
- ****kwargs** – See below

Optional Keyword Arguments

- **positioning** (*string*) One of 'logical' or 'physical'
- **size** (*int*) Size of the object (in pixels)
- **width** (*int*) Width of the object (in pixels)
- **color** (*Color*) Valid `Color` object
- **style** (*LineStyle*) Valid `LineStyle` object

selected = None
True if object is currently selected by the UI.

mouseisdown = None
True if object is tracking UI mouse button as down.

step()
Override in your child class to perform periodic processing.

destroy()
Prevent the sprite from being displayed or checked in collision detection. Once this is called, the sprite can no longer be displayed or used. If you only want to prevent a sprite from being displayed, set the `visible` attribute to *False*.

`positioning`
Whether object was created with 'logical' or 'physical' positioning.

`movable`
Whether object can be moved. Set-able and get-able.

`selectable`
Whether object can be selected by the UI. Set-able and get-able.

`strokable`
Whether the object supports a click-drag input from the UI mouse. Set-able and get-able.

`select`()
Place the object in a 'selected' state.

Param None

Returns None

`unselect`()
Place the object in an 'unselected' state.

Param None

Returns None

`mousedown`()
Inform the object of a 'mouse down' event.

Param None

Returns None

`mouseup`()
Inform the object of a 'mouse up' event.

Param None

Returns None

`processEvent`(*event*)
Inform the object of a generic ggame event.

Parameters **`event`** – The ggame event object to receive and process.

Returns None

This method is intended to be overridden.

`physicalPointTouching`(*ppos*)
Determine if a physical point is considered to be touching this object.

Parameters **`ppos`** (*`tuple(int, int)`*) – Physical screen coordinates.

Return type boolean

Returns True if touching, False otherwise.

This method **must** be overridden.

`translate`(*pdisp*)
Perform necessary processing in response to being moved by the mouse/UI.

Parameters **`pdisp`** (*`tuple(int, int)`*) – Translation vector (x,y) in physical screen units.

Returns None

This method **must** be overridden.

stroke(*ppos*, *pdisp*)
Perform necessary processing in response to click-drag action by the mouse/UI.

Parameters

- **ppos** (*tuple(int,int)*) – Physical coordinates of stroke start.
- **pdisp** (*tuple(int,int)*) – Translation vector of stroke action in physical screen units.

Returns None

This method is intended to be overridden.

canStroke(*ppos*)
Can the object respond to beginning a stroke action at the given position.

Parameters **ppos** (*tuple(int,int)*) – Physical coordinates of stroke start.

Return type Boolean

Returns True if the object can respond, False otherwise.

This method is intended to be overridden.

touchAsset(*force=False*)
Check to see if an asset needs to be updated it and if so (or forced) call the `_updateAsset()` method.

MathApp classes for representing geometric points

2.2.2 Point Objects

These classes are subclasses of *Sprite* and are used to represent points in geometry and mathematics.

_Point

This is the abstract base class for all point classes.

class ggame.point.**_Point**(*asset*, **args*, ***kwargs*)
Abstract base class for all point classes.

Parameters

- **asset** (*Asset*) – A valid ggame Asset object
- **pos** (*tuple(float,float)*) – The position (physical or logical)

step()
Perform periodic processing.

physicalPointTouching(*ppos*)
Determine if a physical point is considered to be touching this point.

Parameters **ppos** (*tuple(int,int)*) – Physical screen coordinates.

Return type boolean

Returns True if touching, False otherwise.

translate(*pdisp*)
Perform necessary processing in response to being moved by the mouse/UI.

Parameters **pdisp** (*tuple(int,int)*) – Translation vector (x,y) in physical screen units.

Returns None

distanceTo(*otherpoint*)
Compute the distance to another `_Point` object.

Parameters **otherpoint** (`_Point`) – A reference to the other `_Point`

Return type float

Returns The distance (in logical units) to the other point

Point

class ggame.point.**Point**(**args*, ***kwargs*)
Basic point object representing any point in a geometrical sense. An instantiated Point object is *callable* and will return a tuple with its logical position as an (x,y) pair.

