

---

# **PyUnity**

***Release 0.2.1***

**Jun 24, 2021**



---

## Contents:

---

|          |                                    |           |
|----------|------------------------------------|-----------|
| <b>1</b> | <b>Installing</b>                  | <b>3</b>  |
| 1.1      | Releases                           | 3         |
| 1.1.1    | v0.2.0                             | 3         |
| 1.1.2    | v0.1.0                             | 4         |
| 1.1.3    | v0.0.5                             | 4         |
| 1.1.4    | v0.0.4                             | 5         |
| 1.1.5    | v0.0.3                             | 5         |
| 1.1.6    | v0.0.2                             | 5         |
| 1.2      | Tutorials                          | 6         |
| 1.2.1    | Tutorial 1: The Basics             | 6         |
| 1.2.1.1  | What is PyUnity?                   | 6         |
| 1.2.1.2  | Basic concepts                     | 6         |
| 1.2.1.3  | Transforms                         | 6         |
| 1.2.1.4  | Code                               | 6         |
| 1.2.1.5  | Rotation                           | 7         |
| 1.2.2    | Tutorial 2: Rendering in Scenes    | 7         |
| 1.2.2.1  | Scenes                             | 7         |
| 1.2.2.2  | Meshes                             | 8         |
| 1.2.2.3  | The MeshRenderer                   | 8         |
| 1.2.2.4  | Debugging                          | 10        |
| 1.2.3    | Tutorial 3: Scripts and Behaviours | 10        |
| 1.2.3.1  | Behaviours                         | 11        |
| 1.2.3.2  | Behaviours vs Components           | 11        |
| 1.2.3.3  | Examples                           | 11        |
| 1.3      | License                            | 12        |
| 1.4      | API reference                      | 13        |
| 1.4.1    | PyUnity package                    | 13        |
| 1.4.1.1  | Version 0.2.1 (in development)     | 13        |
| 1.4.1.2  | Subpackages                        | 15        |
| 1.4.1.3  | Submodules                         | 21        |
| <b>2</b> | <b>Indices and tables</b>          | <b>35</b> |
|          | <b>Python Module Index</b>         | <b>37</b> |
|          | <b>Index</b>                       | <b>39</b> |



PyUnity is a Python implementation of the Unity Engine, written in C++. This is just a fun project and many features have been taken out to make it as easy as possible to create a scene and run it.



# CHAPTER 1

---

## Installing

---

To install PyUnity, use pip:

```
> pip install pyunity
```

Its dependencies are just OpenGL, Pygame and GLFW.

Alternatively, you can clone the repository [here](#). Then run setup.py:

```
> python setup.py install
```

Sometimes on Linux machines, Pygame cannot be installed via pip: in that case, use the package manager. For example, on Ubuntu:

```
> sudo apt-get install python3-pygame
```

## 1.1 Releases

### 1.1.1 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/rayzchen/pyunity/releases/tag/0.2.0>

### 1.1.2 v0.1.0

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

Features:

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

Notes:

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

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

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

### 1.1.3 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/rayzchen/pyunity/releases/tag/0.0.5>

### **1.1.4 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/rayzchen/pyunity/releases/tag/0.0.4>

### **1.1.5 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/rayzchen/pyunity/releases/tag/0.0.3>

### **1.1.6 v0.0.2**

First proper release (v0.0.1 was lost).

Features:

- Documentation

- Meshes

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

## 1.2 Tutorials

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

### 1.2.1 Tutorial 1: The Basics

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

#### 1.2.1.1 What is PyUnity?

PyUnity is a Python port of the [UnityEngine](#), which was originally written in C++. PyUnity has been modified to be easy to use in Python, which means that some features have been removed.

#### 1.2.1.2 Basic concepts

In PyUnity, everything will belong to a `GameObject`. A `GameObject` is a named object that has lots of different things on it that will affect the `GameObject` and other `GameObjects`. Each `GameObject` has its own `Components`, which are like the hardware in a computer. These `Components` can do all sorts of things.

#### 1.2.1.3 Transforms

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

A `Transform` can also have a child. This child is also a `GameObject`'s component. 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.

#### 1.2.1.4 Code

All of that has now been established, so let's start to program 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 Pygame
Trying Pygame as a window provider
Using window provider Pygame
Loaded PyUnity version 0.2.1
```

The output beneath the import is just debug statement, you can turn it off with the environment variable `PYUNITY_DEBUG_INFO` set to `"0"`.

Now we have 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.

### 1.2.1.5 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 what to do, only use the `localEulerAngles` property.

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

## 1.2.2 Tutorial 2: Rendering in Scenes

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.

### 1.2.2.1 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.

### 1.2.2.2 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 3 pieces of data: the vertices (or points), the faces and the normals. Normals are just vectors saying which way the face is pointing.

For this, we don't want to have 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.

### 1.2.2.3 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 colour in RGB.

