PyUnity

Release 0.8.4

Mar 10, 2022

Contents:

1 Version 0.8.4 (in development) 1

2 Disclaimer 3

3 Installing 5

4 Links 7

Python Module Index 69

Index 71

CHAPTER 1

Version 0.8.4 (in development)

PyUnity is a pure Python 3D Game Engine that was inspired by the structure of the Unity Game Engine. This does not mean that PyUnity are bindings for the UnityEngine. However, this project has been made to facilitate any programmer, beginner or advanced, novice or veteran.

CHAPTER 2

Disclaimer

As we have said above, this is not a set of bindings for the UnityEngine, but a pure Python library to aid in making 3D games in Python.

CHAPTER 3

Installing

To install PyUnity for Linux distributions based on Ubuntu or Debian, use:

```
> pip3 install pyunity
```

To install PyUnity for other operating systems, use pip:

```
> pip install pyunity
```

Alternatively, you can clone the repository to build the package from source. The latest version is on the master branch and you can build as follows:

```
> git clone https://github.com/pyunity/pyunity
> git checkout master
> python setup.py install
```

The latest builds are on the `develop` branch which is the default branch. These builds are sometimes broken, so use at your own risk.

```
> git clone https://github.com/pyunity/pyunity
> python setup.py install
```

Its only dependencies are PyOpenGL, PySDL2, GLFW, Pillow and PyGLM. Microsoft Visual C++ Build Tools are required on Windows for building yourself.

CHAPTER 4

Links

For more information check out *the API Documentation*.

If you would like to contribute, please first see the contributing guidelines, check out the latest issues and make a pull request.

4.1 Releases

4.1.1 v0.8.2

Bugfix regarding `Quaternion.FromDir`, `Quaternion.Euler`, `abstractmethod` and 2D depth buffers.

4.1.2 v0.8.1

Bugfix regarding camera position updating and input axes.

4.1.3 v0.8.0

New features:

- Rewrote documentation and docstrings
- Reformatted code
- F string integration
- **`ImmutableStruct` and `ABCMeta` metaclasses**
  - The `ABCMeta` class has more features than the default Python `abc` module.
- Rewrote examples
- **Combined many functions common to both Vector2 and Vector3 into a single Vector class.**

  - If you want to implement your own Vector classes, subclass from Vector and implement the required abstract methods.
- Fixed quaternion and rotation maths
- Input axes and mouse input
- Multiple lights
- Different light types
- Window provider caching and checking
- **Gui components**
  - This includes buttons, checkboxes, images and text boxes
  - Rect transforms can be very flexible
  - Platform-specific font loading
- **Stub package**
  - This will work with editors such as VSCode and PyCharm, just install `pyunity-stubs` from pip

Stub package: https://pypi.org/project/pyunity-stubs

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.8.0

4.1.4 v0.7.1

Extra features used in the PyUnity Editor.

Changes:

- Code of Conduct and Contributing guides
- Rewrote most of the README to clear confusion about what PyUnity really is
- RGB and HSV
- Better GameObject deleting
- ShowInInspector and HideInInspector
- Dynamic lighting

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.7.1

4.1.5 v0.7.0

New features:

- Customizable skybox
- Editor integration
- Rewrote scene saving and loading
- PYUNITY_WINDOW_PROVIDER environment variable
- Fixed example 8

Editor GitHub: https://github.com/pyunity/pyunity-gui

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.7.0

4.1.6 v0.6.0

Project structure update.

New features:

- Replaced Pygame with PySDL2
- Revamped audio module
- Fixed input bugs
- Added scene saving
- Added project saving
- Added project structure
- Automated win32 builds on Appveyor
- Removed redundant code from fixed function pipeline

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.6.0

4.1.7 v0.5.2

Small minor fix of shader inclusion in binary distributions.

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.5.2

4.1.8 v0.5.1

Bugfix that fixes the shaders and dependency management.

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.5.1

4.1.9 v0.5.0

Big rendering update that completely rewrites rendering code and optimizes it.

New features:

- Script loading
- Shaders
- Vertex buffer objects and vertex array objects
- Optimized rendering
- Colours
- Textures
- New lighting system
- New meshes and mesh loading

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.5.0

4.1.10 v0.4.0

Small release that has large internal changes.

New features:

- Added logger
- Moved around files and classes to make it more pythonic
- Rewrote docs
- Fixed huge bug that broke all versions from 0.2.0-0.3.1
- Clarified README.md

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.4.0

4.1.11 v0.3.1

Bugfix on basically everything because 0.3.0 was messed up.

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.3.1

4.1.12 v0.3.0

After a long break, 0.3.0 is finally here!

New features:

- Added key input (not fully implemented)
- Fixed namespace pollution
- Fixed minor bugs
- Window resizing implemented
- New Scene loading interface
- Python 3.9 support
- Finished pxd files
- LGTM Integration
- AppVeyor is now the main builder
- Code is now PEP8-friendly
- Added tests.py
- Cleaned up working directory

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.3.0

4.1.13 v0.2.1

Small bugfix around the AudioClip loading and inclusion of the OGG file in example 8.

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.2.1

4.1.14 v0.2.0

A CI integration update, with automated building from Appveyor and Travis CI.

Features:

- Shaded faces with crisp colours
- PXD files to optimize Cython further (not yet implemented fully)
- Scene changing
- FPS changes
- Better error handling
- Travis CI and AppVeyor integration
- Simple audio handling
- Changelogs in the dist folder of master
- Releases branch for builds from Travis
- Python 3.6 support
- 1 more example, bringing the total to 8

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.2.0

4.1.15 v0.1.0

Cython update, where everything is cythonized. First big update.

Features:

- Much more optimized rendering with Cython
- A new example
- Primitives
- Scaling
- Tutorials
- New color theme for documentation
- Timer decorator
- Non-interactive mode
- Frustrum culling
- Overall optimization

Notes:

- The FPS config will not have a change due to the inability of cyclic imports in Cython.
- You can see the c code used in Cython in the src folder.
- When installing with `setup.py`, you can set the environment variable `a` to anything but an empty string, this will disable recreating the c files. For example:

```
> set a=1
> python setup.py install
```

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.1.0

4.1.16 v0.0.5

Transform updates, with new features extending GameObject positioning.

Features:

- Local transform
- Quaternion
- Better example loader
- Primitive objects in files
- Fixed jittering when colliding from an angle
- Enabled friction (I don't know when it was turned off)
- Remove scenes from SceneManager
- Vector division

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.0.5

4.1.17 v0.0.4

Physics update.

New features:

- Rigidbodies
- Gravity
- Forces
- Optimized collision
- Better documentation
- Primitive meshes
- PyUnity mesh files that are optimized for fast loading
- Pushed GLUT to the end of the list so that it has the least priority
- Fixed window loading
- Auto README.md updater

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.0.4

4.1.18 v0.0.3

More basic things added.

Features:

- Examples (5 of them!)
- Basic physics components

- Lighting
- Better window selection
- More debug options
- File loader for .obj files

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.0.3

4.1.19 v0.0.2

First proper release (v0.0.1 was lost).

Features:

- Documentation
- Meshes

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.0.2

4.2 Tutorials

Here are some tutorials to get you started in using PyUnity. They need no prior knowledge about Unity, but they do require you to be comfortable with using Python.

4.2.1 Tutorial 1: The Basics

Table of Contents

- *What is PyUnity?*
- *Basic concepts*
- *Transforms*
- *Code*
- *Rotation*

In this tutorial you will be learning the basics to using PyUnity, and understanding some key concepts.

What is PyUnity?

PyUnity is a Python implementation of the UnityEngine, which was originally written in C++. PyUnity has been modified to be as easy to use in Python as possible, without sacrificing the flexibility and the versatilility of Unity.

Basic concepts

In PyUnity, everything belongs to a `GameObject`. A GameObject is a named object that has lots of Components on it, each affecting the GameObject and other GameObjects. Components are Python objects that do a specific job, like rendering an object or deleting other GameObjects.

Transforms

Each GameObject has a special component called a `Transform`. A Transform holds information about the GameObject's position, rotation and scale.

A Transform also manages the hierarchy system in PyUnity. Each transforms can have multiple children, which are all Transforms attached to the children GameObjects. All transforms will have a `localPosition`, `localRotation` and `localScale`, which are all relative to their parent. In addition, all Transforms will have a `position`, `rotation` and `scale` property which is measured in global space.

For example, if there is a Transform at 1 unit up from the origin, and its child had a `localPosition` of 1 unit right, then the child would have a `position` of 1 unit up and 1 unit to the right.

Code

All of that has now been established, so let's start programing it all! To start, we need to import PyUnity.

```
>>> from pyunity import *
Loaded config
Trying GLFW as a window provider
GLFW doesn't work, trying PySDL2
Trying PySDL2 as a window provider
Using window provider PySDL2
Loaded PyUnity version 0.4.0
```

Note: the output beneath the import is debug info, you can turn it off with the environment variable `PYUNITY_DEBUG_INFO` set to `"0"`. For example:

```
>>> import os
>>> os.environ["PYUNITY_DEBUG_INFO"] = "0"
>>> from pyunity import *
>>> # No output
```

Now we've loaded the module, we can start creating our GameObjects. To create a GameObject, use the `GameObject` class:

```
>>> root = GameObject("Root")
```

Then we can change its position by accessing its transform. All GameObjects have references to their transform by the `transform` attribute, and all components have a reference to the GameObject and the Transform that they belong to, by the `gameObject` and `transform` attributes. Here's how to make the GameObject positioned 1 unit up, 2 units to the right and 3 units forward:

```
>>> root.transform.localPosition = Vector3(2, 1, 3)
```

A `Vector3` is just a way to represent a 3D vector. In PyUnity the coordinate system is a left-hand Y-axis up system, which is essentially what OpenGL uses, but with the Z-axis flipped.

Then to add a child to the GameObject, specify the parent GameObject as the second argument:

```
>>> child1 = GameObject("Child1", root)
>>> child2 = GameObject("Child2", root)
```

Note: Accessing the `localPosition`, `localRotation` and `localScale` attributes are faster than using the `position`, `rotation` and `scale` properties. Use the local attributes whenever you can.

Rotation

Rotation is measured in Quaternions. Do not worry about these, because they use some very complex maths. All you need to know are these methods:

1. To make a Quaternion that represents no rotation, use `Quaternion.identity()`. This just means no rotation.
2. To make a Quaternion from an axis and angle, use the `Quaternion.FromAxis()` method. What this does is it creates a Quaternion that represents a rotation around an axis clockwise, by `angle` degrees. The axis does not need to be normalized.
3. To make a Quaternion from Euler angles, use `Quaternion.Euler`. This creates a Quaternion from Euler angles, where it is rotated on the Z-axis first, then the X-axis, and finally the Y-axis.

Transforms also have `localEulerAngles` and `eulerAngles` properties, which just represent the Euler angles of the rotation Quaternions. If you don't know how Quaternions work, only use the `eulerAngles` property.

In the next tutorial, we'll be covering how to render things and use a Scene.

4.2.2 Tutorial 2: Rendering in Scenes

Table of Contents

- *Scenes*
- *Meshes*
- *The MeshRenderer*
- *Debugging*

Last tutorial we covered some basic concepts on GameObjects and Transforms, and this time we'll be looking at how to render things in a window.

Scenes

A `Scene` is like a page to draw on: you can add things, remove things and change things. To create a scene, you can call `SceneManager.AddScene`:

```
>>> scene = SceneManager.AddScene("Scene")
```

In your newly created scene, you have 2 GameObjects: a Main Camera, and a Light. These two things can be moved around like normal GameObjects.

Next, let's move the camera back 10 units:

```
>>> scene.mainCamera.transform.localPosition = Vector3(0, 0, -10)
```

`scene.mainCamera` references the `Camera` Component on the Main Camera, so we can access the Transform by using its `transform` attribute.

Meshes

To render anything, we need a model of it. Let's say we want to create a cube. Then we need a model of a cube, or what's called a mesh. Meshes have 4 pieces of data: the vertices (or points), the faces, the normals and the texture coordinates. Normals are just vectors saying which way the face is pointing, and texture coordinates are coordinates to represent how an image is displayed on the surface of a mesh.

For a simple object like a cube, we don't need to create our own mesh. Fortunately there is a method called `Mesh.cube` which creates a cube for us. Here it is:

```
>>> cubeMesh = Mesh.cube(2)
```

The `2` means to create a cube with side lengths of 2. Then, to render this mesh, we need a new Component.

The MeshRenderer

The MeshRenderer is a Component that can render a mesh in the scene. To add a new Component, we can use a method called `AddComponent`:

```
>>> cube = GameObject("cube")
>>> renderer = cube.AddComponent(MeshRenderer)
```

Now we can give our renderer the cube mesh from before.

```
>>> renderer.mesh = cubeMesh
```

Finally, we need a `Material` to use. To create a Material, we need to specify a color in `RGB`.

```
>>> renderer.mat = Material(RGB(255, 0, 0))
```

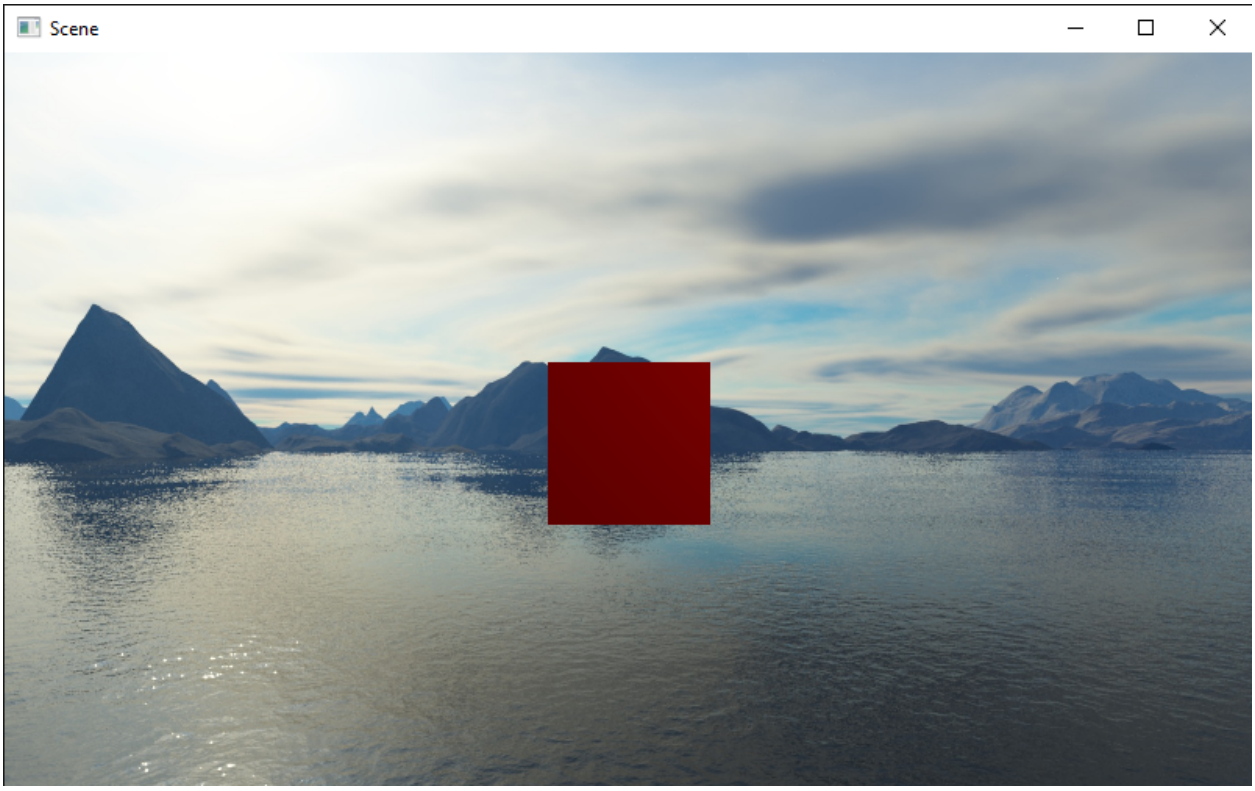
Here I used a red material. Finally we need to add the cube to our scene, otherwise we can't see it in the window:

```
>>> scene.Add(cube)
```