Parameters

- **pos** (*tuple(float, float)*) – Position in physical or logical units.
- ****kwargs** – See below

Optional Keyword Arguments

- **positioning** (*str*) One of 'logical' (default) or 'physical'
- **size** (*int*) Radius of the point (in pixels)
- **color** (*Color*) Valid `Color` object
- **style** (*LineStyle*) Valid `LineStyle` object

Example:

```
"""
Example of using MathApp Point class.
"""
from ggame.asset import Color
from ggame.point import Point
from ggame.mathapp import MathApp

P1 = Point((0, 1), color=Color(0xFF8000, 1.0))
P1.movable = True
# An orange point that can be moved

P2 = Point(lambda: (P1()[0], P1()[1] + 1))
# A point position based on P1
P3 = Point((1, 0))
# A third, fixed point

MathApp().run()
```

ImagePoint

class ggame.point.**ImagePoint**(*url*, **args*, ***kwargs*)

Point **object that uses an image as its on-screen** representation.

Parameters

- **url** (*str*) – Location of an image file (png, jpg)
- ***args** – See below
- ****kwargs** – See below

Required Arguments

- **pos** (*tuple(float,float)*) Position in physical or logical units.

Optional Keyword Arguments

- **positioning** (*str*) One of 'logical' (default) or 'physical'
- **frame** (*Frame*) The sub-frame location of image within the image file
- **qty** (*int*) The number of sub-frames, when used as a sprite sheet
- **direction** (*str*) One of 'horizontal' (default) or 'vertical'
- **margin** (*int*) Pixels between sub-frames if sprite sheet

physicalPointTouching(*ppos*)
Determine if a physical point is considered to be touching point's image.

Parameters **ppos** (*tuple(int, int)*) – Physical screen coordinates.

Return type boolean

Returns True if touching, False otherwise.

Line objects for MathApp applications

2.2.3 Line Objects

This category currently only has one class: *LineSegment*, but will eventually be extended to include at least Line and Ray.

LineSegment

class ggame.line.**LineSegment**(**args*, ***kwargs*)
Create a line segment on the screen. This is a subclass of *Sprite* and *_MathVisual* but most of the inherited members are of little use and are not shown in the documentation.

Parameters

- ***args** – See below
- ****kwargs** – See below

Required Arguments

- **pos** (*tuple(float,float)*) **Starting point of the segment, which may** be a literal tuple of floats, or a reference to any object or function that returns or evaluates to a tuple of floats.
- **end** (*tuple(float,float)*) Ending point of the segment (see above)

Optional Keyword Arguments

- **positioning** (*str*) One of 'logical' or 'physical'
- **style** (*LineStyle*) Valid *LineStyle* object

Example:

```
from ggame.point import Point
from ggame.line import LineSegment
from ggame.mathapp import MathApp

p1 = Point((2,1))
ls = LineSegment(p1, Point((1,1)))

MathApp().run()
```

physicalPointTouching(*ppos*)
This method always returns False.

translate(*pdisp*)
This method is not implemented.

Circle

Circle object for MathApp applications

class ggame.circle.**Circle**(**args*, ***kwargs*)
Create a circle on the screen. This is a subclass of *Sprite* and *_MathVisual* but most of the inherited members are of little use and are not shown in the documentation.

Parameters

- ***args** – See below
- ****kwargs** – See below

Required Arguments

- **pos (*tuple(float,float)*) Center point of the circle, which may** be a literal tuple of floats, or a reference to any object or function that returns or evaluates to a tuple of floats.
- **radius [float or Point] Radius of the circle (logical units)** or a *Point* on the circle.

Optional Keyword Arguments

- **positioning** (*str*) One of 'logical' or 'physical'
- **style** (*LineStyle*) Valid *LineStyle* object
- **color** (*Color*) Valid Color object'

Example:

```
"""
Example of using MathApp Circle class.
"""
from ggame.point import Point
from ggame.circle import Circle
from ggame.mathapp import MathApp

# P1 is the center of the circle
P1 = Point((0.5, 1))
# P2 is on the perimeter
P2 = Point((1.3, 1))
# define a circle from center and perimeter points
C = Circle(P1, P2)
# allow the user to drag the perimeter point to resize the circle
```

(continues on next page)

(continued from previous page)

```
P2.movable = True

MathApp().run()
```

center
An ordered pair (x,y) or `Point` that represents the (logical) circle center. This attribute is only get-able.

radius
A **float** that represents the radius of the circle. This attribugte is only get-able.

step()
Override in your child class to perform periodic processing.

physicalPointTouching(*ppos*)
Determine if a physical point is considered to be touching this object.