```
>>> renderer.mat = Material((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:

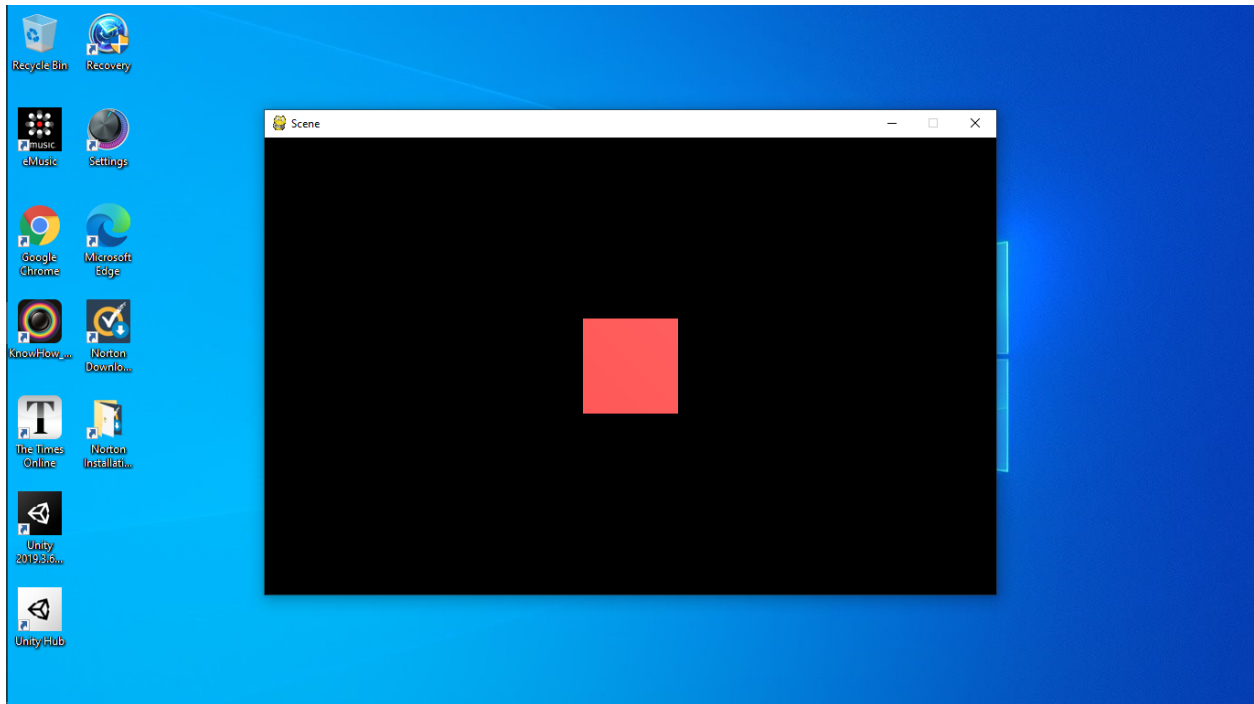
```
>>> from pyunity import *
Loaded config
Trying GLFW as a window provider
GLFW doesn't work, trying Pygame
Trying Pygame as a window provider
Using window provider Pygame
Loaded PyUnity version 0.2.1
>>> scene = SceneManager.AddScene("Scene")
```

(continues on next page)

(continued from previous page)

```
>>> 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((255, 0, 0))
>>> scene.Add(cube)
```

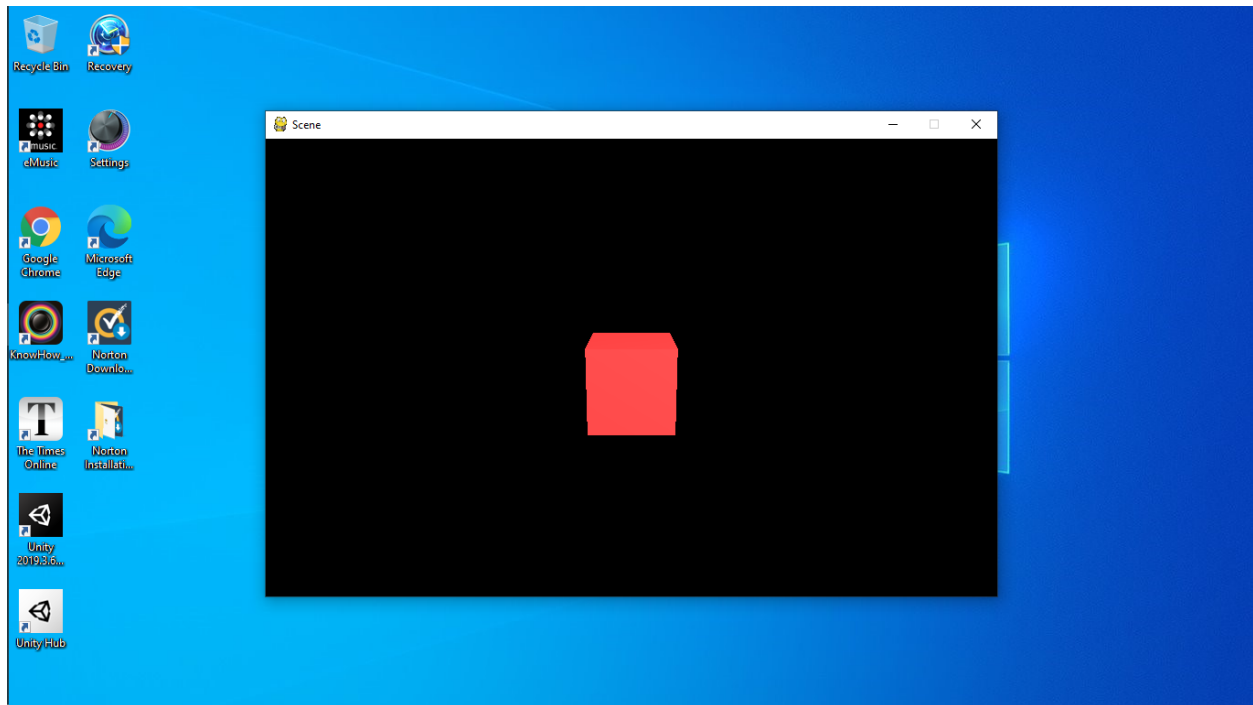
Then, to run our scene, we use `scene.Run()`. And now we have a cube:



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:



### 1.2.2.4 Debugging

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

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

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

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

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

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

Next tutorial, we'll be covering scripts and Behaviours.

