PyUnity

Release 0.9.0

The PyUnity Team

Feb 05, 2023

CONTENTS

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

CHAPTER

ONE

VERSION 0.9.0 (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.

1.1 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.

1.2 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
> pip 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
> pip install .
```

Its only dependencies are PyOpenGL, PySDL2, Pillow and PyGLM. Microsoft Visual C++ Build Tools are required on Windows for building yourself. GLFW can be optionally installed if you would like to use the GLFW window provider.

1.3 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 then make a pull request.

CHAPTER

TWO

RELEASES

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

2.1 Releases

2.1.1 v0.8.3

Bugfix regarding Quaternion.eulerAngles.

To set the rotation of the camera using Euler angles, use `scene.mainCamera.localRotation.SetBackward(Vector3(...))`.

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.9.0

2.1.2 v0.8.2

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

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.8.2

2.1.3 v0.8.1

Bugfix regarding camera position updating and input axes.

Download source code at https://github.com/pyunity/pyunity/releases/tag/0.8.1

2.1.4 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

2.1.5 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

2.1.6 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

2.1.7 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

2.1.8 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

2.1.9 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

2.1.10 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

2.1.11 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

2.1.12 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

2.1.13 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

2.1.14 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

2.1.15 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

2.1.16 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
> pip install .
```

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

2.1.17 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

2.1.18 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

2.1.19 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

2.1.20 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

CHAPTER

THREE

TUTORIALS

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

3.1 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.

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

3.1.1 Tutorial 1: The Basics

Table of Contents

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.

3.1.2 Tutorial 2: Rendering in Scenes

Table of Contents

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))
```

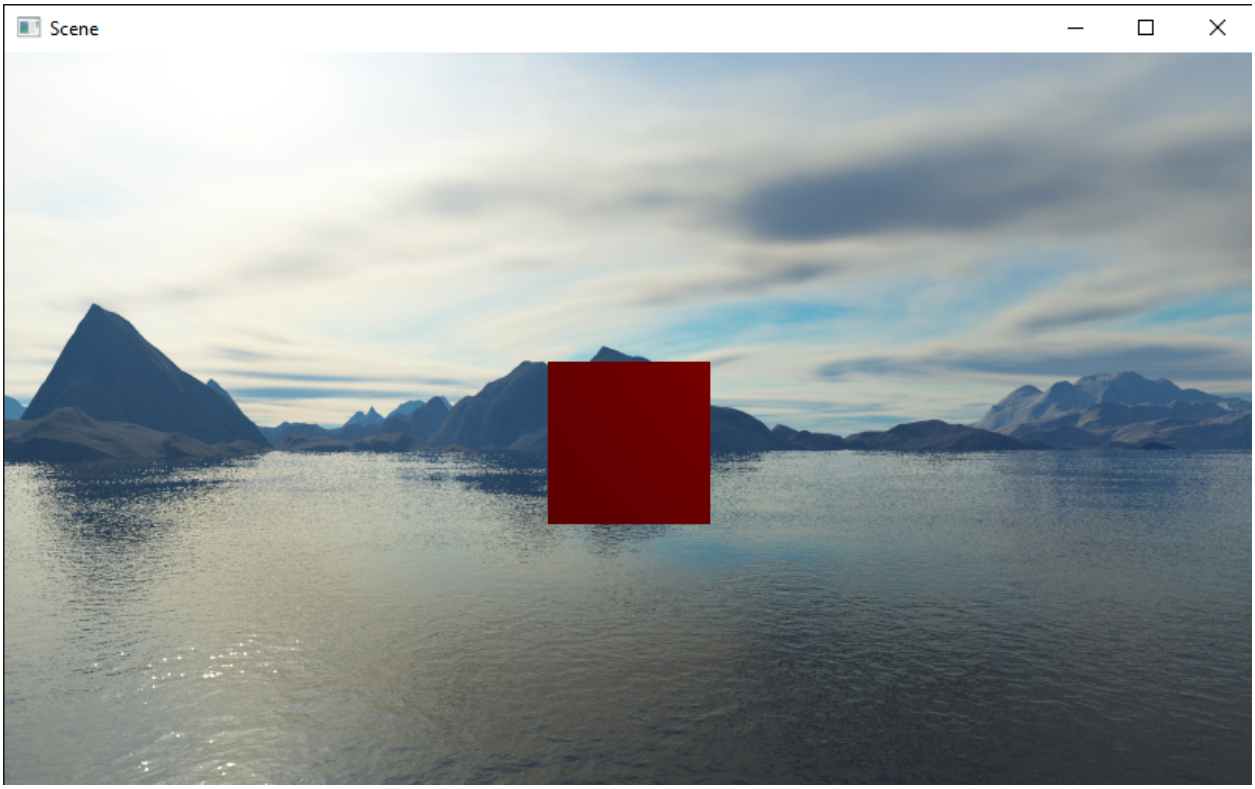
Here I used a red material. Finally we need to add the cube to our scene, otherwise we can't see it in the window:

```
>>> 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)
```

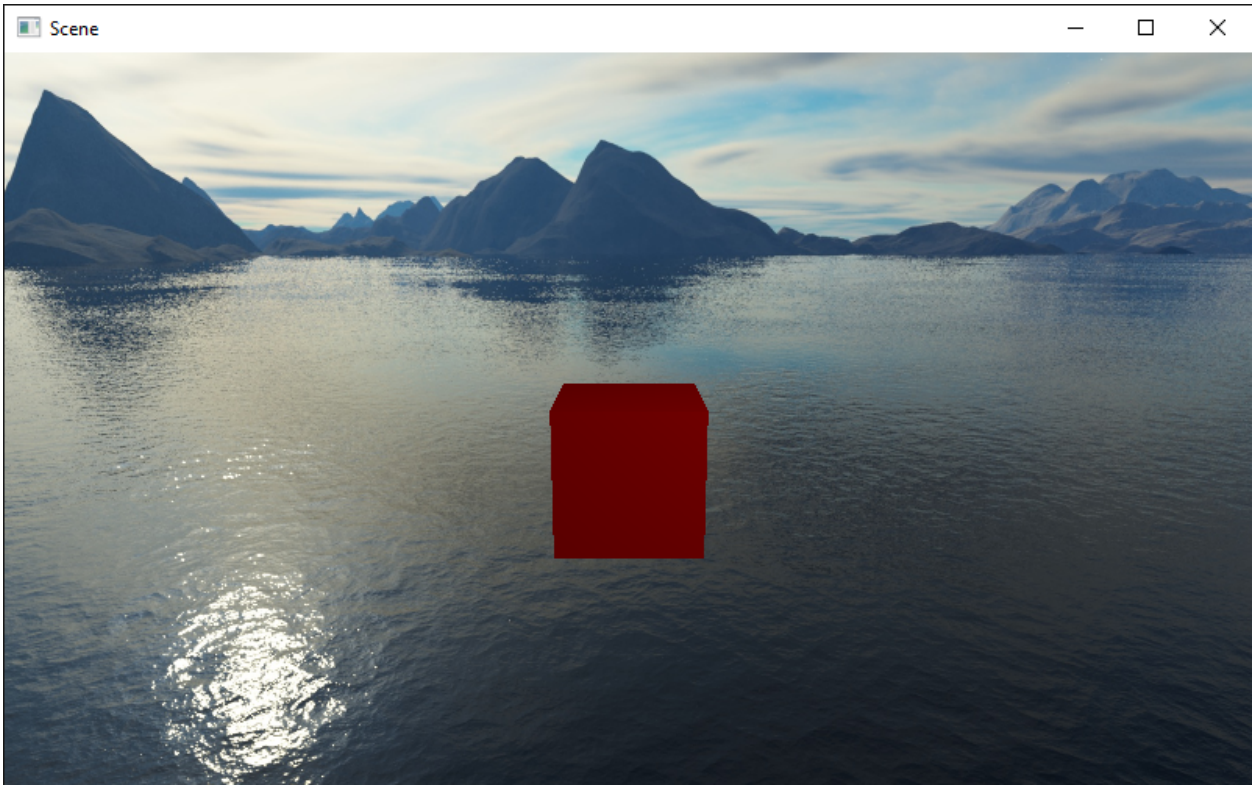
Then, to run our scene, we use `SceneManager.LoadScene(scene)`. And now we have a cube:



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)
```