Parameters **ppos** (`tuple(int, int)`) – Physical screen coordinates.

Return type boolean

Returns True if touching, False otherwise.

This method **must** be overridden.

translate(*pdisp*)
Perform necessary processing in response to being moved by the mouse/UI.

Parameters **pdisp** (`tuple(int, int)`) – Translation vector (x,y) in physical screen units.

Returns None

This method **must** be overridden.

2.2.4 Text Objects

Label

MathApp label classes for displaying text.

class ggame.label.**Label**(**args*, ***kwargs*)
Create a text label on the screen. This is a subclass of `Sprite` and `_MathVisual` but most of the inherited members are of little use and are not shown in the documentation.

Parameters

- ***args** – See below
- ****kwargs** – See below

Required Arguments

- **pos (*tuple(float,float)*) Screen position of the label, which may** be a literal tuple of floats, or a reference to any object or function that returns or evaluates to a tuple of floats.
- **text (*str*) Text to appear in the label. This may be a literal** string or a reference to any object or function that returns or evaluates to a string.

Optional Keyword Arguments

- **positioning** (*str*) One of 'logical' or 'physical'
- **size** (*int*) Size of text font (in pixels)

- **width** (*int*) Width of the label (in pixels)
- **color** (*Color*) Valid `Color` object

Example:

```
"""
Example of using the Label class.
"""

from ggame.asset import Color
from ggame.label import Label
from ggame.mathapp import MathApp

L = Label(
    (20, 80),  # physical location on screen
    "Initial Speed (m/s)",  # text to display
    size=15,  # text size (pixels)
    positioning="physical",  # use physical coordinates
    color=Color(0x202000, 1.0),
)  # text color

MathApp().run()
```

`physicalPointTouching`(*ppos*)

Determine if a physical point is considered to be touching this object.

Parameters **`ppos`** (*`tuple(int, int)`*) – Physical screen coordinates.

Return type boolean

Returns True if touching, False otherwise.

This method **must** be overridden.

`translate`(*pdisp*)

Perform necessary processing in response to being moved by the mouse/UI.

Parameters **`pdisp`** (*`tuple(int, int)`*) – Translation vector (x,y) in physical screen units.

Returns None

This method **must** be overridden.

2.2.5 Indicators

MathApp indicator classes for displaying two-state or boolean values.

ImageIndicator

`class` `ggame.indicator.`**`ImageIndicator`**(*url*, **args*, ***kwargs*)

Use a sprite sheet image to indicate integer or boolean values.

Required Inputs

Parameters

- **`url`** (*`str`*) – Location of image file consisting of a multi-image sprite sheet.
- **`pos`** (*`(float, float)`*) – Position in physical or logical units. May be a `ggame.point.Point` instance, a literal (x,y) pair, or a function that returns an (x,y) pair.

- **value** (*int*) – The state of the indicator. May be a function that returns a suitable (integer) value.
- ****kwargs** – See below

Optional Keyword Arguments

- **positioning** (*str*) One of 'logical' (default) or 'physical'
- **frame** (***:class:'ggame.asset.Frame'***) **Sub-frame location of sub-image within** the main image.
- **qty** (*int*) The number of sub-frames, when used as sprite sheet.
- **direction** (*str*) One of 'horizontal' (default) or 'vertical'.
- **margin** (*int*) Number of pixels between sub-frames if sprite sheet.

Example:

```python
"""
Example of using ImageIndicator class.
"""

from ggame.mathapp import MathApp
from ggame.indicator import ImageIndicator
from ggame.inputpoint import InputImageButton
from ggame.asset import Frame

BUTTON = InputImageButton(
    "images/button-round.png",
    None,
    (40, 105),
    positioning="physical",
    frame=Frame(0, 0, 100, 100),
    qty=2,
)
BUTTON.scale = 0.5

LIGHT = ImageIndicator(
    "images/red-led-off-on.png",
    (100, 100),
    BUTTON,  # button object supplies the indicator state.
    positioning="physical",
    frame=Frame(0, 0, 600, 600),
    qty=2,
)
LIGHT.scale = 0.1

MathApp().run()
```

physicalPointTouching(*ppos*)

Determine if a physical point is considered to be touching this object.

Parameters **ppos** (*tuple(int, int)*) – Physical screen coordinates.

Return type boolean

Returns True if touching, False otherwise.