## 1.2.3 Tutorial 3: Scripts and Behaviours

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

### 1.2.3.1 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 has passed since last frame.

### 1.2.3.2 Behaviours vs Components

Look at the code for the Component class:

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

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

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

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

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

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

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

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

### 1.2.3.3 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((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, dt):
        self.a += 1
        if self.a == 10:
            print(self.transform.position - self.lastPos)
            print(self.transform.rotation.conjugate * self.lastRot)
            print(self.transform.scale / self.lastScl)
            self.a = 0
```

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

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

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

In the next tutorial we'll be looking at physics.

## 1.3 License

MIT License

Copyright (c) 2020 Ray Chen

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



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

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

## 1.4 API reference

### 1.4.1 PyUnity package

#### 1.4.1.1 Version 0.2.1 (in development)

A Python implementation of the Unity Engine that supports different types of windows. Still in development.

#### Importing

The first step in using PyUnity is always importing 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 FreeGLUT as a window provider
FreeGLUT doesn't work, trying GLFW
GLFW doesn't work, trying Pygame
Using window provider Pygame
Loaded PyUnity version 0.2.1
```

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((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 Pygame. 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`.

## Windows

The window is provided by one of three providers: GLFW, Pygame 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 Pygame is tested last.

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

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

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

## Examples

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 Pygame
Using window provider Pygame
Loaded PyUnity version 0.2.1
>>> main()
```

Or from the command line:

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

The 1 just means to load example 1, and there are 8 examples. To load all examples one by one, do not specify a number. If you want to contribute an example, then please create a new pull request.

`pyunity.timer(func)`

Use this decorator to print how long a function takes.

### 1.4.1.2 Subpackages

#### pyunity.physics package

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

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

#### Example

```
>>> cube = GameObject("Cube")
>>> collider = cube.AddComponent(AABBoxCollider)
>>> collider.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 *gravity* set to True.

#### Submodules

##### pyunity.physics.core module

Core classes of the PyUnity physics engine.

**class** `pyunity.physics.core.AABBoxCollider`

Bases: `pyunity.physics.core.Collider`

An axis-aligned box collider that cannot be deformed.

**min**

The corner with the lowest coordinates.

**Type** `Vector3`

**max**

The corner with the highest coordinates.

**Type** `Vector3`

**pos**

The center of the AABBBoxCollider

**Type** *Vector3*

**CheckOverlap** (*other*)

Checks to see if the bounding box of two colliders overlap.

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

**Returns** Whether they are overlapping or not

**Return type** bool

**SetSize** (*min*, *max*)

Sets the size of the collider.

**Parameters**

- **min** (*Vector3*) – The corner with the lowest coordinates.
- **max** (*Vector3*) – The corner with the highest coordinates.

**collidingWith** (*other*)

Check to see if the collider is colliding with another collider.

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

**Returns** Collision data

**Return type** *Manifold* or None

## Notes

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

To check against a SphereCollider, the check is as follows:

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

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

Bases: object

Manages the collisions between all colliders.

**rigidbodies**

Dictionary of rigidbodies and the colliders on the GameObject that the Rigidbody belongs to

**Type** dict

**dummyRigidbody**

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

**Type** *Rigidbody*

**AddPhysicsInfo** (*scene*)

Get all colliders and rigidbodies from a specified scene. This overwrites the collider and rigidbody lists, and so can be called whenever a new collider or rigidbody is added or removed.

**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.

**CheckCollisions** ()

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

**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

**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, so that it is more precise.

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

Bases: *pyunity.core.Component*

Collider base class.

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

Bases: object

Class to store collision data.

**Parameters**

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

**class** pyunity.physics.core.**PhysicMaterial** (*restitution=0.75, friction=1*)

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*

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

Bases: `pyunity.core.Component`

Class to let a GameObject follow physics rules.

**mass**

Mass of the Rigidbody. Defaults to 100

**Type** *int* or *float*

**velocity**

Velocity of the Rigidbody

**Type** *Vector3*

**physicMaterial**

Physics material of the Rigidbody

**Type** *PhysicMaterial*

**position**

Position of the Rigidbody. It is assigned to its GameObject's position when the CollHandler is created

**Type** *Vector3*

**AddForce** (*force*)

Apply a force to the center of the Rigidbody.

**Parameters** **force** (*Vector3*) – Force to apply

**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.

**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

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

Bases: `pyunity.physics.core.Collider`

A spherical collider that cannot be deformed.

**min**

The corner with the lowest coordinates.

**Type** *Vector3*

**max**

The corner with the highest coordinates.

**Type** *Vector3*

**pos**

The center of the SphereCollider

**Type** *Vector3*

**radius**

The radius of the SphereCollider

**Type** *Vector3*

**CheckOverlap** (*other*)

Checks to see if the bounding box of two colliders overlap.

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

**Returns** Whether they are overlapping or not

**Return type** `bool`

**SetSize** (*radius, offset*)

Sets the size of the collider.

**Parameters**

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

**collidingWith** (*other*)

Check to see if the collider is colliding with another collider.

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

**Returns** Collision data

**Return type** *Manifold* or `None`

## Notes

To check against another SphereCollider, the distance and the sum of the radii is checked.

To check against an AABBoxColider, the check is as follows:

1. The sphere's center is checked to see if it is inside the AABB.

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

```
pyunity.physics.core.infinity = inf
```

A representation of infinity

### pyunity.window package

#### pyunity.window

A module used to load the window providers.