The full code:

```
>>> from pyunity import *
Loaded config
Trying GLFW as a window provider
GLFW doesn't work, trying PySDL2
Trying PySDL2 as a window provider
Using window provider PySDL2
Loaded PyUnity version 0.4.0
>>> scene = SceneManager.AddScene("Scene")
>>> scene.mainCamera.transform.localPosition = Vector3(0, 0, -10)
>>> cubeMesh = Mesh.cube(2)
>>> cube = GameObject("Cube")
>>> renderer = cube.AddComponent(MeshRenderer)
>>> renderer.mesh = cubeMesh
>>> renderer.mat = Material(RGB(255, 0, 0))
>>> scene.Add(cube)
```

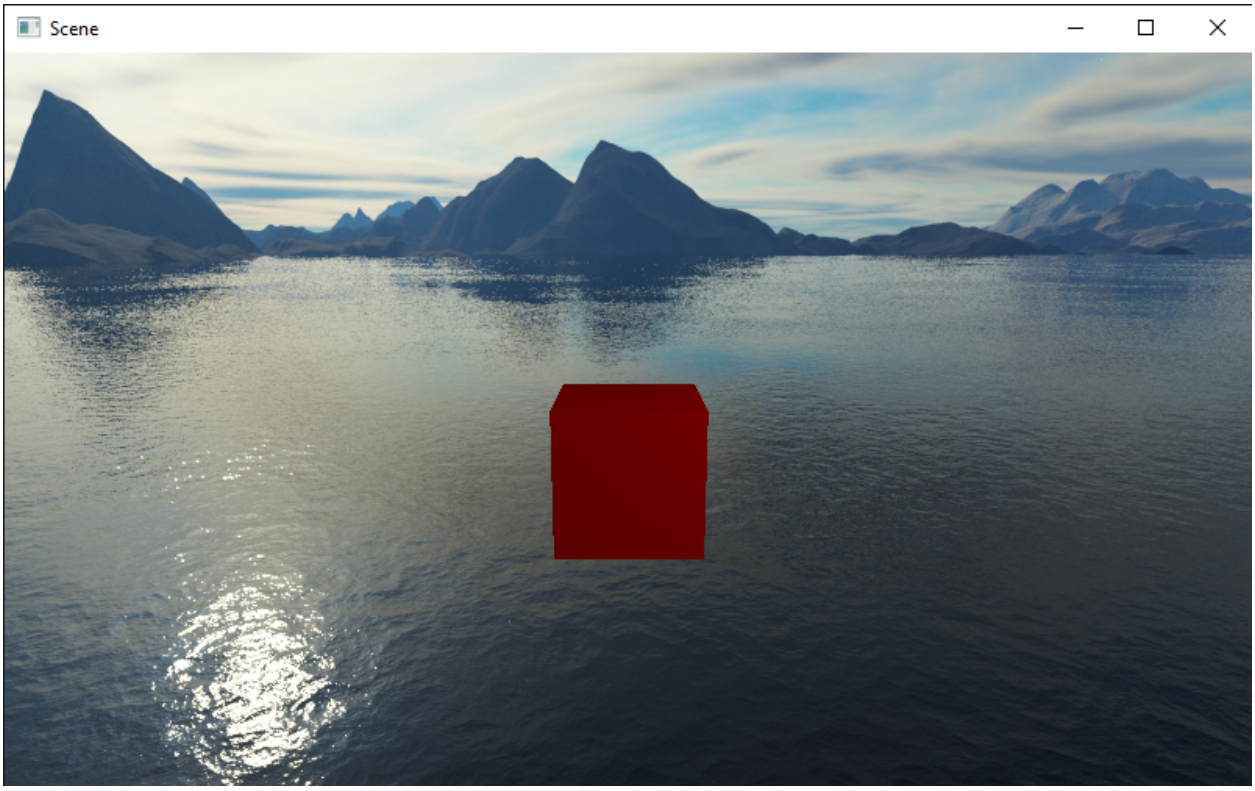
Then, to run our scene, we use `SceneManager.LoadScene(scene)`. And now we have a cube:



To see it better, let's move the camera up a bit and tilt it downwards. Replace the third line with this:

```
>>> scene.mainCamera.transform.localPosition = Vector3(0, 3, -10)
>>> scene.mainCamera.transform.localEulerAngles = Vector3(15, 0, 0)
```

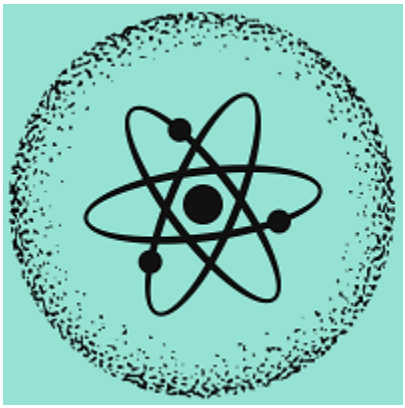
Now we can see it better:



Let's say we want to place an image onto the cube. To do this, we need to change the Material and add a `Texture2D`.

```
>>> renderer.mat = Material(RGB(255, 255, 255), Texture2D("pyunity.png"))
```

Place `pyunity.png` in the same folder as your script and run the code. Here is the image for reference:



And here is the complete code:

```
from pyunity import *

scene = SceneManager.AddScene("Scene")
scene.mainCamera.transform.localPosition = Vector3(0, 0, -10)

cubeMesh = Mesh.cube(2)
cube = GameObject("Cube")
renderer = cube.AddComponent(MeshRenderer)
```

(continues on next page)

(continued from previous page)

```
renderer.mesh = cubeMesh
renderer.mat = Material(RGB(255, 0, 0), Texture2D("pyunity.png"))
scene.Add(cube)

SceneManager.LoadScene(scene)
```

Debugging

If you want to see what you've done already, then you can use a number of debugging methods. The first is to call `scene.List()`:

```
>>> scene.List()
/Main Camera
/Light
/Cube
```

This lists all the Gameobjects in the scene. Then, let's check the cube's components:

```
>>> cube.components
[<Transform position=Vector3(0, 0, 0) rotation=Quaternion(1, 0, 0, 0) scale=Vector3(1,
↪ 1, 1) path="/Cube">, <pyunity.core.MeshRenderer object at 0x0B170CA0>]
```

Finally, let's check the Main Camera's transform.

```
>>> scene.mainCamera.transform
<Transform position=Vector3(0, 3, -10) rotation=Quaternion(0.9914448613738104, 0.
↪13052619222005157, 0.0, 0.0) scale=Vector3(1, 1, 1) path="/Main Camera">
```

Next tutorial, we'll be covering scripts and "Behaviour"s.

4.2.3 Tutorial 3: Scripts and Behaviours

Table of Contents

- *Behaviours*
- *Behaviours vs Components*
- *Examples*

Last tutorial we covered rendering meshes. In this tutorial we will be seeing how to make 2 GameObjects interact with each other.

Behaviours

A `Behaviour` is a Component that you can create yourself. To create a Behaviour, subclass from it:

```
>>> class MyBehaviour(Behaviour):
...     pass
```

In this case the Behaviour does nothing. To make it do something, use the Update function:

```
>>> class Rotator(Behaviour):
...     def Update(self, dt):
...         self.transform.localEulerAngles += Vector3(0, 90, 0) * dt
```

What this does is it rotates the GameObject that the Behaviour is on by 90 degrees each second around the y-axis. The `Update` function takes 1 argument, `dt`, which is how many seconds have passed since the last frame.

Behaviours vs Components

Look at this watered-down version of the `Component` class:

```
class Component:
    def __init__(self):
        self.gameObject = None
        self.transform = None

    def GetComponent(self, component):
        return self.gameObject.GetComponent(component)

    def AddComponent(self, component):
        return self.gameObject.AddComponent(component)
```

A Component has 2 attributes: `gameObject` and `transform`. This is set whenever the Component is added to a GameObject. A Behaviour is subclassed from a Component and so has the same attributes. Each frame, the Scene will call the `Update` function on all Behaviours, passing the time since the last frame in seconds.

When you want to do something at the start of the Scene, use the `Start` function. That will be called right at the start of the scene, when `scene.Run()` is called.

```
>>> class MyBehaviour(Behaviour):
...     def Start(self):
...         self.a = 0
...     def Update(self, dt):
...         print(self.a)
...         self.a += dt
```

The example above will print in seconds how long it had been since the start of the Scene. Note that the order in which all Behaviours' `Start` functions will be the orders of the GameObjects.

With this, you can create all sorts of Components, and because Behaviour is subclassed from Component, you can add a Behaviour to a GameObject with `AddComponent`.

Examples

This creates a spinning cube:

```
>>> class Rotator(Behaviour):
...     def Update(self, dt):
...         self.transform.localEulerAngles += Vector3(0, 90, 135) * dt
...
>>> scene = SceneManager.AddScene("Scene")
>>> cube = GameObject("Cube")
>>> renderer = cube.AddComponent(MeshRenderer)
>>> renderer.mesh = Mesh.cube(2)
>>> renderer.mat = Material(RGB(255, 0, 0))
```

(continues on next page)

(continued from previous page)

```
>>> cube.AddComponent(Rotator)
>>> scene.Add(cube)
>>> scene.Run()
```

This is a debugging Behaviour, which prints out the change in position, rotation and scale each 10 frames:

```
class Debugger(Behaviour):
    lastPos = Vector3.zero()
    lastRot = Quaternion.identity()
    lastScl = Vector3.one()
    a = 0
    def Update(self, dt):
        self.a += 1
        if self.a == 10:
            print(self.transform.position - self.lastPos)
            print(self.transform.rotation.conjugate * self.lastRot)
            print(self.transform.scale / self.lastScl)
            self.a = 0
```

Note that the printed output for non-moving things would be as so:

```
Vector3(0, 0, 0)
Quaternion(1, 0, 0, 0)
Vector3(1, 1, 1)
Vector3(0, 0, 0)
Quaternion(1, 0, 0, 0)
Vector3(1, 1, 1)
Vector3(0, 0, 0)
Quaternion(1, 0, 0, 0)
Vector3(1, 1, 1)
...
```

This means no rotation, position or scale change. It will break when you set the scale to `Vector3(0, 0, 0)`.

In the next tutorial we'll be looking at 2D development.

4.2.4 Tutorial 4: 2D

Table of Contents

- *Data types*
  - *RectOffset*
  - *RectTransform*
  - *Image2D*
  - *Canvas*
- *Code*
  - *Interaction*
- *Anchors*

This tutorial we will be introducing many new components, namely the `RectTransform` and the `Image2D`. There is more to 2D than this, but most of the tutorial is quite dense in new ideas.

Data types

To facilitate positioning objects, we are going to use `RectAnchors` and `RectOffset`. They both subclass `RectData`, which means they have two properties: `min` and `max`. They are both of type `Vector2`. For now, let's ignore the `RectAnchors`.

RectOffset

By ignoring `RectAnchors` we can simplify our offset to a literal rectangle. The `min` value specifies the top left corner of the rectangle, and the `max` value specifies the bottom right corner. In PyUnity, the X axis goes left to right and the Y axis goes top to bottom.

For example, a rect that is 100 pixels by 150 pixels, with a top left corner of (50, 75) would be like this:

```
>>> offset = RectOffset(
...     Vector2(50, 75),
...     Vector2(150, 225) # 100 + 50 and 150 + 75
... )
```

RectTransform

A `RectTransform` has 5 notable properties: `parent`, `anchors`, `offset`, `rotation` and `pivot`. `parent` is a read-only property, which gets the `RectTransform` of its parent, if it has one. `rotation` is a float measured in degrees, and `pivot` is a point between (0.0, 0.0) and (1.0, 1.0) which defines the rotation point.

Image2D

A `RectTransform` can't really do much on its own, so we'll look at the `Image2D` component. This renders a texture in the rect that is defined from the `RectTransform`. If you read tutorial 2, you may have used the `Texture2D` class. Here we can do the exact same:

```
>>> gameObject = GameObject("Image")
>>> transform = gameObject.AddComponent(RectTransform)
>>> transform.offset = RectOffset.Rectangle(
...     Vector2(100, 100), center=Vector2(125, 75))
>>> img = gameObject.AddComponent(Image2D)
>>> img.texture = Texture2D("python.png")
```

Canvas

All 2D renderers must be a descendant of a `Canvas` element, which can customize the rendering of 2D components. We don't need to worry about that too much, except that if we were to create an `Image2D` we must make it as a child or descendant of our canvas.

```
canvas = GameObject("Canvas")
canvas.AddComponent(Canvas)
img = GameObject("Image", canvas)
# And so on...
```

Here the second argument to the `GameObject` constructor specifies its parent, which must be a `GameObject`.

Code

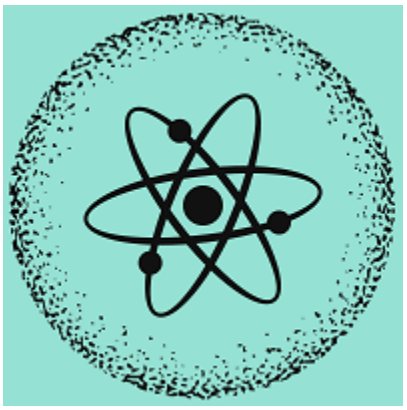
```
from pyunity import *

scene = SceneManager.AddScene("Scene")
canvas = GameObject("Canvas")
canvas.AddComponent(Canvas)
scene.Add(canvas)

gameObject = GameObject("Image", canvas)
transform = gameObject.AddComponent(RectTransform)
transform.offset = RectOffset.Rectangle(
    Vector2(100, 100), center=Vector2(125, 75))
img = gameObject.AddComponent(Image2D)
img.texture = Texture2D("pyunity.png")
scene.Add(gameObject)

SceneManager.LoadScene(scene)
```

PyUnity image:



This is the result:

Interaction

The easiest way to create an interactable image is to use the `Button` class. This will trigger whenever any part of the rect is clicked on. Here is an example:

```
def callback():
    print("Clicked")

# Same canvas and image code as above
...
button = gameObject.AddComponent(Button)
button.callback = callback
```

If you check the docs for the `Button` class, you can see two more attributes: `state` and `button`. This specifies what state and which button must be pressed for the callback to trigger.

If you would like more control over the button, using a Behaviour is easier as it can interact easily with other GameObjects and is created on a per-component basis. However, if you would like more interaction with the mouse, here is a method:

```
class HoverUpdater(Behaviour, GuiComponent):
    def HoverUpdate(self):
        print("Hovering over component")

# Same canvas and image code as above
...
gameObject.AddComponent(HoverUpdater)
```

The `GuiComponent` class defines an abstract method called `HoverUpdate` which is called whenever the mouse

is hovering over a component. This method will be called exactly once per canvas in a single GuiComponent each frame. In fact, this is how the `Button` class is implemented.