This method **must** be overridden.

translate (*pdisp*)
Perform necessary processing in response to being moved by the mouse/UI.

Parameters **pdisp** (*tuple(int, int)*) – Translation vector (x,y) in physical screen units.

Returns None

This method **must** be overridden.

LEDIndicator

class ggame.indicator.**LEDIndicator** (**args*, ***kwargs*)
Subclass of ImageIndicator using a red LED on/off image.

Required Inputs

Parameters

- **pos** (*(float, float)*) – Position in physical or logical units. May be a *ggame.point.Point* instance, a literal (x,y) pair, or a function that returns an (x,y) pair.
- **value** (*int*) – The state of the indicator. May be a function that returns a suitable (integer) value.
- ****kwargs** – See below

Optional Keyword Arguments

- **positioning** (*str*) One of 'logical' (default) or 'physical'

Example:

```
"""
Example of using MathApp LEDIndicator class.
"""
from ggame.mathapp import MathApp
from ggame.indicator import LEDIndicator
from ggame.inputpoint import MetalToggle

TOGGLE = MetalToggle(0, (-1, 0))

SWITCH = LEDIndicator((-1, 0.5), TOGGLE)

MathApp().run()
```

2.2.6 Input Controls

Slider

MathApp input class for accepting user numeric input with a "slider" control.

class ggame.slider.**Slider** (**args*, ***kwargs*)
Create a 'slider' style numeric input control on the screen. This is a subclass of *Sprite* and *_MathVisual* but most of the inherited members are of little use and are not shown in the documentation.

Parameters

- ***args** – See below
- ****kwargs** – See below

Required Arguments

- **pos** (***tuple(float,float)***) **Screen position of the slider, which may** be a literal tuple of floats, or a reference to any object or function that returns or evaluates to a tuple of floats.
- **minval** (*float*) The minimum value of the slider
- **maxval** (*float*) The maximum value of the slider
- **initial** (*float*) The initial value of the slider

Optional Keyword Arguments

- **steps** (*int*) Number of steps between minval and maxval (default 50)
- **leftkey** (***str***) **Name of a keyboard key that will make the** slider control step down (move left) (default None). See `KeyEvent` for a list of names.
- **rightkey** (***str***) **Name of a keyboard key that will make the slider** control step up (move right) (default None)
- **centerkey** (***str***) **Name of a keyboard key that will make the slider** move to its center position (default None)
- **positioning** (*str*) One of 'logical' or 'physical'
- **size** (*int*) Width of the slider (in pixels)
- **color** (*Color*) Valid `Color` object
- **style** (*LineStyle*) Valid `LineStyle` object

Example:

```
"""
Example of using Slider class.
"""

from ggame.slider import Slider
from ggame.mathapp import MathApp

S = Slider(
    (100, 150),  # screen position
    0,  # minimum value
    250,  # maximum value
    125,  # initial value
    positioning="physical",  # use physical coordinates for position
    steps=10,
)  # 10 steps between 125 and 250

MathApp().run()
```

`value`
Report value of the slider. Attribute is get-able and set-able.

`step`()
Override in your child class to perform periodic processing.

`increment`(*step*)
Increment the slider value.

Parameters **`step`** (`float`) – The amount by which the slider control should be adjusted.

Returns None

`select``()`
Place the object in a 'selected' state.

Param None

Returns None

`unselect``()`
Place the object in an 'unselected' state.

Param None

Returns None

`canstroke``(`*ppos*`)`
Function returns true if the given physical position corresponds with a part of the slider that can be dragged (i.e. the thumb).

Parameters **`ppos`** (*`(float, float)`*) – Physical screen coordinates.

Returns True if the position represents a draggable part of the control.

Return type boolean

`stroke``(`*ppos*, *pdisp*`)`
Function performs the action of stroking or click-dragging on the slider control (i.e. dragging the thumb).

Parameters

- **`float) ppos`** (*`(float,`*) – Physical screen coordinates.
- **`float) pdisp`** (*`(float,`*) – Physical displacement vector.

Returns None

`physicalPointTouching``(`*ppos*`)`
Determine if a physical point is considered to be touching this object.

Parameters **`ppos`** (*`tuple(int, int)`*) – Physical screen coordinates.

Return type boolean

Returns True if touching, False otherwise.

This method **must** be overridden.

`physicalPointTouchingThumb``(`*ppos*`)`
Determine if a physical screen location is touching the slider "thumb".