#### Windows

The window is provided by one of three providers: GLFW, Pygame 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 Pygame is tested last.

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

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

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

```
pyunity.window.LoadWindowProvider()
```

Loads an appropriate window provider to use

```
pyunity.window.glfwCheck()
```

Checks to see if GLFW works

```
pyunity.window.glutCheck()
```

Checks to see if FreeGLUT works

```
pyunity.window.pygameCheck()
```

Checks to see if Pygame works

#### Submodules

#### pyunity.window.glfwWindow module

```
class pyunity.window.glfwWindow.Window(config, name)
```

Bases: object

A window provider that uses GLFW.

**Raises** PyUnityException – If the window creation fails



**start** (*update\_func*)

Start the main loop of the window.

**Parameters** **update\_func** (*function*) – The function that calls the OpenGL calls.

### pyunity.window.glutWindow module

**class** pyunity.window.glutWindow.**Window** (*config, name*)

Bases: object

A window provider that uses FreeGLUT.

**display** ()

Function to render in the scene.

**schedule\_update** (*t*)

Starts the window refreshing.

**start** (*update\_func*)

Start the main loop of the window.

**Parameters** **update\_func** (*function*) – The function that calls the OpenGL calls.

### pyunity.window.pygameWindow module

**class** pyunity.window.pygameWindow.**Window** (*config, name*)

Bases: object

A window provider that uses PyGame.

**start** (*update\_func*)

Start the main loop of the window.

**Parameters** **update\_func** (*function*) – The function that calls the OpenGL calls.

#### 1.4.1.3 Submodules

### pyunity.audio module

Classes to manage the playback of audio.

**class** pyunity.audio.**AudioClip** (*file*)

Bases: object

Class to store information about an audio file.

**file**

Name of the file

**Type** str

**sound**

Sound file that can be played with a `pygame.mixer.Channel`. Only set when the AudioClip is in an AudioSource in a running scene.

**Type** pygame.mixer.Sound

**SetSound** (*file*)

Changes the audio file.

**Parameters** **file** (*str*) – Name of the audio file Must be a .ogg file, which can work on any platform.

**Raises**

- `PyUnityException` – If the provided file is not an OGG audio file
- `TypeError` – If the provided file is not of type `str`

**class** `pyunity.audio.AudioSource`

Bases: `pyunity.core.Component`

Manages playback on an AudioSource.

**clip**

Clip to play. Best way to set the clip is to use the `SetClip` function.

**Type** `AudioClip`

**PlayOnStart**

Whether it plays on start or not.

**Type** `bool`

**Loop**

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

**Type** `bool`

**Pause ()**

Pauses the current clip.

**Play ()**

Plays the current clip.

**SetClip (clip)**

Sets the clip to play.

**Parameters** **clip** (`AudioClip`) – Clip to set

**Raises** `TypeError` – If the provided clip is not of type `AudioClip`

**Stop ()**

Stop the current clip.

**UnPause ()**

Unpauses the current clip.

## **pyunity.core module**

Core classes for the PyUnity library.

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

## **Example**

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

```

>>> from pyunity import * # Import
Loaded config
Trying GLFW as a window provider
GLFW doesn't work, trying Pygame
Trying Pygame as a window provider
Using window provider Pygame
Loaded PyUnity version 0.2.1
>>> mat = Material((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(2, 2, 2) path="/Root/Child1">, <pyunity.core.MeshRenderer object at
↳ 0x0B14FCB8>]
>>> child2.transform.children # List child2's children
[<Transform position=Vector3(0, 5, 0) rotation=Quaternion(1, 0, 0, 0) scale=Vector3(3,
↳ 3, 3) path="/Root/Child2/Grandchild">]

```

### **class** pyunity.core.Behaviour

Bases: *pyunity.core.Component*

Base class for behaviours that can be scripted.

#### **gameObject**

GameObject that the component belongs to.

Type *GameObject*

#### **transform**

Transform that the component belongs to.

Type *Transform*

#### **Start()**

Called every time a scene is loaded up.

#### **Update(dt)**

Called every frame.

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

### **class** pyunity.core.Camera

Bases: *pyunity.core.Component*

Component to hold data about the camera in a scene.

**fov**

Fov in degrees measured horizontally. Defaults to 90.

**Type** int

**near**

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

**Type** float

**far**

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

**Type** float

**clearColor**

Tuple of 4 floats of the clear color of the camera. Defaults to (.1, .1, .1, 1). Color mode is RGBA.

**Type** tuple

**class** pyunity.core.Component

Bases: object

Base class for built-in components.

**gameObject**

GameObject that the component belongs to.

**Type** *GameObject*

**transform**

Transform that the component belongs to.

**Type** *Transform*

**AddComponent** (*component*)

Calls *AddComponent* on the component's GameObject.

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

**GetComponent** (*component*)

Calls *GetComponent* on the component's GameObject.

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

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

Bases: object

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

**Parameters**

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

**name**

Name of the GameObject

**Type** str

**components**

List of components

**Type** list

**tag**

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

**Type** *Tag*

**transform**

Transform that belongs to the GameObject

**Type** *Transform*

**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, do a manual loop.

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

**class** pyunity.core.**Light**

Bases: *pyunity.core.Component*

Component to hold data about the light in a scene.

**class** pyunity.core.**Material** (*color*)

Bases: object

Class to hold data on a material.

**color**

A list or tuple of 4 floats that make up a RGBA color.

**Type** list or tuple

**class** pyunity.core.**MeshRenderer**

Bases: *pyunity.core.Component*

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

**mesh**

Mesh that the MeshRenderer will render.