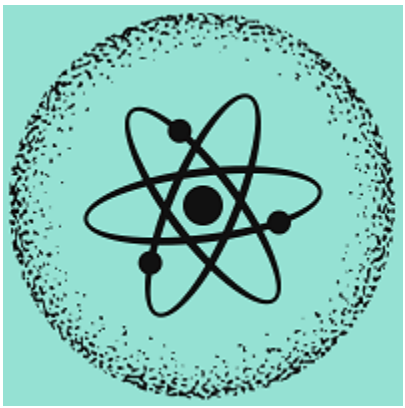
Now we can see it better:



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")
```

(continues on next page)

(continued from previous page)

```
renderer = cube.AddComponent(MeshRenderer)
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 will be covering scripts and how to make a `Behaviour`.

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

3.1.3 Tutorial 3: Scripts and Behaviours

Table of Contents

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):
...         Logger.Log(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))
>>> 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):
        self.a += 1
        if self.a == 10:
            Logger.Log(self.transform.position - self.lastPos)
            Logger.Log(self.transform.rotation.conjugate * self.lastRot)
            Logger.Log(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.

3.1.4 Tutorial 4: 2D

Table of Contents

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.

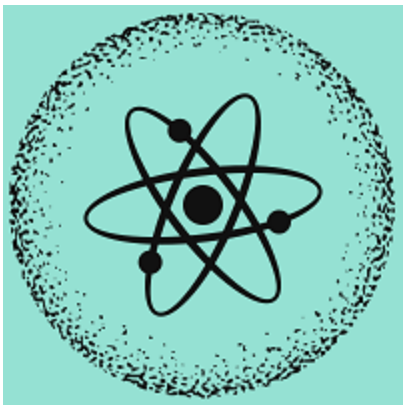
```
canvas = GameObject("Canvas")
canvas.AddComponent(Canvas)
img = GameObject("Image", canvas)
# And so on...
```

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

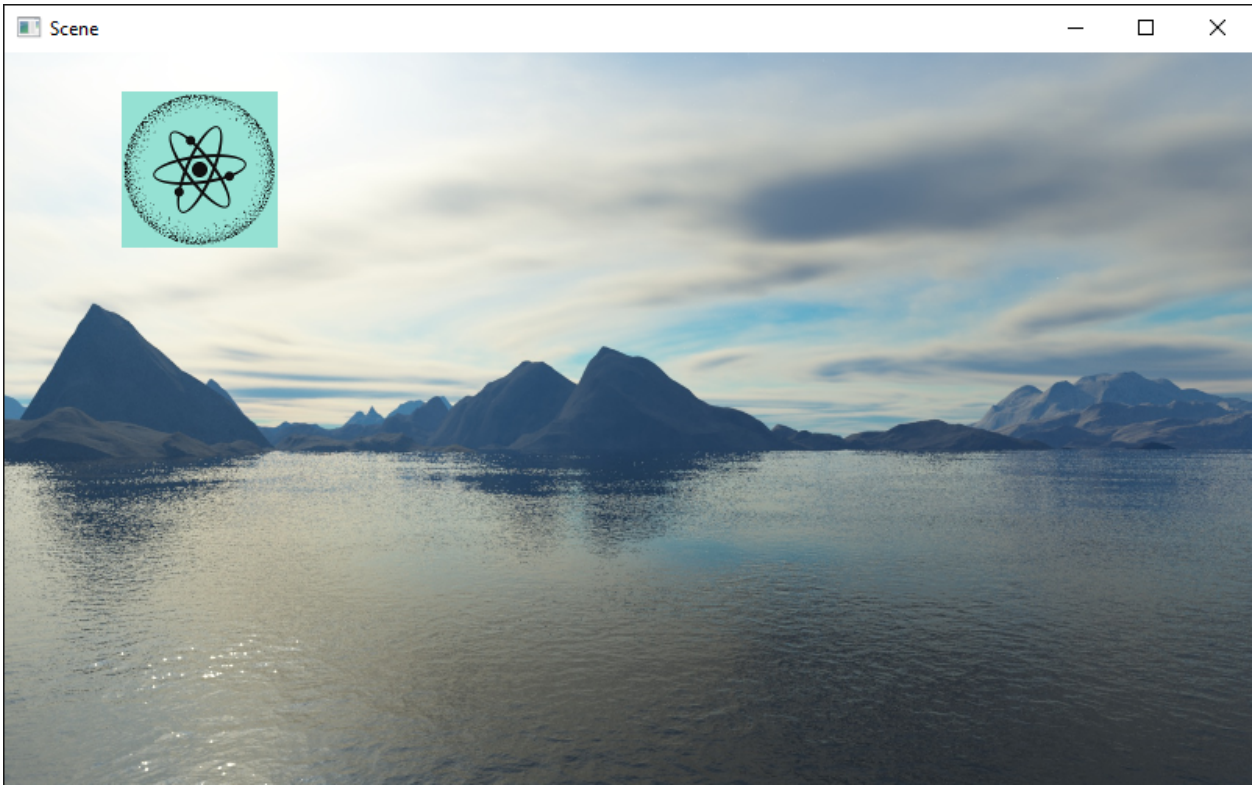
Code

```
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:

```
class CallbackReceiver(Component):
    def callback():
        Logger.Log("Clicked")

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

(continues on next page)

(continued from previous page)

```
receiver = gameObject.AddComponent(CallbackReceiver)
button.callback = Event(receiver.callback)
```

`Button.callback` must be an `Event` object that contains a method of a component added to a GameObject. This is because when saving a PyUnity project, the Python code itself is not saved. It is also easier to reference a component and a method name in the saved scene file.

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):
        Logger.Log("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 each edge 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.

CHAPTER

FOUR

LINKS

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

4.1 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

CHAPTER

FIVE

LICENSE

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

5.1 License

CHAPTER

SIX

API DOCUMENTATION

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

6.1 API Documentation

Information on specific functions, classes, and methods.

6.1.1 Subpackages

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.physics package

Submodules

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.physics.config module

Source code: pyunity/physics/config.py

---

pyunity.physics.config.**gravity = Vector3(0, -9.81, 0)**

Type: *Vector3*

Gravitational constant (9.81 m/s^2)

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.physics.core module

Source code: pyunity/physics/core.py

Core classes of the PyUnity physics engine.

pyunity.physics.core.**Infinity = inf**

Type: float

A representation of infinity

class pyunity.physics.core.**PhysicMaterial**

Bases: object

Class to store data on a collider's material.

Parameters

- **restitution** (*float*) – Bounciness of the material
- **friction** (*float*) – Friction of the material

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**

Bases: object

Class to store collision data.