Anchors

For a 2D rect to scale with the window, we can use the `anchors` property of the `RectTransform`. This has two values like the `offset`, a `min` and a `max`. These two values are between `Vector2(0, 0)` and `Vector2(1, 1)`, where 0 and 1 represent the left and right of the window, or the top and bottom of the window. The offsets are applied where the anchors are.

The easiest way to understand this is when the anchors are a single point. For example, the default anchors are `RectAnchors(Vector2(0, 0), Vector2(0, 0))`. This means both points of the anchors are at `Vector2(0, 0)` so all offsets are calculated from the top left.

If we wanted our rect to be centered in the middle at all times, or be offset from the middle, we can set the anchors to be at `Vector2(0.5, 0.5)`. Likewise, if we wanted our rect to be at the bottom right, we can use `Vector2(1, 1)`.

This applies with two anchors: if we wanted our rect to be 50px away from the edges of the window, we would use anchors of `RectAnchors(Vector2(0, 0), Vector2(1, 1))` and offset of `RectOffset(Vector2(50, 50), Vector2(-50, -50))`. This is how we can control the scaling of a rect with respect to the window size.

This tutorial was quite code-heavy, and it is not quite complete. If you are confused, please join our discord support server at https://discord.com/zTn48BEbF9.

4.3 Links

Here are some links to websites about the PyUnity project:

https://github.com/pyunity/pyunity - GitHub repository

https://pypi.org/project/pyunity - PyPi page

https://discord.gg/zTn48BEbF9 - Discord server

4.4 License

MIT License

Copyright (c) 2020-2021 Ray Chen

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

4.5 API Documentation

Information on specific functions, classes, and methods.

4.5.1 Subpackages

pyunity.physics package

Submodules

pyunity.physics.config module

pyunity.physics.config.**gravity = Vector3(0, -9.81, 0)**
Gravitational constant (9.81 m/s^2)

pyunity.physics.core module

Core classes of the PyUnity physics engine.

pyunity.physics.core.**Infinity = inf**
A representation of infinity

class pyunity.physics.core.**PhysicMaterial**(*restitution=0.75*, *friction=1*, *immutable=False*)
Bases: object

Class to store data on a collider's material. :param restitution: Bounciness of the material :type restitution: float :param friction: Friction of the material :type friction: float

restitution
Bounciness of the material

Type float

friction
Friction of the material

Type float

combine
Combining function. -1 means minimum, 0 means average, and 1 means maximum

Type int

exception(**args*, ***kwargs*)

class pyunity.physics.core.**Manifold**(*a*, *b*, *normal*, *penetration*)
Bases: object

Class to store collision data. :param a: The first collider :type a: Collider :param b: The second collider :type b: Collider :param normal: The collision normal :type normal: Vector3 :param penetration: How much the two colliders overlap :type penetration: float

class pyunity.physics.core.**Collider**(*transform*)
Bases: *pyunity.core.Component*

Collider base class.

pos

collidingWith(*other*)

class pyunity.physics.core.**SphereCollider**(*transform*)
Bases: *pyunity.physics.core.Collider*

A spherical collider that cannot be deformed.

radius
The radius of the SphereCollider

Type *Vector3*

collidingWith(*other*)
Check to see if the collider is colliding with another collider.

Parameters **other** (Collider) – Other collider to check against

Returns Collision data

Return type *Manifold* or None

Notes

To check against another SphereCollider, the distance and the sum of the radii is checked. To check against an AABBoxColider, the check is as follows: 1. The sphere's center is checked to see if it

is inside the AABB.

1. If it is, then the two are colliding.
2. If it isn't, then a copy of the position is clamped to the AABB's bounds.
3. Finally, the distance between the clamped position and the original position is measured.
4. If the distance is bigger than the sphere's radius, then the two are colliding.
5. If not, then they aren't colliding.

class pyunity.physics.core.**AABBoxCollider**(*transform*)
Bases: *pyunity.physics.core.Collider*

An axis-aligned box collider that cannot be deformed.

size
The size of the AABBoxCollider.

Type *Vector3*

collidingWith(*other*)
Check to see if the collider is colliding with another collider.

Parameters **other** (Collider) – Other collider to check against

Returns Collision data

Return type *Manifold* or None

Notes

To check against another AABBoxCollider, the corners are checked to see if they are inside the other collider.

To check against a SphereCollider, the check is as follows: 1. The sphere's center is checked to see if it

is inside the AABB.

1. If it is, then the two are colliding.
2. If it isn't, then a copy of the position is clamped to the AABB's bounds.
3. Finally, the distance between the clamped position and the original position is measured.
4. If the distance is bigger than the sphere's radius, then the two are colliding.
5. If not, then they aren't colliding.

class pyunity.physics.core.**Rigidbody**(*transform*, *dummy=False*)

Bases: *pyunity.core.Component*

Class to let a GameObject follow physics rules.

mass

Mass of the Rigidbody. Defaults to 100

Type int or float

velocity

Velocity of the Rigidbody

Type *Vector3*

physicMaterial

Physics material of the Rigidbody

Type *PhysicMaterial*

Move(*dt*)

Moves all colliders on the GameObject by the Rigidbody's velocity times the delta time. :param dt: Time to simulate movement by :type dt: float

AddForce(*force*)

Apply a force to the center of the Rigidbody. :param force: Force to apply :type force: Vector3

Notes

A force is a gradual change in velocity, whereas an impulse is just a jump in velocity.

AddImpulse(*impulse*)

Apply an impulse to the center of the Rigidbody. :param impulse: Impulse to apply :type impulse: Vector3

Notes

A force is a gradual change in velocity, whereas an impulse is just a jump in velocity.

class pyunity.physics.core.**CollManager**

Bases: object

Manages the collisions between all colliders. .. attribute:: rigidbodies

Dictionary of rigidbodies andthe colliders on the gameObject that the Rigidbody belongs to

type dict

dummyRigidbody

A dummy rigidbody used when a GameObject has colliders but no rigidbody. It has infinite mass

Type *Rigidbody*

AddPhysicsInfo(*scene*)
Get all colliders and rigidbodies from a specified scene. This overwrites the collider and rigidbody lists, and so can be called whenever a new collider or rigidbody is added or removed. :param scene: Scene to search for physics info :type scene: Scene

Notes

This function will overwrite the pre-existing dictionary of rigidbodies. When there are colliders but no rigidbody is on the GameObject, then they are placed in the dictionary with a dummy Rigidbody that has infinite mass and a default physic material. Thus, they cannot move.

GetRestitution(*a*, *b*)
Get the restitution needed for two rigidbodies, based on their combine function :param a: Rigidbody 1 :type a: Rigidbody :param b: Rigidbody 2 :type b: Rigidbody

Returns Restitution

Return type float

CheckCollisions()
Goes through every pair exactly once, then checks their collisions and resolves them.

ResolveCollision(*m*, *rbA*, *rbB*)

correct_inf(*a*, *b*, *correction*, *target*)

Step(*dt*)
Steps through the simulation at a given delta time. :param dt: Delta time to step :type dt: float

Notes

The simulation is stepped 10 times manually by the scene, so it is more precise.

Module contents

A basic 3D Physics engine that uses similar concepts to the Unity Engine itself. Only supports non-rotated colliders.

To create an immoveable object, use math.inf or the provided `Infinity` variable. This will make the object not be able to move, unless you set an initial velocity. Then, the collider will either push everything it collides with, or bounces it back at twice the speed.

Example

```
>>> cube = GameObject("Cube")
>>> collider = cube.AddComponent(AABBoxCollider)
>>> collider.velocity = Vector3.right()
```

Configuration

If you want to change some configurations, import the config file like so:

```
>>> from pyunity.physics import config
```

Inside the config file there are some configurations:

- `gravity` is the gravity of the whole system. It only affects Rigidbodies that have `Rigidbody.gravity` set to True.

pyunity.scenes package

Submodules

pyunity.scenes.scene module

Class to load, render and manage GameObjects and their various components.

You should never use the *Scene* class directly, instead, only use the `SceneManager` class.

class pyunity.scenes.scene.**Scene**(*name*)
Bases: `object`

Class to hold all of the GameObjects, and to run the whole scene.

Parameters **name** (*str*) – Name of the scene

Notes

Create a scene using the SceneManager, and don't create a scene directly using this class.

static Bare(*name*)
Create a bare scene.

Parameters **name** (*str*) – Name of the scene

Returns A bare scene with no GameObjects

Return type *Scene*

rootGameObjects
All GameObjects which have no parent

Add(*gameObject*)
Add a GameObject to the scene.

Parameters **gameObject** (`GameObject`) – The GameObject to add.

Remove(*gameObject*)
Remove a GameObject from the scene.

Parameters **gameObject** (`GameObject`) – GameObject to remove.

Raises `PyUnityException` – If the specified GameObject is not part of the Scene.

Has(*gameObject*)
Check if a GameObject is in the scene.

Parameters **gameObject** (`GameObject`) – Query GameObject

Returns If the GameObject exists in the scene

Return type bool

RegisterLight(*light*)
Register a light for the scene.

Parameters **light** (`Light`) – Light component to register

List()
Lists all the GameObjects currently in the scene.

FindGameObjectsByName(*name*)
Finds all GameObjects matching the specified name.

Parameters **name** (*str*) – Name of the GameObject

Returns List of the matching GameObjects

Return type list

FindGameObjectsByTagName(*name*)
Finds all GameObjects with the specified tag name.

Parameters **name** (*str*) – Name of the tag

Returns List of matching GameObjects

Return type list

Raises GameObjectException – When there is no tag named *name*

FindGameObjectsByTagNumber(*num*)
Gets all GameObjects with a tag of tag *num*.

Parameters **num** (*int*) – Index of the tag

Returns List of matching GameObjects

Return type list

Raises GameObjectException – If there is no tag with specified index.

FindComponentByType(*component*)
Finds the first matching Component that is in the Scene.

Parameters **component** (*type*) – Component type

Returns The matching Component

Return type *Component*

Raises ComponentException – If the component is not found

FindComponentsByType(*component*)
Finds all matching Components that are in the Scene.

Parameters **component** (*type*) – Component type

Returns List of the matching Components

Return type list

inside_frustrum(*renderer*)
Check if the renderer's mesh can be seen by the main camera.

Parameters **renderer** (MeshRenderer) – Renderer to test

Returns If the mesh can be seen

Return type bool

start_scripts()
Start the scripts in the Scene.

Start()
Start the internal parts of the Scene.

update_scripts()
Updates all scripts in the scene.

no_interactive()
Run scene without rendering.

update()
Updating function to pass to the window provider.

Render()
Call the appropriate rendering functions of the Main Camera.

clean_up()
Called when the scene finishes running, or stops running.

pyunity.scenes.sceneManager module

Module that manages creation and deletion of Scenes.

pyunity.scenes.sceneManager.**AddScene**(*sceneName*)
Add a scene to the SceneManager. Pass in a scene name to create a scene.

Parameters **sceneName** (*str*) – Name of the scene

Returns Newly created scene

Return type *Scene*

Raises PyUnityException – If there already exists a scene called *sceneName*

pyunity.scenes.sceneManager.**AddBareScene**(*sceneName*)
Add a scene to the SceneManager. Pass in a scene name to create a scene.

Parameters **sceneName** (*str*) – Name of the scene

Returns Newly created scene

Return type *Scene*

Raises PyUnityException – If there already exists a scene called *sceneName*

pyunity.scenes.sceneManager.**GetSceneByIndex**(*index*)
Get a scene by its index.

Parameters **index** (*int*) – Index of the scene

Returns Specified scene at index *index*

Return type *Scene*

Raises IndexError – If there is no scene at the specified index

pyunity.scenes.sceneManager.**GetSceneByName**(*name*)
Get a scene by its name.

Parameters **name** (*str*) – Name of the scene

Returns Specified scene with name of *name*

Return type *Scene*

Raises KeyError – If there is no scene called *name*

pyunity.scenes.sceneManager.**RemoveScene**(*scene*)
Removes a scene from the SceneManager.

Parameters **scene** (Scene) – Scene to remove

Raises

- TypeError – If the provided scene is not type Scene
- PyUnityException – If the scene is not part of the SceneManager

pyunity.scenes.sceneManager.**RemoveAllScenes**()
Removes all scenes from the SceneManager.

pyunity.scenes.sceneManager.**LoadSceneByName**(*name*)
Loads a scene by its name.

Parameters **name** (*str*) – Name of the scene

Raises

- TypeError – When the provided name is not a string
- PyUnityException – When there is no scene named name

pyunity.scenes.sceneManager.**LoadSceneByIndex**(*index*)
Loads a scene by its index of when it was added to the SceneManager.

Parameters **index** (*int*) – Index of the scene

Raises

- TypeError – When the provided index is not an integer
- PyUnityException – When there is no scene at index index

pyunity.scenes.sceneManager.**LoadScene**(*scene*)
Load a scene by a reference.

Parameters **scene** (Scene) – Scene to be loaded

Raises

- TypeError – When the scene is not of type *Scene*
- PyUnityException – When the scene is not part of the SceneManager. This is checked because the SceneManager has to make some checks before the scene can be run.

pyunity.scenes.sceneManager.**CurrentScene**()
Gets the current scene being run

Module contents

Module to create and load Scenes.

pyunity.values package

Submodules

pyunity.values.abc module

exception pyunity.values.abc.**ABCException**
Bases: Exception

exception pyunity.values.abc.**ABCMessage**
Bases: *pyunity.values.abc.ABCException*

class pyunity.values.abc.**abstractmethod**(*func*)
Bases: object

static getargs(*func*)

class pyunity.values.abc.**abstractproperty**(*func*)
Bases: *pyunity.values.abc.abstractmethod*

class pyunity.values.abc.**ABCMeta**(*fullname*, *bases*, *attrs*, *message=None*)
Bases: type

pyunity.values.other module

class pyunity.values.other.**Clock**
Bases: object

fps

Start(*fps=None*)

Maintain()

class pyunity.values.other.**ImmutableStruct**
Bases: type

pyunity.values.quaternion module

Class to represent a rotation in 3D space.

class pyunity.values.quaternion.**Quaternion**(*w*, *x*, *y*, *z*)
Bases: object

Class to represent a unit quaternion, also known as a versor.

Parameters

- **w** (*float*) – Real value of Quaternion
- **x** (*float*) – x coordinate of Quaternion
- **y** (*float*) – y coordinate of Quaternion
- **z** (*float*) – z coordinate of Quaternion

abs_diff(*other*)

copy()
Deep copy of the Quaternion.

Returns A deep copy

Return type *Quaternion*

normalized()
A normalized Quaternion, for rotations. If the length is 0, then the identity quaternion is returned.

Returns A unit quaternion

Return type *Quaternion*

conjugate
The conjugate of a unit quaternion

RotateVector(*vector*)
Rotate a vector by the quaternion

static FromAxis(*angle*, *a*)
Create a quaternion from an angle and an axis.

Parameters

- **angle** (*float*) – Angle to rotate
- **a** (Vector3) – Axis to rotate about

static Between(*v1*, *v2*)

static FromDir(*v*)

angleAxisPair
Gets or sets the angle and axis pair. Tuple of form (angle, axis).

static Euler(*vector*)
Create a quaternion using Euler rotations.

Parameters **vector** (Vector3) – Euler rotations

Returns Generated quaternion

Return type *Quaternion*

eulerAngles
Gets or sets the Euler Angles of the quaternion

SetBackward(*value*)

static identity()
Identity quaternion representing no rotation

class pyunity.values.quaternion.**QuaternionDiff**(*w*, *x*, *y*, *z*)
Bases: object

pyunity.values.texture module

class pyunity.values.texture.**Material**(*color*, *texture=None*)
Bases: object

Class to hold data on a material.

color
An albedo tint.