**Type** *Mesh*

**mat**

Material to use for the mesh

**Type** *Material*

**render** ()

Render the mesh that the MeshRenderer has.

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

Bases: object

Class to group GameObjects together without referencing the tags.

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

**Raises**

- *ValueError* – If there is no tag name

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

**tagName**

Tag name

**Type** str**tag**

Tag index of the list of tags

**Type** int**static 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.TransformBases: *pyunity.core.Component*

Class to hold data about a GameObject's transformation.

**gameObject**

GameObject that the component belongs to.

**Type** *GameObject***localPosition**

Position of the Transform in local space.

**Type** *Vector3***localRotation**

Rotation of the Transform in local space.

**Type** *Quaternion***localScale**

Scale of the Transform in local space.

**Type** *Vector3***parent**

Parent of the Transform. The hierarchical tree is actually formed by the Transform, not the GameObject.

**Type** *Transform* or None**children**

List of children

**Type** list**FullPath ()**

Gets the full path of the Transform.

**Returns** The full path of the Transform.**Return type** str

**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.

**ReparentTo (parent)**

Reparent a Transform.

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

**eulerAngles**

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

**localEulerAngles**

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

**position**

Position of the Transform in world space.

**rotation**

Rotation of the Transform in world space.

**scale**

Scale of the Transform in world space.

`pyunity.core.tags = ['Default']`

List of current tags

**pyunity.errors module**

Module for all exceptions related to PyUnity.

**exception pyunity.errors.ComponentException**

Bases: *pyunity.errors.PyUnityException*

Class for PyUnity exceptions relating to components.

**exception pyunity.errors.GameObjectException**

Bases: *pyunity.errors.PyUnityException*

Class for PyUnity exceptions relating to GameObjects.

**exception pyunity.errors.PyUnityException**

Bases: *Exception*

Base class for PyUnity exceptions.

**exception pyunity.errors.PyUnityWarning**

Bases: *Warning*

Base class for PyUnity warnings.

**pyunity.loader module**

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

`pyunity.loader.LoadMesh (filename)`

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

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

**Returns** Generated mesh

**Return type** *Mesh*

`pyunity.loader.LoadObj(filename)`

Loads a .obj file to a PyUnity mesh.

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

**Returns** A mesh of the object file

**Return type** *Mesh*

`pyunity.loader.LoadScene(sceneName, filePath=None)`

Load a scene from a file. Uses pickle.

**Parameters** `sceneName` (*str*) – Name of the scene, without the .scene extension

**Returns** Loaded scene

**Return type** *Scene*

## Notes

If there already is a scene called *sceneName*, then no scene will be added.

`class pyunity.loader.Primitives`

Bases: object

`capsule = <pyunity.meshes.Mesh object>`

`cube = <pyunity.meshes.Mesh object>`

`cylinder = <pyunity.meshes.Mesh object>`

`double_quad = <pyunity.meshes.Mesh object>`

`quad = <pyunity.meshes.Mesh object>`

`sphere = <pyunity.meshes.Mesh object>`

`pyunity.loader.SaveMesh(mesh, name, filePath=None)`

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

### Parameters

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

`pyunity.loader.SaveScene(scene, filePath=None)`

Save a scene to a file. Uses pickle.

### Parameters

- **scene** (*Scene*) – Scene to save
- **filePath** (*str*, *optional*) – Pass in `__file__` to save in directory of script, otherwise pass in a directory. If not specified, then the scene is saved in the cwd.



**pyunity.meshes module**

Module for prebuilt meshes.

**class** pyunity.meshes.**Mesh** (*verts, triangles, normals*)

Bases: object

Class to create a mesh for rendering with a MeshRenderer

**Parameters**

- **verts** (*list*) – List of Vector3's containing each vertex
- **triangles** (*list*) – List of ints containing triangles joining up the vertexes. Each int is the index of a vertex above.
- **normals** (*list*) – List of Vector3's containing the normal of each triangle. Unlike Unity, PyUnity uses normals per triangle.

**verts**

List of Vector3's containing each vertex

**Type** list

**triangles**

List of ints containing triangles joining up the vertexes. Each int is the index of a vertex above.

**Type** list

**normals**

List of Vector3's containing the normal of each triangle. Unlike Unity, PyUnity uses normals per triangle.

**Type** list

**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*

**static double\_quad** (*size*)

Creates a two-sided quadrilateral mesh.

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

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

**Return type** *Mesh*

**static quad** (*size*)

Creates a quadrilateral mesh.

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

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

**Return type** *Mesh*

**pyunity.scene module****class** pyunity.scene.**Scene** (*name*)

Bases: object

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

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

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

**Add** (*gameObject*)

Add a GameObject to the scene.

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

Lists all the GameObjects currently in the scene.

**Remove** (*gameObject*)

Remove a GameObject from the scene.

**Parameters** **gameObject** (*GameObject*) – GameObject to remove.**Raises** *PyUnityException* – If the specified GameObject is the Main Camera.**Run** ()

Run the scene and create a window for it.

**Start** ()

Start the internal parts of the Scene.

**inside\_frustrum** (*renderer*)

Check if the renderer's mesh can be seen by the main camera.

**Parameters** **renderer** ([MeshRenderer](#)) – Renderer to test

**Returns** If the mesh can be seen

**Return type** bool

**no\_interactive** ()

**render** ()

Renders all GameObjects with MeshRenderers.

**start\_scripts** ()

Start the scripts in the Scene.

**transform** (*transform*)

Transform the matrix by a specified transform.

**Parameters** **transform** ([Transform](#)) – Transform to move

**update** ()

Updating function to pass to the window provider.