Parameters

- **a** (Collider) – The first collider
- **b** (Collider) – The second collider
- **point** (Vector3) – The collision point
- **normal** (Vector3) – The collision normal
- **penetration** (*float*) – How much the two colliders overlap

class pyunity.physics.core.**Collider**

Bases: *Component*

Collider base class.

offset = None

The offset from the centre of the Collider

Type
Vector3

supportPoint(*direction*)

property pos

property rot

class pyunity.physics.core.**SphereCollider**

Bases: *Collider*

A spherical collider that cannot be deformed.

radius = 0.0

The radius of the SphereCollider

Type
Vector3

SetSize(*radius*, *offset*)

Sets the size of the collider.

Parameters

- **radius** (*float*) – The radius of the collider.
- **offset** (Vector3) – Offset of the collider.

property min

property max

collidingWith(*other*)

supportPoint(*direction*)

class pyunity.physics.core.**BoxCollider**

Bases: *Collider*

An axis-aligned box collider that cannot be deformed.

size = None

The distance between two farthest vertices of the collider

Type
Vector3

SetSize(*size*, *offset*)

Sets the size of the collider.

Parameters

- **size** (Vector3) – The dimensions of the collider.
- **offset** (Vector3) – Offset of the collider.

property min

property max

collidingWith(*other*)

supportPoint(*direction*)

class pyunity.physics.core.**Rigidbody**

Bases: *Component*

Class to let a GameObject follow physics rules.

velocity = None

Velocity of the Rigidbody

Type
Vector3

rotVel = None

Rotational velocity of the Rigidbody

Type
Vector3

force = None

Force acting on the Rigidbody. Reset every frame.

Type
Vector3

torque = None

Rotational force acting on the Rigidbody. Reset every frame.

Type
Vector3

physicMaterial = <pyunity.physics.core.PhysicMaterial object at 0x7f7af4c4bd30>

Physics material of the Rigidbody

Type
PhysicMaterial

property mass

Mass of the Rigidbody. Defaults to 100

property inertia

property pos

property rot

Move(*dt*)

Moves all colliders on the GameObject by the Rigidbody's velocity times the delta time.

Parameters
dt (*float*) – Time to simulate movement by

MovePos(*offset*)

Moves the rigidbody and its colliders by an offset.

Parameters
offset (*Vector3*) – Offset to move

AddForce(*force*, *point=Vector3(0, 0, 0)*)

Apply a force to the center of the Rigidbody.

Parameters

- **force** (Vector3) – Force to apply
- **point** (Vector3, *optional*) – Point relative to center of mass in local space to apply force at

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.

Parameters

impulse (Vector3) – Impulse to apply

Notes

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

class pyunity.physics.core.**SupportPoint**

Bases: *IgnoredMixin*

class pyunity.physics.core.**Triangle**

Bases: *IgnoredMixin*

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

Bases: *IgnoredMixin*

Manages the collisions between all colliders.

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

static supportPoint(*a*, *b*, *direction*)

static nextSimplex(*args*)

static lineSimplex(*args*)

static triSimplex(*args*)

static tetraSimplex(*args*)

static gjk(*a*, *b*)

`static epa`(*a*, *b*)

`static AddEdge`(*edges*, *a*, *b*)

`static barycentric`(*p*, *a*, *b*, *c*)

`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.

Parameters
scene (Scene) – Scene to search for physics info

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

Parameters

- **a** (Rigidbody) – Rigidbody 1
- **b** (Rigidbody) – Rigidbody 2

Returns
Restitution

Return type
float

`CheckCollisions`()

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

`ResolveCollisions`(*a*, *b*, *point*, *restitution*, *normal*, *penetration*)

`correctInf`(*a*, *b*, *correction*, *target*)

`Step`(*dt*)

Steps through the simulation at a given delta time.

Parameters
dt (*float*) – Delta time to step

Notes

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

Module contents

Source code: pyunity/physics/__init__.py

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(BoxCollider)
>>> collider.SetSize(-Vector3.one(), Vector3.one())
>>> 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

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.scenes.runner module

Source code: pyunity/scenes/runner.py

---

exception pyunity.scenes.runner.**ChangeScene**

Bases: Exception

class pyunity.scenes.runner.**Runner**

Bases: object

setScene(*scene*)

setNext(*scene*)

open()

setup()

load()

start()

quit()

class pyunity.scenes.runner.**WindowRunner**

Bases: *Runner*

open()

setup()

load()

start()

quit()

class pyunity.scenes.runner.**NonInteractiveRunner**

Bases: *Runner*

load()

pyunity.scenes.runner.**newRunner**()

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.scenes.scene module

Source code: pyunity/scenes/scene.py

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.

pyunity.scenes.scene.**createTask**(*loop*, *coro*, **args*)

class pyunity.scenes.scene.**Scene**

Bases: *Asset*

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.

GetAssetFile(*gameObject*)

SaveAsset(*ctx*)

static Bare(*name*)

Create a bare scene.

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

Returns
A bare scene with no GameObjects

Return type
Scene

property rootGameObjects

All GameObjects which have no parent

Add(*gameObject*)

Add a GameObject to the scene.

Parameters
gameObject (GameObject) – The GameObject to add.

AddMultiple(**args*)

Add GameObjects to the scene.

Parameters
***args** (*list*) – A list of GameObjects to add.

Destroy(*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

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.

FindComponent(*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

FindComponents(*component*)

Finds all matching Components that are in the Scene.

Parameters
component (*type*) – Component type

Returns
List of the matching Components

Return type
list

`insideFrustrum`(*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

`startOpenGL`()

`startScripts`()

`startLoop`()

`Start`()
Start the internal parts of the Scene. Deprecated in 0.9.0.

`updateScripts`(*loop*)
Updates all scripts in the scene.

`updateFixed`(*loop*)

`Render`(*loop=None*)
Call the appropriate rendering functions of the Main Camera.

Parameters
loop (EventLoop) – Event loop to run `Behaviour.OnPreRender()` and `Behaviour.OnPostRender()` in. If None, the above methods will not be called.

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

pyunity.scenes.sceneManager module

Source code: pyunity/scenes/sceneManager.py

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
PyUnityException – 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.**stopWindow**()

pyunity.scenes.sceneManager.**CurrentScene**()

Gets the current scene being run

Module contents

Source code: pyunity/scenes/__init__.py

Module to create and load Scenes.

Attention: You are viewing PyUnity docs under the develop branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.values package

Submodules

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.values.abc module

Source code: pyunity/values/abc.py

---

exception pyunity.values.abc.**ABCException**

Bases: Exception

exception pyunity.values.abc.**ABCMessage**

Bases: *ABCException*

class pyunity.values.abc.**abstractmethod**

Bases: *IncludeMixin*

static getargs(*func*)

class pyunity.values.abc.**abstractproperty**

Bases: *abstractmethod*

class pyunity.values.abc.**ABCMeta**

Bases: type

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.values.mathf module

Source code: pyunity/values/mathf.py

---

pyunity.values.mathf.**Acos**(*num*)

Returns the angle whose cosine is num. Return value is in radians.

Parameters
num (*float*) – Input number

pyunity.values.mathf.**Asin**(*num*)

Returns the angle whose sine is num. Return value is in radians.

Parameters
num (*float*) – Input number

pyunity.values.mathf.**Atan**(*num*)

Returns the angle whose tangent is num. Return value is in radians.

Parameters
num (*float*) – Input number

pyunity.values.mathf.**Atan2**(*x*, *y*)

Returns the two-argument arctangent of x/y.