Type *Color*

texture
A texture to map onto the mesh provided by a MeshRenderer

Type *Texture2D*

class pyunity.values.texture.**Color**
Bases: object

to_string()

static from_string(*string*)

class pyunity.values.texture.**RGB**(*r*, *g*, *b*)
Bases: *pyunity.values.texture.Color*

A class to represent an RGB color.

Parameters

- **r** (*int*) – Red value (0-255)
- **g** (*int*) – Green value (0-255)
- **b** (*int*) – Blue value (0-255)

to_rgb()

to_hsv()

static from_hsv(*h*, *s*, *v*)

class pyunity.values.texture.**HSV**(*h*, *s*, *v*)
Bases: *pyunity.values.texture.Color*

A class to represent a HSV color.

Parameters

- **h** (*int*) – Hue (0-360)
- **s** (*int*) – Saturation (0-100)
- **v** (*int*) – Value (0-100)

to_rgb()

to_hsv()

static from_rgb(*r*, *g*, *b*)

pyunity.values.vector module

pyunity.values.vector.**clamp**(*x*, *_min*, *_max*)

class pyunity.values.vector.**Vector**
Bases: object

abs()

length()

class pyunity.values.vector.**Vector2**(*x_or_list=None*, *y=None*)
Bases: *pyunity.values.vector.Vector*

copy()
Makes a copy of the Vector2

get_length_sqrd()
Gets the length of the vector squared. This is much faster than finding the length.

Returns The length of the vector squared

Return type float

length
Gets or sets the magnitude of the vector

normalized()
Get a normalized copy of the vector, or Vector2(0, 0) if the length is 0.

Returns A normalized vector

Return type *Vector2*

normalize()
Normalize the vector in place.

normalize_return_length()
Normalize the vector and return its length before the normalization

Returns The length before the normalization

Return type float

get_distance(*other*)
The distance between this vector and the other vector

Returns The distance

Return type float

get_dist_sqrd(*other*)
The distance between this vector and the other vector, squared. It is more efficient to call this than to call *get_distance* and square it.

Returns The squared distance

Return type float

int_tuple
Return the x, y and z values of this vector as ints

rounded
Return the x, y and z values of this vector rounded to the nearest integer

clamp(*min*, *max*)
Clamps a vector between two other vectors, resulting in the vector being as close to the edge of a bounding box created as possible.

Parameters

- **min** (Vector2) – Min vector
- **max** (Vector2) – Max vector

dot(*other*)
Dot product of two vectors.

Parameters **other** (Vector2) – Other vector

Returns Dot product of the two vectors

Return type float

cross(*other*)
Cross product of two vectors. In 2D this is a scalar.

Parameters **other** (Vector2) – Other vector

Returns Cross product of the two vectors

Return type float

static min(*a*, *b*)

static max(*a*, *b*)

static zero()
A vector of zero length

static one()
A vector of ones

static left()
Vector2 pointing in the negative x axis

static right()
Vector2 pointing in the postive x axis

static up()
Vector2 pointing in the postive y axis

static down()
Vector2 pointing in the negative y axis

class pyunity.values.vector.**Vector3**(*x_or_list=None*, *y=None*, *z=None*)
Bases: *pyunity.values.vector.Vector*

copy()
Makes a copy of the Vector3

Returns A shallow copy of the vector

Return type *Vector3*

get_length_sqrd()
Gets the length of the vector squared. This is much faster than finding the length.

Returns The length of the vector squared

Return type float

length
Gets or sets the magnitude of the vector

normalized()
Get a normalized copy of the vector, or Vector3(0, 0, 0) if the length is 0.

Returns A normalized vector

Return type *Vector3*

normalize()
Normalize the vector in place.

normalize_return_length()
Normalize the vector and return its length before the normalization

Returns The length before the normalization

Return type float

get_distance(*other*)
The distance between this vector and the other vector

Returns The distance

Return type float

get_dist_sqrd(*other*)
The distance between this vector and the other vector, squared. It is more efficient to call this than to call *get_distance* and square it.

Returns The squared distance

Return type float

int_tuple
Return the x, y and z values of this vector as ints

rounded
Return the x, y and z values of this vector rounded to the nearest integer

clamp(*min*, *max*)
Clamps a vector between two other vectors, resulting in the vector being as close to the edge of a bounding box created as possible.

Parameters

- **min** (Vector3) – Min vector
- **max** (Vector3) – Max vector

dot(*other*)
Dot product of two vectors.

Parameters **other** (Vector3) – Other vector

Returns Dot product of the two vectors

Return type float

cross(*other*)
Cross product of two vectors

Parameters **other** (Vector3) – Other vector

Returns Cross product of the two vectors

Return type *Vector3*

static min(*a*, *b*)

static max(*a*, *b*)

static zero()
A vector of zero length

static one()
A vector of ones

static forward()
Vector3 pointing in the positive z axis

static back()
Vector3 pointing in the negative z axis

static left()
Vector3 pointing in the negative x axis

static right()
Vector3 pointing in the postive x axis

static up()
Vector3 pointing in the postive y axis

static down()
Vector3 pointing in the negative y axis

Module contents

pyunity.window package

Submodules

pyunity.window.glfwWindow module

Class to create a window using GLFW.

class pyunity.window.glfwWindow.**Window**(*name*, *resize*)
Bases: *pyunity.window.ABCWindow*

A window provider that uses GLFW.

Raises PyUnityException – If the window creation fails

framebuffer_size_callback(*window*, *width*, *height*)

key_callback(*window*, *key*, *scancode*, *action*, *mods*)

mouse_callback(*window*, *button*, *action*, *mods*)

get_mouse(*mousecode*, *keystate*)

check_keys()

check_mouse()

get_key(*keycode*, *keystate*)

get_mouse_pos()

check_quit()

quit()

start(*update_func*)
Start the main loop of the window.

Parameters **update_func** (*function*) – The function that calls the OpenGL calls.

pyunity.window.glutWindow module

Class to create a window using FreeGLUT.

class pyunity.window.glutWindow.**Window**(*name*, *resize*)
Bases: *pyunity.window.ABCWindow*

A window provider that uses FreeGLUT.

start(*update_func*)
Start the main loop of the window.

Parameters **update_func** (*function*) – The function that calls the OpenGL calls.

schedule_update(*t*)
Starts the window refreshing.

display()
Function to render in the scene.

quit()

get_key(*keycode*, *keystate*)

get_mouse(*mousecode*, *keystate*)

pyunity.window.sdl2Window module

Class to create a window using PySDL2.

class pyunity.window.sdl2Window.**Window**(*name*, *resize*)
Bases: *pyunity.window.ABCWindow*

A window provider that uses PySDL2.

quit()

start(*update_func*)

process_keys(*events*)

process_mouse(*events*)

get_key(*keycode*, *keystate*)

get_mouse(*mousecode*, *keystate*)

get_mouse_pos()

pyunity.window.templateWindow module

Template window provider, use this for creating new window providers

class pyunity.window.templateWindow.**Window**(*name*, *resize*)
Bases: *pyunity.window.ABCWindow*

A template window provider.

quit()

start(*update_func*)
Start the main loop of the window.

Parameters **update_func** (*function*) – The function that calls the OpenGL calls.

Module contents

A module used to load the window providers.

The window is provided by one of three providers: GLFW, PySDL2 and GLUT. When you first import PyUnity, it checks to see if any of the three providers work. The testing order is as above, so GLUT is tested last.

To create your own provider, create a class that has the following methods:

- **__init__: initiate your window and** check to see if it works.
- **start: start the main loop in your** window. The first parameter is `update_func`, which is called when you want to do the OpenGL calls.

Check the source code of any of the window providers for an example. If you have a window provider, then please create a new pull request.

pyunity.window.**checkModule**(*name*)

pyunity.window.**glfwCheck**()
Checks to see if GLFW works

pyunity.window.**sdl2Check**()
Checks to see if PySDL2 works

pyunity.window.**glutCheck**()
Checks to see if GLUT works

pyunity.window.**GetWindowProvider**()
Gets an appropriate window provider to use

pyunity.window.**SetWindowProvider**(*name*)

pyunity.window.**CustomWindowProvider**(*cls*)

class pyunity.window.**ABCWindow**(*name*, *resize*)
Bases: object

get_mouse(*mousecode*, *keystate*)

get_key(*keycode*, *keystate*)

get_mouse_pos()

quit()

start(*update_func*)

4.5.2 Submodules

pyunity.audio module

Classes to manage the playback of audio. It uses the sdl2.sdlmixer library. A variable in the `config` module called `audio` will be set to `False` if the mixer module cannot be initialized.

pyunity.audio.**mixer**

class pyunity.audio.**AudioClip**(*path*)
Bases: object

Class to store information about an audio file.

path
Path to the file

Type str

music
Sound chunk that can be played with an SDL2 Mixer Channel. Only set when the AudioClip is played in an *AudioSource*.

Type sdl2.sdlmixer.mixer.Mix_Chunk

class pyunity.audio.**AudioSource**(*transform*)
Bases: *pyunity.core.Component*

Manages playback on an AudioSource.

clip
Clip to play. Best way to set the clip is to use the *SetClip()* function.

Type *AudioClip*

playOnStart
Whether it plays on start or not.

Type bool

loop
Whether it loops or not. This is not fully supported.

Type bool

SetClip(*clip*)
Sets a clip for the AudioSource to play.

Parameters **clip** (AudioClip) – AudioClip to play

Play()
Plays the AudioClip attached to the AudioSource.

Stop()
Stops playing the AudioClip attached to the AudioSource.

Pause()
Pauses the AudioClip attached to the AudioSource.

UnPause()
Unpauses the AudioClip attached to the AudioSource.

Playing
Gets if the AudioSource is playing.

class pyunity.audio.**AudioListener**(*transform*)
Bases: *pyunity.core.Component*

Class to receive audio events and to base spatial sound from. By default the Main Camera has an AudioListener, but you can also remove it and add a new one to another GameObject in a Scene. There can only be one AudioListener, otherwise sound is disabled.

Init()
Initializes the AudioListener.

DeInit()
Stops all AudioSources and frees memory that is used by the AudioClips.

pyunity.core module

Core classes for the PyUnity library.

This module has some key classes used throughout PyUnity, and have to be in the same file due to references both ways. Usually when you create a scene, you should never create Components directly, instead add them with AddComponent.

Example

To create a GameObject with 2 children, one of which has its own child, and all have MeshRenderers:

```
>>> from pyunity import * # Import
Loaded config
Trying GLFW as a window provider
GLFW doesn't work, trying PySDL2
Trying PySDL2 as a window provider
Using window provider PySDL2
Loaded PyUnity version 0.8.4
>>> mat = Material(RGB(255, 0, 0)) # Create a default material
>>> root = GameObject("Root") # Create a root GameObjects
>>> child1 = GameObject("Child1", root) # Create a child
>>> child1.transform.localPosition = Vector3(-2, 0, 0) # Move the child
>>> renderer = child1.AddComponent(MeshRenderer) # Add a renderer
>>> renderer.mat = mat # Add a material
>>> renderer.mesh = Mesh.cube(2) # Add a mesh
>>> child2 = GameObject("Child2", root) # Create another child
>>> renderer = child2.AddComponent(MeshRenderer) # Add a renderer
>>> renderer.mat = mat # Add a material
>>> renderer.mesh = Mesh.quad(1) # Add a mesh
>>> grandchild = GameObject("Grandchild", child2) # Add a grandchild
>>> grandchild.transform.localPosition = Vector3(0, 5, 0) # Move the grandchild
>>> renderer = grandchild.AddComponent(MeshRenderer) # Add a renderer
>>> renderer.mat = mat # Add a material
>>> renderer.mesh = Mesh.cube(3) # Add a mesh
>>> root.transform.List() # List all GameObjects
/Root
/Root/Child1
/Root/Child2
/Root/Child2/Grandchild
>>> child1.components # List child1's components
[<Transform position=Vector3(-2, 0, 0) rotation=Quaternion(1, 0, 0, 0)␣
↪scale=Vector3(1, 1, 1) path="/Root/Child1">, <pyunity.core.MeshRenderer object at␣
↪0x0A929460>]
>>> child2.transform.children # List child2's children
[<Transform position=Vector3(0, 5, 0) rotation=Quaternion(1, 0, 0, 0) scale=Vector3(1,
↪ 1, 1) path="/Root/Child2/Grandchild">]
```

class pyunity.core.**Tag**(*tagNumOrName*)

Bases: object

Class to group GameObjects together without referencing the tags.

Parameters **tagNumOrName** (*str or int*) – Name or index of the tag

Raises

- ValueError – If there is no tag name
- IndexError – If there is no tag at the provided index
- TypeError – If the argument is not a str or int

tagName

Tag name

Type str

tag

Tag index of the list of tags

Type int

classmethod AddTag(*name*)
Add a new tag to the tag list.

Parameters **name** (*str*) – Name of the tag

Returns The tag index

Return type int

class pyunity.core.**GameObject**(*name='GameObject'*, *parent=None*)
Bases: object

Class to create a GameObject, which is an object with components.

Parameters

- **name** (*str, optional*) – Name of GameObject
- **parent** (*GameObject or None*) – Parent of GameObject

name
Name of the GameObject

Type str

components
List of components

Type list

tag
Tag that the GameObject has (defaults to tag 0 or Default)

Type *Tag*

transform
Transform that belongs to the GameObject

Type *Transform*

static BareObject(*name='GameObject'*)
Create a bare GameObject with no components or attributes.

Parameters **name** (*str*) – Name of the GameObject

AddComponent(*componentClass*)
Adds a component to the GameObject. If it is a transform, set GameObject's transform to it.

Parameters **componentClass** (*Component*) – Component to add. Must inherit from *Component*

GetComponent(*componentClass*)
Gets a component from the GameObject. Will return first match. For all matches, use *GetComponents*.

Parameters **componentClass** (*Component*) – Component to get. Must inherit from *Component*

Returns The specified component, or *None* if the component is not found

Return type *Component* or None

RemoveComponent(*componentClass*)
Removes the first matching component from a GameObject.

Parameters **componentClass** (*type*) – Component to remove

Raises

- `ComponentException` – If the GameObject doesn't have the specified component
- `ComponentException` – If the specified component is a Transform

GetComponents(*componentClass*)
Gets all matching components from the GameObject.

Parameters **componentClass** (`Component`) – Component to get. Must inherit from *Component*

Returns A list of all matching components

Return type list

RemoveComponents(*componentClass*)
Removes all matching component from a GameObject.

Parameters **componentClass** (*type*) – Component to remove

Raises `ComponentException` – If the specified component is a Transform

class pyunity.core.**HideInInspector**(*type=None*, *default=None*)
Bases: `object`

An attribute that should be saved when saving a project, but not shown in the Inspector of the PyUnityEditor.

type
Type of the variable

Type type

default
Default value (will be set to the Behaviour)

Type Any

name
None

Type NoneType

class pyunity.core.**ShowInInspector**(*type=None*, *default=None*, *name=None*)
Bases: *pyunity.core.HideInInspector*

An attribute that should be saved when saving a project, and shown in the Inspector of the PyUnityEditor.

type
Type of the variable

Type type

default
Default value (will be set to the Behaviour)

Type Any

name
Alternate name shown in the Inspector

Type str

class pyunity.core.**Component**(*transform*, *is_dummy=False*)
Bases: `object`

Base class for built-in components.

gameObject
GameObject that the component belongs to.