Parameters **`float) ppos`** (*`(float,`*) – Physical screen coordinates.

Returns True if touching, False otherwise.

Return type boolean

`translate``(`*pdisp*`)`
Perform necessary processing in response to being moved by the mouse/UI.

Parameters **`pdisp`** (*`tuple(int, int)`*) – Translation vector (x,y) in physical screen units.

Returns None

This method **must** be overridden.

`destroy``()`
Prevent the sprite from being displayed or checked in collision detection. Once this is called, the sprite can no longer be displayed or used. If you only want to prevent a sprite from being displayed, set the `visible` attribute to *False*.

InputNumeric

MathApp input classes for accepting user numeric and pushbutton input.

class ggame.input.**InputNumeric**(*pos*, *val*, ***kwargs*)

Create a *Label* that can be selected to accept new **numeric** input from the keyboard. This is a subclass of *Sprite* and *_MathVisual* but most of the inherited members are of little use and are not shown in the documentation.

Parameters

- **pos** (*tuple(float,float)*) – Screen position of the control, which may be a literal tuple of floats, or a reference to any object or function that returns or evaluates to a tuple of floats.
- **val** (*float*) – Initial value of the input text
- ****kwargs** – See below

Optional Keyword Arguments

- **fmt** (*str*) a Python format string (default is {0:.2f})
- **positioning** (*str*) One of 'logical' or 'physical'
- **size** (*int*) Size of text font (in pixels)
- **width** (*int*) Width of the label (in pixels)
- **color** (*Color*) Valid *Color* object

Example:

```
"""
Example of using InputNumeric class.
"""

from ggame.input import InputNumeric
from ggame.mathapp import MathApp

P = InputNumeric(
    (300, 275),  # screen coordinates of input
    3.14,  # initial value
    positioning="physical",
)  # use physical coordinates

MathApp().run()
```

processEvent(*event*)

Inform the object of a generic ggame event.

Parameters **event** – The ggame event object to receive and process.

Returns None

This method is intended to be overridden.

select()

Place the object in a 'selected' state.

Param None

Returns None

unselect()

Place the object in an 'unselected' state.

Param None

Returns None

InputButton

class ggame.input.**InputButton**(*callback*, **args*, ***kwargs*)

Create a *Label* that can be clicked with a mouse to execute a user-defined function. This is a subclass of *Sprite* and *_MathVisual* but most of the inherited members are of little use and are not shown in the documentation.

Parameters

- **callback** (*function*) – A reference to a function to execute, passing this button object, when the button is clicked
- **pos** (*tuple(float, float)*) – Screen position of the control, which may be a literal tuple of floats, or a reference to any object or function that returns or evaluates to a tuple of floats.
- **val** (*float*) – Initial value of the input text
- ***args** – See below
- ****kwargs** – See below

Required Arguments

- **pos (*tuple(float,float)*) Screen position of the button, which may** be a literal tuple of floats, or a reference to any object or function that returns or evaluates to a tuple of floats.
- **text (*str*) Text to appear in the button. This may be a literal** string or a reference to any object or function that returns or evaluates to a string.

Optional Keyword Arguments

- **positioning** (*str*) One of 'logical' or 'physical'
- **size** (*int*) Size of text font (in pixels)
- **width** (*int*) Width of the label (in pixels)
- **color** (*Color*) Valid *Color* object

Example:

```python
"""
Example of using MathApp InputButton class.
"""
from ggame.input import InputButton
from ggame.mathapp import MathApp


def pressbutton(_button):
    """
    Callback function executed when button is pressed.
    """
    print("InputButton pressed!")
```

(continues on next page)

(continued from previous page)

```
InputButton(
    pressbutton,  # reference to handler
    (20, 80),  # physical location on screen
    "Press Me",  # text to display
    size=15,  # text size (pixels)
    positioning="physical",
)  # use physical coordinates

MathApp().run()
```

select()

Place the object in a 'selected' state.

Param None

Returns None

MathApp GUI input pushbutton and toggle controls.

InputImageButton

class ggame.inputpoint.**InputImageButton**(*url*, *callback*, **args*, ***kwargs*)

ImagePoint **object that uses an image as its on-screen and** activates a callback function when pressed.