Parameters

- **x** (*float*) – Input x
- **y** (*float*) – Input y

pyunity.values.mathf.**Ceil**(*num*)

Returns the smallest integer greater than or equal to num.

Parameters
num (*float*) – Input number

pyunity.values.mathf.**Clamp**(*num*, *a*, *b*)

Returns a if num is smaller than or equal to a, b if num is greater than or equal to b, or num if it is between a and b.

Parameters

- **num** (*float*) – Input number
- **a** (*float*) – Lower bound
- **b** (*float*) – Upper bound

pyunity.values.mathf.**Clamp01**(*num*)

Returns num clamped between 0 and 1.

Parameters
num (*float*) – Input number

pyunity.values.mathf.**Cos**(*num*)

Returns the cosine of num. Must be passed in radians.

Parameters
num (*float*) – Input number

pyunity.values.mathf.**DeltaAngle**(*a*, *b*)

Calculates the shortest difference between two given angles given in degrees.

Parameters

- **a** (*float*) – Input a
- **b** (*float*) – Input b

pyunity.values.mathf.**Exp**(*num*)

Returns e raised to the power of num.

Parameters
num (*float*) – Exponent

pyunity.values.mathf.**Floor**(*num*)

Returns the largest integer smaller than or equal to num.

Parameters
num (*float*) – Input number

pyunity.values.mathf.**InverseLerp**(*num*, *a*, *b*)

Determines where num lies between two points a and b.

Parameters

- **num** (*float*) – Number to check
- **a** (*float*) – Lower bound
- **b** (*float*) – Upper bound

pyunity.values.mathf.**Lerp**(*num*, *a*, *b*)

Linearly interpolates between a and b by num.

Parameters

- **num** (*float*) – Amount to interpolate by
- **a** (*float*) – Lower bound
- **b** (*float*) – Upper bound

pyunity.values.mathf.**LerpUnclamped**(*num*, *a*, *b*)

Linearly interpolates between a and b by num with no limit for num.

Parameters

- **num** (*float*) – Amount to interpolate by
- **a** (*float*) – Lower bound
- **b** (*float*) – Upper bound

pyunity.values.mathf.**Log**(*num*)

Returns the base 10 logarithm of num.

Parameters
num (*float*) – Input number

pyunity.values.mathf.**Sign**(*num*)

Returns the sign of num (either -1 or 1, or 0 if num is 0).

Parameters
num (*float*) – Input number

pyunity.values.mathf.**Sin**(*num*)

Returns the sine of num. Must be passed in radians.

Parameters
num (*float*) – Input number

pyunity.values.mathf.**SmoothStep**(*num*)

Used in conjunction with *Lerp()* to smoothly interpolate between two numbers. This function takes a number between 0 and 1 and returns a number between 0 and 1, which has a steeper graph around 0.5 and a smoother graph near 0 and 1.

Parameters
num (*float*) – Input number (between 0 and 1)

Notes

This uses the mathematical equation f(x) = 3x^2 - 2x^3.

pyunity.values.mathf.**Sqrt**(*num*)

Returns the square root of num.

Parameters
num (*float*) – Input number

pyunity.values.mathf.**Tan**(*num*)

Returns the tangent of num. Must be passed in radians.

Parameters
num (*float*) – Input number

class pyunity.values.mathf.**SmoothDamper**

Bases: object

SmoothDamp(*current*, *target*, *smoothTime*, *dt*)

reset()

pyunity.values.other module

Source code: pyunity/values/other.py

---

class pyunity.values.other.**IgnoredMixin**

Bases: object

class pyunity.values.other.**IncludeMixin**

Bases: object

class pyunity.values.other.**IncludeInstanceMixin**

Bases: object

class pyunity.values.other.**Clock**

Bases: object

property fps

Start(*fps=None*)

Maintain()

class pyunity.values.other.**LockedLiteral**

Bases: object

class pyunity.values.other.**SavableStruct**

Bases: object

fromDict(*factory*, *attrs*, *instanceCheck=None*)

class pyunity.values.other.**StructEntry**

Bases: object

ignore = <pyunity.values.other.StructEntry._ignoredEntry object>

Type: _ignoredEntry

class pyunity.values.other.**ImmutableStruct**

Bases: type

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.values.quaternion module

Source code: pyunity/values/quaternion.py

Class to represent a rotation in 3D space.

class pyunity.values.quaternion.**Quaternion**

Bases: *LockedLiteral*

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

absDiff(*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

property 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*)

property angleAxisPair

Gets 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

property eulerAngles

Gets the Euler angles of the quaternion

static identity()

Identity quaternion representing no rotation

class pyunity.values.quaternion.**QuaternionDiff**

Bases: object

pyunity.values.vector module

Source code: pyunity/values/vector.py

---

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

Clamp a value between a minimum and a maximum

pyunity.values.vector.**conv**(*num*)

Convert float to string and removing decimal place as necessary.

class pyunity.values.vector.**Vector**

Bases: *LockedLiteral*

abs()

length()

property intTuple

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

replace(*num*, *value*)

class pyunity.values.vector.**Vector2**

Bases: *Vector*

replace(*num*, *value*)

copy()

Makes a copy of the Vector2

getLengthSqrd()

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

property length

Gets 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

getDistance(*other*)

The distance between this vector and the other vector

Returns
The distance

Return type
float

getDistSqrd(*other*)

The distance between this vector and the other vector, squared. It is more efficient to call this than to call *Vector2.getDistance()* and square it.

Returns
The squared distance

Return type
float

clamp(*min*, *max*)

Returns a clamped 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

Returns
A vector inside or on the surface of the bounding box specified by min and max.

Return type
Vector3

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**

Bases: *Vector*

`replace`(*num*, *value*)

`copy`()

Makes a copy of the Vector3

Returns
A shallow copy of the vector

Return type
Vector3

`getLengthSqrd`()

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

`property length`

Gets 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

`getDistance`(*other*)

The distance between this vector and the other vector

Returns
The distance

Return type
float

`getDistSqrd`(*other*)

The distance between this vector and the other vector, squared. It is more efficient to call this than to call *Vector3.getDistance()* and square it.

Returns
The squared distance

Return type
float

`clamp`(*min*, *max*)

Returns a clamped 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

Returns
A vector inside or on the surface of the bounding box specified by min and max.

Return type
Vector3

`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

Source code: pyunity/values/__init__.py

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.window package

Subpackages

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.window.providers package

Module contents

Source code: pyunity/window/providers/__init__.py

---

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

pyunity.window.providers.**sort**(*x*)

pyunity.window.providers.**getProviders**()

Submodules

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.window.abc module

Source code: pyunity/window/abc.py

Abstract base class for window providers. Imported into `pyunity.Window`.

class pyunity.window.abc.**ABCWindow**

Bases: `object`

Abstract base class that window providers should subclass and define methods of.

Parameters

name (*str*) – Name to display on window title.

setResize(*resize*)

Sets the resize function, which has a signature of `def Resize(width, height):`. This should be bound to the appropriate callback handler of the window.

Parameters
resize (*function*) – Resize function

getMouse(*mousecode*, *keystate*)

Get mouse state for specific mouse button and state.

Parameters

- **mousecode** (MouseCode) – Query button
- **keystate** (KeyState) – Query state

Returns
If the query button matches the query state. Note that both `KeyState.PRESS` and `KeyState.DOWN` will match a query of `KeyState.PRESS` because when a button is first hit it is still pressed down.