Type *GameObject*

transform
Transform that the component belongs to.

Type *Transform*

AddComponent(*component*)
Calls *AddComponent* on the component's GameObject.

Parameters **component** (Component) – Component to add. Must inherit from *Component*

GetComponent(*component*)
Calls *GetComponent* on the component's GameObject.

Parameters **componentClass** (Component) – Component to get. Must inherit from *Component*

RemoveComponent(*component*)
Calls *RemoveComponent* on the component's GameObject.

Parameters **component** (Component) – Component to remove. Must inherit from *Component*

GetComponents(*component*)
Calls *GetComponents* on the component's GameObject.

Parameters **componentClass** (Component) – Component to get. Must inherit from *Component*

RemoveComponents(*component*)
Calls *RemoveComponents* on the component's GameObject.

Parameters **component** (Component) – Component to remove. Must inherit from *Component*

scene
Get either the scene of the GameObject or the current running scene.

class pyunity.core.**SingleComponent**(*transform*, *is_dummy=False*)
Bases: *pyunity.core.Component*

Represents a component that can be added only once.

class pyunity.core.**Transform**(*transform=None*)
Bases: *pyunity.core.SingleComponent*

Class to hold data about a GameObject's transformation.

gameObject
GameObject that the component belongs to.

Type *GameObject*

localPosition
Position of the Transform in local space.

Type *Vector3*

localRotation
Rotation of the Transform in local space.

Type *Quaternion*

localScale
Scale of the Transform in local space.

Type *Vector3*

parent
Parent of the Transform. The hierarchical tree is actually formed by the Transform, not the GameObject. Do not modify this attribute.

Type *Transform* or None

children
List of children

Type list

position
Position of the Transform in world space.

rotation
Rotation of the Transform in world space.

localEulerAngles
Rotation of the Transform in local space. It is measured in degrees around x, y, and z.

eulerAngles
Rotation of the Transform in world space. It is measured in degrees around x, y, and z.

scale
Scale of the Transform in world space.

ReparentTo(*parent*)
Reparent a Transform.

Parameters **parent** (Transform) – The parent to reparent to.

List()
Prints the Transform's full path from the root, then lists the children in alphabetical order. This results in a nice list of all GameObjects.

GetDescendants()
Iterate through all descedants of this Transform.

FullPath()
Gets the full path of the Transform.

Returns The full path of the Transform.

Return type str

LookAtTransform(*transform*)
Face towards another transform's position.

Parameters **transform** (Transform) – Transform to face towards

Notes

The rotation generated may not be upright, and to fix this just use `transform.rotation.eulerAngles *= Vector3(1, 1, 0)` which will remove the Z component of the Euler angles.

LookAtGameObject(*gameObject*)
Face towards another GameObject's position. See *Transform.LookAtTransform* for details.

Parameters **gameObject** (GameObject) – GameObject to face towards

LookAtPoint(*vec*)
Face towards a point. See *Transform.LookAtTransform* for details.

Parameters **vec** (Vector3) – Point to face towards

LookInDirection(*vec*)
Face in a vector direction (from origin to point). See *Transform.LookAtTransform* for details.

Parameters **vec** (Vector3) – Direction to face in

class pyunity.core.**LightType**
Bases: enum.IntEnum

An enumeration.

Point = 0

Directional = 1

Spot = 2

class pyunity.core.**Light**(*transform*, *is_dummy=False*)
Bases: *pyunity.core.SingleComponent*

Component to hold data about the light in a scene.

intensity
Intensity of light

Type int

color
Light color (will mix with material color)

Type *Color*

type
Type of light (currently only Point and Directional are supported)

Type *LightType*

type = 1

class pyunity.core.**MeshRenderer**(*transform*, *is_dummy=False*)
Bases: *pyunity.core.SingleComponent*

Component to render a mesh at the position of a transform.

mesh
Mesh that the MeshRenderer will render.

Type *Mesh*

mat
Material to use for the mesh

Type *Material*

Render()
Render the mesh that the MeshRenderer has.

pyunity.errors module

Module for all exceptions and warnings related to PyUnity.

exception pyunity.errors.**PyUnityException**
Bases: Exception

Base class for PyUnity exceptions.

exception pyunity.errors.**ComponentException**
Bases: *pyunity.errors.PyUnityException*

Class for PyUnity exceptions relating to components.

exception pyunity.errors.**GameObjectException**
Bases: *pyunity.errors.PyUnityException*

Class for PyUnity exceptions relating to GameObjects.

pyunity.files module

Module to load files and scripts. Also manages project structure.

pyunity.files.**convert**(*type*, *list*)
Converts a Python array to a C type from ctypes.

Parameters

- **type** (*_ctypes.PyCSimpleType*) – Type to cast to.
- **list** (*list*) – List to cast

Returns A C array

Return type object

class pyunity.files.**Behaviour**(*transform*, *is_dummy=False*)
Bases: *pyunity.core.Component*

Base class for behaviours that can be scripted.

gameObject
GameObject that the component belongs to.

Type *GameObject*

transform
Transform that the component belongs to.

Type *Transform*

Start()
Called every time a scene is loaded up.

Update(*dt*)
Called every frame.

Parameters **dt** (*float*) – Time since last frame, sent by the scene that the Behaviour is in.

FixedUpdate(*dt*)
Called every frame, in each physics step.

Parameters **dt** (*float*) – Length of this physics step

LateUpdate(*dt*)
Called every frame, after physics processing.

Parameters **dt** (*float*) – Time since last frame, sent by the scene that the Behaviour is in.

class pyunity.files.**Scripts**
Bases: object

Utility class for loading scripts in a folder.

static CheckScript(*text*)
Check if `text` is a valid script for PyUnity.

Parameters **text** (*list*) – List of lines

Returns If script is valid or not.

Return type bool

Notes

This function checks each line to see if it matches at least one of these criteria:

1. The line is an `import` statement
2. The line is just whitespace or blank
3. The line is just a comment preceded by whitespace or nothing
4. The line is a class definition
5. The line has an indentation at the beginning

These checks are essential to ensure no malicious code is run to break the PyUnity engine.

static LoadScripts(*path*)
Loads all scripts found in `path`.

Parameters **path** (*Pathlike*) – A path to a folder containing all the scripts

Returns A module that contains all the imported scripts

Return type ModuleType

Notes

This function will add a module to `sys.modules` that is called `PyUnityScripts`, and can be imported like any other module. The module will also have a variable called `__pyunity__` which shows that it is from PyUnity and not a real module. If an existing module named `PyUnityScripts` is present and does not have the `__pyunity__` variable set, then a warning will be issued and it will be replaced.

class pyunity.files.**Texture2D**(*path_or_im*)
Bases: object

Class to represent a texture.

load()
Loads the texture and sets up an OpenGL texture name.

setImg(*im*)

use()
Binds the texture for usage. The texture is reloaded if it hasn't already been.

class pyunity.files.**Skybox**(*path*)
Bases: object

Skybox model consisting of 6 images

compile()

use()

class pyunity.files.**Prefab**(*gameObject*, *components*)
Bases: object

Prefab model

class pyunity.files.**File**(*path*, *type*, *uuid=None*)
Bases: object

class pyunity.files.**Project**(*path*, *name*)
Bases: object

import_file(*localPath*, *type*, *uuid=None*)

reimport_file(*localPath*)

get_file_obj(*uuid*)

write_project()

static from_folder(*filePath*)

save_mat(*mat*, *name*)

load_mat(*file*)

pyunity.gui module

class pyunity.gui.**Canvas**(*transform*, *is_dummy=False*)
Bases: *pyunity.core.Component*

A Component that manages GUI interactions and 2D rendering. Only GameObjects which are a descendant of a Canvas will be rendered.

Update(*updated*)
Check if any components have been clicked on.

Parameters **updated** (*list*) – List of already updated GameObjects.

class pyunity.gui.**RectData**(*min_or_both=None*, *max=None*)
Bases: object

Class to represent a 2D rect.

Parameters

- **min_or_both** (Vector2 *or* RectData) – Minimum value, or another RectData object
- **max** (Vector2 *or None*) – Maximum value. Default is None

class pyunity.gui.**RectAnchors**(*min_or_both=None*, *max=None*)
Bases: *pyunity.gui.RectData*

A type of RectData which represents the anchor points of a RectTransform.

SetPoint(*p*)
Changes both the minimum and maximum anchor points.

Parameters **p** (Vector2) – Point

RelativeTo(*other*)
Get RectData of another Rect relative to the anchor points.

Parameters **other** (RectData) – Querying rect

Returns Relative rect to this

Return type *RectData*

class pyunity.gui.**RectOffset**(*min_or_both=None*, *max=None*)
Bases: *pyunity.gui.RectData*

Rect to represent the offset from the anchor points of a RectTransform.

static Rectangle(*size*, *center=Vector2(0, 0)*)
Create a rectangular RectOffset.

Parameters

- **size** (*float or Vector2*) – Size of offset
- **center** (*Vector2, optional*) – Central point of RectOffset, by default Vector2.zero()

Returns The generated RectOffset

Return type *RectOffset*

Move(*pos*)
Move the RectOffset by a specified amount.

Parameters **pos** (Vector2) –

SetCenter(*pos*)
Sets the center of the RectOffset. The size is preserved.

Parameters **pos** (Vector2) – Center point of the RectOffset

SetPoint(*pos*)

class pyunity.gui.**RectTransform**(*transform*)
Bases: *pyunity.core.SingleComponent*

A Component that represents the size, position and orientation of a 2D object.

anchors
Anchor points of the RectTransform. Measured between Vector2(0, 0) and Vector2(1, 1)

Type *RectAnchors*

offset
Offset vectors representing the offset of opposite corners from the anchors. Measured in pixels

Type *RectOffset*

pivot
Point in which the object rotates around. Measured between Vector2(0, 0) and Vector2(1, 1)

Type *Vector2*

rotation
Rotation in degrees

Type float

parent

GetRect()
Gets screen coordinates of the bounding box.

Returns Screen coordinates

Return type *RectData*

class pyunity.gui.**GuiComponent**(*transform*, *is_dummy=False*)
Bases: *pyunity.core.Component*

A Component that represents a clickable area.

HoverUpdate()

class pyunity.gui.**NoResponseGuiComponent**(*transform*, *is_dummy=False*)
Bases: *pyunity.gui.GuiComponent*

A Component that blocks all clicks that are behind it.

HoverUpdate()
Empty HoverUpdate function. This is to ensure nothing happens when the component is clicked, and so components behind won't be updated.

class pyunity.gui.**Image2D**(*transform*)
Bases: *pyunity.gui.NoResponseGuiComponent*

A 2D image component, which is uninteractive.

texture
Texture to render

Type *Texture2D*

depth
Z ordering of image. Higher depths are drawn on top.

Type float

class pyunity.gui.**Button**(*transform*)
Bases: *pyunity.gui.GuiComponent*

A Component that calls a function when clicked.

callback
Callback function

Type FunctionType

state
Which state triggers the callback

Type *KeyState*

mouseButton
Which mouse button triggers the callback

Type *MouseCode*

pressed
If the button is pressed down or not

Type bool

state = 1

mouseButton = 1

HoverUpdate()

class pyunity.gui.**WinFontLoader**
Bases: pyunity.gui._FontLoader

classmethod LoadFile(*name*)
Use the Windows registry to find a font file name.

Parameters **name** (*str*) – Font name. This is not the same as the file name.

Returns Font file name

Return type str

Raises PyUnityException – If the font is not found

class pyunity.gui.**UnixFontLoader**
Bases: pyunity.gui._FontLoader

classmethod LoadFile(*name*)
Use fc-match to find the font file name.

Parameters **name** (*str*) – Font name. This is not the same as the file name.

Returns Font file name

Return type str

Raises PyUnityException – If the font is not found

class pyunity.gui.**FontLoader**
Bases: *pyunity.gui.UnixFontLoader*

class pyunity.gui.**Font**(*name*, *size*, *imagefont*)
Bases: object

Font object to represent font data.

_font
Image font object. Do not use unless you know what you are doing.

Type ImageFont.FreeTypeFont

name
Font name

Type str

size
Font size, in points

Type int

class pyunity.gui.**TextAlign**
Bases: enum.IntEnum

An enumeration.

Left = 1

Center = 2

Right = 3

class pyunity.gui.**Text**(*transform*)
Bases: *pyunity.gui.NoResponseGuiComponent*

Component to render text.

font
Font object to render

Type *Font*

text
Contents of the Text

Type str

color
Fill color

Type *Color*

depth
Z ordering of the text. Higher values are on top.

Type float

centeredX
How to align in the X direction

Type *TextAlign*

centeredY
How to align in the Y direction

Type *TextAlign*

rect
RectTransform of the GameObject. Can be None

Type *RectTransform*

texture
Texture of the text, to save computation time.

Type *Texture2D*

Notes

Modifying *font*, *text*, or *color* will call *GenTexture()*.

centeredX = 1

centeredY = 2

GenTexture()
Generate a Texture2D to render.

class pyunity.gui.**CheckBox**(*transform*, *is_dummy=False*)
Bases: *pyunity.gui.GuiComponent*

A component that updates the Image2D of its GameObject when clicked.

checked
Current state of the checkbox

Type bool

HoverUpdate()
Inverts checked and updates the texture of the Image2D, if there is one.

class pyunity.gui.**Gui**
Bases: object

Helper class to create GUI GameObjects. Do not instantiate.

classmethod MakeButton(*name*, *scene*, *text='Button'*, *font=None*, *color=None*, *texture=None*)
Create a Button GameObject and add all relevant GameObjects to the scene.

Parameters

- **name** (*str*) – Name of the GameObject
- **scene** (Scene) – Scene to add all generated GameObjects to
- **text** (*str, optional*) – Text content of the button, by default "Button"
- **font** (Font, *optional*) – Default font to use, if None then "Arial" is used
- **color** (Color, *optional*) – Fill color of the button text, by default black
- **texture** (Texture2D, *optional*) – Texture for the button background.

Returns A tuple containing the *RectTransform* of button, the *Button* component and the *Text* component.

Return type tuple

Notes

This will create 3 GameObjects in this hierarchy:

```
<specified button name>
|- Button
|- Text
```

The generated GameObject can be accessed from the `gameObject` property of the returned components. The `Button` GameObject will have two components, *Button* and *RectTransform*. The `Button` GameObject will have two components, *Image2D* and *RectTransform*.

classmethod MakeCheckBox(*name*, *scene*)
Create a CheckBox GameObject and add the appropriate components needed.

Parameters

- **name** (*str*) – Name of GameObject
- **scene** (Scene) – Scene to add GameObject to

Returns A tuple of the *RectTransform* as well as the *CheckBox* component.

Return type tuple

Notes

The generated GameObject can be accessed from the `gameObject` property of the returned components. The GameObject will have 3 properties added: a *RectTransform*, a *CheckBox* and an *Image2D*.

pyunity.input module

class pyunity.input.**KeyState**
Bases: enum.IntEnum

An enumeration.

UP = 1

DOWN = 2

PRESS = 3

NONE = 4

class pyunity.input.**KeyCode**
Bases: enum.IntEnum

An enumeration.