Required Arguments

Parameters

- **url** (*str*) – Location of an image file (png, jpg)
- **callback** (*function*) – Reference to a function to execute, passing this button object.
- ***args** – See below
- ****kwargs** – See below

Required Arguments

- **pos** (*tuple(float,float)*) Position in physical or logical units.

Optional Keyword Arguments

- **positioning** (*str*) One of 'logical' (default) or 'physical'
- **frame** (*Frame*) The sub-frame location of image within the image file
- **qty** (*int*) The number of sub-frames, when used as a sprite sheet
- **direction** (*str*) One of 'horizontal' (default) or 'vertical'
- **margin** (*int*) Pixels between sub-frames if sprite sheet

Example:

```
"""
Example of using the InputImageButton class.
"""
from ggame.inputpoint import InputImageButton
from ggame.mathapp import MathApp
```

(continues on next page)

(continued from previous page)

```
from ggame.asset import Frame


def pressbutton(_button):
    """
    Callback function executed when button is pressed.
    """
    print("Button Pressed!")


BUTTON = InputImageButton(
    "images/button-round.png", pressbutton, (0, 0), frame=Frame(0, 0, 100, 100),␣
↪qty=2
)
BUTTON.scale = 0.5

MathApp().run()
```

select()

Place the object in a 'selected' state.

Param None

Returns None

InputImageToggle

class ggame.inputpoint.**InputImageToggle**(*url*, *statelist*, *initindex*, **args*, ***kwargs*)

Spritesheet-based input control that toggles through a set of states with each mouseclick.

Required Inputs

Parameters

- **url** (*str*) – Location of image file consisting of a multi-image sprite sheet.
- **statelist** (*list*) – List of values to correspond with toggle states.
- **initindex** (*int*) – Index to initial toggle state.
- **pos** (*(float,float)*) – Position in physical or logical units. May be a *ggame.point.Point* instance, a literal (x,y) pair, or a function that returns an (x,y) pair.
- ****kwargs** – See below

Optional Keyword Arguments

- **positioning** (*str*) One of 'logical' (default) or 'physical'
- **frame** (*:class:'ggame.asset.Frame'*) **Sub-frame location of sub-image within** the main image.
- **direction** (*str*) One of 'horizontal' (default) or 'vertical'.
- **margin** (*int*) Number of pixels between sub-frames if sprite sheet.

select()

Place the object in a 'selected' state.

Param None

Returns None

MetalToggle

class ggame.inputpoint.**MetalToggle**(*initindex*, **args*, ***kwargs*)
Metal toggle input control that toggles through two states with each mouseclick.

Required Inputs

Parameters

- **initindex** (*int*) – Index to initial toggle state (0 or 1).
- **pos** (*(float,float)*) – Position in physical or logical units. May be a *ggame.point.Point* instance, a literal (x,y) pair, or a function that returns an (x,y) pair.

Example:

```
"""
Example of using MathApp LEDIndicator class.
"""
from ggame.mathapp import MathApp
from ggame.indicator import LEDIndicator
from ggame.inputpoint import MetalToggle

TOGGLE = MetalToggle(0, (-1, 0))

SWITCH = LEDIndicator((-1, 0.5), TOGGLE)

MathApp().run()
```

GlassButton

class ggame.inputpoint.**GlassButton**(*callback*, **args*, ***kwargs*)
Glass button input control that triggers a callback when clicked.

Required Inputs

Parameters

- **callback** (*function*) – Reference to a function to execute, passing this button object as an argument.
- **pos** (*(float,float)*) – Position in physical or logical units. May be a *ggame.point.Point* instance, a literal (x,y) pair, or a function that returns an (x,y) pair.

Example:

```
"""
Example of using GlassButton class.
"""
from ggame.inputpoint import GlassButton
from ggame.mathapp import MathApp


def pressbutton(_button):
    """
    Callback function executed when button is pressed.
    """
    print("Button Pressed!")
```

(continues on next page)

(continued from previous page)

```
BUTTON = GlassButton(pressbutton, (0, 0))

MathApp().run()
```

2.2.7 Time Utility

MathApp class for creating periodic and timed callbacks.

Timer

class ggame.timer.**Timer**

The Timer class instantiates an object whose basic function is to report the number of seconds since its creation by calling the object as a function with an empty argument list.

The Timer class accepts no arguments during creation.