**update\_scripts** ()

Updates all scripts in the scene.

`pyunity.scene.SceneManager = <pyunity.scene.SceneManager object>`

Manages all scene additions and changes

## pyunity.vector3 module

A class to store x, y and z values, with a lot of utility functions.

**class** `pyunity.vector3.Vector3` (*x\_or\_list=None, y=None, z=None*)

Bases: `object`

**static back** ()

Vector3 pointing in the negative z axis

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

Clamps a vector between two other vectors, resulting in the vector being as close to the edge of a bounding box created as possible.

**Parameters**

- **min** ([Vector3](#)) – Min vector
- **max** ([Vector3](#)) – Max vector

**copy** ()

Makes a copy of the Vector3

**Returns** A shallow copy of the vector

**Return type** [Vector3](#)

**cross** (*other*)

Cross product of two vectors

**Parameters** **other** ([Vector3](#)) – Other vector

**Returns** Cross product of the two vectors

**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

**static down** ()

Vector3 pointing in the negative y axis

**static forward** ()

Vector3 pointing in the positive z axis

**get\_dist\_sqrd** (*other*)

The distance between this vector and the other vector, squared. It is more efficient to call this than to call *get\_distance* and square it.

**Returns** The squared distance

**Return type** float

**get\_distance** (*other*)

The distance between this vector and the other vector

**Returns** The distance

**Return type** float

**get\_length\_sqrd** ()

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

**Returns** The length of the vector squared

**Return type** float

**int\_tuple**

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

**static left** ()

Vector3 pointing in the negative x axis

**length**

Gets or sets the magnitude of the vector

**normalize\_return\_length** ()

Normalize the vector and return its length before the normalization

**Returns** The length before the normalization

**Return type** float

**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*

**static one** ()

A vector of ones

**static right** ()

Vector3 pointing in the positive x axis

**rounded**

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

**static up()**

Vector3 pointing in the postive y axis

**static zero()**

A vector of zero length

`pyunity.vector3.clamp(x, _min, _max)`

Clamp a value between a minimum and a maximum

**pyunity.quaternion module**

**class** `pyunity.quaternion.Quaternion(w, x, y, z)`

Bases: `object`

Class to represent a 4D Quaternion.

**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

**static Euler** (*vector*)

Create a quaternion using Euler rotations.

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

**Returns** Generated quaternion

**Return type** *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

**RotateVector** (*vector*)

Rotate a vector by the quaternion

**angleAxisPair**

Gets or sets the angle and axis pair.

**Notes**

When getting, it returns a tuple in the form of (*angle, x, y, z*). When setting, assign like `q.eulerAngles = (angle, vector)`.

**conjugate**

The conjugate of a unit quaternion

**copy()**

Deep copy of the Quaternion.

**Returns** A deep copy

**Return type** *Quaternion*

**eulerAngles**

Gets or sets the Euler Angles of the quaternion

**static identity()**

Identity quaternion representing no rotation

**normalized()**

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

**Returns** A unit quaternion

**Return type** *Quaternion*

## CHAPTER 2

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





### p

- `pyunity`, [13](#)
- `pyunity.audio`, [21](#)
- `pyunity.core`, [22](#)
- `pyunity.errors`, [27](#)
- `pyunity.loader`, [27](#)
- `pyunity.meshes`, [29](#)
- `pyunity.physics`, [15](#)
- `pyunity.physics.core`, [15](#)
- `pyunity.quaternion`, [33](#)
- `pyunity.scene`, [30](#)
- `pyunity.vector3`, [31](#)
- `pyunity.window`, [20](#)
- `pyunity.window.glfwWindow`, [20](#)
- `pyunity.window.glutWindow`, [21](#)
- `pyunity.window.pygameWindow`, [21](#)



## A

AABBoxCollider (class in *pyunity.physics.core*), 15  
 Add() (*pyunity.scene.Scene* method), 30  
 AddComponent() (*pyunity.core.Component* method), 24  
 AddComponent() (*pyunity.core.GameObject* method), 25  
 AddForce() (*pyunity.physics.core.Rigidbody* method), 18  
 AddImpulse() (*pyunity.physics.core.Rigidbody* method), 18  
 AddPhysicsInfo() (*pyunity.physics.core.CollManager* method), 16  
 AddTag() (*pyunity.core.Tag* static method), 26  
 angleAxisPair (*pyunity.quaternion.Quaternion* attribute), 33  
 AudioClip (class in *pyunity.audio*), 21  
 AudioSource (class in *pyunity.audio*), 22

## B

back() (*pyunity.vector3.Vector3* static method), 31  
 Behaviour (class in *pyunity.core*), 23

## C

Camera (class in *pyunity.core*), 23  
 capsule (*pyunity.loader.Primitives* attribute), 28  
 CheckCollisions() (*pyunity.physics.core.CollManager* method), 17  
 CheckOverlap() (*pyunity.physics.core.AABBoxCollider* method), 16  
 CheckOverlap() (*pyunity.physics.core.SphereCollider* method), 19  
 children (*pyunity.core.Transform* attribute), 26  
 clamp() (in module *pyunity.vector3*), 33  
 clamp() (*pyunity.vector3.Vector3* method), 31

clearColor (*pyunity.core.Camera* attribute), 24  
 clip (*pyunity.audio.AudioSource* attribute), 22  
 Collider (class in *pyunity.physics.core*), 17  
 collidingWith() (*pyunity.physics.core.AABBoxCollider* method), 16  
 collidingWith() (*pyunity.physics.core.SphereCollider* method), 19  
 CollManager (class in *pyunity.physics.core*), 16  
 color (*pyunity.core.Material* attribute), 25  
 combine (*pyunity.physics.core.PhysicMaterial* attribute), 18  
 Component (class in *pyunity.core*), 24  
 ComponentException, 27  
 components (*pyunity.core.GameObject* attribute), 24  
 conjugate (*pyunity.quaternion.Quaternion* attribute), 33  
 copy() (*pyunity.quaternion.Quaternion* method), 33  
 copy() (*pyunity.vector3.Vector3* method), 31  
 cross() (*pyunity.vector3.Vector3* method), 31  
 cube (*pyunity.loader.Primitives* attribute), 28  
 cube() (*pyunity.meshes.Mesh* static method), 29  
 cylinder (*pyunity.loader.Primitives* attribute), 28