A = 1

B = 2

C = 3

D = 4

E = 5

F = 6

G = 7

H = 8

I = 9

J = 10

K = 11

L = 12

M = 13

N = 14

O = 15

P = 16

Q = 17

R = 18

S = 19

T = 20

U = 21

V = 22

W = 23

X = 24

Y = 25

Z = 26

Space = 27

Alpha0 = 28

Alpha1 = 29

Alpha2 = 30

Alpha3 = 31

Alpha4 = 32

Alpha5 = 33

Alpha6 = 34

Alpha7 = 35

Alpha8 = 36

Alpha9 = 37

F1 = 38

F2 = 39

F3 = 40

F4 = 41

F5 = 42

F6 = 43

F7 = 44

F8 = 45

F9 = 46

F10 = 47

F11 = 48

F12 = 49

Keypad0 = 50

Keypad1 = 51

Keypad2 = 52

Keypad3 = 53

Keypad4 = 54

Keypad5 = 55

Keypad6 = 56

Keypad7 = 57

Keypad8 = 58

Keypad9 = 59

Up = 60

Down = 61

Left = 62

Right = 63

class pyunity.input.**MouseCode**

Bases: enum.IntEnum

An enumeration.

Left = 1

Middle = 2

Right = 3

class pyunity.input.**KeyboardAxis**(*name*, *speed*, *positive*, *negative*)
Bases: object

get_value(*dt*)

class pyunity.input.**Input**
Bases: object

classmethod GetKey(*keycode*)
Check if key is pressed at moment of function call

Parameters **keycode** (KeyCode) – Key to query

Returns If the key is pressed

Return type boolean

classmethod GetKeyUp(*keycode*)
Check if key was released this frame.

Parameters **keycode** (KeyCode) – Key to query

Returns If the key was released

Return type boolean

classmethod GetKeyDown(*keycode*)
Check if key was pressed down this frame.

Parameters **keycode** (KeyCode) – Key to query

Returns If the key was pressed down

Return type boolean

classmethod GetKeyState(*keycode*, *keystate*)
Check key state at moment of function call

Parameters

- **keycode** (KeyCode) – Key to query
- **keystate** (KeyState) – Keystate, either KeyState.PRESS, KeyState.UP or KeyState.DOWN

Returns If the key state matches

Return type boolean

classmethod GetMouse(*mousecode*)
Check if mouse button is pressed at moment of function call

Parameters **mousecode** (MouseCode) – Mouse button to query

Returns If the mouse button is pressed

Return type boolean

classmethod GetMouseUp(*mousecode*)
Check if mouse button was released this frame.

Parameters **mousecode** (MouseCode) – Mouse button to query

Returns If the mouse button was released

Return type boolean

classmethod GetMouseDown(*mousecode*)
Check if mouse button was pressed down this frame.

Parameters **mousecode** (MouseCode) – Mouse button to query

Returns If the mouse button was pressed down

Return type boolean

classmethod GetMouseState(*mousecode*, *mousestate*)
Check for mouse button state at moment of function call

Parameters

- **mousecode** (MouseCode) – Key to query
- **mousestate** (KeyState) – Keystate, either KeyState.PRESS, KeyState.UP or KeyState.DOWN

Returns If the mouse button state matches

Return type boolean

classmethod GetAxis(*axis*)
Get the value for the specified axis. This is always between -1 and 1.

Parameters **axis** (*str*) – Specified axis

Returns Axis value

Return type float

Raises PyUnityException – If the axis is not a valid axis

classmethod GetRawAxis(*axis*)
Get the raw value for the specified axis. This is always either -1, 0 or 1.

Parameters **axis** (*str*) – Specified axis

Returns Raw axis value

Return type float

Raises PyUnityException – If the axis is not a valid axis

classmethod UpdateAxes(*dt*)

pyunity.loader module

Utility functions related to loading and saving PyUnity meshes and scenes.

This will be imported as pyunity.Loader.

pyunity.loader.**LoadObj**(*filename*)
Loads a .obj file to a PyUnity mesh.

Parameters **filename** (*str*) – Name of file

Returns A mesh of the object file

Return type *Mesh*

pyunity.loader.**SaveObj**(*mesh*, *name*, *filePath=None*)

pyunity.loader.**LoadMesh**(*filename*)
Loads a .mesh file generated by *SaveMesh*. It is optimized for faster loading.

Parameters **filename** (*str*) – Name of file relative to the cwd

Returns Generated mesh

Return type *Mesh*

pyunity.loader.**SaveMesh**(*mesh*, *name*, *filePath=None*)
Saves a mesh to a .mesh file for faster loading.

Parameters

- **mesh** (Mesh) – Mesh to save
- **name** (*str*) – Name of the mesh
- **filePath** (*str, optional*) – Pass in *__file__* to save in directory of script, otherwise pass in the path of where you want to save the file. For example, if you want to save in C:Downloads, then give "C:Downloadsmesh.mesh". If not specified, then the mesh is saved in the cwd.

pyunity.loader.**GetImports**(*file*)

pyunity.loader.**SaveSceneToProject**(*scene*, *filePath=None*, *name=None*)

pyunity.loader.**SaveAllScenes**(*name*, *filePath=None*)

pyunity.loader.**GetId**(*ids*, *obj*)

pyunity.loader.**SaveScene**(*scene*, *project*)

class pyunity.loader.**ObjectInfo**(*uuid*, *type*, *attrs*)
Bases: object

pyunity.loader.**components = {'AABBoxCollider': <class 'pyunity.physics.core.AABBoxCollide**
List of all components by name

pyunity.loader.**parse_string**(*string*)

pyunity.loader.**LoadProject**(*filePath*)

class pyunity.loader.**Primitives**
Bases: object

Primitive preloaded meshes. Do not instantiate this class.

pyunity.logger module

Utility functions to log output of PyUnity.

This will be imported as pyunity.Logger.

pyunity.logger.**get_tmp**()

class pyunity.logger.**Level**(*abbr*, *name*)
Bases: object

Represents a level or severity to log. You should never instantiate this directly, instead use one of *Logging.OUTPUT*, *Logging.INFO*, *Logging.DEBUG*, *Logging.ERROR* or *Logging.WARN*.

class pyunity.logger.**Special**(*func*)
Bases: object

Class to represent a special line to log. You should never instantiate this class, instead use one of *Logger.RUNNING_TIME*.

pyunity.logger.**Log**(**message*)
Logs a message with level OUTPUT.

pyunity.logger.**LogLine**(*level*, **message*, *silent=False*)
Logs a line in *latest.log* found in these two locations: Windows: `%appdata%\PyUnity\Logs\latest.log` Other: `/tmp/pyunity/logs/latest.log`

Parameters **level** (Level) – Level or severity of log.

pyunity.logger.**LogException**(*e*)
Log an exception.

Parameters **e** (*Exception*) – Exception to log

pyunity.logger.**LogSpecial**(*level*, *type*)
Log a line of level *level* with a special line that is generated at runtime.

Parameters

- **level** (Level) – Level of log
- **type** (Special) – The special line to log

pyunity.logger.**Save**()
Saves a new log file with a timestamp of initializing PyUnity for the first time.

pyunity.logger.**SetStream**(*s*)

pyunity.logger.**ResetStream**()

pyunity.meshes module

Module for meshes created at runtime.

class pyunity.meshes.**Mesh**(*verts*, *triangles*, *normals*, *texcoords=None*)
Bases: object

Class to create a mesh for rendering with a MeshRenderer

Parameters

- **verts** (*list*) – List of Vector3's containing each vertex
- **triangles** (*list*) – List of ints containing triangles joining up the vertices. Each int is the index of a vertex above.
- **normals** (*list*) – List of Vector3's containing the normal of each vertex.

verts
List of Vector3's containing each vertex

Type list

triangles
List of lists containing triangles joining up the vertices. Each int is the index of a vertex above. The list is two-dimesional, meaning that each item in the list is a list of three ints.

Type list

normals
List of Vector3's containing the normal of each vertex.

Type list

texcoords
List of lists containing the texture coordinate of each vertex. The list is two-dimesional, meaning that each item in the list is a list of two floats.

Type list (optional)

Notes

When any of the mesh attributes are updated while a scene is running, you must use `compile(force=True)` to update the mesh so that it is displayed correctly.

```
>>> mesh = Mesh.cube(2)
>>> mesh.vertices[1] = Vector3(2, 0, 0)
>>> mesh.compile(force=True)
```

compile(*force=False*)

draw()

copy()
Create a copy of the current Mesh.

Returns Copy of the mesh

Return type *Mesh*

static quad(*size*)
Creates a quadrilateral mesh.

Parameters **size** (*float*) – Side length of quad

Returns A quad centered at Vector3(0, 0) with side length of `size` facing in the direction of the negative z axis.

Return type *Mesh*

static double_quad(*size*)
Creates a two-sided quadrilateral mesh.

Parameters **size** (*float*) – Side length of quad

Returns A double-sided quad centered at Vector3(0, 0) with side length of `size`.

Return type *Mesh*

static cube(*size*)
Creates a cube mesh.

Parameters **size** (*float*) – Side length of cube

Returns A cube centered at Vector3(0, 0, 0) that has a side length of `size`

Return type *Mesh*

pyunity.render module

Classes to aid in rendering in a Scene.

pyunity.render.**gen_buffers**(*mesh*)
Create buffers for a mesh.

Parameters **mesh** (Mesh) – Mesh to create buffers for

Returns Tuple containing a vertex buffer object and an index buffer object.

Return type tuple

pyunity.render.**gen_array**()
Generate a vertex array object.

Returns

A vertex buffer object of floats. Has 3 elements:

```
# vertex    # normal    # texcoord
x, y, z,    a, b, c,    u, v
```

Return type Any

class pyunity.render.**Shader**(*vertex*, *frag*, *name*)
Bases: object

compile()
Compiles shader and generates program. Checks for errors.

Notes

This function will not work if there is no active framebuffer.

static fromFolder(*path*, *name*)
Create a Shader from a folder. It must contain `vertex.glsl` and `fragment.glsl`.

Parameters

- **path** (*str*) – Path of folder to load
- **name** (*str*) – Name to register this shader to. Used with *Camera.SetShader*.

setVec3(*var*, *val*)
Set a `vec3` uniform variable.

Parameters

- **var** (*bytes*) – Variable name
- **val** (*Any*) – Value of uniform variable

setMat4(*var*, *val*)
Set a `mat4` uniform variable.

Parameters

- **var** (*bytes*) – Variable name
- **val** (*Any*) – Value of uniform variable

setInt(*var*, *val*)
Set an `int` uniform variable.

Parameters

- **var** (*bytes*) – Variable name
- **val** (*Any*) – Value of uniform variable

setFloat(*var*, *val*)
Set a `float` uniform variable.

Parameters

- **var** (*bytes*) – Variable name
- **val** (*Any*) – Value of uniform variable

use()
Compile shader if it isn't compiled, and load it into OpenGL.

pyunity.render.**compile_shaders**()

class pyunity.render.**Camera**(*transform*)
Bases: *pyunity.core.SingleComponent*

Component to hold data about the camera in a scene.

near
Distance of the near plane in the camera frustrum. Defaults to 0.05.

Type float

far
Distance of the far plane in the camera frustrum. Defaults to 100.

Type float

clearColor
The clear color of the camera. Defaults to (0, 0, 0).

Type *RGB*

setup_buffers()
Creates 2D quad VBO and VAO for GUI.

fov
FOV of camera

orthoSize

Resize(*width*, *height*)
Resizes the viewport on screen size change.

Parameters

- **width** (*int*) – Width of new window
- **height** (*int*) – Height of new window

getMatrix(*transform*)
Generates model matrix from transform.

get2DMatrix(*rectTransform*)
Generates model matrix from RectTransform.

getViewMat()
Generates view matrix from Transform of camera.

UseShader(*name*)
Sets current shader from name.

Render(*renderers*, *lights*)
Render specific renderers, taking into account light positions.

Parameters

- **renderers** (*List[MeshRenderer]*) – Which meshes to render

- **lights** (*List[Light]*) – Lights to load into shader

Render2D(*canvases*)
Render all Image2D and Text components in specified canvases.

Parameters **canvases** (*List[Canvas]*) – Canvases to process. All processed GameObjects are cached to prevent duplicate rendering.

class pyunity.render.**Screen**
Bases: object

pyunity.settings module

class pyunity.settings.**LiveDict**(*d*, *parent=None*)
Bases: object

update()

todict()

keys()

values()

items()

pop(*item*)

class pyunity.settings.**Database**(*path*)
Bases: *pyunity.settings.LiveDict*

update()

refresh()

4.6 Indices and tables

- genindex
- modindex
- search

Python Module Index

p

pyunity.audio, 42
pyunity.core, 43
pyunity.errors, 49
pyunity.files, 50
pyunity.gui, 52
pyunity.input, 57
pyunity.loader, 61
pyunity.logger, 62
pyunity.meshes, 63
pyunity.physics, 29
pyunity.physics.config, 26
pyunity.physics.core, 26
pyunity.render, 64
pyunity.scenes, 33
pyunity.scenes.scene, 30
pyunity.scenes.sceneManager, 32
pyunity.settings, 67
pyunity.values, 40
pyunity.values.abc, 33
pyunity.values.other, 34
pyunity.values.quaternion, 34
pyunity.values.texture, 35
pyunity.values.vector, 36
pyunity.window, 41
pyunity.window.glfwWindow, 40
pyunity.window.glutWindow, 40
pyunity.window.sdl2Window, 41
pyunity.window.templateWindow, 41

Index

Symbols

_font (*pyunity.gui.Font attribute*), 55

A

A (*pyunity.input.KeyCode attribute*), 58
AABBoxCollider (*class in pyunity.physics.core*), 27
ABCException, 33
ABCMessage, 33
ABCMeta (*class in pyunity.values.abc*), 34
ABCWindow (*class in pyunity.window*), 42
abs() (*pyunity.values.vector.Vector method*), 36
abs_diff() (*pyunity.values.quaternion.Quaternion method*), 34
abstractmethod (*class in pyunity.values.abc*), 34
abstractproperty (*class in pyunity.values.abc*), 34
Add() (*pyunity.scenes.scene.Scene method*), 30
AddBareScene() (*in module pyunity.scenes.sceneManager*), 32
AddComponent() (*pyunity.core.Component method*), 47
AddComponent() (*pyunity.core.GameObject method*), 45
AddForce() (*pyunity.physics.core.Rigidbody method*), 28
AddImpulse() (*pyunity.physics.core.Rigidbody method*), 28
AddPhysicsInfo() (*pyunity.physics.core.CollManager method*), 29
AddScene() (*in module pyunity.scenes.sceneManager*), 32
AddTag() (*pyunity.core.Tag class method*), 44
Alpha0 (*pyunity.input.KeyCode attribute*), 58
Alpha1 (*pyunity.input.KeyCode attribute*), 58
Alpha2 (*pyunity.input.KeyCode attribute*), 58
Alpha3 (*pyunity.input.KeyCode attribute*), 58
Alpha4 (*pyunity.input.KeyCode attribute*), 59
Alpha5 (*pyunity.input.KeyCode attribute*), 59
Alpha6 (*pyunity.input.KeyCode attribute*), 59
Alpha7 (*pyunity.input.KeyCode attribute*), 59
Alpha8 (*pyunity.input.KeyCode attribute*), 59
Alpha9 (*pyunity.input.KeyCode attribute*), 59
anchors (*pyunity.gui.RectTransform attribute*), 53
angleAxisPair (*pyunity.values.quaternion.Quaternion attribute*), 35
AudioClip (*class in pyunity.audio*), 42
AudioListener (*class in pyunity.audio*), 43
AudioSource (*class in pyunity.audio*), 42