Return type
bool

Notes

A good starting point is this example function:

```
def getMouse(self, mousecode, keystate):
    mouse = mouseMap[mousecode]
    if keystate == KeyState.PRESS:
        if self.mouse[mouse] in [KeyState.PRESS, KeyState.DOWN]:
            return True
    if self.mouse[mouse] == keystate:
        return True
    return False
```

where `mouseMap` is a mapping of `MouseCode` to the window provider's own representation of mouse buttons, *self.mouse* is a mapping of the window provider's own representation of mouse buttons to `KeyState`. This makes it easy to both query and set the keystates of the mouse.

getKey(*keycode*, *keystate*)

Get key state for specific key and state.

Parameters

- **keycode** (KeyCode) – Query key
- **keystate** (KeyState) – Query state

Returns
If the query key matches the query state. Note that both `KeyState.PRESS` and `KeyState.DOWN` will match a query of `KeyState.PRESS` because when a key is first hit it is still pressed down.

Return type
bool

Notes

A good starting point is this example function:

```
def getKey(self, keycode, keystate):
    key = keyMap[keycode]
    if keystate == KeyState.PRESS:
        if self.keys[key] in [KeyState.PRESS, KeyState.DOWN]:
            return True
    if self.keys[key] == keystate:
        return True
    return False
```

where `keyMap` is a mapping of `KeyCode` to the window provider's own representation of keys, *self.keys* is a mapping of the window provider's own representation of keys to `KeyState`. This makes it easy to both query and set the keystates of the keyboard.

getMousePos()

Get a tuple of (x, y) representing the position of the mouse inside the window.

Returns
Mouse coordinates

Return type
tuple

refresh()

Refreshes and redraws the screen.

updateFunc()

Update the input of keys and mouse. Also checks to quit. Don't close the window in this method. Close it in *quit()* instead.

Raises
PyUnityExit – When the window should

quit()

Closes the window.

Module contents

Source code: pyunity/window/__init__.py

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 `updateFunc`, 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.**GetWindowProvider**()

Gets an appropriate window provider to use

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

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

6.1.2 Submodules

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.audio module

Source code: pyunity/audio.py

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.

class pyunity.audio.**AudioClip**

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**

Bases: *Component*

Manages playback on an AudioSource.

clip = None

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

Type
AudioClip

playOnStart = False

Whether it plays on start or not.

Type
bool

loop = False

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.

property Playing

Gets if the AudioSource is playing.

class pyunity.audio.**AudioListener**

Bases: *SingleComponent*

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

Source code: pyunity/core.py

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 0.9.0
>>> 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.MeshRenderer object at 0x00000170E4199CF0>]
>>> 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**

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

`tags = ['Default']`
Type: list

List of current tags

`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.SavesProjectID`
Bases: object

`class pyunity.core.GameObject`
Bases: *SavesProjectID*

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

classmethod 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 *GameObject.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. To remove all matching components, use *GameObject.RemoveComponents()*.

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**

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

Set when `Component.__init_subclass__` is excecuted

Type
NoneType

class pyunity.core.**ShowInInspector**

Bases: *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.**ComponentType**

Bases: *ABCMeta*

class pyunity.core.**Component**

Bases: *SavesProjectID*

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

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

GetComponent(*component*)

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

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

RemoveComponent(*component*)

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

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

GetComponents(*component*)

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

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

RemoveComponents(*component*)

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

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

property scene

Get the scene of the GameObject.

class pyunity.core.**SingleComponent**

Bases: *Component*

Represents a component that can be added only once.

class pyunity.core.**Transform**

Bases: *SingleComponent*

Class to hold data about a GameObject's transformation.

gameObject

GameObject that the component belongs to.

Type
GameObject

localPosition = None

Position of the Transform in local space.

Type
Vector3

localRotation = None

Rotation of the Transform in local space.

Type
Quaternion

`localScale = None`

Scale of the Transform in local space.

Type
Vector3

`parent = None`

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

`property position`

Position of the Transform in world space.

`property rotation`

Rotation of the Transform in world space.

`property localEulerAngles`

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

`property eulerAngles`

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

`property 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

pyunity.errors module

Source code: pyunity/errors.py

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: *PyUnityException*

Class for PyUnity exceptions relating to components.

exception pyunity.errors.**GameObjectException**

Bases: *PyUnityException*

Class for PyUnity exceptions relating to GameObjects.

exception pyunity.errors.**ProjectParseException**

Bases: *PyUnityException*

Class for PyUnity project parsing exceptions.

exception pyunity.errors.**PyUnityExit**

Bases: *PyUnityException*

Exception for breaking out of the main loop and shutting PyUnity down.

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.events module

Source code: pyunity/events.py

---

class pyunity.events.**Event**

Bases: object

trigger()

callSoon()

pyunity.events.**wrap**(*func*)

class pyunity.events.**EventLoopManager**

Bases: object

current = None

Type: None

exceptions = []

Type: list

exceptionLock = <unlocked _thread.RLock object owner=0 count=0>

Type: RLock

waitingLock = <unlocked _thread.RLock object owner=0 count=0>

Type: RLock

schedule(**funcs*, *main=False*, *ups=None*, *waitFor=None*)

addLoop(*loop*)

start()

quit()

class pyunity.events.**EventLoop**

Bases: _UnixSelectorEventLoop

async shutdown(*signal=None*)

handleException(*context*)

pyunity.events.**StartCoroutine**(*coro*)

class pyunity.events.**WaitForSeconds**

Bases: object

class pyunity.events.**WaitForEventLoop**

Bases: object

class pyunity.events.**WaitForUpdate**

Bases: *WaitForEventLoop*

class pyunity.events.**WaitForFixedUpdate**

Bases: *WaitForEventLoop*

class pyunity.events.**WaitForRender**

Bases: *WaitForEventLoop*

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.files module

Source code: pyunity/files.py

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**

Bases: *Component*

Base class for behaviours that can be scripted.

Awake()

Called every time a scene is loaded up, regardless whether the Behaviour is enabled or not. Cannot be an `async` function.

async Start()

Called every time a scene is loaded up. Only called when the Behaviour is enabled. Can be either a normal function or an `async` function.

async Update(*dt*)

Called every frame. Can be either a normal function or an `async` function.

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

async FixedUpdate(*dt*)

Called every frame, in each physics step. Can be either a normal function or an `async` function.

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

async LateUpdate(*dt*)

Called every frame, after physics processing. Can be either a normal function or an async function.

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

async OnPreRender()

Called before rendering happens. Can be either a normal function or an async function.

async OnPostRender()

Called after rendering happens. Can be either a normal function or an async function.

OnDestroy()

Called at the end of each Scene. Cannot be an async function.

class pyunity.files.Scripts

Bases: object

Utility class for loading scripts in a folder.

template = 'from pyunity import *\n\nclass {}(Behaviour):\n async def Start(self):\n pass\n\n async def Update(self, dt):\n pass\n'

Type: str

var = {}

Type: dict

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 GenerateModule()

static LoadScript(*path*, *force=False*)

Loads a PyUnity script by path.

Parameters

- **path** (*Pathlike*) – A path to a PyUnity script
- **force** (*bool*) – Continue on error

Returns
The compiled PyUnity script

Return type
type

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.

static Reset()

class pyunity.files.**Asset**

Bases: *SavesProjectID*

GetAssetFile(*gameObject*)

SaveAsset(*ctx*)

class pyunity.files.**Texture2D**

Bases: *Asset*

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.