Example of use:

```
"""
Example of using the Timer class.
"""

from ggame.timer import Timer
from ggame.mathapp import MathApp


def timercallback(t):
    """
    Callback function to receive notification of timer expiration.
    """
    print("time's up at", t.time, "seconds!")


TIMER = Timer()
# Execute timercallback after 5 seconds
TIMER.callAfter(5, timercallback)

MathApp().run()
```

reset()

Set the reference time to the MathApp current time. If the timer is reset before the app initializes then do nothing.

Returns None

time

Attribute is always updated with the number of seconds since the timer was created.

step()

Override in your child class to perform periodic processing.

callAfter(*delay*, *callback*, *periodic=False*)

Set a callback to occur either once or periodically. The callback function should accept a single parameter which will be a reference to the timer object that called it.

Parameters

- **`delay`** (*`float`*) – The number of seconds to wait before executing the callback function
- **`callback`** (*`function`*) – The callback function to call on timer expiration
- **`periodic`** (*`boolean`*) – Set True if the callback function should be executed periodically

Returns None

`callAt`(*calltime*, *callback*)
Set a callback to occur at a specific time (seconds since the Timer object was created or since its `reset()` method was called.

Parameters

- **`time`** (*`float`*) – The time to wait since timer creation or reset before calling the callback function.
- **`callback`** (*`function`*) – The callback function to call

Returns None

`callEvery`(*period*, *callback*)
Set a callback to occur periodically. The callback function should accept a single parameter which will be a reference to the timer object that called it.

Parameters

- **`period`** (*`float`*) – The number of seconds to wait before each execution of the callback function
- **`callback`** (*`function`*) – The callback function to call on timer expiration

Returns None

2.3 ggLogic Digital Logic

These MathApp-based digital logic classes are experimental.

2.3.1 BoolNOT

class `ggame.logic.`**`BoolNOT`**(**args*, ***kwargs*)
Logical NOT boolean gate.

2.3.2 BoolNOR

class `ggame.logic.`**`BoolNOR`**(**args*, ***kwargs*)
Logical NOR boolean gate. Multiple inputs.

2.3.3 BoolNAND

class `ggame.logic.`**`BoolNAND`**(**args*, ***kwargs*)
Logical NAND boolean gate. Multiple inputs.

2.3.4 BoolAND

class ggame.logic.**BoolAND**(**args*, ***kwargs*)
Logical AND boolean gate. Multiple inputs.

2.3.5 BoolSRFF

2.4 ggAstro Astronautics

ggame extensions for modeling spacecraft in planetary orbit

2.4.1 Planet

class ggame.astro.**Planet**(***kwargs*)
Initialize the Planet object.

Optional keyword parameters are supported:

Parameters

- **viewscale** (*float*) – Pixels per meter in graphics display. Default is 10.
- **radius** (*float*) – Radius of the planet in meters. Default is Earth radius.
- **planetmass** (*float*) – Mass of the planet in kg. Default is Earth mass.
- **color** (*int*) – Color of the planet. Default is greenish (0x008040).
- **viewalt** (*float*) – Altitude of initial viewpoint in meters. Default is rocket altitude.
- **viewanom** (*float*) – True anomaly (angle) of initial viewpoint in radians. Default is the rocket anomaly.
- **viewanomd** (*float*) – True anomaly (angle) of initial viewpoing in degrees. Default is the rocket anomaly.

Example:

```
"""
Example of using Planet class.
"""
from ggame.astro import Planet

EARTH = Planet(viewscale=0.00005)
EARTH.run()
```

run(*userfunc=None*)
Execute the Planet (and Rocket) simulation without setting the initial view.

runWithRocket(*rocket=None*)
Execute the Planet (and Rocket) simulation.

Optional parameters

Parameters **rocket** (Rocket) – Reference to a Rocket object - sets the initial view

Returns None

2.4.2 Rocket

class `ggame.astro.`**`Rocket`**(*planet*, ***kwargs*)

Rocket is a class for simulating the motion of a projectile through space, acted upon by arbitrary forces (thrust) and by gravitaitonal attraction to a single planetary object.

Required parameters:

Parameters **`planet`** (`Planet`) – Reference to a *`Planet`* object.