## D

display() (*pyunity.window.glutWindow.Window* method), 21  
 dot() (*pyunity.vector3.Vector3* method), 32  
 double\_quad (*pyunity.loader.Primitives* attribute), 28  
 double\_quad() (*pyunity.meshes.Mesh* static method), 29  
 down() (*pyunity.vector3.Vector3* static method), 32  
 dummyRigidbody (*pyunity.physics.core.CollManager* attribute), 16

## E

Euler() (*pyunity.quaternion.Quaternion* static method), 33

eulerAngles (*pyunity.core.Transform* attribute), 27  
 eulerAngles (*pyunity.quaternion.Quaternion* attribute), 34

## F

far (*pyunity.core.Camera* attribute), 24  
 file (*pyunity.audio.AudioClip* attribute), 21  
 FindGameObjectsByName () (*pyunity.scene.Scene* method), 30  
 FindGameObjectsByTagName () (*pyunity.scene.Scene* method), 30  
 FindGameObjectsByTagNumber () (*pyunity.scene.Scene* method), 30  
 forward () (*pyunity.vector3.Vector3* static method), 32  
 fov (*pyunity.core.Camera* attribute), 24  
 friction (*pyunity.physics.core.PhysicMaterial* attribute), 18  
 FromAxis () (*pyunity.quaternion.Quaternion* static method), 33  
 FullPath () (*pyunity.core.Transform* method), 26

## G

GameObject (*class in pyunity.core*), 24  
 gameObject (*pyunity.core.Behaviour* attribute), 23  
 gameObject (*pyunity.core.Component* attribute), 24  
 gameObject (*pyunity.core.Transform* attribute), 26  
 GameObjectException, 27  
 get\_dist\_sqrd () (*pyunity.vector3.Vector3* method), 32  
 get\_distance () (*pyunity.vector3.Vector3* method), 32  
 get\_length\_sqrd () (*pyunity.vector3.Vector3* method), 32  
 GetComponent () (*pyunity.core.Component* method), 24  
 GetComponent () (*pyunity.core.GameObject* method), 25  
 GetRestitution () (*pyunity.physics.core.CollManager* method), 17  
 glfwCheck () (*in module pyunity.window*), 20  
 glutCheck () (*in module pyunity.window*), 20

## I

identity () (*pyunity.quaternion.Quaternion* static method), 34  
 infinity (*in module pyunity.physics.core*), 20  
 inside\_frustum () (*pyunity.scene.Scene* method), 30  
 int\_tuple (*pyunity.vector3.Vector3* attribute), 32

## L

left () (*pyunity.vector3.Vector3* static method), 32

length (*pyunity.vector3.Vector3* attribute), 32  
 Light (*class in pyunity.core*), 25  
 List () (*pyunity.core.Transform* method), 26  
 List () (*pyunity.scene.Scene* method), 30  
 LoadMesh () (*in module pyunity.loader*), 27  
 LoadObj () (*in module pyunity.loader*), 28  
 LoadScene () (*in module pyunity.loader*), 28  
 LoadWindowProvider () (*in module pyunity.window*), 20  
 localEulerAngles (*pyunity.core.Transform* attribute), 27  
 localPosition (*pyunity.core.Transform* attribute), 26  
 localRotation (*pyunity.core.Transform* attribute), 26  
 localScale (*pyunity.core.Transform* attribute), 26  
 Loop (*pyunity.audio.AudioSource* attribute), 22

## M

Manifold (*class in pyunity.physics.core*), 17  
 mass (*pyunity.physics.core.Rigidbody* attribute), 18  
 mat (*pyunity.core.MeshRenderer* attribute), 25  
 Material (*class in pyunity.core*), 25  
 max (*pyunity.physics.core.AABBBoxCollider* attribute), 15  
 max (*pyunity.physics.core.SphereCollider* attribute), 19  
 Mesh (*class in pyunity.meshes*), 29  
 mesh (*pyunity.core.MeshRenderer* attribute), 25  
 MeshRenderer (*class in pyunity.core*), 25  
 min (*pyunity.physics.core.AABBBoxCollider* attribute), 15  
 min (*pyunity.physics.core.SphereCollider* attribute), 19  
 Move () (*pyunity.physics.core.Rigidbody* method), 18  
 MovePos () (*pyunity.physics.core.Rigidbody* method), 19

## N

name (*pyunity.core.GameObject* attribute), 24  
 near (*pyunity.core.Camera* attribute), 24  
 no\_interactive () (*pyunity.scene.Scene* method), 31  
 normalize\_return\_length () (*pyunity.vector3.Vector3* method), 32  
 normalized () (*pyunity.quaternion.Quaternion* method), 34  
 normalized () (*pyunity.vector3.Vector3* method), 32  
 normals (*pyunity.meshes.Mesh* attribute), 29