GetAssetFile(*gameObject*)

SaveAsset(*ctx*)

classmethod FromOpenGL(*texture*)

class pyunity.files.**Skybox**

Bases: `object`

Skybox model consisting of 6 images

names = ['right.jpg', 'left.jpg', 'top.jpg', 'bottom.jpg', 'front.jpg', 'back.jpg']
Type: list

points = [-1, 1, -1, -1, -1, -1, 1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, -1, 1, -1, -1, -1, -1, 1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, 1, -1, -1, 1, 1, -1, -1, 1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 1, -1, -1, -1, -1, 1, -1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 1, -1, -1, -1, -1, -1, 1, -1, 1, -1, 1, 1, -1, 1, 1, 1, 1, 1, 1, -1, 1, 1, -1, 1, -1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, -1, -1, -1, 1, 1, -1, 1]
Type: list

compile()

use()

class pyunity.files.**Prefab**

Bases: *Asset*

Prefab model

Contains(*obj*)

Instantiate(*scene=None*, *parent=None*, *position=Vector3(0, 0, 0)*, *rotation=Quaternion(1, 0, 0, 0)*, *scale=Vector3(1, 1, 1)*, *worldSpace=False*)

Instantiate this prefab.

Parameters

- **scene** (Scene, *optional*) – The scene to instantiate in. If None, the current scene is selected.
- **parent** (GameObject, *optional*) – The parent to instantiate the Prefab under. If None, the prefab will be instantiated at the root of the scene.
- **position** (Vector3, *optional*) – Position of the newly created GameObject, by default Vector3.zero()
- **rotation** (Quaternion, *optional*) – Rotation of the newly created GameObject, by default Quaternion.identity()
- **scale** (Vector3, *optional*) – Scale of the newly created GameObject, by default Vector3.one()
- **worldSpace** (*bool*, *optional*) – Whether the above 3 properties are world space or local space, by default False

Returns
The newly created GameObject

Return type
GameObject

Raises
PyUnityException – If `scene` is None but no scene is running

GetAssetFile(*gameObject*)

SaveAsset(*ctx*)

class pyunity.files.**ProjectSavingContext**

Bases: object

class pyunity.files.**File**

Bases: object

pyunity.files.**checkScene**(*func*)

class pyunity.files.**Project**

Bases: object

property assets

Write()

ImportFile(*file*, *uuid=None*, *write=True*)

ImportAsset(*asset*, *gameObject=None*, *filename=None*)

SetAsset(*file*, *obj*)

GetUuid(*obj*)

static FromFolder(*folder*)

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.gui module

Source code: pyunity/gui.py

---

pyunity.gui.**createTask**(*loop*, *coro*)

class pyunity.gui.**Canvas**

Bases: *Component*

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

Update(*loop*)

Check if any components have been hovered over.

class pyunity.gui.**RectData**

Bases: `object`

Class to represent a 2D rect.

Parameters

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

size()

SetPoint(*pos*)

Changes both the minimum and maximum points.

Parameters
pos (Vector2) – Point

class pyunity.gui.**RectAnchors**

Bases: *RectData*

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

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**

Bases: *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

class pyunity.gui.**RectTransform**

Bases: *SingleComponent*

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

anchors = None

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

Type
RectAnchors

offset = None

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

Type
RectOffset

pivot = None

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

Type
Vector2

rotation = 0.0

Rotation in degrees

Type
float

property parent

GetRect(*bb=None*)

Gets screen coordinates of the bounding box not including offset.

Parameters
bb (*Vector2, optional*) – Bounding box to base anchors off of

Returns
Screen coordinates

Return type
RectData

class pyunity.gui.**GuiComponent**

Bases: *Component*

A Component that represents a clickable area.

HoverUpdate()

class pyunity.gui.**NoResponseGuiComponent**

Bases: *GuiComponent*

A Component that blocks all clicks that are behind it.

async 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.**GuiRenderComponent**

Bases: *NoResponseGuiComponent*

A Component that renders something in its RectTransform.

flipX = 0

Type: int

flipY = 0

Type: int

PreRender()

class pyunity.gui.**Image2D**

Bases: *GuiRenderComponent*

A 2D image component, which is uninteractive.

texture = None

Texture to render

Type
Texture2D

depth = 0.0

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

Type
float

class pyunity.gui.**RenderTarget**

Bases: *GuiRenderComponent*

flipY = 1

Type: int

PreRender()

saveImg(*path*)

genBuffers(*force=False*)

setSize(*size*)

class pyunity.gui.**Button**

Bases: *GuiComponent*

A Component that calls a function when clicked.

callback = None

Callback function. Must be a method of a Component.

Type
Event

state = KeyState.UP

Which state triggers the callback

Type
KeyState

mouseButton = MouseCode.Left

Which mouse button triggers the callback

Type
MouseCode

pressed = False

If the button is pressed down or not

Type
bool

async HoverUpdate()

class pyunity.gui.**WinFontLoader**

Bases: _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: _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: *UnixFontLoader*

class pyunity.gui.**Font**

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: IntEnum

An enumeration.

Left = 1

Type: int

Center = 2

Type: int

Right = 3

Type: int

Top = 1

Type: int

Bottom = 3

Type: int

class pyunity.gui.**Text**

Bases: *GuiRenderComponent*

Component to render text.

font = None

Font object to render

Type
Font

text = Text

Contents of the Text

Type
str

color = None

Fill color

Type
Color

depth = 0.1

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

Type
float

centeredX = TextAlign.Left

How to align in the X direction

Type
TextAlign

centeredY = TextAlign.Center

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()*.

PreRender()

GenTexture()

Generate a Texture2D to render.

class pyunity.gui.**CheckBox**

Bases: *GuiComponent*

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

checked = False

Current state of the checkbox

Type
bool

async 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*.

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.input module

Source code: pyunity/input.py

---

class pyunity.input.**KeyState**

Bases: IntEnum

An enumeration.

UP = 1

Type: int

DOWN = 2

Type: int

PRESS = 3

Type: int

NONE = 4

Type: int

class pyunity.input.**KeyCode**

Bases: IntEnum

An enumeration.

A = 1

Type: int

B = 2

Type: int

C = 3

Type: int

D = 4

Type: int

E = 5

Type: int

F = 6

Type: int

G = 7

Type: int

H = 8

Type: int

I = 9

Type: int

J = 10

Type: int

K = 11

Type: int

L = 12

Type: int

M = 13

Type: int

N = 14

Type: int

O = 15

Type: int

P = 16

Type: int

Q = 17

Type: int

R = 18
Type: int

S = 19
Type: int

T = 20
Type: int

U = 21
Type: int

V = 22
Type: int

W = 23
Type: int

X = 24
Type: int

Y = 25
Type: int

Z = 26
Type: int

Space = 27
Type: int

Alpha0 = 28
Type: int

Alpha1 = 29
Type: int

Alpha2 = 30
Type: int

Alpha3 = 31
Type: int

Alpha4 = 32
Type: int

Alpha5 = 33
Type: int

Alpha6 = 34
Type: int

Alpha7 = 35
Type: int

Alpha8 = 36
Type: int

Alpha9 = 37

Type: int

F1 = 38

Type: int

F2 = 39

Type: int

F3 = 40

Type: int

F4 = 41

Type: int

F5 = 42

Type: int

F6 = 43

Type: int

F7 = 44

Type: int

F8 = 45

Type: int

F9 = 46

Type: int

F10 = 47

Type: int

F11 = 48

Type: int

F12 = 49

Type: int

Keypad0 = 50

Type: int

Keypad1 = 51

Type: int

Keypad2 = 52

Type: int

Keypad3 = 53

Type: int

Keypad4 = 54

Type: int

Keypad5 = 55

Type: int

Keypad6 = 56

Type: int

Keypad7 = 57

Type: int

Keypad8 = 58

Type: int

Keypad9 = 59

Type: int

Up = 60

Type: int

Down = 61

Type: int

Left = 62

Type: int

Right = 63

Type: int

class pyunity.input.**MouseCode**

Bases: IntEnum

An enumeration.