Optional keyword parameters are supported:

Parameters

- **`bitmap`** (*`str`*) – Url of a suitable bitmap image for the rocket (png recommended), default is *images/rocket.png*
- **`bitmapscale`** (*`float`*) – Scale factor for bitmap. Default is 0.1
- **`velocity`** (*`float`*) – Initial rocket speed. Default is zero.
- **`directiond`** (*`float`*) – Initial rocket direction in degrees. Default is zero.
- **`direction`** (*`float`*) – Initial rocket direction in radians. Default is zero.
- **`tanomalyd`** (*`float`*) – Initial rocket true anomaly in degrees. Default is 90.
- **`tanomaly`** (*`float`*) – Initial rocket true anomaly in radians. Default is pi/2.
- **`altitude`** (*`float`*) – Initial rocket altitude in meters. Default is zero.
- **`showstatus`** (*`bool`*) – Boolean displays flight parameters on screen. Default is True.
- **`int) statuspos`** (*`(int,`*) – Tuple with screen x,y coordinates of flight parameters. Default is upper left.
- **`statuslist`** (*`list[str]`*) – List of status names to include in flight parameters. Default is all, consisting of: “velocity”, “acceleration”, “course”, “altitude”, “thrust”, “mass”, “trueanomaly”, “scale”, “timezoom”, “shiptime”
- **`leftkey`** (*`str`*) – A *`ggame.event.KeyEvent`* key identifier that will serve as the “rotate left” key while controlling the ship. Default is ‘left arrow’.
- **`rightkey`** (*`str`*) – A *`ggame.event.KeyEvent`* key identifier that will serve as the “rotate right” key while controlling the ship. Default is ‘right arrow’.

Following parameters may be set as a constant value, or pass in the name of a function that will return the value dynamically or the name of a *ggmath* UI control that will return the value.

Parameters

- **`or float timezoom`** (*`function`*) – Scale factor for time zoom. Factor = 10^timezoom
- **`or float heading`** (*`function`*) – Direction to point the rocket in (must be radians)
- **`or float mass`** (*`fucntion`*) – Mass of the rocket (must be kg)
- **`or float thrust`** (*`function`*) – Thrust of the rocket (must be N)

Animation related parameters may be ignored if no sprite animation:

Parameters

- **`bitmapframe`** (`Frame`) – ((x1,y1),(x2,y2)) tuple defines a region in the bitmap
- **`bitmapqty`** (*`int`*) – Number of bitmaps – used for animation effects

- **bitmapdir** (*str*) – “horizontal” or “vertical” use with animation effects. Default is “horizontal”
- **bitmapmargin** (*int*) – Pixels between successive animation frames
- **tickrate** (*float*) – Frequency of spacecraft dynamics calculations (Hz)

Example:

```
"""
Example of using Rocket class.
"""
from ggame.astro import Planet, Rocket

EARTH = Planet(viewscale=0.00005)
EARTH.run()

ROCKET = Rocket(EARTH, altitude=400000, velocity=7670, timezoom=2)
```

planet = None
Reference to an app object of *Planet* class

tickrate = None
Target animation frames per second.

localheading = None
Heading (angle) of travel

timezoom = None
Reference to a function that will dynamically determine the time zoom factor. An integer: 1,2,3 faster, -1, slower.

heading = None
Reference to a function that will dynamically calculate spaceship orientation or heading (in radians)

mass = None
Reference to a function that will dynamically calculate the spaceship mass (in kg).

thrust = None
Reference to a funtion that will dynamically calculate the rocket thrust (in N).

addStatusReport (*statuslist*, *statusfuncs*, *statusselect*)
Accept list of all status names, all status text functions, and the list of status names that have been selected for display.

Parameters

- **statuslist** (*list[str]*) – List of status names to include in flight parameters. Default is all, consisting of: “velocity”, “acceleration”, “course”, “altitude”, “thrust”, “mass”, “trueanomaly”, “scale”, “timezoom”, and “shiptime”.
- **statusfuncs** (*list[func]*) – List of function references corresponding to statuslist parameter.
- **statusselect** (*list[str]*) – List of names chosen from statuslist to display.

Returns None

xyposition
Report the x,y tuple for logical position of the spaceship.

tanomalyd
Report/set the spaceship position as a direction relative to central body (degrees).

`altitude`
Report/set the spaceship altitude of planet surface in meters.

`velocity`
Report the spaceship velocity scalar in m/s.

`acceleration`
Report the spaceship acceleration scalar in m/s.

`tanomaly`
Report/set the spaceship position as a direction relative to central body (radians).

`r`
Report the spaceship distance (radius) from central body center of mass.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index