## O

one () (*pyunity.vector3.Vector3* static method), 32

## P

parent (*pyunity.core.Transform* attribute), 26  
 Pause () (*pyunity.audio.AudioSource* method), 22  
 PhysicMaterial (*class in pyunity.physics.core*), 17

physicMaterial (*pyunity.physics.core.Rigidbody attribute*), 18  
 Play() (*pyunity.audio.AudioSource method*), 22  
 PlayOnStart (*pyunity.audio.AudioSource attribute*), 22  
 pos (*pyunity.physics.core.AABBBoxCollider attribute*), 15  
 pos (*pyunity.physics.core.SphereCollider attribute*), 19  
 position (*pyunity.core.Transform attribute*), 27  
 position (*pyunity.physics.core.Rigidbody attribute*), 18  
 Primitives (*class in pyunity.loader*), 28  
 pygameCheck() (*in module pyunity.window*), 20  
 pyunity (*module*), 13  
 pyunity.audio (*module*), 21  
 pyunity.core (*module*), 22  
 pyunity.errors (*module*), 27  
 pyunity.loader (*module*), 27  
 pyunity.meshes (*module*), 29  
 pyunity.physics (*module*), 15  
 pyunity.physics.core (*module*), 15  
 pyunity.quaternion (*module*), 33  
 pyunity.scene (*module*), 30  
 pyunity.vector3 (*module*), 31  
 pyunity.window (*module*), 20  
 pyunity.window.glfwWindow (*module*), 20  
 pyunity.window.glutWindow (*module*), 21  
 pyunity.window.pygameWindow (*module*), 21  
 PyUnityException, 27  
 PyUnityWarning, 27

## Q

quad (*pyunity.loader.Primitives attribute*), 28  
 quad() (*pyunity.meshes.Mesh static method*), 29  
 Quaternion (*class in pyunity.quaternion*), 33

## R

radius (*pyunity.physics.core.SphereCollider attribute*), 19  
 Remove() (*pyunity.scene.Scene method*), 30  
 render() (*pyunity.core.MeshRenderer method*), 25  
 render() (*pyunity.scene.Scene method*), 31  
 ReparentTo() (*pyunity.core.Transform method*), 27  
 restitution (*pyunity.physics.core.PhysicMaterial attribute*), 18  
 right() (*pyunity.vector3.Vector3 static method*), 32  
 rigidbodies (*pyunity.physics.core.CollManager attribute*), 16  
 Rigidbody (*class in pyunity.physics.core*), 18  
 RotateVector() (*pyunity.quaternion.Quaternion method*), 33  
 rotation (*pyunity.core.Transform attribute*), 27  
 rounded (*pyunity.vector3.Vector3 attribute*), 32  
 Run() (*pyunity.scene.Scene method*), 30

## S

SaveMesh() (*in module pyunity.loader*), 28  
 SaveScene() (*in module pyunity.loader*), 28  
 scale (*pyunity.core.Transform attribute*), 27  
 Scene (*class in pyunity.scene*), 30  
 SceneManager (*in module pyunity.scene*), 31  
 schedule\_update() (*pyunity.window.glfwWindow.Window method*), 21  
 SetClip() (*pyunity.audio.AudioSource method*), 22  
 SetSize() (*pyunity.physics.core.AABBBoxCollider method*), 16  
 SetSize() (*pyunity.physics.core.SphereCollider method*), 19  
 SetSound() (*pyunity.audio.AudioClip method*), 21  
 sound (*pyunity.audio.AudioClip attribute*), 21  
 sphere (*pyunity.loader.Primitives attribute*), 28  
 SphereCollider (*class in pyunity.physics.core*), 19  
 Start() (*pyunity.core.Behaviour method*), 23  
 Start() (*pyunity.scene.Scene method*), 30  
 start() (*pyunity.window.glfwWindow.Window method*), 20  
 start() (*pyunity.window.glutWindow.Window method*), 21  
 start() (*pyunity.window.pygameWindow.Window method*), 21  
 start\_scripts() (*pyunity.scene.Scene method*), 31  
 Step() (*pyunity.physics.core.CollManager method*), 17  
 Stop() (*pyunity.audio.AudioSource method*), 22

## T

Tag (*class in pyunity.core*), 25  
 tag (*pyunity.core.GameObject attribute*), 25  
 tag (*pyunity.core.Tag attribute*), 26  
 tagName (*pyunity.core.Tag attribute*), 26  
 tags (*in module pyunity.core*), 27  
 timer() (*in module pyunity*), 15  
 Transform (*class in pyunity.core*), 26  
 transform (*pyunity.core.Behaviour attribute*), 23  
 transform (*pyunity.core.Component attribute*), 24  
 transform (*pyunity.core.GameObject attribute*), 25  
 transform() (*pyunity.scene.Scene method*), 31  
 triangles (*pyunity.meshes.Mesh attribute*), 29

## U

UnPause() (*pyunity.audio.AudioSource method*), 22  
 up() (*pyunity.vector3.Vector3 static method*), 33  
 Update() (*pyunity.core.Behaviour method*), 23  
 update() (*pyunity.scene.Scene method*), 31  
 update\_scripts() (*pyunity.scene.Scene method*), 31

## V

Vector3 (*class in pyunity.vector3*), 31

`velocity` (*pyunity.physics.core.Rigidbody attribute*),  
18  
`verts` (*pyunity.meshes.Mesh attribute*), 29

## W

`Window` (*class in pyunity.window.glfwWindow*), 20  
`Window` (*class in pyunity.window.glutWindow*), 21  
`Window` (*class in pyunity.window.pygameWindow*), 21

## Z

`zero()` (*pyunity.vector3.Vector3 static method*), 33