Left = 1

Type: int

Middle = 2

Type: int

Right = 3

Type: int

class pyunity.input.**KeyboardAxis**

Bases: object

getValue(*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

mousePosition = None

Type: None

classmethod **UpdateAxes**(*dt*)

Attention: You are viewing PyUnity docs under the `develop` branch. As such, they are only applicable if you installed from source. Go to https://docs.pyunity.x10.bz/en/latest/ for the most recent release.

pyunity.loader module

Source code: pyunity/loader.py

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 (*Pathlike*) – Name of file

Returns
A mesh of the object file

Return type
Mesh

pyunity.loader.**SaveObj**(*mesh*, *path*)

Save a PyUnity Mesh to a .obj file.

Parameters

- **mesh** (Mesh) – Mesh to save
- **path** (*Pathlike*) – Path to save mesh

pyunity.loader.**LoadStl**(*filename*)

Loads a .stl mesh to a PyUnity mesh.

Parameters
filename (*Pathlike*) – Path to PyUnity mesh.

Raises
PyUnityException – If the file format is incorrect

pyunity.loader.**SaveStl**(*mesh*, *path*)

Save a PyUnity Mesh to a .stl file.

Parameters

- **mesh** (Mesh) – Mesh to save
- **path** (*Pathlike*) – Path to save mesh

pyunity.loader.**LoadMesh**(*filename*)

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

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

Returns
Generated mesh

Return type
Mesh

pyunity.loader.**SaveMesh**(*mesh*, *path*)

Saves a mesh to a .mesh file for faster loading.

Parameters

- **mesh** (Mesh) – Mesh to save
- **path** (*Pathlike*) – Path to save mesh

class pyunity.loader.**Primitives**

Bases: object

Primitive preloaded meshes. Do not instantiate this class.

cube = <pyunity.meshes.Mesh object>

Type: *Mesh*

quad = <pyunity.meshes.Mesh object>

Type: *Mesh*

doubleQuad = <pyunity.meshes.Mesh object>

Type: *Mesh*

sphere = <pyunity.meshes.Mesh object>

Type: *Mesh*

capsule = <pyunity.meshes.Mesh object>

Type: *Mesh*

cylinder = <pyunity.meshes.Mesh object>

Type: *Mesh*

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

pyunity.loader.**parseString**(*string*, *project=None*)

pyunity.loader.**parseStringFallback**(*string*, *project*, *fallback*)

class pyunity.loader.**ObjectInfo**

Bases: object

class SkipConv

Bases: object

static convString(*v*)

pyunity.loader.**SaveMat**(*material*, *project*, *filename*)

pyunity.loader.**LoadMat**(*path*, *project*)

pyunity.loader.**SavePrefab**(*prefab*, *path*, *project*)

pyunity.loader.**LoadPrefab**(*path*, *project*)

pyunity.loader.**savable = (<class 'pyunity.meshes.Color'>, <class 'pyunity.values.vector.Vector3'>, <class 'pyunity.values.quaternion.Quaternion'>, <class 'bool'>, <class 'int'>, <class 'str'>, <class 'float'>, <class 'list'>, <class 'tuple'>, <class 'pyunity.core.SavesProjectID'>, <class 'pyunity.loader.ObjectInfo.SkipConv'>)**

Type: tuple

All savable types that will not be saved as UUIDs

pyunity.loader.**SaveGameObjects**(*gameObjects*, *data*, *project*)

pyunity.loader.**LoadObjectInfos**(*file*)

pyunity.loader.**instanceCheck**(*type_*, *value*)

pyunity.loader.**GetComponentMap**()

pyunity.loader.**LoadGameObjects**(*data*, *project*)

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

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

pyunity.loader.**GenerateProject**(*name*, *force=True*)

pyunity.loader.**SaveProject**(*project*)

pyunity.loader.**LoadProject**(*folder*, *remove=True*)

pyunity.loader.**LoadScene**(*sceneFile*, *project*)

pyunity.logger module

Source code: pyunity/logger.py

Utility functions to log output of PyUnity.

This will be imported as `pyunity.Logger`.

pyunity.logger.**getDataFolder**()

class pyunity.logger.**Level**

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**

Bases: object

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

class pyunity.logger.**Elapsed**

Bases: object

tick()

pyunity.logger.**Log**(**message*, *stacklevel=1*)

Logs a message with level OUTPUT.

pyunity.logger.**LogLine**(*level*, **message*, *stacklevel=1*, *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*, *stacklevel=1*, *silent=False*)

Log an exception.

Parameters
e (*Exception*) – Exception to log

pyunity.logger.**LogTraceback**(*exctype*, *value*, *tb*)

Log an exception.

Parameters

- **exctype** (*type*) – Type of exception that is to be raised
- **value** (*Any*) – Value of the exception contents
- **tb** (*traceback*) – Traceback object to log

Notes

This function is not meant to be used by general users.

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.

class pyunity.logger.**TempRedirect**

Bases: object

get()

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

pyunity.logger.**ResetStream**()

pyunity.meshes module

Source code: pyunity/meshes.py

Module for meshes created at runtime and their various attributes.

class pyunity.meshes.**Mesh**

Bases: *Asset*

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

GetAssetFile(*gameObject*)

SaveAsset(*ctx*)

static quad(*size*)

Creates a quadrilateral mesh.

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

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

Return type
Mesh

static doubleQuad(*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 cylinder(*radius*, *height*, *detail=32*)

static sphere(*size*, *detail=16*)

static capsule(*radius*, *height*, *detail=16*)

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

class pyunity.meshes.**Material**

Bases: *Asset*

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

GetAssetFile(*gameObject*)

SaveAsset(*ctx*)

class pyunity.meshes.**Color**

Bases: object

toString()

static fromString(*string*)

class pyunity.meshes.**RGB**

Bases: *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)

toRGB()

toHSV()

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

class pyunity.meshes.**HSV**

Bases: *Color*

A class to represent a HSV color.

Parameters

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

toRGB()

toHSV()

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

class pyunity.meshes.**MeshRenderer**

Bases: *SingleComponent*

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

mesh = None

Mesh that the MeshRenderer will render.

Type
Mesh

mat = <pyunity.meshes.Material object at 0x7f7af4d004c0>

Material to use for the mesh

Type
Material

DefaultMaterial = <pyunity.meshes.Material object>

Type: *Material*

Render()

Render the mesh that the MeshRenderer has.

pyunity.render module

Source code: pyunity/render.py

Classes to aid in rendering in a Scene.

pyunity.render.**fillScreen**(*scale=1*)

class pyunity.render.**Shader**

Bases: object

VERSION = '1.10'

Type: str

loadCache(*file*)

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** (`Vector3`) – Value of uniform variable

setMat3(*var*, *val*)

Set a `mat3` uniform variable.

Parameters

- **var** (*bytes*) – Variable name
- **val** (*glm.mat3*) – Value of uniform variable

setMat4(*var*, *val*)

Set a `mat4` uniform variable.

Parameters

- **var** (*bytes*) – Variable name
- **val** (*glm.mat4*) – Value of uniform variable

setInt(*var*, *val*)

Set an `int` uniform variable.

Parameters

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

setFloat(*var*, *val*)

Set a `float` uniform variable.