B

B (*pyunity.input.KeyCode attribute*), 58
back() (*pyunity.values.vector.Vector3 static method*), 39
Bare() (*pyunity.scenes.scene.Scene static method*), 30
BareObject() (*pyunity.core.GameObject static method*), 45
Behaviour (*class in pyunity.files*), 50
Between() (*pyunity.values.quaternion.Quaternion static method*), 35
Button (*class in pyunity.gui*), 54

C

C (*pyunity.input.KeyCode attribute*), 58
callback (*pyunity.gui.Button attribute*), 54
Camera (*class in pyunity.render*), 66
Canvas (*class in pyunity.gui*), 52
Center (*pyunity.gui.TextAlign attribute*), 55
centeredX (*pyunity.gui.Text attribute*), 56
centeredY (*pyunity.gui.Text attribute*), 56
check_keys() (*pyunity.window.glfwWindow.Window method*), 40
check_mouse() (*pyunity.window.glfwWindow.Window method*), 40
check_quit() (*pyunity.window.glfwWindow.Window method*), 40
CheckBox (*class in pyunity.gui*), 56

CheckCollisions() (*pyunity.physics.core.CollManager method*), 29
checked (*pyunity.gui.CheckBox attribute*), 56
checkModule() (*in module pyunity.window*), 42
CheckScript() (*pyunity.files.Scripts static method*), 51
children (*pyunity.core.Transform attribute*), 48
clamp() (*in module pyunity.values.vector*), 36
clamp() (*pyunity.values.vector.Vector2 method*), 37
clamp() (*pyunity.values.vector.Vector3 method*), 39
clean_up() (*pyunity.scenes.scene.Scene method*), 32
clearColor (*pyunity.render.Camera attribute*), 66
clip (*pyunity.audio.AudioSource attribute*), 42
Clock (*class in pyunity.values.other*), 34
Collider (*class in pyunity.physics.core*), 26
collidingWith() (*pyunity.physics.core.AABBoxCollider method*), 27
collidingWith() (*pyunity.physics.core.Collider method*), 26
collidingWith() (*pyunity.physics.core.SphereCollider method*), 27
CollManager (*class in pyunity.physics.core*), 28
Color (*class in pyunity.values.texture*), 35
color (*pyunity.core.Light attribute*), 49
color (*pyunity.gui.Text attribute*), 56
color (*pyunity.values.texture.Material attribute*), 35
combine (*pyunity.physics.core.PhysicMaterial attribute*), 26
compile() (*pyunity.files.Skybox method*), 51
compile() (*pyunity.meshes.Mesh method*), 64
compile() (*pyunity.render.Shader method*), 65
compile_shaders() (*in module pyunity.render*), 66
Component (*class in pyunity.core*), 46
ComponentException, 50
components (*in module pyunity.loader*), 62
components (*pyunity.core.GameObject attribute*), 45
conjugate (*pyunity.values.quaternion.Quaternion attribute*), 34
convert() (*in module pyunity.files*), 50
copy() (*pyunity.meshes.Mesh method*), 64
copy() (*pyunity.values.quaternion.Quaternion method*), 34
copy() (*pyunity.values.vector.Vector2 method*), 36
copy() (*pyunity.values.vector.Vector3 method*), 38
correct_inf() (*pyunity.physics.core.CollManager method*), 29
cross() (*pyunity.values.vector.Vector2 method*), 37
cross() (*pyunity.values.vector.Vector3 method*), 39
cube() (*pyunity.meshes.Mesh static method*), 64
CurrentScene() (*in module pyunity.scenes.sceneManager*), 33
CustomWindowProvider() (*in module pyunity.window*), 42

D

D (*pyunity.input.KeyCode attribute*), 58
Database (*class in pyunity.settings*), 67
default (*pyunity.core.HideInInspector attribute*), 46
default (*pyunity.core.ShowInInspector attribute*), 46
DeInit() (*pyunity.audio.AudioListener method*), 43
depth (*pyunity.gui.Image2D attribute*), 54
depth (*pyunity.gui.Text attribute*), 56
Directional (*pyunity.core.LightType attribute*), 49
display() (*pyunity.window.glutWindow.Window method*), 40
dot() (*pyunity.values.vector.Vector2 method*), 37
dot() (*pyunity.values.vector.Vector3 method*), 39
double_quad() (*pyunity.meshes.Mesh static method*), 64
Down (*pyunity.input.KeyCode attribute*), 59
DOWN (*pyunity.input.KeyState attribute*), 57
down() (*pyunity.values.vector.Vector2 static method*), 38
down() (*pyunity.values.vector.Vector3 static method*), 39
draw() (*pyunity.meshes.Mesh method*), 64
dummyRigidbody (*pyunity.physics.core.CollManager attribute*), 28

E

E (*pyunity.input.KeyCode attribute*), 58
Euler() (*pyunity.values.quaternion.Quaternion static method*), 35
eulerAngles (*pyunity.core.Transform attribute*), 48
eulerAngles (*pyunity.values.quaternion.Quaternion attribute*), 35
exception() (*pyunity.physics.core.PhysicMaterial method*), 26

F

F (*pyunity.input.KeyCode attribute*), 58
F1 (*pyunity.input.KeyCode attribute*), 59
F10 (*pyunity.input.KeyCode attribute*), 59
F11 (*pyunity.input.KeyCode attribute*), 59
F12 (*pyunity.input.KeyCode attribute*), 59
F2 (*pyunity.input.KeyCode attribute*), 59
F3 (*pyunity.input.KeyCode attribute*), 59
F4 (*pyunity.input.KeyCode attribute*), 59
F5 (*pyunity.input.KeyCode attribute*), 59
F6 (*pyunity.input.KeyCode attribute*), 59
F7 (*pyunity.input.KeyCode attribute*), 59
F8 (*pyunity.input.KeyCode attribute*), 59
F9 (*pyunity.input.KeyCode attribute*), 59
far (*pyunity.render.Camera attribute*), 66
File (*class in pyunity.files*), 52

FindComponentByType() (*pyunity.scenes.scene.Scene method*), 31
FindComponentsByType() (*pyunity.scenes.scene.Scene method*), 31
FindGameObjectsByName() (*pyunity.scenes.scene.Scene method*), 31
FindGameObjectsByTagName() (*pyunity.scenes.scene.Scene method*), 31
FindGameObjectsByTagNumber() (*pyunity.scenes.scene.Scene method*), 31
FixedUpdate() (*pyunity.files.Behaviour method*), 50
Font (*class in pyunity.gui*), 55
font (*pyunity.gui.Text attribute*), 55
FontLoader (*class in pyunity.gui*), 55
forward() (*pyunity.values.vector.Vector3 static method*), 39
fov (*pyunity.render.Camera attribute*), 66
fps (*pyunity.values.other.Clock attribute*), 34
framebuffer_size_callback() (*pyunity.window.glfwWindow.Window method*), 40
friction (*pyunity.physics.core.PhysicMaterial attribute*), 26
from_folder() (*pyunity.files.Project static method*), 52
from_hsv() (*pyunity.values.texture.RGB static method*), 36
from_rgb() (*pyunity.values.texture.HSV static method*), 36
from_string() (*pyunity.values.texture.Color static method*), 35
FromAxis() (*pyunity.values.quaternion.Quaternion static method*), 35
FromDir() (*pyunity.values.quaternion.Quaternion static method*), 35
fromFolder() (*pyunity.render.Shader static method*), 65
FullPath() (*pyunity.core.Transform method*), 48

G

G (*pyunity.input.KeyCode attribute*), 58
GameObject (*class in pyunity.core*), 45
gameObject (*pyunity.core.Component attribute*), 46
gameObject (*pyunity.core.Transform attribute*), 47
gameObject (*pyunity.files.Behaviour attribute*), 50
GameObjectException, 50
gen_array() (*in module pyunity.render*), 65
gen_buffers() (*in module pyunity.render*), 64
GenTexture() (*pyunity.gui.Text method*), 56
get2DMatrix() (*pyunity.render.Camera method*), 66
get_dist_sqrd() (*pyunity.values.vector.Vector2 method*), 37
get_dist_sqrd() (*pyunity.values.vector.Vector3 method*), 38
get_distance() (*pyunity.values.vector.Vector2 method*), 37
get_distance() (*pyunity.values.vector.Vector3 method*), 38
get_file_obj() (*pyunity.files.Project method*), 52
get_key() (*pyunity.window.ABCWindow method*), 42
get_key() (*pyunity.window.glfwWindow.Window method*), 40
get_key() (*pyunity.window.glutWindow.Window method*), 41
get_key() (*pyunity.window.sdl2Window.Window method*), 41
get_length_sqrd() (*pyunity.values.vector.Vector2 method*), 36
get_length_sqrd() (*pyunity.values.vector.Vector3 method*), 38
get_mouse() (*pyunity.window.ABCWindow method*), 42
get_mouse() (*pyunity.window.glfwWindow.Window method*), 40
get_mouse() (*pyunity.window.glutWindow.Window method*), 41
get_mouse() (*pyunity.window.sdl2Window.Window method*), 41
get_mouse_pos() (*pyunity.window.ABCWindow method*), 42
get_mouse_pos() (*pyunity.window.glfwWindow.Window method*), 40
get_mouse_pos() (*pyunity.window.sdl2Window.Window method*), 41
get_tmp() (*in module pyunity.logger*), 62
get_value() (*pyunity.input.KeyboardAxis method*), 60
getargs() (*pyunity.values.abc.abstractmethod static method*), 34
GetAxis() (*pyunity.input.Input class method*), 61
GetComponent() (*pyunity.core.Component method*), 47
GetComponent() (*pyunity.core.GameObject method*), 45
GetComponents() (*pyunity.core.Component method*), 47
GetComponents() (*pyunity.core.GameObject method*), 46
GetDescendants() (*pyunity.core.Transform method*), 48
GetId() (*in module pyunity.loader*), 62
GetImports() (*in module pyunity.loader*), 62
GetKey() (*pyunity.input.Input class method*), 60
GetKeyDown() (*pyunity.input.Input class method*), 60
GetKeyState() (*pyunity.input.Input class method*), 60

GetKeyUp() (*pyunity.input.Input class method*), 60
getMatrix() (*pyunity.render.Camera method*), 66
GetMouse() (*pyunity.input.Input class method*), 60
GetMouseDown() (*pyunity.input.Input class method*), 61
GetMouseState() (*pyunity.input.Input class method*), 61
GetMouseUp() (*pyunity.input.Input class method*), 60
GetRawAxis() (*pyunity.input.Input class method*), 61
GetRect() (*pyunity.gui.RectTransform method*), 53
GetRestitution() (*pyunity.physics.core.CollManager method*), 29
GetSceneByIndex() (*in module pyunity.scenes.sceneManager*), 32
GetSceneByName() (*in module pyunity.scenes.sceneManager*), 32
getViewMat() (*pyunity.render.Camera method*), 66
GetWindowProvider() (*in module pyunity.window*), 42
glfwCheck() (*in module pyunity.window*), 42
glutCheck() (*in module pyunity.window*), 42
gravity (*in module pyunity.physics.config*), 26
Gui (*class in pyunity.gui*), 56
GuiComponent (*class in pyunity.gui*), 54

H

H (*pyunity.input.KeyCode attribute*), 58
Has() (*pyunity.scenes.scene.Scene method*), 30
HideInInspector (*class in pyunity.core*), 46
HoverUpdate() (*pyunity.gui.Button method*), 54
HoverUpdate() (*pyunity.gui.CheckBox method*), 56
HoverUpdate() (*pyunity.gui.GuiComponent method*), 54
HoverUpdate() (*pyunity.gui.NoResponseGuiComponent method*), 54
HSV (*class in pyunity.values.texture*), 36

I

I (*pyunity.input.KeyCode attribute*), 58
identity() (*pyunity.values.quaternion.Quaternion static method*), 35
Image2D (*class in pyunity.gui*), 54
ImmutableStruct (*class in pyunity.values.other*), 34
import_file() (*pyunity.files.Project method*), 52
Infinity (*in module pyunity.physics.core*), 26
Init() (*pyunity.audio.AudioListener method*), 43
Input (*class in pyunity.input*), 60
inside_frustrum() (*pyunity.scenes.scene.Scene method*), 31
int_tuple (*pyunity.values.vector.Vector2 attribute*), 37
int_tuple (*pyunity.values.vector.Vector3 attribute*), 39
intensity (*pyunity.core.Light attribute*), 49
items() (*pyunity.settings.LiveDict method*), 67

J

J (*pyunity.input.KeyCode attribute*), 58

K

K (*pyunity.input.KeyCode attribute*), 58
key_callback() (*pyunity.window.glfwWindow.Window method*), 40
KeyboardAxis (*class in pyunity.input*), 60
KeyCode (*class in pyunity.input*), 58
Keypad0 (*pyunity.input.KeyCode attribute*), 59
Keypad1 (*pyunity.input.KeyCode attribute*), 59
Keypad2 (*pyunity.input.KeyCode attribute*), 59
Keypad3 (*pyunity.input.KeyCode attribute*), 59
Keypad4 (*pyunity.input.KeyCode attribute*), 59
Keypad5 (*pyunity.input.KeyCode attribute*), 59
Keypad6 (*pyunity.input.KeyCode attribute*), 59
Keypad7 (*pyunity.input.KeyCode attribute*), 59
Keypad8 (*pyunity.input.KeyCode attribute*), 59
Keypad9 (*pyunity.input.KeyCode attribute*), 59
keys() (*pyunity.settings.LiveDict method*), 67
KeyState (*class in pyunity.input*), 57

L

L (*pyunity.input.KeyCode attribute*), 58
LateUpdate() (*pyunity.files.Behaviour method*), 50
Left (*pyunity.gui.TextAlign attribute*), 55
Left (*pyunity.input.KeyCode attribute*), 59
Left (*pyunity.input.MouseCode attribute*), 59
left() (*pyunity.values.vector.Vector2 static method*), 38
left() (*pyunity.values.vector.Vector3 static method*), 39
length (*pyunity.values.vector.Vector2 attribute*), 36
length (*pyunity.values.vector.Vector3 attribute*), 38
length() (*pyunity.values.vector.Vector method*), 36
Level (*class in pyunity.logger*), 62
Light (*class in pyunity.core*), 49
LightType (*class in pyunity.core*), 49
List() (*pyunity.core.Transform method*), 48
List() (*pyunity.scenes.scene.Scene method*), 30
LiveDict (*class in pyunity.settings*), 67
load() (*pyunity.files.Texture2D method*), 51
load_mat() (*pyunity.files.Project method*), 52
LoadFile() (*pyunity.gui.UnixFontLoader class method*), 55
LoadFile() (*pyunity.gui.WinFontLoader class method*), 54
LoadMesh() (*in module pyunity.loader*), 61
LoadObj() (*in module pyunity.loader*), 61
LoadProject() (*in module pyunity.loader*), 62

LoadScene() (*in module pyunity.scenes.sceneManager*), 33
LoadSceneByIndex() (*in module pyunity.scenes.sceneManager*), 33
LoadSceneByName() (*in module pyunity.scenes.sceneManager*), 33
LoadScripts() (*pyunity.files.Scripts static method*), 51
localEulerAngles (*pyunity.core.Transform attribute*), 48
localPosition (*pyunity.core.Transform attribute*), 47
localRotation (*pyunity.core.Transform attribute*), 47
localScale (*pyunity.core.Transform attribute*), 47
Log() (*in module pyunity.logger*), 63
LogException() (*in module pyunity.logger*), 63
LogLine() (*in module pyunity.logger*), 63
LogSpecial() (*in module pyunity.logger*), 63
LookAtGameObject() (*pyunity.core.Transform method*), 48
LookAtPoint() (*pyunity.core.Transform method*), 49
LookAtTransform() (*pyunity.core.Transform method*), 48
LookInDirection() (*pyunity.core.Transform method*), 49
loop (*pyunity.audio.AudioSource attribute*), 43