Parameters

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

use()

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

pyunity.render.**compileShaders**()

pyunity.render.**compileSkyboxes**()

pyunity.render.**resetShaders**()

pyunity.render.**resetSkyboxes**()

class pyunity.render.**LightType**

Bases: IntEnum

An enumeration.

Point = 0

Type: int

Directional = 1

Type: int

Spot = 2

Type: int

class pyunity.render.**Light**

Bases: *SingleComponent*

Component to hold data about the light in a scene.

intensity = 20

Intensity of light

Type
int

color = RGB(255, 255, 255)

Light color (will mix with material color)

Type
Color

type = LightType.Directional

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

Type
LightType

setupBuffers(*depthMapSize*)

class pyunity.render.**Camera**

Bases: *SingleComponent*

Component to hold data about the camera in a scene.

near = 0.05

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

Type
float

far = 200.0

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

Type
float

clearColor = None

The clear color of the camera. Defaults to black.

Type
Color

shader = <pyunity.render.Shader object at 0x7f7af4c8e5e0>

The shader to use for 3D objects.

Type
Shader

skyboxEnabled = True

Toggle skybox on or off. Defaults to True.

Type
bool

skybox = <pyunity.files.Skybox object at 0x7f7af65d5f70>

Selected skybox to render.

Type
Skybox

ortho = False

Orthographic or perspective proection. Defaults to False.

Type
bool

customProjMat

If not None, will be used over any other type of projection matrix. Unavailable from the Editor and also not saved.

Type
glm.mat4 or None

shadows = False

Whether to render depthmaps and use them. Defaults to True.

Type
bool

canvas = None

Target canvas to render. Defaults to None.

Type
Canvas

depthMapSize = 1024

Depth map texture size. Do not modify after scene has started. Defaults to 1024.

Type
int

setupBuffers()

Creates 2D quad VBO and VAO for GUI.

property fov

FOV of camera

property 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.

SetupShader(*lights*)

SetupDepthShader(*light*)

Draw(*renderers*)

Draw specific renderers, taking into account light positions.

Parameters

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

DrawDepth(*renderers*)

RenderDepth(*renderers*, *lights*)

RenderScene(*renderers*, *lights*)

Render(*renderers*, *lights*)

RenderSkybox()

Render2D()

Draw all Image2D and Text components in the Camera's target canvas.

If the Camera has no Canvas, this function does nothing.

Setup2D()

Draw2D(*renderers*)

class pyunity.render.Screen

Bases: object

width = 800

Type: int

height = 500

Type: int

size = Vector2(800, 500)

Type: *Vector2*

aspect = 1.6

Type: float

pyunity.resources module

Source code: pyunity/resources.py

---

pyunity.resources.**getPath**(*local*)

pyunity.settings module

Source code: pyunity/settings.py

---

class pyunity.settings.**LiveDict**

Bases: object

update()

todict()

keys()

values()

items()

pop(*item*)

class pyunity.settings.**Database**

Bases: *LiveDict*

update()

refresh()

6.1.3 Module contents

Source code: pyunity/__init__.py

Version 0.9.0 (in development)

PyUnity is a pure Python 3D Game Engine that was inspired by the structure of the Unity Game Engine. It aims to be as close as possible to Unity itself. 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.

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.

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 install pyunity
```

Alternatively, you can clone the repository here to build the package from source. Then use `setup.py` to build. Note that it will install Cython to compile.

```
> pip 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
> pip install .
```

Its only dependencies are PyOpenGL, PySDL2, Pillow and PyGLM. Microsoft Visual C++ Build Tools are required on Windows for building yourself, but it can be disabled by setting the `cython` environment variable to `0`, at the cost of being less optimized. GLFW can be optionally installed if you would like to use the GLFW window provider.

Importing

To start using PyUnity, you must import it. A standard way to import is like so:

```
>>> from pyunity import *
```

Debug information is turned on by default. If you want to turn it off, set the PYUNITY_DEBUG_MODE environment variable to `"0"`. This is the output with debugging:

```
Loaded config
Trying GLFW as a window provider
GLFW doesn't work, trying PySDL2
Using window provider PySDL2
Loaded PyUnity version 0.9.0
```

If debugging is off, there is no output:

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

Scenes

All PyUnity projects start with a scene. To add a scene, do this:

```
>>> scene = SceneManager.AddScene("Scene 1")
```

Then, let's move the camera backwards 10 units.

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

Finally, add a cube at the origin:

```
>>> cube = GameObject("Cube")
>>> renderer = cube.AddComponent(MeshRenderer)
>>> renderer.mesh = Mesh.cube(2)
>>> renderer.mat = Material(RGB(255, 0, 0))
>>> scene.Add(cube)
```

To see what you have added to the scene, call `scene.List()`:

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

Finally, to run the scene, call `scene.Run()`. The window that is created is one of FreeGLUT, GLFW or PySDL2. The window is selected on module initialization (see Windows subheading).

Behaviours

To create your own PyUnity script, create a class that inherits from Behaviour. Usually in Unity, you would put the class in its own file, but Python can't do something like that, so put all of your scripts in one file. Then, to add a script, just use `AddComponent()`. Do not put anything in the `__init__` function, instead put it in `Start()`. The `Update()` function receives one parameter, `dt`, which is the same as `Time.deltaTime` in Unity.

Windows

The window is provided by one of three providers: GLFW, PySDL2 and FreeGLUT. When you first import PyUnity, it checks to see if any of the three providers work. The testing order is as above, so FreeGLUT 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 would like to contribute a new window provider, then please create a pull request.

Environment variables

Here is a list of environment variables used by PyUnity:

- **PYUNITY_TESTING** (default: unset) When set, the following features are either disabled or ignored:
  - Window provder selection
  - Audio
  - Font loading
- **PYUNITY_DEBUG_MODE** (default: 1) Disables debug output if set to "0". Debug output has the code |D| in the log file.
- **PYUNITY_AUDIO** (default: 1) If set to "0", sdlmixer won't be loaded and `config.audio` is set to `False`.
- **PYUNITY_GL_CONTEXT** (default: unset) Set when the OpenGL context is enabled. Usually not used except by wrapper scripts as Behaviours only update while a valid context exists.
- **PYUNITY_CHECK_WINDOW** (default: 0) If set to "1", forces window provider selection regardless if `windowProvider` is set in `settings.json`. If set to "0", window provider selection is triggered only if `windowProvider` doesn't already exist in `settings.json`.
- **PYUNITY_INTERACTIVE** (default: 1) If set to "0", window providing is disabled and scenes are run without any OpenGL rendering.
- **PYUNITY_SPHINX_CHECK** (default: unset) Used by sphinx to fix some bugs that occur during documentation generation.
- **PYUNITY_CHANGE_MODULE** (default: 1) Change the `__module__` attribute of all imported objects to `pyunity`. If set to "0", this is disabled.

Examples

Examples are located at subfolders in pyunity/examples so do be sure to check them out as a starting point.

To run an example, import it like so:

```
>>> from pyunity.examples.example1 import main
Loaded config
Trying FreeGLUT as a window provider
FreeGLUT doesn't work, trying GLFW
GLFW doesn't work, trying PySDL2
```

(continues on next page)

(continued from previous page)

```
Using window provider PySDL2
Loaded PyUnity version 0.9.0
>>> main()
```

Or from the command line:

```
> python -m pyunity 1
```

The 1 just means to load example 1, and there are 9 examples. To load all examples one by one, do not specify a number. If you want to contribute an example, then please create a pull request.

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z