M

M (*pyunity.input.KeyCode attribute*), 58
Maintain() (*pyunity.values.other.Clock method*), 34
MakeButton() (*pyunity.gui.Gui class method*), 56
MakeCheckBox() (*pyunity.gui.Gui class method*), 57
Manifold (*class in pyunity.physics.core*), 26
mass (*pyunity.physics.core.Rigidbody attribute*), 28
mat (*pyunity.core.MeshRenderer attribute*), 49
Material (*class in pyunity.values.texture*), 35
max() (*pyunity.values.vector.Vector2 static method*), 37
max() (*pyunity.values.vector.Vector3 static method*), 39
Mesh (*class in pyunity.meshes*), 63
mesh (*pyunity.core.MeshRenderer attribute*), 49
MeshRenderer (*class in pyunity.core*), 49
Middle (*pyunity.input.MouseCode attribute*), 60
min() (*pyunity.values.vector.Vector2 static method*), 37
min() (*pyunity.values.vector.Vector3 static method*), 39
mixer (*in module pyunity.audio*), 42
mouse_callback() (*pyunity.window.glfwWindow.Window method*), 40
mouseButton (*pyunity.gui.Button attribute*), 54
MouseCode (*class in pyunity.input*), 59
Move() (*pyunity.gui.RectOffset method*), 53
Move() (*pyunity.physics.core.Rigidbody method*), 28
music (*pyunity.audio.AudioClip attribute*), 42

N

N (*pyunity.input.KeyCode attribute*), 58
name (*pyunity.core.GameObject attribute*), 45
name (*pyunity.core.HideInInspector attribute*), 46
name (*pyunity.core.ShowInInspector attribute*), 46
name (*pyunity.gui.Font attribute*), 55
near (*pyunity.render.Camera attribute*), 66
no_interactive() (*pyunity.scenes.scene.Scene method*), 32
NONE (*pyunity.input.KeyState attribute*), 58
NoResponseGuiComponent (*class in pyunity.gui*), 54
normalize() (*pyunity.values.vector.Vector2 method*), 37
normalize() (*pyunity.values.vector.Vector3 method*), 38
normalize_return_length() (*pyunity.values.vector.Vector2 method*), 37
normalize_return_length() (*pyunity.values.vector.Vector3 method*), 38
normalized() (*pyunity.values.quaternion.Quaternion method*), 34
normalized() (*pyunity.values.vector.Vector2 method*), 36
normalized() (*pyunity.values.vector.Vector3 method*), 38
normals (*pyunity.meshes.Mesh attribute*), 63

O

O (*pyunity.input.KeyCode attribute*), 58
ObjectInfo (*class in pyunity.loader*), 62
offset (*pyunity.gui.RectTransform attribute*), 53
one() (*pyunity.values.vector.Vector2 static method*), 38
one() (*pyunity.values.vector.Vector3 static method*), 39
orthoSize (*pyunity.render.Camera attribute*), 66

P

P (*pyunity.input.KeyCode attribute*), 58
parent (*pyunity.core.Transform attribute*), 48
parent (*pyunity.gui.RectTransform attribute*), 53
parse_string() (*in module pyunity.loader*), 62
path (*pyunity.audio.AudioClip attribute*), 42
Pause() (*pyunity.audio.AudioSource method*), 43
PhysicMaterial (*class in pyunity.physics.core*), 26
physicMaterial (*pyunity.physics.core.Rigidbody attribute*), 28
pivot (*pyunity.gui.RectTransform attribute*), 53
Play() (*pyunity.audio.AudioSource method*), 43
Playing (*pyunity.audio.AudioSource attribute*), 43
playOnStart (*pyunity.audio.AudioSource attribute*), 43
Point (*pyunity.core.LightType attribute*), 49
pop() (*pyunity.settings.LiveDict method*), 67

pos (*pyunity.physics.core.Collider attribute*), 26
position (*pyunity.core.Transform attribute*), 48
Prefab (*class in pyunity.files*), 52
PRESS (*pyunity.input.KeyState attribute*), 58
pressed (*pyunity.gui.Button attribute*), 54
Primitives (*class in pyunity.loader*), 62
process_keys() (*pyunity.window.sdl2Window.Window method*), 41
process_mouse() (*pyunity.window.sdl2Window.Window method*), 41
Project (*class in pyunity.files*), 52
pyunity.audio (*module*), 42
pyunity.core (*module*), 43
pyunity.errors (*module*), 49
pyunity.files (*module*), 50
pyunity.gui (*module*), 52
pyunity.input (*module*), 57
pyunity.loader (*module*), 61
pyunity.logger (*module*), 62
pyunity.meshes (*module*), 63
pyunity.physics (*module*), 29
pyunity.physics.config (*module*), 26
pyunity.physics.core (*module*), 26
pyunity.render (*module*), 64
pyunity.scenes (*module*), 33
pyunity.scenes.scene (*module*), 30
pyunity.scenes.sceneManager (*module*), 32
pyunity.settings (*module*), 67
pyunity.values (*module*), 40
pyunity.values.abc (*module*), 33
pyunity.values.other (*module*), 34
pyunity.values.quaternion (*module*), 34
pyunity.values.texture (*module*), 35
pyunity.values.vector (*module*), 36
pyunity.window (*module*), 41
pyunity.window.glfwWindow (*module*), 40
pyunity.window.glutWindow (*module*), 40
pyunity.window.sdl2Window (*module*), 41
pyunity.window.templateWindow (*module*), 41
PyUnityException, 49

Q

Q (*pyunity.input.KeyCode attribute*), 58
quad() (*pyunity.meshes.Mesh static method*), 64
Quaternion (*class in pyunity.values.quaternion*), 34
QuaternionDiff (*class in pyunity.values.quaternion*), 35
quit() (*pyunity.window.ABCWindow method*), 42
quit() (*pyunity.window.glfwWindow.Window method*), 40
quit() (*pyunity.window.glutWindow.Window method*), 41
quit() (*pyunity.window.sdl2Window.Window method*), 41
quit() (*pyunity.window.templateWindow.Window method*), 41

R

R (*pyunity.input.KeyCode attribute*), 58
radius (*pyunity.physics.core.SphereCollider attribute*), 27
rect (*pyunity.gui.Text attribute*), 56
RectAnchors (*class in pyunity.gui*), 52
Rectangle() (*pyunity.gui.RectOffset static method*), 53
RectData (*class in pyunity.gui*), 52
RectOffset (*class in pyunity.gui*), 53
RectTransform (*class in pyunity.gui*), 53
refresh() (*pyunity.settings.Database method*), 67
RegisterLight() (*pyunity.scenes.scene.Scene method*), 30
reimport_file() (*pyunity.files.Project method*), 52
RelativeTo() (*pyunity.gui.RectAnchors method*), 52
Remove() (*pyunity.scenes.scene.Scene method*), 30
RemoveAllScenes() (*in module pyunity.scenes.sceneManager*), 33
RemoveComponent() (*pyunity.core.Component method*), 47
RemoveComponent() (*pyunity.core.GameObject method*), 45
RemoveComponents() (*pyunity.core.Component method*), 47
RemoveComponents() (*pyunity.core.GameObject method*), 46
RemoveScene() (*in module pyunity.scenes.sceneManager*), 32
Render() (*pyunity.core.MeshRenderer method*), 49
Render() (*pyunity.render.Camera method*), 66
Render() (*pyunity.scenes.scene.Scene method*), 32
Render2D() (*pyunity.render.Camera method*), 67
ReparentTo() (*pyunity.core.Transform method*), 48
ResetStream() (*in module pyunity.logger*), 63
Resize() (*pyunity.render.Camera method*), 66
ResolveCollision() (*pyunity.physics.core.CollManager method*), 29
restitution (*pyunity.physics.core.PhysicMaterial attribute*), 26
RGB (*class in pyunity.values.texture*), 35
Right (*pyunity.gui.TextAlign attribute*), 55
Right (*pyunity.input.KeyCode attribute*), 59
Right (*pyunity.input.MouseCode attribute*), 60
right() (*pyunity.values.vector.Vector2 static method*), 38
right() (*pyunity.values.vector.Vector3 static method*), 39

Rigidbody (*class in pyunity.physics.core*), 28
rootGameObjects (*pyunity.scenes.scene.Scene attribute*), 30
RotateVector() (*pyunity.values.quaternion.Quaternion method*), 35
rotation (*pyunity.core.Transform attribute*), 48
rotation (*pyunity.gui.RectTransform attribute*), 53
rounded (*pyunity.values.vector.Vector2 attribute*), 37
rounded (*pyunity.values.vector.Vector3 attribute*), 39

S

S (*pyunity.input.KeyCode attribute*), 58
Save() (*in module pyunity.logger*), 63
save_mat() (*pyunity.files.Project method*), 52
SaveAllScenes() (*in module pyunity.loader*), 62
SaveMesh() (*in module pyunity.loader*), 62
SaveObj() (*in module pyunity.loader*), 61
SaveScene() (*in module pyunity.loader*), 62
SaveSceneToProject() (*in module pyunity.loader*), 62
scale (*pyunity.core.Transform attribute*), 48
Scene (*class in pyunity.scenes.scene*), 30
scene (*pyunity.core.Component attribute*), 47
schedule_update() (*pyunity.window.glutWindow.Window method*), 40
Screen (*class in pyunity.render*), 67
Scripts (*class in pyunity.files*), 50
sdl2Check() (*in module pyunity.window*), 42
SetBackward() (*pyunity.values.quaternion.Quaternion method*), 35
SetCenter() (*pyunity.gui.RectOffset method*), 53
SetClip() (*pyunity.audio.AudioSource method*), 43
setFloat() (*pyunity.render.Shader method*), 65
setImg() (*pyunity.files.Texture2D method*), 51
setInt() (*pyunity.render.Shader method*), 65
setMat4() (*pyunity.render.Shader method*), 65
SetPoint() (*pyunity.gui.RectAnchors method*), 52
SetPoint() (*pyunity.gui.RectOffset method*), 53
SetStream() (*in module pyunity.logger*), 63
setup_buffers() (*pyunity.render.Camera method*), 66
setVec3() (*pyunity.render.Shader method*), 65
SetWindowProvider() (*in module pyunity.window*), 42
Shader (*class in pyunity.render*), 65
ShowInInspector (*class in pyunity.core*), 46
SingleComponent (*class in pyunity.core*), 47
size (*pyunity.gui.Font attribute*), 55
size (*pyunity.physics.core.AABBoxCollider attribute*), 27
Skybox (*class in pyunity.files*), 51
Space (*pyunity.input.KeyCode attribute*), 58
Special (*class in pyunity.logger*), 62
SphereCollider (*class in pyunity.physics.core*), 27
Spot (*pyunity.core.LightType attribute*), 49
Start() (*pyunity.files.Behaviour method*), 50
Start() (*pyunity.scenes.scene.Scene method*), 31
Start() (*pyunity.values.other.Clock method*), 34
start() (*pyunity.window.ABCWindow method*), 42
start() (*pyunity.window.glfwWindow.Window method*), 40
start() (*pyunity.window.glutWindow.Window method*), 40
start() (*pyunity.window.sdl2Window.Window method*), 41
start() (*pyunity.window.templateWindow.Window method*), 41
start_scripts() (*pyunity.scenes.scene.Scene method*), 31
state (*pyunity.gui.Button attribute*), 54
Step() (*pyunity.physics.core.CollManager method*), 29
Stop() (*pyunity.audio.AudioSource method*), 43

T

T (*pyunity.input.KeyCode attribute*), 58
Tag (*class in pyunity.core*), 44
tag (*pyunity.core.GameObject attribute*), 45
tag (*pyunity.core.Tag attribute*), 44
tagName (*pyunity.core.Tag attribute*), 44
texcoords (*pyunity.meshes.Mesh attribute*), 64
Text (*class in pyunity.gui*), 55
text (*pyunity.gui.Text attribute*), 55
TextAlign (*class in pyunity.gui*), 55
texture (*pyunity.gui.Image2D attribute*), 54
texture (*pyunity.gui.Text attribute*), 56
texture (*pyunity.values.texture.Material attribute*), 35
Texture2D (*class in pyunity.files*), 51
to_hsv() (*pyunity.values.texture.HSV method*), 36
to_hsv() (*pyunity.values.texture.RGB method*), 36
to_rgb() (*pyunity.values.texture.HSV method*), 36
to_rgb() (*pyunity.values.texture.RGB method*), 36
to_string() (*pyunity.values.texture.Color method*), 35
todict() (*pyunity.settings.LiveDict method*), 67
Transform (*class in pyunity.core*), 47
transform (*pyunity.core.Component attribute*), 47
transform (*pyunity.core.GameObject attribute*), 45
transform (*pyunity.files.Behaviour attribute*), 50
triangles (*pyunity.meshes.Mesh attribute*), 63
type (*pyunity.core.HideInInspector attribute*), 46
type (*pyunity.core.Light attribute*), 49
type (*pyunity.core.ShowInInspector attribute*), 46

U

U (*pyunity.input.KeyCode attribute*), 58

UnixFontLoader (*class in pyunity.gui*), 55
UnPause() (*pyunity.audio.AudioSource method*), 43
Up (*pyunity.input.KeyCode attribute*), 59
UP (*pyunity.input.KeyState attribute*), 57
up() (*pyunity.values.vector.Vector2 static method*), 38
up() (*pyunity.values.vector.Vector3 static method*), 39
Update() (*pyunity.files.Behaviour method*), 50
Update() (*pyunity.gui.Canvas method*), 52
update() (*pyunity.scenes.scene.Scene method*), 32
update() (*pyunity.settings.Database method*), 67
update() (*pyunity.settings.LiveDict method*), 67
update_scripts() (*pyunity.scenes.scene.Scene method*), 31
UpdateAxes() (*pyunity.input.Input class method*), 61
use() (*pyunity.files.Skybox method*), 52
use() (*pyunity.files.Texture2D method*), 51
use() (*pyunity.render.Shader method*), 66
UseShader() (*pyunity.render.Camera method*), 66

V

V (*pyunity.input.KeyCode attribute*), 58
values() (*pyunity.settings.LiveDict method*), 67
Vector (*class in pyunity.values.vector*), 36
Vector2 (*class in pyunity.values.vector*), 36
Vector3 (*class in pyunity.values.vector*), 38
velocity (*pyunity.physics.core.Rigidbody attribute*), 28
verts (*pyunity.meshes.Mesh attribute*), 63

W

W (*pyunity.input.KeyCode attribute*), 58
Window (*class in pyunity.window.glfwWindow*), 40
Window (*class in pyunity.window.glutWindow*), 40
Window (*class in pyunity.window.sdl2Window*), 41
Window (*class in pyunity.window.templateWindow*), 41
WinFontLoader (*class in pyunity.gui*), 54
write_project() (*pyunity.files.Project method*), 52

X

X (*pyunity.input.KeyCode attribute*), 58

Y

Y (*pyunity.input.KeyCode attribute*), 58

Z

Z (*pyunity.input.KeyCode attribute*), 58
zero() (*pyunity.values.vector.Vector2 static method*), 38
zero() (*pyunity.values.vector.Vector3 static method*), 39